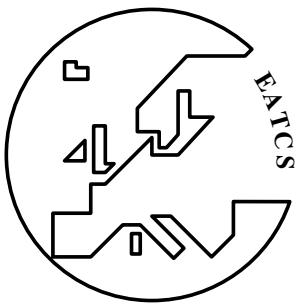
48th International Colloquium on Automata, Languages, and Programming

ICALP 2021, July 12–16, 2021, Glasgow, Scotland (Virtual Conference)

Edited by
Nikhil Bansal
Emanuela Merelli
James Worrell



Editors

Nikhil Bansal
CWI Amsterdam, Netherlands
bansal@gmail.com

Emanuela Merelli
University of Camerino, Italy
emanuela.merelli@unicam.it

James Worrell
University of Oxford, UK
james.ben.worrell@gmail.com

ACM Classification 2012
Theory of Computation

ISBN 978-3-95977-195-5

Published online and open access by
Schloss Dagstuhl – Leibniz-Zentrum für Informatik GmbH, Dagstuhl Publishing, Saarbrücken/Wadern, Germany. Online available at https://www.dagstuhl.de/dagpub/978-3-95977-195-5.

Publication date
July, 2021

Bibliographic information published by the Deutsche Nationalbibliothek
The Deutsche Nationalbibliothek lists this publication in the Deutsche Nationalbibliografie; detailed bibliographic data are available in the Internet at https://portal.dnb.de.

Digital Object Identifier: 10.4230/LIPIcs.ICALP.2021.0

ISBN 978-3-95977-195-5 **ISSN 1868-8969** **https://www.dagstuhl.de/lipics**

LIPIcs – Leibniz International Proceedings in Informatics

LIPIcs is a series of high-quality conference proceedings across all fields in informatics. LIPIcs volumes are published according to the principle of Open Access, i.e., they are available online and free of charge.

ISSN 1868-8969

https://www.dagstuhl.de/lipics

Contents

Track B: Automata, Logic, Semantics, and Theory of Programming

Preface

This volume contains the papers presented at the *48th International Colloquium on Automata, Languages and Programming (**ICALP 2021**)*, held *virtually*, hosted by the University of Glasgow, UK, during July 12–16, 2021. ICALP is a series of annual conferences of the *European Association for Theoretical Computer Science (EATCS)*, which first took place in 1972.

This year, the ICALP program consisted of two tracks:

- Track A: Algorithms, Complexity, and Games
- Track B: Automata, Logic, Semantics, and Theory of Programming

In response to the call for papers, a total of 362 submissions were received: 261 for Track A and 101 for Track B. Each submission was assigned to at least three Program Committee members, aided by 761 external subreviewers. The committees decided to accept 137 papers for inclusion in the scientific program: 108 papers for Track A and 29 for Track B. The selection was made by the Program Committees based on originality, quality, and relevance to theoretical computer science. The quality of the manuscripts was very high indeed, and many deserving papers could not be selected.

The EATCS sponsored awards for both a best paper and a best student paper in each of the two tracks, selected by the Program Committees.

The **best paper awards** were given to the following papers:

Track A: Sayan Bhattacharya and Peter Kiss. *Deterministic Rounding of Dynamic Fractional Matchings.*

Track B: Antoine Amarilli, Louis Jachiet and Charles Paperman. *Dynamic Membership for Regular Languages.*

The **best student paper awards**, for papers that are solely authored by students, were given to the following paper:

Track A: Or Zamir. *Breaking the 2^n barrier for 5-coloring and 6-coloring.*

Track B: *none.*

Apart from the contributed talks, ICALP 2021 included invited presentations by Christel Baier (Technical University of Dresden), Andrei Bulatov (Simon Fraser University, Canada), Keren Censor-Hillel (Technion, Israel), Toniann Pitassi (University of Toronto, Canada), Adi Shamir (Weizmann Institute of Science, Israel), David Woodruff (Carnegie Mellon University, USA).

This volume contains all the contributed papers presented at the conference, papers that accompany the invited talks of Christel Baier, Andrei Bulatov, Keren Censor-Hillel, Adi Shamir, David Woodruff, and an abstract of the invited presentation of Toniann Pitassi.

The program of ICALP 2021 also included presentations of the EATCS Award 2021 to Toniann Pitassi, the Presburger Award 2021 to Shayan Oveis Gharan, the EATCS Distinguished Dissertation Awards to Talya Eden, Marie Fortin, Vera Traub, and the induction of new EATCS Fellows Luca Aceto, Rajeev Alur, Samir Khuller, David Peleg, Davide Sangiorgi, Saket Saurabh.

The following workshops were held as satellite events of ICALP 2021 on July 12, 2021:

- Algorithmic Aspects of Temporal Graphs IV (AATG)
- Verification of Session Types (VEST)
- Programming Research in Mainstream Languages (PRiML)

48th International Colloquium on Automata, Languages, and Programming (ICALP 2021).
Editors: Nikhil Bansal, Emanuela Merelli, and James Worrell
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

- Graph Width Parameters: from Structure to Algorithms (GWP)
- Combinatorial Reconfiguration
- Formal methods education on-line: Tips, Tricks and Tools
- Flavours of Uncertainty in Verification, Planning and Optimization (FUNCTION)

We wish to thank all authors who submitted extended abstracts for consideration, the Program Committees for their scholarly effort, and all the referees who assisted the Program Committees in the evaluation process.

We are also grateful to the Conference General Chair, Simon Gay, and his colleagues from the School of Computing Science, University of Glasgow, for organizing ICALP 2021, and to the Scottish Informatics and Computer Science Alliance (SICSA) for sponsoring participation by PhD students from Scotland.

We would like to thank Anca Muscholl, the Chair of the ICALP Steering Committee, for her continuous support and Artur Czumaj, the president of EATCS, for his generous advice on the organization of the conference.

July 2021

Nikhil Bansal
Emanuela Merelli
James Worrell

Organization

Program Committee

Track A

Nikhil Bansal	CWI Amsterdam, Netherlands, Chair
Yossi Azar	Tel Aviv University, Israel
Luca Becchetti	Sapienza University of Rome, Italy
Alexander Belov	University of Latvia, Latvia
Eric Blais	University of Waterloo, Canada
Niv Buchbinder	Tel Aviv University, Israel
Kevin Buchin	TU Eindhoven, Netherlands
Parinya Chalermsook	Aalto University, Finland
Vincent Cohen-Addad	Google Research, Switzerland
Shahar Dobzinski	Weizmann Institute, Israel
Ran Duan	Tsinghua University, China
Vida Dujmovic	University of Ottawa, Canada
Yuval Filmus	Technion, Israel
Samuel Fiorini	Université libre de Bruxelles, Belgium
Andreas Galanis	University of Oxford, UK
Mika Göös	EPFL, Switzerland
Inge Li Gørtz	TU Denmark, Denmark
Heng Guo	University of Edinburgh, UK
Prahladh Harsha	TIFR, Mumbai, India
Sungjin Im	UC Merced, USA
Stacey Jeffery	CWI Amsterdam, Netherlands
Michael Kapralov	EPFL, Switzerland
Iordanis Kerenidis	CNRS – Université Paris Diderot, France
Stefan Kratsch	HU Berlin, Germany
Ravi Kumar	Google Research, USA
Silvio Lattanzi	Google Research, Switzerland
Shi Li	SUNY Buffalo, USA
Konstantin Makarychev	Northwestern University, USA
Marcin Mucha	University of Warsaw, Poland
Wolfgang Mulzer	FU Berlin, Germany
Jesper Nederlof	Utrecht University, Netherlands
Aleksandar Nikolov	University of Toronto, Canada
Neil Olver	LSE, UK
Rasmus Pagh	IT University of Copenhagen, Denmark
Merav Parter	Weizmann Institute, Israel
Alexandros Psomas	Purdue University, USA
Barna Saha	UC Berkeley, USA
Thatchaphol Saranurak	University of Michigan, USA
Rahul Savani	University of Liverpool, UK
Mohit Singh	Georgia Tech, USA
Sahil Singla	IAS/Princeton, USA
Noah Stephens-Davidowitz	Cornell University, USA
László Végh	LSE, UK
Meirav Zehavi	Ben-Gurion University, Israel

Track B

James Worrell	University of Oxford, UK, Chair
Parosh Aziz Abdulla	Uppsala University, Sweden
S. Akshay	Indian Institute of Technology Bombay, India
Nathalie Bertrand	Inria Rennes, France
Michael Blondin	Université de Sherbrooke, Canada
Olivier Carton	IRIF, Université de Paris, France
Corina Cîrstea	University of Southampton, UK
Ugo Dal Lago	University of Bologna and Inria, Italy
Dana Fisman	Ben Gurion University, Israel
Paul Gastin	LSV, ENS Paris-Saclay, France
Stefan Göller	University of Kassel, Germany
Radha Jagadeesan	DePaul University Chicago, USA
Bakhadyr Khoussainov	University of Auckland, New Zealand
Emanuel Kieronski	Wroclaw University, Poland
Bartek Klin	Warsaw University, Poland
Barbara König	University of Duisburg-Essen, Germany
Laura Kovacs	Vienna University of Technology, Austria
Christoph Löding	RWTH Aachen University, Germany
Sebastian Maneth	University of Bremen, Germany
Richard Mayr	University of Edinburgh, UK
Annabelle McIver	Macquarie University, Australia
Madhavan Mukund	Chennai Mathematical Institute, India
Sophie Pinchinat	IRISA, Université de Rennes, France
Cristian Riveros	Pontificia Universidad Católica de Chile, Chile
Davide Sangiorgi	University of Bologna and Inria, Italy
Lijun Zhang	Institute of Software, Chinese Academy of Sciences, China

Organizing Committee

Simon Gay	University of Glasgow, Chair
Oana Andrei	University of Glasgow
Ornela Dardha	University of Glasgow
Jessica Enright	University of Glasgow
David Manlove	University of Glasgow
Ciaran McCreesh	University of Glasgow
Kitty Meeks	University of Glasgow
Alice Miller	University of Glasgow
Gethin Norman	University of Glasgow
Sofiat Olaosebikan	University of Glasgow
Michele Sevegnani	University of Glasgow

Steering Committee

Anca Muscholl	Bordeaux University, France, Chair
Christel Baier	TU Dresden, Germany
Javier Esparza	TUM Munich, Germany
Paola Flocchini	University of Ottawa, Canada
Leslie Ann Goldberg	Oxford University, United Kingdom
Thore Husfeldt	Lund University, Sweden and
	IT University of Copenhagen, Denmark
Giuseppe Italiano	Università di Roma Tor Vergata, Italy
Stefano Leonardi	Sapienza University of Rome, Italy
Emanuela Merelli	University of Camerino, Italy
Luke Ong	Oxford University, United Kingdom
Paul Spirakis	University of Liverpool, United Kingdom and
	University of Patras, Greece
Christos Zaroliagis	University of Patras and CTI, Greece

Financial Sponsors

University of Glasgow
Scottish Informatics and Computer Science Alliance (SICSA)

Additional Reviewers

Sivert Aasnaess
Samson Abramsky
Bahareh Afshari
Susanne Albers
Helmut Alt
Pablo Andres-Martinez
Spyros Angelopoulos
Diego Arroyuelo
Hagit Attiya
Guy Avnj
Miriam Backens
Eric Balkanski
Yair Bartal
Saugata Basu
Curtis Bechtel
Dylan Bellier
Benjamin Aram Berendsohn
Dietmar Berwanger
Aditya Bhaskara
Kshipra Bhawalkar
Hadley Black
Hans L. Bodlaender
Benedikt Bollig
Xavier Bonnetain
Elette Boyle
Vladimir Braverman
Karl Bringmann
Sam Buss
Christopher Cade
Arnaud Carayol
Arnaud Casteigts
Keren Censor-Hillel
Jérémie Chalopin
Karthekeyan Chandrasekaran
Evangelos Chatziafratis
Lijie Chen
Kuan Cheng
Nai-Hui Chia
Man Kwun Chiu
Marek Chrobak
Jonas Cleve
Ilan Cohen
Arjan Cornelissen
Radu Curticapean
Yuval Dagan
Chen Dan
Debarati Das
Amir Abboud
Marek Adamczyk
Hee-Kap Ahn
Maryam Aliakbarpour
Antoine Amarilli
Haris Angelidakis
Joran van Apeldoorn
Srinivasan Arunachalam
Martin Avanzini
Jasmijn Baaijens
Arturs Backurs
Lucas Bang
Libor Barto
Tugkan Batu
Ryan Beckett
Michael Benedikt
Christoph Berkholz
Siddharth Bhandari
Anup Bhattacharya
Marcin Bienkowski
Markus Bläser
Greg Bodwin
Marthe Bonamy
Adam Bouland
Jendrik Brachter
Alex Bredariol Grilo
Joshua Brody
Jaroslaw Byrka
Clément Canonne
Marco Carmosino
Simon Castellan
Deeparnab Chakrabarty
T-H. Hubert Chan
Zachary Chase
Vaggos Chatziafratis
Sitan Chen
Yun Kuen Cheung
Flavio Chierichetti
Aruni Choudhary
Lorenzo Clemente
Christian Coester
Sarel Cohen
Daniel Cranston
Wojciech Czerwiński
Francesco Dagnino
Stefan Dantchev
Syamantak Das
Karthik Abinav Sankararaman
Bharat Adsul
Gorjan Alagic
Josh Alman
Alexandr Andoni
Patrizio Angelini
Manuel Aprile
Sepehr Assadi
Chen Avin
Yakov Babichenko
Patrick Baillot
Hideo Bannai
Henning Basold
Veronica Becher
Soheil Behnezhad
Huxley Bennett
Aaron Bernstein
Amey Bhangale
Sayan Bhattacharya
Philip Bille
Achim Blumensath
Andrej Bogdanov
Vincenzo Bonifaci
Mathilde Bouvel
Joshua Brakensiek
Marco Bressan
Maike Buchin
Karthik C. S.
Yixin Cao
Katrin Casel
Ilaria Castellani
Diptarka Chakraborty
Timothy M. Chan
Eshan Chattopadhyay
Chandra Chekuri
Yu Chen
Leroy Chew
Dmitry Chistikov
Sherman S. M. Chow
Andrea Clementi
Edith Cohen
Thomas Colcombet
Emilio Cruciani
Artur Czumaj
Victor Dalmau
Luc Dartois
Vrunda Dave

Laure Daviaud
Anuj Dawar
Bart de Keijzer
Vladimir Deineko
Daniele Dell'Erba
Yotam Dikstein
Yann Disser
Gaëtan Douéneau-Tabot
Yfke Dulek
Thomas Dybdahl Ahle
Marwa El Halabi
Matthias Englert
David Eppstein
Hossein Esfandiari
Guy Even
Yuri Faenza
John Fearnley
Andreas Emil Feldmann
Hendrik Fichtenberger
Aris Filos-Ratsikas
Nick Fischer
Lukas Fleischer
Kyle Fox
Hadar Frenkel
Takuro Fukunaga
Ameet Gadekar
Sainyam Galhotra
Chaya Ganesh
Leszek Gasieniec
Simon Gay
Rohan Ghuge
András Gilyén
Paul Goldberg
Alexander Golovnev
Mayank Goswami
Garance Gourdel
Ben Green
Joshua Grogin
Jakob Grue Simonsen
Luciano Gualà
Pierre Guillon
Manoj Gupta
Rohit Gurjar
Peter Habermehl
Yassine Hamoudi
Nathaniel Harms
Kun He
Tyler Helmuth
Duc A. Hoang
Ruben Hoeksma
Seok-Hee Hong

Bernardo David
Anindya De
Mateus De Oliveira
Argyrios Deligkas
Amit Deshpande
Michael Dinitz
Brijesh Dongol
Anne Driemel
Nicolas Dupin
Nadav Dym
Michael Elkin
Alessandro Epasto
Leah Epstein
Kousha Etessami
Shai Evra
Rolf Fagerberg
Cristina Feier
Henning Fernau
Andrés Fielbaum
Arnold Filtser
Orr Fischer
Jacob Focke
Cole Franks
Zachary Friggstad
Martin Fürer
Travis Gagie
Moses Ganardi
Vijay Ganesh
Dmitry Gavinsky
Badih Ghazi
George Giakkoupis
Emmanuel Godard
Elazar Goldenberg
Daniel Gonçalves
Daniel Gottesman
Julien Grange
Alejandro Grez
Martin Grohe
Yan Gu
Giulio Guerrieri
Xiangyu Guo
Sushmita Gupta
Gregory Gutin
Daniel Hader
Sariel Har-Peled
Zen Harper
Falko Hegerfeld
Loic Helouet
Martin Hoefer
Lukáš Holík
Kaave Hosseini

Sami Davies
Mark de Berg
Ronald de Wolf
Holger Dell
Nishanth Dikkala
Irit Dinur
Michal Dory
Guillaume Ducoffe
Zdenek Dvorak
Talya Eden
Alina Ene
Naomi Ephraim
Thomas Erlebach
Hiroshi Eto
Tomer Ezra
Yaron Fairstein
Sándor Fekete
Diodato Ferraioli
Nathanaël Fijalkow
Jeremy Fineman
Bailey Flanigan
Fedor Fomin
Fabrizio Frati
Hongfei Fu
Federico Fusco
Martin Gairing
Arun Ganesh
Sumegha Garg
Pawel Gawrychowski
Prantar Ghosh
Daniel Gibney
Jan Goedgebeur
Petr Golovach
Gramoz Goranci
Themistoklis Gouleakis
Jan Grebik
Sander Gribling
Allan Grønlund
Yong Gu
Shibashis Guha
Anupam Gupta
Tom Gur
Stefan Haar
Amar Hadzihasanovic
Marcel Hark
Masahito Hasegawa
Marc Heinrich
Shuichi Hirahara
Peter Hoefner
Jacob Holm
Jun-Ting Hsieh

Justin Hsu
Edin Husic
Rahul Ilango
Shweta Jain
Bart M. P. Jansen
Rajesh Jayaram
Shaofeng H.-C. Jiang
Stasys Jukna
Panagiotis Kanellopoulos
Adam Karczmarz
Manuel Kauers
Nathaniel Kell
Thomas Kesselheim
Sanjeev Khanna
Benny Kimelfeld
Aleks Kissinger
Kim-Manuel Klein
Kristin Knorr
Yusuke Kobayashi
Zhuan Khye Koh
Christian Konrad
Martin Koutecky
Robert Krauthgamer
Ravishankar Krishnaswamy
Ariel Kulik
Amit Kumar
Ron Kupfer
Anthony Labarre
Jakub Łącki
Michael Lampis
John Lapinskas
Hung Le
Euiwoong Lee
Karoliina Lehtinen
Stefano Leucci
Jason Li
Yi Li
Depeng Liu
Tianren Liu
Yunchao Liu
Philipp Loick
Pinyan Lu
Cong Ma
Calum MacRury
Florin Manea
Yuchen Mao
Eric Martin
Samuel McCauley
Kitty Meeks
Raghu Meka

Zhiyi Huang
Tony Huynh
Fotis Iliopoulos
Siddhartha Jain
David N. Jansen
Artur Jeż
Ce Jin
Jean Christoph Jung
Sampath Kannan
Petteri Kaski
Tali Kaufman
Dominik Kempa
Bas Ketsman
Seri Khoury
Evangelos Kipouridis
Hartmut Klauck
Peter Kling
Tomohiro Koana
Alexander Koch
Christian Komusiewicz
Tuukka Korhonen
Lukasz Kowalik
Klaus Kriegel
Gregory Kucherov
Janardhan Kulkarni
Marvin Künnemann
Dietrich Kuske
Arnaud Labourel
Bundit Laekhanukit
Jonas Landman
Kasper Green Larsen
Jonathan Leake
Orlando Lee
Jean-Simon Lemay
Roie Levin
Ray Li
Jiabao Lin
Jingcheng Liu
Yang Liu
William Lochet
Florian Lonsing
Kelin Luo
Weiyun Ma
Florent Madelaine
Alessio Mansutti
Mathieu Mari
Luke Mathieson
Andrew McGregor
Nicole Megow
Themistoklis Melissourgos

Paul Hunter
Rasmus Ibsen-Jensen
Sidharth Jaggi
Mikolas Janota
Klaus Jansen
Haotian Jiang
Tibor Jordan
Benjamin Lucien Kaminski
Upendra Kapshikar
Alexander Kauer
Telikepalli Kavitha
Batya Kenig
Arindam Khan
Eun Jung Kim
Sándor Kisfaludi-Bak
Pieter Kleer
Marina Knittel
Naoki Kobayashi
Tomasz Kociumaka
Eitan Kondratovsky
Michal Koucky
Laszlo Kozma
Klaus Kriegel
Manfred Kufleitner
Akash Kumar
Denis Kuperberg
Thijs Laarhoven
Oded Lachish
Guillaume Lagarde
Roman Langrehr
Thomas Lavastida
Dabeen Lee
Troy Lee
Christoph Lenzen
Nathan Lhote
Wenzheng Li
Andre Linhares
Siqi Liu
Yanyi Liu
Markus Lohrey
Shachar Lovett
Alessandro Luongo
Will Ma
Konstantinos Mamouras
Pasin Manurangsi
Nicolas Markey
Gilbert Maystre
Ian McQuillan
Arne Meier
Stefan Mengel

Julian Mestre
Benjamin Miller
Michael Mislove
Matthias Mnich
Tobias Mömke
Benoit Montagu
Antoine Mottet
Sagnik Mukhopadhyay
Muhammad Najib
Bento Natura
Jelani Nelson
Huy Nguyen
Dolav Nitay
Zeev Nutov
Gergely Odor
Nicolas Ollinger
Ly Orgo
Soumyabrata Pal
Fahad Panolan
Francesco Pasquale
Andreas Pavlogiannis
Pan Peng
Jorge Pérez
Seth Pettie
Veronika Pillwein
Paolo Pistone
Chara Podimata
Igor Potapov
Emmanouil Pountourakis
Eric Price
Manish Purohit
Karin Quaas
Jakub Radoszewski
Mikhail Raskin
Luca Reggio
Colin Riba
Andrei Romashchenko
Adi Rosen
Peter Rossmanith
Marc Roth
Aviad Rubinstein
Mathieu Sablik
Abdallah Saffidine
Ken Sakayori
Sai Sandeep
Rahul Santhanam
Thomas Sauerwald
Raghuvansh Saxena
Lena Schlipf
Daniel Schoepflin

Othon Michail
Till Miltzow
Slobodan Mitrović
Eugenio Moggi
Morteza Monemizadeh
Pat Morin
Amer Mouawad
Daniel Nagaj
Vasileios Nakos
Gonzalo Navarro
Daniel Neuen
Rad Niazadeh
Damian Niwinski
Johannes Obenaus
Yoshio Okamoto
Izhar Oppenhein
Shayan Oveis Gharan
Dömötör Pálvölgyi
Pedro Paredes
Boaz Patt-Shamir
Max Pedersen
Richard Peng
Will Perkins
Frank Pfenning
Laureline Pinault
Thomas Place
Adam Polak
Aditya Potukuchi
M. Praveen
Probst
David Purser
Kent Quanrud
Prasad Raghavendra
Ankit Singh Rawat
Vojtech Rehak
Liam Roditty
Dana Ron
Will Rosenbaum
Günter Rote
Ron Rothblum
Ignaz Rutter
Sushant Sachdeva
Prakash Saivasan
Mohammad Salavatipour
Piotr Sankowski
Luigi Santocanale
Saket Saurabh
Jonathan Scarlett
Melanie Schmidt
Tselil Schramm

Jakub Michaliszyn
Dor Minzer
Shuichi Miyazaki
Divyarthi Mohan
Benjamin Monmege
Benjamin Moseley
Giorgos Mousa
Viswanath Nagarajan
Shyam Narayanan
Florian Nelles
Alantha Newman
Andrey Nikolaev
Navid Nouri
Nidia Obscura Acosta
Karolina Okrasa
Sebastian Ordyniak
Luca Padovani
Debmalya Panigrahi
Kanstantsin Pashkovich
Erik Paul
Binghui Peng
Ron Peretz
William Pettersson
Long Pham
François Pirot
Wanchote Po Jiamjitrak
Ilia Ponomarenko
Amaury Pouly
Nicola Prezza
Gabriele Puppis
Sharon Qian
Tahiry Rabehaja
Akshay Ramachandran
Saurabh Ray
Christian Retoré
Marcel Roeloffzen
Chuitian Rong
Ansis Rosmanis
Lior Rotem
Thomas Rothvoss
Sagi Saadon
Kunihiko Sadakane
Yoshifumi Sakai
Ville Salo
Ocan Sankur
Ramprasad Saptharishi
Joe Sawada
Nicolas Schabanel
Sylvain Schmitz
Tom Schrijvers

Steffen Schuldenzucker
Francois Schwarzentruber
Chris Schwiegelshohn
Helmut Seidl
C. Seshadhri
Ariel Shaulker
Takeharu Shiraga
Aaron Sidford
Jamie Sikora
Sunil Simon
Nodari Sitchinava
Alexander Skopalik
Christian Sohler
Frank Sommer
Florian Speelman
Ramanathan Thinniyam Srinivasan
Teresa Anna Steiner
Hsin-Hao Su
Jukka Suomela
Zoya Svitkina
John Sylvester
Navid Talebanfard
Jakub Tarnawski
Sharma V. Thankachan
Bernardo Toninho
Ilkka Törmä
Ohad Trabelsi
Kostas Tsichlas
Marc Uetz
Przemysław Uznański
Jan van den Brand
Gabriele Vanoni
Margus Veanes
Oleg Verbitsky
Jan Vondrak
David Wajc
Haitao Wang
Justin Ward
Fan Wei
Nicole Wein
Piotr Wieczorek
Max Willert
Michal Wlodarczyk
Marcin Wrochna
Christian Wulff-Nilsen
Chaoping Xing
Yinzhan Xu
Yutaro Yamaguchi
Cedric Yen-Yu Lin
Fang Yu

Hans-Jörg Schurr
Pascal Schweitzer
Stefan Schwoon
Geraud Senizergues
Hadas Shachnai
Sarai Sheinvald
Arseny Shur
Anastasios Sidiropoulos
Francesco Silvestri
Kirill Simonov
Rene Sitters
Michał Skrzypczak
Mehdi Soleimanifar
Yifan Song
Joachim Spoerhase
Srikanth Srinivasan
Frank Stephan
Scott Summers
S P Suresh
Chaitanya Swamy
Dániel Szilágyi
Seiichiro Tani
Yin Tat Lee
Thomas Thierauf
Jacobo Torán
Csaba Toth
Vera Traub
Madhur Tulsiani
Sander Uijlen
Danny Vainstein
Erik Jan van Leeuwen
Yann Vaxès
Erik Vee
Jamie Vicary
Hoa Vu
Tomasz Walen
Jiaheng Wang
John Watrous
Pascal Weil
Baruch Weizman
Andreas Wiese
Sarah Winter
Sam Chiu-Wai Wong
Fei Wu
Jiayi Xian
Chao Xu
Jie Xue
Katsuhisa Yamanaka
Sorrachai Yingchareonthawornchai
Nengkun Yu

Ariel Schvartzman
Robert Schweller
Saeed Seddighin
Olivier Serre
Shuai Shao
Yixin Shen
Salomon Sickert
Sebastian Siebertz
Paris Siminelakis
Alex Simpson
D Sivakumar
Benjamin Smith
Shay Solomon
Zhao Song
A V Sreejith
Clifford Stein
Donald Stull
Xiaorui Sun
Akira Suzuki
Michelle Sweering
Avishay Tal
Runzhou Tao
Justin Thaler
Clayton Thomas
Hazem Torfah
Noam Touitou
Ashutosh Trivedi
Andrea Turrini
Chris Umans
Ali Vakilian
Rob van Stee
Daniel Vaz
Niccolò Veltri
Roland Vincze
Nikhil Vyas
Di Wang
Joshua Wang
Karol Węgrzycki
Oren Weimann
Stefan Weltge
Sebastian Wild
Thorsten Wißmann
Damien Woods
Zhilin Wu
Mingyu Xiao
Haifeng Xu
Shota Yamada
Kuan Yang
Yuichi Yoshida
Yelena Yuditsky

Viktor Zamaraev
Luca Zanetti
David Zeng
Fred Zhang
Hongyu Zheng
Stanislav Živný
Giacomo Zambelli
Gianluigi Zavattaro
Mark Zhandry
Tianyi Zhang
Li Zhou
Anna Zych
Or Zamir
Peter Zeman
Chihao Zhang
Yuhao Zhang
Dmitriy Zhuk

List of Authors

Amir Abboud (7)
Weizmann Institute of Science, Rehovot, Israel

Samson Abramsky (115)
Department of Computer Science,
University of Oxford, UK

Dimitris Achlioptas (8)
Department of Informatics &
Telecommunications, University of Athens,
Greece

Deeksha Adil (9)
University of Toronto, Canada

Pankaj K. Agarwal (10, 11)
Department of Computer Science,
Duke University, Durham, NC, USA

Marc Aiguier (132)
Université Paris-Saclay, CentraleSupélec,
MICS, France

Shyan Akmal (12, 13)
MIT, EECS and CSAIL, Cambridge, MA, USA

Noga Alon (14)
Department of Mathematics,
Princeton University, NJ, USA;
Schools of Mathematics and Computer Science,
Tel Aviv University, Israel

Antoine Amarilli (116)
LTCI, Télécom Paris, Institut Polytechnique de
Paris, France

Markus Anders (15, 16)
TU Kaiserslautern, Germany;
TU Darmstadt, Germany

Antonios Antoniadis (17)
University of Twente, The Netherlands

Vahid R. Asadi (18)
Simon Fraser University, Burnaby, Canada

Sepehr Assadi (19)
Department of Computer Science,
Rutgers University, Piscataway, NJ, USA

Mitali Bafna (20)
Harvard University, Cambridge, MA, USA

Christel Baier (1)
Technische Universität Dresden, Germany

Tanvi Bajpai (21)
University of Illinois, Urbana-Champaign,
Urbana, IL, USA

Paolo Baldan (117)
Dipartimento di Matematica,
University of Padova, Italy

Borja Balle (118)
DeepMind, London, UK

Etienne Bamas (22)
EPFL, Lausanne, Switzerland

Sayan Bandyapadhyay (23)
Department of Informatics,
University of Bergen, Norway

Gabriel Bathie (119)
École normale supérieure de Lyon, France

Eleni Batziou (24)
Technical University of Munich, Germany

Soheil Behnezhad (19)
Department of Computer Science,
University of Maryland, College Park, MD, USA

Ran Ben Basat (25)
University College London, UK

Matthias Bentert (26)
Faculty IV, Algorithmics and Computational
Complexity, Technische Universität Berlin,
Germany

Mario Berta (82)
Department of Computing,
Imperial College London, UK;
IQIM, California Institute of Technology,
Pasadena, CA, USA;
AWS Center for Quantum Computing,
Pasadena, CA, USA

Sayan Bhattacharya (27)
Department of Computer Science,
University of Warwick, Coventry, UK

Marcin Bienkowski (28)
Institute of Computer Science,
University of Wrocław, Poland

Andreas Björklund (29)
Lund, Sweden

Guy Blanc (30)
Stanford University, CA, USA

48th International Colloquium on Automata, Languages, and Programming (ICALP 2021).
Editors: Nikhil Bansal, Emanuela Merelli, and James Worrell
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Joakim Blikstad (31)
KTH Royal Institute of Technology, Stockholm, Sweden

Manuel Bodirsky (120)
Institut für Algebra, TU Dresden, Germany

Andrej Bogdanov (33)
Department of Computer Science and Engineering and Institute of Theoretical Computer Science and Communications, Chinese University of Hong Kong, China

Cosmin Bonchiş (135)
West University of Timişoara, Romania

Édouard Bonnet (34, 35)
Univ Lyon, CNRS, ENS de Lyon, Université Claude Bernard Lyon 1, LIP UMR5668, France

Sébastien Bouchard (36)
Univ. Bordeaux, Bordeaux INP, CNRS, LaBRI, UMR5800, F-33400 Talence, France

Joshua Brakensiek (37)
Computer Science Department, Stanford University, CA, USA

Cornelius Brand (38)
Charles University, Prague, Czech Republic

Alex Brandts (121)
Department of Computer Science, University of Oxford, UK

Karl Bringmann (39, 40, 41)
Saarland University, Saarland Informatics Campus, Saarbrücken, Germany; Max-Planck-Institute for Informatics, Saarland Informatics Campus, Saarbrücken, Germany

Moritz Buchem (42)
Maastricht University, Maastricht, The Netherlands

Andrei A. Bulatov (2)
School of Computing Science, Simon Fraser University, Burnaby, Canada

Brian Bullins (9)
Toyota Technological Institute at Chicago, IL, USA

Jan Böker (32)
RWTH Aachen University, Germany

Massimo Cairo (43)
Department of Computer Science, University of Helsinki, Finland

Antonin Callard (122)
Université Paris-Saclay, ENS Paris-Saclay, Département Informatique, 91190 Gif-sur-Yvette, France

Marco Carmosino (44)
Department of Computer Science, Boston University, MA, USA

Antonio Casares (123)
LaBRI, Université de Bordeaux, France

Katrin Casel (91)
Hasso Plattner Institute, University of Potsdam, Germany

Ruoxu Cen (45)
Duke University, Durham, NC, USA

Keren Censor-Hillel (3, 46)
Department of Computer Science, Technion, Haifa, Israel

Deeparnab Chakrabarty (21)
Dartmouth College, Hanover, NH, USA

Timothy M. Chan (47)
University of Illinois at Urbana-Champaign, IL, USA

Panagiotis Charalampopoulos (48)
The Interdisciplinary Center Herzliya, Israel

Krishnendu Chatterjee (124)
IST Austria, Klosterneuburg, Austria

Chandra Chekuri (21, 49, 50)
University of Illinois, Urbana-Champaign, Urbana, IL, USA

Lijie Chen (51, 52)
MIT, Cambridge, MA, USA

Yu Chen (53)
Department of Computer and Information Science, University of Pennsylvania, Philadelphia, PA, USA

Kuan Cheng (54)
Peking University, Beijing, China

Yu Cheng (45)
University of Illinois at Chicago, IL, USA

Andrew M. Childs (55)
Joint Center for Quantum Information and Computer Science, Department of Computer Science, and Institute for Advanced Computer Studies, University of Maryland, College Park, MD, USA

George Christodoulou (56)
University of Liverpool, UK

Luca Ciccone (125)
University of Torino, Italy

Lorenzo Clemente (126)
University of Warsaw, Poland

Christian Coester (57)
CWI, Amsterdam, The Netherlands

Thomas Colcombet (123, 127)
CNRS, IRIF, Université de Paris, France

Adrian Crăciun (135)
West University of Timişoara, Romania

Wojciech Czerwiński (128, 129)
University of Warsaw, Poland

Artur Czumaj (58)
Department of Computer Science and DIMAP,
University of Warwick, Coventry, UK

Gianlorenzo D'Angelo (59)
Gran Sasso Science Institute, L'Aquila, Italy

Mina Dalirrooyfard (60)
MIT, Cambridge, MA, USA

Christoph Damerius (61)
University of Hamburg, Germany

Debarati Das (39)
Basic Algorithm Research Copenhagen (BARC),
University of Copenhagen, Denmark

Ewan Davies (62)
Department of Computer Science,
University of Colorado, Boulder, CO, USA

Ronald de Wolf (110)
QuSoft, CWI, Amsterdam, The Netherlands;
University of Amsterdam, The Netherlands

Yoann Dieudonné (36)
MIS Lab., Université de Picardie Jules Verne,
Amiens, France

Clemens Dubslaff (1)
Technische Universität Dresden, Germany

Orr Dunkelman (4)
Computer Science Department,
University of Haifa, Israel

Jonas Ellert (63)
Department of Computer Science,
Technical University of Dortmund, Germany

Matthias Englert (17)
University of Warwick, Coventry, UK

Jacob Evald (64)
BARC, University of Copenhagen, Denmark

Alireza Farhadi (54)
University of Maryland, College Park, MD, USA

Omar Fawzi (82)
Univ Lyon, ENS Lyon, UCBL, CNRS, Inria,
LIP, F-69342, Lyon Cedex 07, France

Nathanaël Fijalkow (123)
CNRS, LaBRI, Université de Bordeaux, France;
The Alan Turing Institute of Data Science,
London, UK

Johannes Fischer (63)
Department of Computer Science,
Technical University of Dortmund, Germany

Fedor V. Fomin (23)
Department of Informatics,
University of Bergen, Norway

Dimitris Fotakis (65)
National Technical University of Athens, Greece

Viktor Fredslund-Hansen (64)
BARC, University of Copenhagen, Denmark

Dominik D. Freydenberger (130)
Loughborough University, UK

Tobias Friedrich (66, 91)
Hasso Plattner Institute,
University of Potsdam, Germany

Zachary Friggstad (67)
Department of Computer Science,
University of Alberta, Edmonton, Canada

Hu Fu (68)
ITCS, Shanghai University of Finance and
Economics, China

Florian Funke (1)
Technische Universität Dresden, Germany

Arun Ganesh (70)
Department of Computer Science,
University of California at Berkeley, CA, USA

Arnab Ganguly (71)
Department of Computer Science,
University of Wisconsin – Whitewater, WI, USA

Robert Ganian (72)
Algorithms and Complexity Group,
TU Wien, Austria

Paritosh Garg (22)
EPFL, Lausanne, Switzerland

Paweł Gawrychowski (48)
University of Wrocław, Poland

Colin Geniet (35)
University of Warsaw, Poland

Zeev Geyzel (4)
Mobileye, an Intel company, Jerusalem, Israel

Uma Girish (73)
Department of Computer Science, Princeton University, NJ, USA

Sergey Goncharov (131)
University Erlangen-Nürnberg, Germany

Alexandre Goy (132)
Université Paris-Saclay, CentraleSupélec, MICS, France

Andrei Graur (14)
Department of Management Science and Engineering, Stanford University, CA, USA

Nick Gravin (74)
ITCS, Shanghai University of Finance and Economics, China

Sander Gribling (110)
IRIF, Université de Paris, CNRS, Paris, France

Martin Grohe (134)
RWTH Aachen University, Germany

Erich Grädel (133)
RWTH Aachen University, Germany

Yong Gu (76)
Institute for Interdisciplinary Information Sciences, Tsinghua University, Beijing, China

Yuzhou Gu (75)
MIT, Cambridge, MA, USA

Anupam Gupta (77)
Computer Science Department, Carnegie Mellon University, Pittsburgh, PA, USA

Venkatesan Guruswami (37)
Computer Science Department, Carnegie Mellon University, Pittsburgh, PA, USA

Maximilian Probst Gutenberg (64)
ETH Zurich, Switzerland

Bernd Gärtner (69)
Institute of Theoretical Computer Science, Department of Computer Science, ETH Zürich, Switzerland

Andreas Göbel (66, 91)
Hasso Plattner Institute, University of Potsdam, Germany

Bernhard Haeupler (78)
Carnegie Mellon University, Pittsburgh, PA, USA; ETH Zürich, Switzerland

MohammadTaghi Hajiaghayi (54)
University of Maryland, College Park, MD, USA

Thekla Hamm (72)
Algorithms and Complexity Group, TU Wien, Austria

Kristoffer Arnsfelt Hansen (24)
Aarhus University, Denmark

Maria Hartmann (79)
Institut für Informatik, Freie Universität Berlin, Germany

Sebastian Haslebacher (69)
Department of Computer Science, ETH Zürich, Switzerland

Monika Henzinger (124)
Faculty of Computer Science, University of Vienna, Austria

D. Ellis Hershkowitz (78)
Carnegie Mellon University, Pittsburgh, PA, USA

Hung P. Hoang (69)
Institute of Theoretical Computer Science, Department of Computer Science, ETH Zürich, Switzerland

Kenneth Hoover (44)
Department of Computer Science, University of California, San Diego, CA, USA

Xiao Hu (11)
Duke University, Durham, NC, USA

Zhenjiang Hu (143)
Key Laboratory of High Confidence Software Technologies (MoE), Department of Computer Science and Technology, Peking University, Beijing, China

Chien-Chung Huang (80)
CNRS, ENS, PSL, Paris, France

Shih-Han Hung (55)
Joint Center for Quantum Information and Computer Science, Department of Computer Science, and Institute for Advanced Computer Studies, University of Maryland, College Park, MD, USA

Kasper Høgh (24)
Aarhus University, Denmark

Sharat Ibrahimpur (81)
Department of Combinatorics and Optimization, University of Waterloo, Canada

Russell Impagliazzo (44)
Department of Computer Science, University of California, San Diego, CA, USA

Gabriel Istrate (135)
West University of Timişoara, Romania

Louis Jachiet (116)
LTCI, Télécom Paris, Institut Polytechnique de Paris, France

Simon Jantsch (1)
Technische Universität Dresden, Germany

Arthur Jaquard (127)
Université de Paris, CNRS, IRIF, F-75006, Paris, France

Hyejung H. Jee (82)
Department of Computing, Imperial College London, UK

Ce Jin (12)
MIT, EECS and CSAIL, Cambridge, MA, USA

Zhengzhong Jin (54)
Johns Hopkins University, Baltimore, MD, USA

Dominik Kaaser (61)
University of Hamburg, Germany

Valentine Kabanets (44)
School of Computing Science, Simon Fraser University, Burnaby, Canada

Sagar Sudhir Kale (124)
Faculty of Computer Science, University of Vienna, Austria

Tobias Kappé (142)
Department of Computer Science, Cornell University, Ithaca, NY, USA

Adam Karczmarz (83)
Institute of Informatics, University of Warsaw, Poland

Petteri Kaski (29)
Department of Computer Science, Aalto University, Espoo, Finland

Tali Kaufman (84)
Department of Computer Science, Bar-Ilan University, Ramat-Gan, Israel

Jenny Kaufmann (60)
Harvard University, Cambridge, MA, USA

Telikepalli Kavitha (85)
Tata Institute of Fundamental Research, Mumbai, India

Ken-ichi Kawarabayashi (86)
National Institute of Informatics, Tokyo, Japan

Chaya Keller (4)
Department of Computer Science, Ariel University, Israel

Nathan Keller (4)
Department of Mathematics, Bar Ilan University, Ramat Gan, Israel

Sanjeev Khanna (53)
Department of Computer and Information Science, University of Pennsylvania, Philadelphia, PA, USA

Sandra Kiefer (134)
University of Warsaw, Poland; RWTH Aachen University, Germany

Eun Jung Kim (35)
Université Paris-Dauphine, PSL University, CNRS UMR7243, LAMSADE, Paris, France

Peter Kiss (27)
Department of Computer Science, University of Warwick, Coventry, UK

Peter Kling (61)
University of Hamburg, Germany

Fabian Klute (72)
Deptartment of Information and Computing Sciences, Utrecht University, The Netherlands

Simon Knäuer (120)
Institut für Algebra, TU Dresden, Germany

Gillat Kol (52)
Princeton University, NJ, USA

Antonina Kolokolova (44)
Department of Computer Science, Memorial University of Newfoundland, St. John's, Canada

George Kontogeorgiou (58)
Mathematics Institute, University of Warwick, Coventry, UK

Tuukka Korhonen (87)
Department of Computer Science, University of Helsinki, Finland

Panagiotis Kostopanagiotis (65)
National Technical University of Athens, Greece

Michal Koucký (88)
Computer Science Institute, Charles University, Prague, Czech Republic

Elias Koutsoupias (56, 57)
University of Oxford, UK

Annamária Kovács (56)
Goethe University, Frankfurt am Main, Germany

Dexter Kozen (142)
Department of Computer Science, Cornell University, Ithaca, NY, USA

Jakub Kozik (89)
Theoretical Computer Science Department, Faculty of Mathematics and Computer Science, Jagiellonian University, Kraków, Poland

László Kozma (79)
Institut für Informatik, Freie Universität Berlin, Germany

Artur Kraska (28)
Institute of Computer Science, University of Wrocław, Poland

Martin S. Krejca (66)
Sorbonne University, CNRS, LIP6, Paris, France

Karel Král (88)
Computer Science Institute, Charles University, Prague, Czech Republic

Adam Kurpisz (90)
Department of Mathematics, ETH Zürich, Switzerland

Rasmus Kyng (9)
ETH Zurich, Switzerland

Arnaud Labourel (36)
Aix Marseille Univ, Université de Toulon, CNRS, LIS, Marseille, France

Clara Lacroce (118)
School of Computer Science, McGill University, Montréal, Canada; Mila, Montréal, Canada

J. A. Gregor Lagodzinski (91)
Hasso Plattner Institute, University of Potsdam, Germany

Michael Lampis (92)
Université Paris-Dauphine, PSL University, CNRS, LAMSADE, 75016, Paris, France

Jane Lange (30)
MIT, Cambridge, MA, USA

Sławomir Lasota (128)
University of Warsaw, Poland

Reut Levi (93)
Efi Arazi School of Computer Science, The Interdisciplinary Center Herzliya, Israel

Tongyang Li (55)
Joint Center for Quantum Information and Computer Science, Department of Computer Science, and Institute for Advanced Computer Studies, University of Maryland, College Park, MD, USA;
Center for Theoretical Physics, MIT, Cambridge, MA, USA

Xin Li (54)
Johns Hopkins University, Baltimore, MD, USA

Yangjia Li (136)
University of Tartu, Estonia;
SKLCS, Institute of Software, CAS, Beijing, China

Yinan Li (110)
Graduate School of Mathematics, Nagoya University, Japan

Hsiang-Hsuan Liu (28)
Utrecht University, The Netherlands

Zhenjian Lu (51, 94)
University of Warwick, Coventry, UK

Xin Lyu (51)
Tsinghua University, Beijing, China

Bruce M. Maggs (70)
Department of Computer Science, Duke University, Durham, NC, USA;
Emerald Innovations, Cambridge, MA, USA

Rupak Majumdar (1)
MPI-SWS, Kaiserslautern, Germany

Noa Marelly (46)
Department of Computer Science, Technion, Haifa, Israel

Mathieu Mari (80)
University of Warsaw, Poland

Dániel Marx (95)
CISPA Helmholtz Center for Information Security, Saarland Informatics Campus, Saarbrücken, Germany

Claire Mathieu (80)
CNRS, IRIF, Université de Paris, France

Nicolaos Matsakis (17)
Athens, Greece

Theo McKenzie (96)
Department of Mathematics, University of California, Berkeley, CA, USA

Michael Mitzenmacher (25)
Harvard University, Cambridge, MA, USA

Sidhanth Mohanty (96)
Department of Computer Science, University of California, Berkeley, CA, USA

Bojan Mohar (86)
Department of Mathematics, Simon Fraser University, Burnaby, Canada; IMFM, Department of Mathematics, University of Ljubljana, Slovenia

Benjamin Monmege (137)
Aix Marseille Univ, Université de Toulon, CNRS, LIS, France

Benjamin Moseley (77, 97)
Tepper School of Business, Carnegie Mellon University, Pittsburgh, PA, USA

Antoine Mottet (129, 138)
Department of Algebra, Faculty of Mathematics and Physics, Charles University, Prague, Czech Republic

Shay Mozes (48)
The Interdisciplinary Center Herzliya, Israel

Lovro Mrkonjić (133)
RWTH Aachen University, Germany

Ansh Nagda (53)
University of Washington, Seattle, WA, USA

Tomáš Nagy (138)
Institut für Diskrete Mathematik und Geometrie, Technische Universität Wien, Austria; Department of Algebra, Faculty of Mathematics and Physics, Charles University, Prague, Czech Republic

Yonatan Nakar (98)
Tel Aviv University, Israel

Vasileios Nakos (41, 65)
Saarland University, Saarland Informatics Campus, Saarbrücken, Germany; Max Planck Institute for Informatics, Saarland Informatics Campus, Saarbrücken, Germany

Roman Nedela (86)
Univeristy of West Bohemia, Pilsen, Czech Republic

Maryam Negahbani (21)
Dartmouth College, Hanover, NH, USA

Eike Neumann (99)
Max Planck Institute for Software Systems, Saarland Informatics Campus, Saarbrücken, Germany

Ilan Newman (100)
University of Haifa, Israel

Lê Thành Dũng (Tito) Nguyễn (139)
Laboratoire d'informatique de Paris Nord, Villetaneuse, France

André Nichterlein (26)
Faculty IV, Algorithmics and Computational Complexity, Technische Universität Berlin, Germany

Harold Nieuwboer (110)
Korteweg-de Vries Institute for Mathematics and QuSoft, University of Amsterdam, The Netherlands

Takaaki Nishimoto (101)
RIKEN Center for Advanced Intelligence Project, Tokyo, Japan

Camille Noûs (139)
Laboratoire Cogitamus, Université Volante, Sevran, France

Igor C. Oliveira (51, 94)
University of Warwick, Coventry, UK

Izhar Oppenheim (84)
Department of Mathematics, Ben-Gurion University of the Negev, Be'er Sheva, Israel

Łukasz Orlikowski (128)
University of Warsaw, Poland

Joël Ouaknine (99)
Max Planck Institute for Software Systems, Saarland Informatics Campus, Saarbrücken, Germany

Luca Padovani (125)
University of Torino, Italy

Prakash Panangaden (118)
School of Computer Science,
McGill University, Montréal, Canada;
Mila, Montréal, Canada

Debmalya Panigrahi (45, 70)
Duke University, Durham, NC, USA

Charles Paperman (116)
Univ. Lille, CNRS, INRIA, Centrale Lille, UMR 9189 CRIStAL, F-59000 Lille, France

Marcus Pappik (66)
Hasso Plattner Institute,
University of Potsdam, Germany

Irene Parada (72)
TU Eindhoven, The Netherlands

Dmitry Paramonov (52)
Princeton University, NJ, USA

Ojas Parekh (102)
Sandia National Laboratories,
Albuquerque, NM, USA

Julie Parreaux (137)
Aix Marseille Univ, Université de Toulon, CNRS, LIS, France

Paweł Parys (140)
Institute of Informatics,
University of Warsaw, Poland

Dhrumil Patel (71)
School of EECS, Louisiana State University,
Baton Rouge, LA, USA

Mike Paterson (58)
Department of Computer Science and DIMAP,
University of Warwick, Coventry, UK

Andrzej Pelc (36)
Département d'informatique,
Université du Québec en Outaouais,
Gatineau, Canada

Will Perkins (62)
Department of Mathematics, Statistics, and Computer Science, University of Illinois at Chicago, IL, USA

Enoch Peserico (103)
Università degli Studi di Padova, Italy

Liat Peterfreund (130)
DI ENS, ENS Paris, CNRS, PSL University, INRIA, France

Daniela Petrişan (132)
Université de Paris, IRIF, France

Seth Pettie (104, 105)
University of Michigan, Ann Arbor, MI, USA

Georgios Piliouras (65)
Singapore University of Technology and Design, Singapore

Michael Pinsker (138)
Institut für Diskrete Mathematik und Geometrie, Technische Universität Wien, Austria;
Department of Algebra, Faculty of Mathematics and Physics, Charles University, Prague, Czech Republic

Jakob Piribauer (1)
Technische Universität Dresden, Germany

Toniann Pitassi (5)
University of Toronto, Canada

Debashmita Poddar (59)
Gran Sasso Science Institute, L'Aquila, Italy

Adam Polak (75, 106)
École Polytechnique Fédérale de Lausanne, Switzerland

Aaron Potechin (90)
Department of Computer Science,
University of Chicago, IL, USA

Pierre Pradic (139)
Department of Computer Science,
University of Oxford, UK

Gautam Prakriya (33)
Institute of Theoretical Computer Science and Communications, Chinese University of Hong Kong, China

Kevin Pratt (38)
Carnegie Mellon University,
Pittsburgh, PA, USA

Ian Pratt-Hartmann (141)
Department of Computer Science,
University of Manchester, UK;
Institute of Computer Science,
University of Opole, Poland

Doina Precup (118)
School of Computer Science,
McGill University, Montréal, Canada;
Mila, Montréal, Canada

Kirk Pruhs (97)
University of Pittsburgh, PA, USA

Karin Quaas (129)
University of Leipzig, Germany

Kent Quanrud (49, 50)
Purdue University, West Lafayette, IN, USA

Guillaume Rabusseau (118)
DIRO, Université de Montréal,
Montréal, Canada;
CIFAR AI Chair, Mila, Montréal, Canada

Francesco Ranzato (117)
Dipartimento di Matematica,
University of Padova, Italy

Ran Raz (73)
Department of Computer Science,
Princeton University, NJ, USA

Luca Reggio (115)
Department of Computer Science,
University of Oxford, UK

Hanlin Ren (76)
Institute for Interdisciplinary Information
Sciences, Tsinghua University, Beijing, China

Malte Renken (26)
Faculty IV, Algorithmics and Computational
Complexity, Technische Universität Berlin,
Germany

Pierre-Alain Reynier (137)
Aix Marseille Univ, Université de Toulon, CNRS,
LIS, France

Nicolás Rivera (107)
Department of Computer Science & Technology,
University of Cambridge, UK;
Instituto de Ingeniería Matemática,
University of Valparaíso, Chile

Romeo Rizzi (43)
Department of Computer Science,
University of Verona, Italy

Lars Rohwedder (22, 42, 106)
EPFL, Lausanne, Switzerland

Dana Ron (98)
Tel Aviv University, Israel

Eyal Ronen (4)
School of Computer Science,
Tel Aviv University, Israel

Marc Roth (108)
Merton College, University of Oxford, UK

Aviad Rubinstein (54)
Stanford University, CA, USA

Sebastian Rudolph (120)
Computational Logic Group,
TU Dresden, Germany

Amin Saberi (109)
Stanford University, CA, USA

Sushant Sachdeva (9)
University of Toronto, Canada

Alireza Samadian (97)
University of Pittsburgh, PA, USA

Sai Sandeep (37)
Computer Science Department,
Carnegie Mellon University,
Pittsburgh, PA, USA

Govind S. Sankar (95)
Indian Institute of Technology Madras,
Chennai, India

Thomas Sauerwald (107)
Department of Computer Science & Technology,
University of Cambridge, UK

Raghuvansh R. Saxena (52)
Princeton University, NJ, USA

Philipp Schepper (95)
CISPA Helmholtz Center for Information
Security, Saarland Informatics Campus,
Saarbrücken, Germany;
Saarbrücken Graduate School of Computer
Science, Saarland Informatics Campus, Germany

Todd Schmid (142)
Department of Computer Science,
University College London, UK

Johannes Schmitt (108)
Mathematical Institute,
University of Bonn, Germany

Florian Schneider (61)
University of Hamburg, Germany

Roy Schwartz (46)
Department of Computer Science,
Technion, Haifa, Israel

Pascal Schweitzer (15, 16)
TU Kaiserslautern, Germany;
TU Darmstadt, Germany

Michele Scquizzato (103)
Università degli Studi di Padova, Italy

Saeed Seddighin (54)
Toyota Technological Institute,
Chicago, IL, USA

Rahul Shah (71)
School of EECS, Louisiana State University,
Baton Rouge, LA, USA

Adi Shamir (4)
Department of Computer Science,
Weizmann Institute of Science,
Rehovot, Israel

Igor Shinkar (18)
Simon Fraser University, Burnaby, Canada

Alexandra Silva (142)
Department of Computer Science,
University College London, UK

Kirill Simonov (23)
Department of Informatics,
University of Bergen, Norway

Corwin Sinnamon (79)
Department of Computer Science,
Princeton University, NJ, USA

Stavros Sintos (11)
University of Chicago, IL, USA

Stratis Skoulakis (65)
Singapore University of Technology and Design,
Singapore

Michał Skrzypczak (126)
University of Warsaw, Poland

Jasper Slusallek (40)
Saarland University, Saarland Informatics
Campus, Saarbrücken, Germany

Zhao Song (52)
Institute for Advanced Study,
Princeton, NJ, US

Carlo Sparaciari (82)
Department of Computing,
Imperial College London, UK;
Department of Physics and Astronomy,
University College London, UK

Tatiana Starikovskaya (119)
DIENS, École normale supérieure de Paris, PSL
Research University, France

Alex Steiger (10)
Department of Computer Science,
Duke University, Durham, NC, USA

Kevin Sun (45)
Duke University, Durham, NC, USA

Alexander Svozil (124)
Faculty of Computer Science,
University of Vienna, Austria

Chaitanya Swamy (67, 81)
Department of Combinatorics and Optimization,
University of Waterloo, Canada

John Sylvester (107)
Department of Computer Science & Technology,
University of Cambridge, UK;
School of Computing Science,
University of Glasgow, UK

Yasuo Tabei (101)
RIKEN Center for Advanced Intelligence
Project, Tokyo, Japan

Li-Yang Tan (30)
Stanford University, CA, USA

Zhihao Gavin Tang (68, 74)
ITCS, Shanghai University of Finance and
Economics, China

Robert E. Tarjan (79)
Department of Computer Science,
Princeton University, NJ, USA;
Intertrust Technologies, Sunnyvale, CA, USA

Ran J. Tessler (4)
Department of Mathematics,
Weizmann Institute of Science, Rehovot, Israel

Sharma V. Thankachan (71)
Department of Computer Science,
University of Central Florida, Orlando, FL, USA

Stéphan Thomassé (35)
Univ Lyon, CNRS, ENS de Lyon,
Université Claude Bernard Lyon 1,
LIP UMR5668, France

Kevin Thompson (102)
Sandia National Laboratories, Albuquerque,
NM, USA

Alexandru I. Tomescu (43)
Department of Computer Science,
University of Helsinki, Finland

Tigran Tonoyan (46)
Department of Computer Science,
Technion, Haifa, Israel

Dominique Unruh (136)
University of Tartu, Estonia

Joran van Apeldoorn (110)
Institute for Information Law and QuSoft,
University of Amsterdam, The Netherlands

Pascal Vanier (122)
Normandie Univ, UNICAEN, ENSICAEN,
CNRS, GREYC, 14000 Caen, France

Shay Vargaftik (25)
VMware Research, Herzliya, Israel

Nithin Varma (100)
University of Haifa, Israel

Virginia Vassilevska Williams (7, 13, 47)
MIT, EECS and CSAIL, Cambridge, MA, US

Pavel Veselý (17)
Computer Science Institute of
Charles University, Prague, Czech Republic

Cosimo Vinci (59)
Gran Sasso Science Institute, L'Aquila, Italy

Emanuele Viola (111)
Khoury College of Computer Sciences,
Northeastern University, Boston, MA, USA

Birgit Vogtenhuber (72)
Graz University of Technology, Austria

Tjark Vredeveld (42)
Maastricht University, Maastricht,
The Netherlands

Nikhil Vyas (20)
MIT, Cambridge, MA, USA

Jens Vygen (80)
Research Institute for Discrete Mathematics &
Hausdorff Center for Mathematics,
University of Bonn, Germany

David Wajc (78, 109)
Stanford University, CA, USA

Michael Walter (110)
KdVI, ITFA, ILLC, and QuSoft,
University of Amsterdam, The Netherlands

Dingyu Wang (104)
University of Michigan, Ann Arbor, MI, USA

Kangning Wang (74)
Department of Computer Science,
Duke University, Durham, NC, USA

Yuyan Wang (97)
Carnegie Mellon University,
Pittsburgh, PA, USA

Rémi Watrigant (35)
Univ Lyon, CNRS, ENS de Lyon,
Université Claude Bernard Lyon 1,
LIP UMR5668, France

Oren Weimann (48)
University of Haifa, Israel

Philip Wellnitz (108)
Max Planck Institute for Informatics,
Saarland Informatics Campus (SIC),
Saarbrücken, Germany

Florian Wetzels (15)
TU Kaiserslautern, Germany

Andreas Wiese (42)
University of Chile, Santiago, Chile

Virginia Vassilevska Williams (75)
MIT, Cambridge, MA, USA

Elias Samuel Wirth (90)
Institute of Mathematics, TU Berlin, Germany

David P. Woodruff (6, 112)
Carnegie Mellon University,
Pittsburgh, PA, USA

James Worrell (99)
Department of Computer Science,
Oxford University, UK

Michał Wrona (138)
Theoretical Computer Science Department,
Jagiellonian University, Kraków, Poland

Hongxun Wu (68)
IIIS, Tsinghua University, Beijing, China

Jinzhao Wu (68)
Peking University, Beijing, China

Christian Wulff-Nilsen (64)
BARC, University of Copenhagen, Denmark

Karol Węgrzycki (106)
Saarland University, Saarland Informatics
Campus, Saarbrücken, Germany;
Max Planck Institute for Informatics, Saarland
Informatics Campus, Saarbrücken, Germany

Han Xu (143)
Department of Computer Science and
Technology, Peking University, Beijing, China

Yinzhan Xu (47, 75)
MIT, Cambridge, MA, USA

Jun Yang (11)
Duke University, Durham, NC, USA

Longhui Yin (104, 105)
Tsinghua University, Beijing, China

Huacheng Yu (52)
Princeton University, NJ, USA

Or Zamir (113)
Blavatnik School of Computer Science,
Tel Aviv University, Israel

Kostas Zampetakis (8)
Department of Computer Science & Engineering,
University of California Santa Cruz, CA, USA

Peter Zeman (86)
Department of Applied Mathematics,
Faculty of Mathematics and Physics,
Charles University, Prague, Czech Republic

Wei Zhan (73)
Department of Computer Science,
Princeton University, NJ, USA

Linpeng Zhang (117)
Department of Computer Science,
University College London, UK

Qianfan Zhang (68)
IIIS, Tsinghua University, Beijing, China

Tianyi Zhang (114)
Tsinghua University, Beijing, China

Yu Zheng (54)
Johns Hopkins University, Baltimore, MD, USA

Rudy Zhou (77)
Tepper School of Business,
Carnegie Mellon University,
Pittsburgh, PA, USA

Samson Zhou (112)
Carnegie Mellon University,
Pittsburgh, PA, USA

Robin Ziemek (1)
Technische Universität Dresden, Germany

Elia C. Zirondelli (43)
Department of Mathematics,
University of Trento, Italy;
Department of Computer Science,
University of Verona, Italy

Philipp Zschoche (26)
Faculty IV, Algorithmics and Computational
Complexity, Technische Universität Berlin,
Germany

Stanislav Živný (121)
Department of Computer Science,
University of Oxford, UK

From Verification to Causality-Based Explications

Christel Baier
Technische Universität Dresden, Germany

Clemens Dubslaff
Technische Universität Dresden, Germany

Florian Funke
Technische Universität Dresden, Germany

Simon Jantsch
Technische Universität Dresden, Germany

Rupak Majumdar
MPI-SWS, Kaiserslautern, Germany

Jakob Piribauer
Technische Universität Dresden, Germany

Robin Ziemek
Technische Universität Dresden, Germany

Abstract

In view of the growing complexity of modern software architectures, formal models are increasingly used to understand *why* a system works the way it does, opposed to simply verifying *that* it behaves as intended. This paper surveys approaches to formally explicate the observable behavior of reactive systems. We describe how Halpern and Pearl's notion of actual causation inspired verification-oriented studies of cause-effect relationships in the evolution of a system. A second focus lies on applications of the Shapley value to responsibility ascriptions, aimed to measure the influence of an event on an observable effect. Finally, formal approaches to probabilistic causation are collected and connected, and their relevance to the understanding of probabilistic systems is discussed.

2012 ACM Subject Classification Theory of computation → Logic and verification

Keywords and phrases Model Checking, Causality, Responsibility, Counterfactuals, Shapley value

Digital Object Identifier 10.4230/LIPIcs.ICALP.2021.1

Category Invited Talk

Funding This work was funded by DFG grant 389792660 as part of TRR 248 – CPEC (see `https://perspicuous-computing.science`), the Cluster of Excellence EXC 2050/1 (CeTI, project ID 390696704, as part of Germany's Excellence Strategy), DFG-projects BA-1679/11-1 and BA-1679/12-1, and the European Research Council under the Grant Agreement 610150 (`http://www.impact-erc.eu/`) (ERC Synergy Grant ImPACT).

1 Introduction

Modern software systems are increasingly complex and even small changes to a system or its environment may lead to unforeseen and disastrous behaviors. As software controls more aspects of our lives everyday, it is desirable – and for widespread acceptance in societal decisions, eventually inevitable – to have comprehensive and powerful techniques available to understand what a system does.

The field of formal methods has developed a portfolio of tools that provide confidence in the working of complex software systems. In formal methods, one builds a formal model of a system and specifies its desired behavior in an appropriate (temporal) logical formalism. Algorithmic techniques such as model checking [12, 31] can answer the question whether the model satisfies the specification, or in other words, whether the system behaves as intended, often in a "push button" way. Moreover, an important aspect of these algorithms is that they can produce independently verifiable justifications of their outcome, such as *counterexamples* or *certificates* to justify the violation or correctness of a property, respectively. Since the earliest successes of model checking, the availability of counterexample traces was stated as a major advantage for the method over deductive verification [30]. As model checkers became

more complex, concerns about their correct implementation led to research on producing certificates for correctness. Examples are inductive invariants or derivations in a deductive system [62, 64, 44, 70] that can be checked independently from the verification process.

While certificates and counterexample traces can provide a useful explication about the behavior of a system, they only provide rudimentary understanding of *why* a system works the way it does. In epistemic terms, the outcome of model checking applied to a system and a specification provides *knowledge that* a system satisfies a specification or not in terms of an assertion (whether the system satisfies the specification) and a justification (certificate or counterexample) to increase the belief in the result. However, model checking usually does not provide *understanding* on *why* a system behaves in a certain way. Such an understanding can be obtained by causal links between possible events and their observed outcome.[1]

The need to better understand why a system is correct or incorrect has led to a broad research program on models and reasoning methods that aim to provide such knowledge of causes (see, e.g., [97, 98]). The goal of this survey is to summarize research on causal reasoning in the field of verification and highlight the challenges that lie ahead.

The first step in understanding knowledge of causes is the mathematical formulation and study of *causality*. Grasping the intuitive concept of cause-effect relationships in a formal model has proved notoriously difficult. Centuries of philosophical reasoning on the subject have distilled the *counterfactuality principle* [67, 68, 88] as a central feature of what constitutes an actual cause: if the cause had not occurred, then the effect would not have happened. While the counterfactuality principle was generally agreed upon, a rigorous mathematical formulation was developed only recently, through the seminal work of Halpern and Pearl and their coworkers [95, 45, 46, 55]. In a nutshell, they model causal systems using *structural equation models*, and provide a set of axioms to characterize when an event is an actual cause of another. We provide a summary of the foundations of causality and some of their applications in verification in Section 2.

While causality is a qualitative concept, in that an event is an actual cause of another or is not, more recent work considers *quantitative* measures of *responsibility*. Responsibility measures the relative importance that an event had in causing another event. In other words, the responsibility of an acting agent gauges what fraction of an observable effect can be attributed to that agent's behavior. Here, an agent could be, e.g., an individual, a coalition, a software component, or device in a computer network. Chockler and Halpern [23] define the *degree of responsibility* of an actual cause in the Halpern-Pearl sense based on the cardinality of the smallest witness change that makes an event a cause of another. A more recent strand for formalizing responsibility is based on the *Shapley value* [106]. In cooperative game theory, the Shapley value measures the influence of an agent on the outcome jointly brought about by the agents and is classically used to find a fair division of a cost or a surplus among them. The appeal of the Shapley value stems on the one hand from its uniqueness with respect to a relatively simple set of axioms, and on the other hand from its seemingly universal applicability. Research on employing Shapley-like values for the explication of machine-learning predictions [117, 35, 91, 1, 110] and the behavior of formal models [119, 11, 93, 10] is currently very active. A summary of applications of Shapley values in the verification context is provided in Section 3.

[1] Epistemologists since Gettier [47] will recognize that justified true belief does not constitute a comprehensive theory of knowledge. For the same reason, a theory of understanding as knowledge of causes is a matter of vigorous debate, with Gettier-like counterexamples [52, 100, 101]. These subtle epistemic issues are orthogonal to our work.

From a systematic viewpoint, causality and responsibility can be understood in either a *backward* or a *forward* manner [113]. In the backward or *ex post* setting, an effect has already transpired, and the goal is to describe its causes and determine their relative influence in producing the effect. The actual causation framework by Halpern and Pearl follows this paradigm, and therefore also most approaches presented in Section 2. In the forward or *ex ante* setting, a reasoning model includes possible contingencies and the goal is to characterize the global power of agents and events in affecting the outcome. The forward responsibility in game structures (see Section 3.1) and the importance value for temporal logics (see Section 3.2) pursue this pattern. There are also attempts to express forward-looking causality notions by structural equations in the context of *accountability* [73]. Seen from an operational angle, the distinction between backward and forward notions loosely relates to when the causal analysis is executed. Backward notions tend to be applicable at inspection time, e.g., to guide the debugging process in post mortem analyses. Forward notions are prone to be used at design time of the system, laying out general phenomena of its inner workings.

Finally, we consider causality in the setting of probabilistic models. Unlike the deterministic setting, mathematical notions of causality and responsibility are less understood. There is widespread agreement in the philosophy literature that a quintessential characteristic of causes in the probabilistic setting is the *probability-raising property* [111, 21, 107, 39]: an occurrence of the cause should increase the probability of subsequently observing the effect. Nevertheless, it has also been observed that simply taking probabilities often leads to counterintuitive phenomena owing to mutual dependencies and latent correlations with other events [104, 21, 107]. Section 4 discusses attempts to formulate probability-raising approaches to causation for operational probabilistic models. These approaches tend to produce forward notions since probabilities inherently refer to a collection of evolutions in which an event happened. There are numerous philosophical accounts on actual probabilistic causation [89, 84, 42]. In terms of formal models, Pearl's early notion of actual causality in terms of *causal beams* [96, 97] entailed probabilistic flavor, and the *causal probabilistic logic* of [115, 14] describes a language for reasoning about probabilistic causation. Nonetheless, we are aware of only a few works that study a probabilistic version of causality in operational models (see, e.g., [75, 2, 37, 9]). Along these lines, we point out open directions for research that focus on the operational point of view.

2 Counterfactual Notions of Causality

An important starting point for the study of causality in formal methods is the influential work by Halpern and Pearl [58, 59, 60, 56, 57] on *actual causality*, henceforth abbreviated *HP causality*. We provide a brief and informal overview of their definition.

Halpern and Pearl use *structural equation systems* as a modeling language for causal models. A causal model relies on exogeneous variables U and endogeneous variables V, representing external or independent factors and internal factors, respectively. The value of each endogeneous variable $x \in V$ is specified by a deterministic function f_x that may depend on exogeneous variables and on endogeneous variables that are preceding x with respect to a fixed order on V. Intuitively, a causal model can be thought of as an arithmetic circuit whose primary inputs are the exogeneous variables and where some of whose internal nodes are labeled by the endogeneous variables. The circuit then specifies the functions defining the endogeneous variables as well as the dependencies between the variables.

More formally, let $M = (U, V, \{f_x\}_{x \in V})$ be a causal model. Given a formula φ over the exogeneous and endogeneous variables (in some appropriate logic), and a *context* $\vec{u}$ that assigns values to all variables in U, the goal of actual causality is to state whether an assignment of values $\vec{X} = \vec{x}$ to a subset $X \subseteq V$ is a cause of φ. Halpern and Pearl define $\vec{X} = \vec{x}$ to be a *cause* of a formula φ in $(M, \vec{u})$ if the following three axioms hold.

AC1 both the cause and the effect are true: the model $(M, \vec{u})$ satisfies $\vec{X} = \vec{x}$ as well as φ,
AC2 the principle of counterfactual dependence (discussed below), and
AC3 causes are minimal: no partial assignment of $\vec{X} = \vec{x}$ satisfies AC1 and AC2.

The key to AC2 is captured by the notion of *interventions*, describing a direct assignment of values to some endogeneous variables while disregarding their defining functions. Formally, $[\vec{Y} \leftarrow \vec{y}]$ stands for the intervention on variables $Y \subseteq V$ by assigning them values $\vec{y}$ and leaving all other values for variables $V \setminus Y$ to follow from their defining functions. Then, $[\vec{Y} \leftarrow \vec{y}]\psi$ describes the impact of an intervention on a formula ψ. An intervention thus can represent a counterfactual: *what if* variables in Y took values $\vec{y}$ instead of their actual values? Turning back to the definition of actual causes for φ, axiom AC2 now requires the existence of an intervention $[\vec{X} \leftarrow \vec{x}']$ on the variables in X such that the effect φ is not observable, i.e., $[\vec{X} \leftarrow \vec{x}']\neg\varphi$ holds in $(M, \vec{u})$. The precise definition of AC2 is, however, more involved and several variants exist for AC2 to account for different settings and applications.[2]

2.1 Instances of HP Causality in Verification

Principles of causality have been used, often implicitly, in formal verification for a long time. An early example is *program slicing* (see, e.g., [61]) where by following program dependencies one aims to identify approximations of an actual cause for reaching a program location. Causality is also a key concept in error localization, the problem of reducing a counterexample trace for ease of debugging [120, 13, 105, 53, 118, 72, 121]. A correspondence of causality in counterexample traces to finding minimal `UNSAT` cores has been identified in [16]. Early and influential work on causality in formal verification is exemplified by research on *vacuity* and *coverage*. Vacuity [17, 81, 102] explicates whether a positive verification result originates from an unintended trivial behavior. Coverage [65, 25, 27, 26] is dual to vacuity, and explicates whether certain parts of the system were not relevant for the successful result. While for determining vacuity one considers small changes to the specification and checks whether these change the result, coverage is obtained by perturbations to the system rather than the specification and is actually a particular instance of HP causality.

Temporal logics play a crucial role in the verification context to describe properties of and requirements on the system. Common temporal logics are, e.g., *computation tree logic* (CTL) [29] or *linear temporal logic* (LTL) [99]. In LTL, e.g., $\neg E \,\mathcal{U}\, C$ describes that an effect E does not occur before a cause C and $\Diamond E$ stands for the effect E to eventually occur.

From Coverage to HP Causality. Coverage itself is a concept with a manifold of incarnations and we focus here on the formalization by [24], where the connection of coverage to HP causality has been addressed. The operational model is provided by a Kripke structure K, i.e., a finite directed graph over states labeled by atomic propositions. Further, we are given an atomic proposition q and a specification ψ expressed in an appropriate (temporal)

[2] Halpern and Pearl's definition of causality underwent a considerable amount of development over the past 20 years, primarily varying AC2. One usually distinguishes the *original* version [58], the *updated* version [59, 60], and the *modified* version [56, 57] (alongside variations by other authors [54, 63]).

logic over the set of atomic propositions such that K satisfies ψ. Then, a state s of K is said to be *q-covered* if changing the truth value of q in s leads to a structure that does not satisfy ψ. Considering the hypothetical mutant system in which q takes the opposite value in s corresponds to a counterfactual notion from the causality literature. Yet, coverage only allows simple counterfactuals containing *individual* changes to the system. As pointed out in [24], it is for this reason that coverage at times fails to express deeper dependencies involved in the satisfaction of ψ.

To define a *cause* in this setting, one can consider the following simple causal model: for each state s, there is one endogeneous variable v_s, which specifies whether the value of q in s is swapped in contrast to the original structure K, or not. One first refers to a context where all variables v_s are set to `false` and then considers possible swap operations. From K and ψ one can derive a Boolean function φ over the endogeneous variables V such that an instantiation $I\colon V \to \{\texttt{true}, \texttt{false}\}$ satisfies φ if and only if swapping the truth value of q exactly in states s with $I(v_s) = \texttt{true}$ leads to a structure satisfying ψ. Now s is a *cause of ψ with respect to q* [24, Definition 3.2] if there exists a set of variables Y such that $[Y \leftarrow \texttt{true}]\varphi$ and $[Y \cup \{v_s\} \leftarrow \texttt{true}]\neg\varphi$ hold. In other words, s is a cause if there exists a set of states S' (corresponding to variables Y) such that swapping q in S' leads to a structure satisfying ψ[3], but swapping the value of q in S' and s gives a structure falsifying ψ. These two conditions postulate precisely the axiom AC2, which takes a simpler form than usual thanks to the lack of higher-order dependencies among the variables in this causal model. In the presented causal model, axiom AC1 holds by the assumption that K satisfies ψ, and the minimality axiom AC3 is trivially fulfilled as only single states are considered as potential causes.

While this causal model is very simple, in particular it does not include any dependencies in between variables, the work in [24] shows that even such restricted models are useful.

Fault Localization. The causality interpretation of coverage presented above takes a forward-looking perspective in that changes to the system are globally tested against the given specification. In [16], a similar approach is applied to the backward-oriented setting of fault localization, i.e., the problem of pointing out those parts of a (finite) counterexample trace π that are most relevant for violation of a given linear-time specification φ. In this incarnation of HP causality, the endogeneous variables V contain a variable $v_{(s,q)}$ for each pair consisting of a state s and atomic proposition q of the Kripke structure. These variables can take values $\{\texttt{true}, \texttt{false}\}$ and the interpretation is exactly as before, namely $v_{(s,q)} = \texttt{true}$ means that the truth value of q in state s is changed in contrast to the initial context. Moreover, the axiom AC2 takes the same form as in the previous case. Specifically, it expresses that there is a set of variables $Y \subseteq V$ such that changing the truth value for the corresponding state-proposition pairs lets π still violate φ, while additionally swapping q in s leads to π satisfying φ (interpreted over a weak semantics of LTL on finite paths).

Counterfactual Reasoning for Configurable Systems. Nowadays, almost every practical software system is configurable, let it be using `#ifdef` constraints or through *features* [5]. Features inherently have a designated meaning, usually expressed by their name, e.g., a "verbose" feature indicates that the software will expose additional information during runtime. Debugging configurable systems is challenging, as the number of possible systems suffers from an exponential blowup in the number of features. While there are specifically tailored methods for analyzing configurable systems [112], e.g., through model checking [32, 38],

[3] The updated version of HP causality [59, 60] would require this condition also for all subsets of S'.

research on identifying root causes in configurable systems on the abstraction level of features is still in its infancy. Such a causal analysis can provide useful insights for debugging: developers can focus on the parts implementing the features identified to be responsible for the bug, and users can obtain suggestions to reconfigure the system to not expose the bug. First ideas to explicate which feature activations and deactivations cause an effect in configurable systems were described in [6]. There, the set of feature configurations with observable effect is obtained by configurable systems analysis, e.g., through family-based verification [32, 112, 38, 28]. Exploiting the Boolean case of HP causality [40, 69], those partial feature configurations can be determined where the corresponding systems all show the effect (see AC1), for which there is a reconfiguration that does not exhibit the effect (AC2), and that are minimal (AC3).

2.2 Further Approaches Inspired by HP Causality

The work [34] presents a formal definition of actual causes in the setting of concurrent interacting programs. Originating from logs written by the concurrent system, the goal is to localize causes in those *program actions* that are most relevant for the violation of a desirable property. The approach is investigated in detail for the prominent class of safety properties, with a view towards legal accountability in security-critical systems.

In [78], a causality-based approach to explain *timed diagnostic traces* has been presented, which are used as counterexamples for model-checking results in timed systems. Such traces represent a set of violating executions and the goal of [78] is to compute the parts that can be considered causal for violating the property.

A different definition inspired by HP causality was used in [86]. There, causes for reachability properties are formulas of a temporal logic called *event order logic*, used to describe temporal relations between events. Algorithms to compute causes in this sense were also studied in [15, 77], and the approach was extended to handle general LTL formulas as effects rather than just reachability in [20].

In [49, 50] the authors argue that the HP causality, which is propositional in nature, is not the ideal starting point for a framework of causality in formal verification. They present a formalism which is based on counterfactual reasoning, uses system traces as first-class objects and is designed to work for compositional systems. In [51] the formalism is further generalized by defining abstract *counterfactual builders*, which specify what alternative scenarios should be considered for counterfactual reasoning. Further, [51] also considers hyperproperties as specifications. While hyperproperties are useful to specifying system properties, it was observed in [33] that they can also be used to formalize causality. Similar observations have been made for probabilistic causation [2, 37].

3 Responsibility and Shapley-like Ascriptions

While the previous section defined and applied qualitative concepts of causation, this section shifts the focus towards quantitative approaches of *responsibility*. Loosely speaking, responsibility refers to a numerical value designed to measure how much weight an event had in producing an effect, relative to concurring or competing events linked to the same effect. There is widespread agreement that a necessary condition for assuming responsibility is causal relevance of the event in question to the effect [41, 18]. As a consequence, the term *responsibility* usually builds directly or indirectly on concepts of causality. While the specific numerical value in a notion of responsibility may not have a semantic content, it can order the events in terms of their causal relevance.

Chockler and Halpern [23] introduced the notion of *degree of responsibility*, which is attributed to actual causes in causal models of HP causality. This degree measures how many changes to the evolution of events are necessary until counterfactual values for the actual cause change the observable effect. In [24] this notion is combined with the study of mutant coverage to build a degree of responsibility in CTL model checking assigned to state-proposition pairs (see also Section 2.1).

The degree of responsibility measures the influence of an event by looking at how many further counterfactual changes are (minimally) required to swap the effect, but is does not take into account how many such minimal sets of changes exist. One can argue that a cause is individually more influential if it admits *many* such sets since this means less dependencies on other events. This rationale has generated an active strand in formalization of responsibility based on the *Shapley value* [106]. The Shapley value is a central solution concept from theoretical economics and was originally designed to find a fair distribution of a financial surplus that was brought about cooperatively by a number of producers.

Formally, a *cooperative game* with n players is a mapping $g\colon 2^{[n]} \to \mathbb{R}$ such that $g(\emptyset) = 0$, where $[n] = \{1, \ldots, n\}$. The value $g(C)$ is meant to represent the surplus (or, depending on the specific situation, the cost) that the coalition $C \subseteq [n]$ can ensure upon acting collaboratively. The Shapley value of player i is then defined as

$$\mathrm{Sh}(i) \;=\; \frac{1}{n!} \cdot \sum_{\pi \in S_n} g(\pi_{\geq i}) - g(\pi_{\geq i} \setminus \{i\}) \tag{1}$$

where S_n denotes the set of self-bijections $[n] \to [n]$ and where $\pi_{\geq i} = \{j \in [n] \mid \pi(j) \geq \pi(i)\}$ for a given $\pi \in S_n$. Intuitively, $g(\pi_{\geq i}) - g(\pi_{\geq i} \setminus \{i\})$ describes the marginal contribution of player i to the coalition $\pi_{\geq i}$. The Shapley value takes the average of all such marginal contributions. Thus, $\mathrm{Sh}(i)$ is a measure for the overall influence of player i in the game g.

The general setup of cooperative games as real-valued functions on the power set of $[n]$ makes the Shapley value amenable to measuring the influence of abstract players in formalized situations of collaborative interaction. This rationale has recently been invoked for the interpretation of machine learning models [117, 35, 91, 1, 110]. In this case, the players are the input parameters to a machine learning model, and the Shapley value has the goal to measure the influence of each parameter on the output of the model.

This section outlines three approaches that employ the Shapley value as a means to distribute an overall effect into individual responsibilities. In Section 3.1 the general setting of an *extensive form game* is chosen and responsibilities are attributed to its players with respect to producing a certain outcome. Section 3.2 discusses a notion of importance of states for the satisfaction of an LTL property in a Kripke structure. Section 3.3 finally presents an extension of the Shapley value that can be used to define responsibilities in a setting of continuously varying parameters.

3.1 Responsibility in Game Structures

As summarized in Section 2, causal models are by now a fundamental building block for notions of actual causation in the verification domain. However, in complex scenarios that involve cooperative interaction, non-cooperative competition, and imperfect information, they fall short of modeling various natural features such as temporal sequentiality, knowledge, and agency. The work [10] presents an approach to establish notions of responsibility in these strategic settings by passing to *extensive form games* [116, 80]. These provide a popular formalism for studying the dynamics that underlie strategic interaction in the presence of

competing objectives. In a nutshell, an extensive form game is an explicit presentation of the strategic scenario in terms of a tree structure whose edges describe the transitions between states when actions are taken, certain states may be indistinguishable for the players given their knowledge, and each path from the root to a leaf is associated with an outcome. Apart from being a highly expressive model, a century of research on the subject has generated a rich set of solution concepts on which a study of responsibility can build, primarily following economic rationales.

In [10] three responsibility notions are defined with respect to an event E that is encoded by a binary labelling on the leaves of the game tree, i.e., E took place on a play or not. All three notions follow the common two-step process consisting of first defining (qualitatively) what it means for a coalition C to be responsible and then extracting (quantitatively) an individual responsibility value through an application of the Shapley value on coalition responsibilities. That is, one takes the cooperative game g to take binary values $\{0, 1\}$ depending on whether or not a coalition is responsible. They also share the counterfactual paradigm in that a necessary condition for being responsible for the occurrence of E is the power to preclude E. While the notions can be ordered according to their logical strength, they are perhaps best explained along two lines of distinction given by the *temporal perspective* and *epistemic state*.

The temporal perspective can be either *forward-looking* or *backward-looking* [113]. For forward-looking notions one attains a prospective, *ex ante* viewpoint that studies the preclusive power for the game as a whole. The forward-looking notion put forth in [10] is called *forward responsibility* and requires the coalition to possess a strategy that globally avoids E. In contrast, backward-looking notions consider a specific play from a retrospective, *ex post* viewpoint and study who was responsible for E as the play evolved.

Depending on how the epistemic state is taken into account, [10] distinguishes *strategic* backward responsibility and *causal* backward responsibility. In order for a coalition to be strategically backward responsible, it must have had the power to avoid E at some point on the play and it must have been aware of this fact *given its epistemic knowledge*. In situations of imperfect information, this latter condition is crucial for arriving at a *responsibility-as-capacity* notion [113] in a strategic sense that goes beyond a mere counterfactuality check: when one does not know all relevant information, one can even bring about E inadvertently or unintendedly. Causal backward responsibility essentially drops the latter requirement in that the coalition is able to avoid E from some point on, everything else held fixed. This corresponds to the *responsibility-as-cause* from the classification presented in [113].

There is a translation of a causal model into an extensive form game under which causal backward responsibility corresponds exactly to but-for causes [10]. It can therefore happen that a player is causally backward responsible without belonging to an actual cause in the HP causality sense [59] and, therefore, with degree of responsibility 0 in the sense of [23]. In the prototypical example in which Suzy and Billy both throw rocks at a bottle and Suzy's stone hits first, Billy's degree of responsibility is 0, while both are attributed causal backward responsibility 1/2. Since both players acted in exactly the same way based on the same information, there is reason to favor the latter symmetric notion, and avoid actual causation *en route* to accurately model intuitive responsibility concepts. A detailed comparison to causal models, other notions of responsibility in strategic games of imperfect information [119], and proof-theoretic approaches to formalize responsibility [19, 94] is given in [10].

3.2 The Importance Value for Temporal Logics

The paradigm of passing from binary coalitional responsibilities to quantitative individual responsibilities by virtue of the Shapley value is also applied in [93] to model check Kripke structures against temporal logic specifications. The resulting notion is called the *importance value* and measures the influence of a state in a system for the satisfaction of a given specification. Intuitively, a state is important in this framework if the way that the nondeterministic choices of the state are resolved has a large impact on whether the given specification is met.

Formally, let K be a Kripke structure with states S and a dedicated initial state, and φ be an LTL formula. Then one defines the cooperative game $g\colon 2^S \to \{0,1\}$ using an induced two-player game as follows. For a set of states $C \subseteq S$ we let $\mathcal{G}_C$ be the two-player game over the arena K where player `SAT` controls the states in C, player `UNSAT` controls the states in $S \backslash C$ and the winning condition is φ. Then, $g(C) = 1$ if player `SAT` wins $\mathcal{G}_C$, and $g(C) = 0$ otherwise. With this definition, the importance value $\mathcal{I}(s)$ of a state $s \in S$ with respect to K and φ is defined to be the Shapley value of player s in g (see Equation (1)). The notion can be straightforwardly extended to define the importance of a set of states $P_i \subseteq S$, where $S = P_1 \dot\cup \ldots \dot\cup P_n$ is a given partition of the state space. This generalization is intended to take an existing compositional structure of the system appropriately into account.

The work [93] studies the associated computational problems of deciding whether $\mathcal{I}(s) > 0$ (called the *usefulness problem*) and deciding whether $\mathcal{I}(s) > \eta$ for a rational threshold η. The intrinsic complexity of solving two-player LTL-games (the decision problem is 2EXPTIME-complete) carries over to these problems. This computational intractability of the importance value motivates further studying the complexity when restricted to fragments of LTL, and tight complexity results were shown in [93] for a wide range of specifications.

In [93], the presented framework is also applied to CTL model checking of *modal transition systems* (MTS) [85]. MTSs have two levels of nondeterminism: the standard nondeterminism of the underlying graph and additionally a choice on which of the transitions in a state are actually included in the system. The latter kind of nondeterminism is used to design a two-player game where one player tries to satisfy the CTL specification and the other player tries to violate it. However, since the semantics of CTL relies on infinite trees and the order in which the branches are evolving has a strong impact on which player wins, there does not appear to be a natural candidate game that proceeds in a turn-by-turn fashion. Hence [93] considers *one-shot games* in which the players commit to a valid set of transitions in the states under their control once at the beginning of their play. This determines once more a binary cooperative game g that induces importance values in the same way as for LTL.

There is a straightforward generalization of the importance value to a $2\frac{1}{2}$-player game $\mathcal{G}$ in which the actions taken by the players are associated with probability distributions over the states. In this formalism, the players each make non-deterministic choices among its available actions, but the actual successor state then depends on a random choice according to the associated distribution. Given an LTL specification, the goal of `SAT` is to maximize the probability that the resulting path satisfies the specification, while `UNSAT` tries to minimize it. These $2\frac{1}{2}$-player games are *determined* in a quantitative sense [92]: the maximal probability that can be enforced by `SAT` against all strategies of `UNSAT` is 1 minus the minimal probability that can be enforced by `UNSAT` against all strategies of `SAT`. This probability is called the *value* $\mathrm{val}(\mathcal{G})$ of the game (see also the survey [22]). Let $S = S_{\texttt{SAT}} \dot\cup S_{\texttt{UNSAT}}$ be the partition of the states of $\mathcal{G}$ into those under control of `SAT` and `UNSAT`, respectively. For a subset $C \subseteq S_{\texttt{SAT}}$ the value $g(C)$ is then defined as $\mathrm{val}(\mathcal{G}_C)$, where $\mathcal{G}_C$ is the $2\frac{1}{2}$-player game obtained from $\mathcal{G}$ by putting the states in $S \backslash C$ under the control of player `UNSAT`. Taking Shapley values as above then induces the importance value of a state in $S_{\texttt{SAT}}$.

3.3 Attributing Responsibility in Continuous Models

The Shapley value [106] is an inherently discrete solution concept. On the other hand, realistic formal models of reactive systems often entail continuous features such as timing [4, 36, 90], physical phenomena [3, 108], or parametric dependencies [71, 48, 79]. Notions of responsibility for these models therefore tend to require new mathematical approaches if the continuous nature is to be taken into account appropriately.

The continuous scenario seen from an economic angle generalizes (discrete) cooperative games: rather than just participating in a coalition, the n players of a game each pick a value v_i from a continuous domain $D_i \subseteq \mathbb{R}$ including 0 and the (generalized) game then determines a collective surplus or cost based on this input. This is formally described by a continuously differentiable function $g\colon D_1 \times \ldots \times D_n \to \mathbb{R}$ such that $g(0, \ldots, 0) = 0$. Economists usually take the domains to be of the form $D_i = [0, m_i)$ for some maximal input $m_i \in \mathbb{R}$, and further assume g to be non-decreasing with non-negative range. The *Aumann-Shapley value* [7] is a generalization of the Shapley value designed to provide a solution to the question how the value $g(v_1, \ldots, v_n)$ should be "fairly" distributed among the players. It is one instance of what is called a *cost-sharing scheme* and admits an axiomatization in the spirit of its discrete predecessor [43, 110].

Inspired by this model, the work [11] presents an approach to measure the relative importance of the parameters on the behavior of *parametric Markov chains* for a wide range of properties, including ω-regular specifications, specifications in probabilistic CTL, and on expected rewards. Here, a parametric Markov chain is a directed graph where each edge is assigned a probability that may depend on a set of parameters such that for each instantiation of the parameters the probabilities outgoing from a state sum up to 1. A parametric Markov chain instantiated with fixed values for the parameters then coincides with a *discrete-time Markov chain* (DTMC). For this purpose the aforementioned assumptions on g must be relaxed: the continuously differentiable function $g\colon D \to \mathbb{R}$ has arbitrary domain $D \subseteq \mathbb{R}^n$ and is not subject to monotonicity and non-negativity restrictions. This also means that the canonical baseline value 0 for v_i is not always available anymore. The responsibility problem in this generalized setting then reads as follows: given g and two parameter choices $v, v' \in D$, how *responsible* is the i-th parameter for the observable change $g(v') - g(v)$? In this slightly generalized form, the Aumann-Shapley value of the i-th parameter is defined as

$$\mathrm{AS}_i(g, v, v') = (v'_i - v_i) \cdot \int_0^1 \partial_i g\left(v + \alpha(v' - v)\right) d\alpha. \tag{2}$$

The integrand involves the i-th partial derivative of g and intuitively measures the marginal contribution of the i-th parameter at the points lying between v and v'. The integral then takes the average of these contributions along the straight line from v to v'. While taking the straight line is desirable in an economic context to meet the *average cost for homogeneous goods axiom* [43], this axiom is often void of meaning when applied to formal systems. When one replaces the straight line in Equation (2) with an arbitrary (monotonic) path from v to v', then one speaks of *path attribution schemes* [109]. Of course, taking different paths induces different attributions, and which ones should be considered worthwhile depends on the specific scenario. This could for instance be due to potential restrictions on the way that changes on the parameters can be implemented in practice. The work [11] applies these path attribution schemes to the function induced by ω-regular or probabilistic CTL specifications on a parametric Markov chain. The set of axioms presented there is adjusted to this particular situation and justifies why one can conceive the value $\mathrm{AS}_i(g, v, v')$ as the fraction of the observable effect $g(v') - g(v)$ that is produced by the i-th parameter.

It is noteworthy, however, that the approach put forth in [11] is by no means specific to the context of parametric Markov chains. Any scenario in which continuously varying parameters determine a value can in principle be handled similarly. Of course, which path attribution schemes should be regarded as meaningful needs to be checked case-by-case, and corresponding axiomatizations should be chosen with care. But it is no accident that the main decidability result in [11] is formulated in terms of path attribution schemes on functions in n independent variables – a generality that makes the approach potentially applicable for a range of similar problems.

4 Probabilistic Causation

As seen in the preceding sections, notions of causality and responsibility have been widely explored in the non-probabilistic setting. In contrast, there have been far less attempts at defining a suitable notion of causes for probabilistic operational systems such as Markov chains. However, probabilistic theories of causation have been considered in various philosophical accounts [111, 21, 107, 39]. One central idea behind these theories is the *probability-raising principle*, which goes back to Reichenbach [103, 104]. It states that causes should raise the probability of their effects. After observing a cause C, the probability of an effect E is higher than after observing that the cause has not occurred. Formulated with conditional probabilities, this can be written as

$$\Pr(E \mid C) > \Pr(E \mid \neg C), \qquad \text{or equivalently} \qquad \Pr(E \mid C) > \Pr(E).$$

For the conditional probabilities to be well-defined, it is necessary that $\Pr(C) > 0$ and $\Pr(\neg C) > 0$. Later on, we will make sure that the events conditioned on have positive probability. Note that if $\Pr(C) > 0$ and $\Pr(E \mid C) > \Pr(E)$, it already follows that $\Pr(\neg C) > 0$. Defining $p \stackrel{\text{def}}{=} \Pr(C)$, the equivalence of the two inequalities follows from the equation $\Pr(E) = p \cdot \Pr(E \mid C) + (1-p) \cdot \Pr(E \mid \neg C)$, which implies

$$\Pr(E \mid C) - \Pr(E) = (1-p)(\Pr(E \mid C) - \Pr(E \mid \neg C)).$$

The probability-raising principle alone, however, cannot distinguish between cause and effect as it holds if and only if $\Pr(C \mid E) > \Pr(C \mid \neg E)$ as well. For this reason, additional conditions have to be imposed for causal reasoning. One key condition is temporal priority, which prescribes that a cause has to occur *before* the effect.

This section formalizes both the probability-raising principle as well as the requirement of temporal priority for probabilistic operational models. We draw connections between different ideas from the literature to provide an overview over basic probabilistic notions of causality in the context of formal verification. For this, we assume to have given a DTMC $\mathcal{M}$ with a probability distribution over initial states. This way, the sets Π_φ of paths starting in initial states and fulfilling an LTL property φ are measurable [114] and have a well-defined probability value $\Pr_{\mathcal{M}}(\Pi_\varphi)$, which we also denote by $\Pr_{\mathcal{M}}(\varphi)$. Applying the probability-raising principle and expressing the temporal priority using LTL leads to the following first definition of causality in DTMCs for reachability properties.

▶ **Definition 1** (reachability-cause). *Let $\mathcal{M}$ be a DTMC with state space S and let $C, E \subseteq S$ be two disjoint sets of states. Then C is a* reachability-cause *of E if $\Pr_{\mathcal{M}}(\neg E \,\mathcal{U}\, C) > 0$ and*

$$\Pr_{\mathcal{M}}(\Diamond E \mid \neg E \,\mathcal{U}\, C) > \Pr_{\mathcal{M}}(\Diamond E). \tag{3}$$

Note that Equation (3) implies that $\Pr_{\mathcal{M}}\big(\neg(\neg E \,\mathcal{U}\, C)\big) > 0$. This ensures that also $\Pr_{\mathcal{M}}\big(\Diamond E \mid \neg(\neg E \,\mathcal{U}\, C)\big)$ is well-defined and so Equation (3) is equivalent to $\Pr_{\mathcal{M}}(\Diamond E \mid \neg E \,\mathcal{U}\, C) > \Pr_{\mathcal{M}}\big(\Diamond E \mid \neg(\neg E \,\mathcal{U}\, C)\big)$. If there are no paths first reaching the effect E and afterwards the cause C, e.g., because the states in the effect E are absorbing, Equation (3) simplifies to $\Pr_{\mathcal{M}}(\Diamond E \mid \Diamond C) > \Pr_{\mathcal{M}}(\Diamond E)$.

In this treatment of reachability properties, a cause C specifies the set of finite executions ending in C that cause the subsequent extension to an infinite execution to satisfy $\Diamond E$. This idea can be lifted to the treatment of causes of arbitrary events in $\mathcal{M}$ specified by a measurable set of infinite paths $\mathcal{L} \subseteq S^\omega$. A cause is then a set of finite paths $\Gamma \subseteq S^+$. Besides the probability-raising property, the temporal priority condition needs to be included. For path properties this needs extra consideration. While for a cause $\Gamma \subseteq S^+$ it is clear that the cause is observed once a finite path in Γ is generated in a DTMC, this is not the case for the effect $\mathcal{L} \subseteq S^\omega$ as it consists of infinite executions. However, it seems natural to say that the effect occurred on a finite path δ whenever $\Pr_{\mathcal{M}}(\mathcal{L} \mid \delta) = 1$, i.e., if a generated finite path ensures that almost all infinite extensions belong to $\mathcal{L}$. Here, we used δ to also denote the event of all infinite paths having δ as a prefix. Analogously, for a set of finite paths Γ, we denote by Γ also the event of all infinite paths with a prefix in Γ. Consequently, $\neg\Gamma$ denotes the event of all infinite paths that have no prefix in Γ. The discussed treatment of temporal priority is now used in the following definition of a cause in a DTMC.

▶ **Definition 2** (global PR-cause). *Let $\mathcal{M}$ be a DTMC with state space S, let $\Gamma \subseteq S^+$ be a non-empty set of finite paths, and let $\mathcal{L} \subseteq S^\omega$ be a measurable set of paths. Then, Γ is a* global probability-raising cause (global PR-cause) *for $\mathcal{L}$ if the following two conditions hold:*
PAC1: $\Pr_{\mathcal{M}}(\mathcal{L} \mid \Gamma) > \Pr_{\mathcal{M}}(\mathcal{L})$, *and*
PAC2: *for all $\gamma \in \Gamma$, no proper prefix γ' of γ satisfies $\Pr_{\mathcal{M}}(\mathcal{L} \mid \gamma') = 1$.*
As Γ is a set of finite paths in $\mathcal{M}$, the cylinder set spanned by each $\gamma' \in \Gamma$ has positive probability. So, the conditional probabilities $\Pr_{\mathcal{M}}(\mathcal{L} \mid \Gamma)$ and $\Pr_{\mathcal{M}}(\mathcal{L} \mid \gamma')$ in Definition 2 are well-defined.

Axiom PAC1 expresses the probability-raising principle. It implies that $\Pr_{\mathcal{M}}(\neg\Gamma) > 0$. As this ensures that all necessary conditional probabilities are well-defined, PAC1 is equivalent to the probability-raising condition $\Pr_{\mathcal{M}}(\mathcal{L} \mid \Gamma) > \Pr_{\mathcal{M}}(\mathcal{L} \mid \neg\Gamma)$. Axiom PAC2 captures that the effect must not occur before the cause.

The requirement on the cause in the provided definition is of *global* nature: the cause Γ as a whole has to guarantee the probability-raising property with respect to the effect. Single elements $\gamma \in \Gamma$, however, do not necessarily guarantee that the probability of the effect has been raised. Furthermore, under mild assumptions, the definition subsumes the treatment of reachability properties above:

▶ **Proposition 3.** *Let $\mathcal{M}$ be a DTMC with state space S and let $C, E \subseteq S$ be disjoint. Assume that no state in $s \in S \setminus E$ satisfies $\Pr_{\mathcal{M},s}(\Diamond E) = 1$. Then the following are equivalent:*
1. *C is a reachability-cause for E.*
2. *The set Γ of finite paths in $(S \setminus (C \cup E))^* C$ is a global PR-cause for the set $\mathcal{L}$ of paths satisfying $\Diamond E$.*

In the proposition above, $\Pr_{\mathcal{M},s}$ denotes the probability measure induced by $\mathcal{M}$ with assuming s as the unique initial state. A related notion of causality based on probability-raising in DTMCs has been introduced by Kleinberg et al. in a series of papers [75, 76, 74, 66, 122]. Here, probabilistic CTL is used to describe the cause C and the effect E via state formulas. We can describe both events also directly as sets of states in the DTMC by considering exactly those states that fulfil the corresponding probabilistic CTL formula. For reachability

properties, the set C is then said to be a cause of E if $\Pr_{\mathcal{M},c}(\Diamond E) > \Pr_{\mathcal{M}}(\Diamond E)$ for all $c \in C$. So, the requirement for this notion of causality is *local*: reaching any state $c \in C$ has to ensure that the probability of reaching E afterwards is raised. In case $C = \{c\}$ is a singleton disjoint from E, this notion agrees with Definition 1.

Adapting PAC1 to sets of paths and including the temporal priority requirement that the effect does not occur before the cause (PAC2 as before), we obtain the following definition of causality:

▶ **Definition 4** (local PR-cause). *Let $\mathcal{M}$ be a DTMC with state space S, $\Gamma \subseteq S^+$ a set of finite paths, and let $\mathcal{L} \subseteq S^\omega$ be a measurable set of paths. Then Γ is a* local probability-raising cause *(local PR-cause) for $\mathcal{L}$ if*
PAC1$^{\text{loc}}$: *for all $\gamma \in \Gamma$ we have $\Pr_{\mathcal{M}}(\mathcal{L} \mid \gamma) > \Pr_{\mathcal{M}}(\mathcal{L})$, and*
PAC2: *for all $\gamma \in \Gamma$ no proper prefix γ' of γ satisfies $\Pr_{\mathcal{M}}(\mathcal{L} \mid \gamma') = 1$.*

Axiom PAC1$^{\text{loc}}$ can be seen as the local version of PAC1. Clearly, PAC1$^{\text{loc}}$ implies PAC1. Furthermore, PAC1$^{\text{loc}}$ implies that $\Pr_{\mathcal{M}}(\neg\gamma) > 0$ for all $\gamma \in \Gamma$. Hence, we could equivalently reformulate PAC1$^{\text{loc}}$ as $\Pr_{\mathcal{M}}(\mathcal{L} \mid \gamma) > \Pr_{\mathcal{M}}(\mathcal{L} \mid \neg\gamma)$ for all $\gamma \in \Gamma$.

The work by Kleinberg et al. proceeds relative to an explicit probability value $p > \Pr_{\mathcal{M}}(\mathcal{L})$ such that $\Pr_{\mathcal{M}}(\mathcal{L} \mid \gamma) \geqslant p$ for all γ in a cause Γ. The higher this value p lies above $\Pr_{\mathcal{M}}(\mathcal{L})$, the greater is the amount by which all elements of Γ are guaranteed to raise the probability of the effect $\mathcal{L}$.

Such a reference to a specific threshold value p has also been incorporated into a notion of *p-causes* in [9]. Motivated by monitoring applications (see, e.g., [87]), the underlying idea is that notions of causality could be used to foresee undesirable behavior. If a cause for an erroneous execution is observed, countermeasures can be taken before the error actually occurs. Here it is particularly useful to specify a sensitivity p that expresses how likely an error is after observing the cause. In addition, the occurrence of an erroneous execution should not stay undetected. Therefore, an additional condition is imposed on p-causes: almost all executions that exhibit the error should have a prefix in the cause. Together with the temporal priority of the cause (PAC2) as before, these requirements lead to the following definition:

▶ **Definition 5** (p-cause). *Let $\mathcal{M}$ be a DTMC with state space S and $p \in (0, 1]$. A non-empty set $\Gamma \subseteq S^+$ is a* p-cause *for a measurable set $\mathcal{L} \subseteq S^\omega$ if*
PAC1$^{\text{p}}$: *for all $\gamma \in \Gamma$ we have $\Pr_{\mathcal{M},s_0}(\mathcal{L} \mid \gamma) \geqslant p$,*
PAC2: *for all $\gamma \in \Gamma$ no proper prefix γ' of γ satisfies $\Pr_{\mathcal{M}}(\mathcal{L} \mid \gamma') = 1$, and*
PAC3: $\Pr_{\mathcal{M}}(\Gamma \mid \mathcal{L}) = 1$.

Besides the practicality in monitoring applications, condition PAC3 also adds the counterfactual idea to the definition. Since almost all executions in $\mathcal{L}$ have a prefix in Γ, the effect occurs with probability 0 if the cause was not observed. Condition PAC1$^{\text{p}}$ is a third variant of the probability-raising requirement that compares the probability of the effect after observing an element of the cause to a specific threshold value p rather than the overall probability $\Pr_{\mathcal{M}}(\mathcal{L})$. In case $p > \Pr_{\mathcal{M}}(\mathcal{L})$, this variant implies PAC1 and PAC1$^{\text{loc}}$.

For ω-regular $\mathcal{L}$ there always exist p-causes for any $p \in (0, 1]$. The reason is that then almost all paths in $\mathcal{L}$ have a prefix π with $\Pr_{\mathcal{M}}(\mathcal{L} \mid \pi) = 1$. Choosing the shortest of such prefixes in accordance with condition PAC2 yields a 1-cause and hence a p-cause for any p. For $p < 1$, there are multiple p-causes in general. In [9], the problem to find cost-optimal p-causes with respect to a variety of cost measures is addressed.

The relationship between the different notions of causes is summarized in the following proposition. It is a direct consequence of the implications between the axioms used in the definitions that have been discussed so far.

▶ **Proposition 6.** *Let $\mathcal{M}$ be a DTMC with state space S, $\Gamma \subseteq S^+$, and let $\mathcal{L} \subseteq S^\omega$ be a measurable set of paths. Then the following statements hold:*

1. *If Γ is a p-cause for $\mathcal{L}$ for some $p > \Pr_{\mathcal{M}}(\mathcal{L})$, then Γ is also a local PR-cause for $\mathcal{L}$.*
2. *If Γ is a local PR-cause for $\mathcal{L}$, then Γ is also a global PR-cause for $\mathcal{L}$.*
3. *If Γ is a singleton, then Γ is a local PR-cause for $\mathcal{L}$ iff Γ is a global PR-cause for $\mathcal{L}$.*

The probabilistic notions of causality discussed in this section naturally constitute forward-looking notions: the probability-raising principle inherently addresses the behavior of a system across multiple executions, and causes are prone to exhibiting a predictive character. These notions can be useful in inferring causal dependencies in data series [75, 74, 66] and predicting undesirable behavior of reactive systems through runtime monitoring [9]. Nevertheless, as far as formal probabilistic models are concerned, a comprehensive study of cause-effect relationships is still at the beginning.

5 Concluding Remarks

This article gave an overview of recent trends in causality-based reasoning in the verification context. The focus of this article was on concepts that aim to explicate *why* a system exhibits a specific observable behavior and to which degree individual agents of a system can be held *responsible* for it. For non-probabilistic formal models, concepts of causation have been introduced in multiple facets and examined for manifold applications. To increase the power of causal inferences, a more systematic study relating forward and backward notions of causality would be highly beneficial.

Compared to the non-probabilistic setting, research on probabilistic causation in stochastic operational models is still in its infancy. While the techniques presented here are limited to purely probabilistic models (Markov chains), an examination of causality in probabilistic models with nondeterminism (Markov decision processes) is largely open. A first step in this direction is a formalization of action causes as a hyperproperty in Markov decision processes [37]. Another important future direction is to reason about cause-effect relationships in hidden Markovian models where states (and events) are not fully observable.

Another research strand not covered in this article are causality-based verification techniques (see, e.g., [82, 83]) that rely on the successive identification of cause-effect relationships between events to generate a causality-based proof for the satisfaction or violation of a system property. Along these lines, the work [8] presents a causality-based technique for solving symbolically expressed, infinite-state two-player reachability games. Applying this paradigm also in a probabilistic setting is a promising direction of study.

References

1 Kjersti Aas, Martin Jullum, and Anders Løland. Explaining individual predictions when features are dependent: More accurate approximations to Shapley values, 2020. `arXiv:1903.10464`.

2 Erika Ábrahám and Borzoo Bonakdarpour. HyperPCTL: A temporal logic for probabilistic hyperproperties. In *Proc. of the 15th Intern. Conf. on Quantitative Evaluation of Systems (QEST)*, pages 20–35. Springer, 2018.

3 R. Alur, C. Courcoubetis, N. Halbwachs, T.A. Henzinger, P.-H. Ho, X. Nicollin, A. Olivero, J. Sifakis, and S. Yovine. The algorithmic analysis of hybrid systems. *Theoretical Computer Science*, 138(1):3–34, 1995. Hybrid Systems. `doi:10.1016/0304-3975(94)00202-T`.

4 Rajeev Alur and David L. Dill. A Theory of Timed Automata. *Theoretical Computer Science*, 126(2):183–235, 1994. doi:10.1016/0304-3975(94)90010-8.

5 Sven Apel and Christian Kästner. An Overview of Feature-Oriented Software Development. *Journal of Object Technology*, 8:49–84, 2009.

6 Uwe Aßmann, Christel Baier, Clemens Dubslaff, Dominik Grzelak, Simon Hanisch, Ardhi P. P. Hartono, Stefan Köpsell, Tianfang Lin, and Thorsten Strufe. *Tactile computing: Essential building blocks for the Tactile Internet*, chapter 13, pages 301–326. Academic Press, 2021.

7 R. J. Aumann and L. S. Shapley. *Values of Non-Atomic Games*. Princeton University Press, 1974. URL: http://www.jstor.org/stable/j.ctt13x149m.

8 Christel Baier, Norine Coenen, Bernd Finkbeiner, Florian Funke, Simon Jantsch, and Julian Siber. Causality-based Game Solving. In *Proc. of the 33rd Intern. Conf. on Computer Aided Verification (CAV)* (to appear), 2021.

9 Christel Baier, Florian Funke, Simon Jantsch, Jakob Piribauer, and Robin Ziemek. Probabilistic causes in Markov chains, 2021. arXiv:2104.13604.

10 Christel Baier, Florian Funke, and Rupak Majumdar. A Game-Theoretic Account of Responsibility Allocation. In *Proc. of the 30th International Joint Conference on Artificial Intelligence (IJCAI)* (to appear), 2021.

11 Christel Baier, Florian Funke, and Rupak Majumdar. Responsibility Attribution in Parameterized Markovian Models. In *Proc. of the 35th AAAI Conference on Artificial Intelligence (AAAI)* (to appear), 2021.

12 Christel Baier and Joost-Pieter Katoen. *Principles of Model Checking*. MIT Press, 2008.

13 Thomas Ball, Mayur Naik, and Sriram K. Rajamani. From symptom to cause: Localizing errors in counterexample traces. *SIGPLAN Not.*, 38(1):97–105, 2003. doi:10.1145/640128.604140.

14 Sander Beckers and Joost Vennekens. A General Framework for Defining and Extending Actual Causation Using CP-Logic. *Int. J. Approx. Reasoning*, 77(C):105–126, October 2016. doi:10.1016/j.ijar.2016.05.008.

15 Adrian Beer, Stephan Heidinger, Uwe Kühne, Florian Leitner-Fischer, and Stefan Leue. Symbolic Causality Checking Using Bounded Model Checking. In *Model Checking Software*, pages 203–221. Springer, 2015.

16 Ilan Beer, Shoham Ben-David, Hana Chockler, Avigail Orni, and Richard Trefler. Explaining counterexamples using causality. In *Proc. of the 21st Intern. Conf. on Computer Aided Verification (CAV)*, pages 94–108. Springer, 2009. doi:10.1007/978-3-642-02658-4_11.

17 Ilan Beer, Shoham Ben-David, Cindy Eisner, and Yoav Rodeh. Efficient Detection of Vacuity in ACTL Formulas. In *Proc. of the 9th Intern. Conf. on Computer Aided Verification (CAV)*, pages 279–290, 1997. doi:10.1007/3-540-63166-6_28.

18 Matthew Braham and Martin van Hees. An Anatomy of Moral Responsibility. *Mind*, 121 (483):601–634, 2012.

19 Jan Broersen. Deontic epistemic stit logic distinguishing modes of mens rea. *Journal of Applied Logic*, 9(2):137–152, 2011. doi:10.1016/j.jal.2010.06.002.

20 Georgiana Caltais, Sophie Linnea Guetlein, and Stefan Leue. Causality for General LTL-definable Properties. In *Proc. of the 3rd Workshop on formal reasoning about Causation, Responsibility, and Explanations in Science and Technology (CREST)*, pages 1–15, 2018. doi:10.4204/EPTCS.286.1.

21 Nancy Cartwright. Causal Laws and Effective Strategies. *Noûs*, 13(4):419–437, 1979. doi:10.1093/0198247044.003.0002.

22 Krishnendu Chatterjee and Thomas A. Henzinger. A Survey of Stochastic ω-Regular Games. *J. Comput. Syst. Sci.*, 78(2):394–413, 2012. doi:10.1016/j.jcss.2011.05.002.

23 Hana Chockler and Joseph Y. Halpern. Responsibility and Blame: A Structural-Model Approach. *J. Artif. Int. Res.*, 22(1):93–115, October 2004.

24 Hana Chockler, Joseph Y. Halpern, and Orna Kupferman. What causes a system to satisfy a specification? *ACM Transactions on Computational Logic*, 9(3):20:1–20:26, 2008. doi:10.1145/1352582.1352588.

25 Hana Chockler, Orna Kupferman, Robert P. Kurshan, and Moshe Y. Vardi. A Practical Approach to Coverage in Model Checking. In *Proc. of the 13th Intern. Conf. on Computer Aided Verification (CAV)*, pages 66–78, 2001. doi:10.1007/3-540-44585-4_7.

26 Hana Chockler, Orna Kupferman, and Moshe Vardi. Coverage Metrics for Formal Verification. *Intern. Journal on Software Tools for Technology Transfer (STTT)*, 8(4–5):373–386, 2006.

27 Hana Chockler, Orna Kupferman, and Moshe Y. Vardi. Coverage Metrics for Temporal Logic Model Checking. In *Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, pages 528–542, 2001. doi:10.1007/3-540-45319-9_36.

28 Philipp Chrszon, Clemens Dubslaff, Sascha Klüppelholz, and Christel Baier. ProFeat: feature-oriented engineering for family-based probabilistic model checking. *Formal Aspects of Computing*, 30(1):45–75, 2018.

29 Edmund M. Clarke, E. Allen Emerson, and A. Prasad Sistla. Automatic verification of finite-state concurrent systems using temporal logic specifications. *ACM Trans. Program. Lang. Syst.*, 8:244–263, 1986.

30 Edmund M. Clarke, Orna Grumberg, Kenneth L. McMillan, and Xudong Zhao. Efficient Generation of Counterexamples and Witnesses in Symbolic Model Checking. In *Proc. of the 32nd Annual ACM/IEEE Design Automation Conf. (DAC)*, pages 427–432, New York, NY, USA, 1995. ACM. doi:10.1145/217474.217565.

31 Edmund M. Clarke, Thomas A. Henzinger, Helmut Veith, and Roderick Bloem. *Handbook of Model Checking*. Springer Publishing Company, Incorporated, 1st edition, 2018.

32 Andreas Classen, Patrick Heymans, Pierre-Yves Schobbens, Axel Legay, and Jean-François Raskin. Model Checking Lots of Systems: Efficient Verification of Temporal Properties in Software Product Lines. In *Proc. of the 32nd Intern. Conf. on Software Engineering (ICSE)*, pages 335–344. ACM, 2010.

33 Norine Coenen. Causality and Hyperproperties. In Gregor Gössler, Stefan Leue, and Shin Nakajima, editors, *Causal Reasoning in Systems (NII Shonan Meeting 139)*, 2019. URL: https://shonan.nii.ac.jp/seminars/139/.

34 Anupam Datta, Deepak Garg, Dilsun Kaynar, Divya Sharma, and Arunesh Sinha. Program Actions as Actual Causes: A Building Block for Accountability. In *Proc. of the 28th IEEE Computer Security Foundations Symp. (CSF)*, pages 261–275, 2015. doi:10.1109/CSF.2015.25.

35 Anupam Datta, Shayak Sen, and Yair Zick. Algorithmic Transparency via Quantitative Input Influence: Theory and Experiments with Learning Systems. In *Proc. of the 37th IEEE Symp. on Security and Privacy (SP)*, pages 598–617, 2016. doi:10.1109/SP.2016.42.

36 David L. Dill. Timing Assumptions and Verification of Finite-State Concurrent Systems. In *Automatic Verification Methods for Finite State Systems*, LNCS. Springer, 1990. doi:10.1007/3-540-52148-8_17.

37 Rayna Dimitrova, Bernd Finkbeiner, and Hazem Torfah. Probabilistic Hyperproperties of Markov Decision Processes. In *Proc. of the 18th Intern. Symp. on Automated Technology for Verification and Analysis (ATVA)*, volume 12302 of *LNCS*, pages 484–500. Springer, 2020.

38 Clemens Dubslaff, Christel Baier, and Sascha Klüppelholz. Probabilistic model checking for feature-oriented systems. *Trans. Aspect-Oriented Software Development*, 12:180–220, 2015.

39 Ellery Eells. *Probabilistic Causality*. Cambridge Studies in Probability, Induction and Decision Theory. Cambridge University Press, 1991.

40 Thomas Eiter and Thomas Lukasiewicz. Complexity results for explanations in the structural-model approach. *Artificial Intelligence*, 154(1-2):145–198, 2004. doi:10.1016/j.artint.2003.06.002.

41 Joel Feinberg. *Doing & Deserving: Essays in the Theory of Responsibility*. Princeton University Press, Princeton, USA, 1970.

42 Luke Fenton-Glynn. A Proposed Probabilistic Extension of the Halpern and Pearl Definition of 'Actual Cause'. *British Journal for the Philosophy of Science*, 68(4):1061–1124, 2017. doi:10.1093/bjps/axv056.

43 Eric Friedman and Hervé Moulin. Three methods to share joint costs or surplus. *Journal of Economic Theory*, 87(2):275–312, 1999.

44 Florian Funke, Simon Jantsch, and Christel Baier. Farkas Certificates and Minimal Witnesses for Probabilistic Reachability Constraints. In *Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*. Springer, 2020. doi:10.1007/978-3-030-45190-5_18.

45 David Galles and Judea Pearl. Axioms of Causal Relevance. *Artif. Intell.*, 97(1–2):9–43, December 1997. doi:10.1016/S0004-3702(97)00047-7.

46 David Galles and Judea Pearl. An Axiomatic Characterization of Causal Counterfactuals. *Foundations of Science*, 3:151–182, 1998. doi:10.1023/A:1009602825894.

47 Edmund Gettier. Is justified true belief knowledge? *Analysis*, 23:121–123, 1963.

48 Robert Givan, Sonia Leach, and Thomas Dean. Bounded-parameter Markov decision processes. *Artificial Intelligence*, 122(1):71–109, 2000. doi:10.1016/S0004-3702(00)00047-3.

49 Gregor Gössler and Daniel Le Métayer. A General Trace-Based Framework of Logical Causality. In *Formal Aspects of Component Software*, pages 157–173. Springer, 2014.

50 Gregor Gössler and Daniel Le Métayer. A General Framework for Blaming in Component-Based Systems. *Science of Computer Programming*, 113:223–235, 2015. doi:10.1016/j.scico.2015.06.010.

51 Gregor Gössler and Jean-Bernard Stefani. Causality Analysis and Fault Ascription in Component-Based Systems. *Theoretical Computer Science*, 837:158–180, 2020. doi:10.1016/j.tcs.2020.06.010.

52 Stephen R. Grimm. Understanding as Knowledge of Causes. In Abrol Fairweather, editor, *Virtue Epistemology Naturalized: Bridges Between Virtue Epistemology and Philosophy of Science*, pages 329–345. Springer, 2014. doi:10.1007/978-3-319-04672-3_19.

53 Alex Groce. Error Explanation with Distance Metrics. In *Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, pages 108–122, Berlin, Heidelberg, 2004. Springer Berlin Heidelberg.

54 Ned Hall. Structural Equations and Causation. *Philosophical Studies*, 132(1):109–136, 2007. URL: http://www.jstor.org/stable/25471849.

55 Joseph Y. Halpern. Axiomatizing Causal Reasoning. *Journal of Artif. Intell. Research*, 12(1):317–337, 2000.

56 Joseph Y. Halpern. A Modification of the Halpern-Pearl Definition of Causality. In *Proc. of the 24th International Joint Conference on Artificial Intelligence (IJCAI)*, pages 3022–3033. AAAI Press, 2015.

57 Joseph Y. Halpern. *Actual Causality*. MIT Press, 2016.

58 Joseph Y. Halpern and Judea Pearl. Causes and Explanations: A Structural-Model Approach: Part i: Causes. In *Proc. of the 17th Conf. on Uncertainty in AI (UAI)*, pages 194–202. Morgan Kaufmann Publishers Inc., 2001.

59 Joseph Y. Halpern and Judea Pearl. Causes and Explanations: A Structural-Model Approach. Part I: Causes. *The British Journal for the Philosophy of Science*, 56(4):843–887, 2005.

60 Joseph Y. Halpern and Judea Pearl. Causes and Explanations: A Structural-Model Approach. Part II: Explanations. *The British Journal for the Philosophy of Science*, 56(4):889–911, 2005.

61 Mark Harman and Robert Hierons. An overview of program slicing. *Software Focus*, 2(3):85–92, 2001. doi:10.1002/swf.41.

62 Thomas A. Henzinger, Ranjit Jhala, Rupak Majumdar, George C. Necula, Grégoire Sutre, and Westley Weimer. Temporal-Safety Proofs for Systems Code. In *Proc. of the 14th Intern. Conf. on Computer Aided Verification (CAV)*, volume 2404 of *LNCS*, pages 526–538. Springer, 2002. doi:10.1007/3-540-45657-0_45.

63 Christopher Hitchcock. The Intransitivity of Causation Revealed in Equations and Graphs. *The Journal of Philosophy*, 98(6):273–299, 2001. URL: http://www.jstor.org/stable/2678432.

64 Martin Hofmann, Christian Neukirchen, and Harald Rueß. Certification for μ-Calculus with Winning Strategies. In *Proc. of the 23rd Intern. Symp. on Model Checking Software (SPIN)*, volume 9641 of *LNCS*, pages 111–128. Springer, 2016. doi:10.1007/978-3-319-32582-8_8.

65 Yatin Hoskote, Timothy Kam, Pei-Hsin Ho, and Xudong Zhao. Coverage Estimation for Symbolic Model Checking. In *Proc. of the 36th Annual ACM/IEEE Design Automation Conf. (DAC)*, pages 300–305, 1999. doi:10.1145/309847.309936.

66 Yuxiao Huang and Samantha Kleinberg. Fast and Accurate Causal Inference from Time Series Data. In *Proc. of the 28th Intern. Florida AI Research Society Conf. (FLAIRS)*, pages 49–54. AAAI Press, 2015. URL: http://www.aaai.org/ocs/index.php/FLAIRS/FLAIRS15/paper/view/10434.

67 David Hume. *A Treatise of Human Nature*. John Noon, 1739.

68 David Hume. *An Enquiry Concerning Human Understanding*. London, 1748.

69 Amjad Ibrahim and Alexander Pretschner. From checking to inference: Actual causality computations as optimization problems. In *Proc. of the 18th Intern. Symp. on Automated Technology for Verification and Analysis (ATVA)*, pages 343–359. Springer, 2020. doi:10.1007/978-3-030-59152-6_19.

70 Simon Jantsch, Florian Funke, and Christel Baier. Minimal Witnesses for Probabilistic Timed Automata. In *Proc. of the 18th Intern. Symp. on Automated Technology for Verification and Analysis (ATVA)*, pages 501–517. Springer, 2020. doi:10.1007/978-3-030-59152-6_28.

71 Bengt Jonsson and Kim G. Larsen. Specification and refinement of probabilistic processes. In *Proc. of the 6th Annual IEEE Symp. on Logic in Computer Science (LICS)*, pages 266–277, 1991.

72 Manu Jose and Rupak Majumdar. Cause clue clauses: error localization using maximum satisfiability. In *Proc. of the 32nd ACM SIGPLAN Conf. on Programming Language Design and Implementation (PLDI)*, pages 437–446. ACM, 2011. doi:10.1145/1993498.1993550.

73 Severin Kacianka, Amjad Ibrahim, and Alexander Pretschner. Expressing Accountability Patterns using Structural Causal Models, 2020. arXiv:2005.03294.

74 Samantha Kleinberg. A Logic for Causal Inference in Time Series with Discrete and Continuous Variables. In *Proc. of the 22nd International Joint Conference on Artificial Intelligence (IJCAI)*, pages 943–950, 2011. doi:10.5591/978-1-57735-516-8/IJCAI11-163.

75 Samantha Kleinberg and Bud Mishra. The Temporal Logic of Causal Structures. In *Proc. of the 25th Conf. on Uncertainty in AI (UAI)*, pages 303–312, 2009.

76 Samantha Kleinberg and Bud Mishra. The Temporal Logic of Token Causes. In *Proc. of the 12th Intern. Conf. on Principles of Knowledge Representation and Reasoning (KR)*, 2010.

77 Martin Kölbl and Stefan Leue. An Efficient Algorithm for Computing Causal Trace Sets in Causality Checking. In *Proc. of the 17th Intern. Symp. on Automated Technology for Verification and Analysis (ATVA)*, pages 171–186. Springer, 2019. doi:10.1007/978-3-030-31784-3_10.

78 Martin Kölbl, Stefan Leue, and Robert Schmid. Dynamic Causes for the Violation of Timed Reachability Properties. In *Proc. of the 18th Intern. Conf. on Formal Modeling and Analysis of Timed Systems (FORMATS)*, pages 127–143. Springer, 2020. doi:10.1007/978-3-030-57628-8_8.

79 Igor Kozine and Lev Utkin. Interval-Valued Finite Markov Chains. *Reliable Computing*, 8:97–113, April 2002. doi:10.1023/A:1014745904458.

80 Harold W. Kuhn. *Extensive Games and the Problem of Information*, pages 193–216. Princeton University Press, Princeton, 1953. doi:10.1515/9781400881970-012.

81 Orna Kupferman and Moshe Y. Vardi. Vacuity Detection in Temporal Model Checking. In *Proc. of the 10th IFIP Working Conf. on Correct Hardware Design and Verification Methods (CHARME)*, pages 82–96, 1999.

82 Andrey Kupriyanov and Bernd Finkbeiner. Causality-Based Verification of Multi-threaded Programs. In *Proc. of the 24th Intern. Conf. on Concurrency Theory (CONCUR)*, LNCS, pages 257–272. Springer, 2013. doi:10.1007/978-3-642-40184-8_19.

83 Andrey Kupriyanov and Bernd Finkbeiner. Causal Termination of Multi-threaded Programs. In *Proc. of the 26th Intern. Conf. on Computer Aided Verification (CAV)*, LNCS, pages 814–830. Springer, 2014. doi:10.1007/978-3-319-08867-9_54.

84 Igal Kvart. Causation: Probabilistic and Counterfactual Analyses. In *Causation and Counterfactuals*, pages 359–387. MIT Press, Cambridge, MA, USA, 2004.

85 Kim G. Larsen and Bent Thomsen. A Modal Process Logic. In *Proc. of the 3rd Annual Symp. on Logic in Computer Science (LICS)*, pages 203–210, 1988. doi:10.1109/LICS.1988.5119.

86 Florian Leitner-Fischer and Stefan Leue. Causality Checking for Complex System Models. In *Proc. of the 14th Intern. Conf. on Verification, Model Checking, and Abstract Interpretation (VMCAI)*, pages 248–267, 2013. doi:10.1007/978-3-642-35873-9_16.

87 Martin Leucker and Christian Schallhart. A brief account of runtime verification. *The Journal of Logic and Algebraic Programming*, 78(5):293–303, 2009. doi:10.1016/j.jlap.2008.08.004.

88 David Lewis. Causation. *Journal of Philosophy*, 70(17):556–567, 1973. doi:10.2307/2025310.

89 David Lewis. Postscripts to 'Causation'. In *Philosophical Papers Vol. II*. Oxford University Press, 1986.

90 Harry R. Lewis. A logic of concrete time intervals. In *Proc. of the 5th Annual IEEE Symp. on Logic in Computer Science (LICS)*, pages 380–389, 1990. doi:10.1109/LICS.1990.113763.

91 Scott M. Lundberg and Su-In Lee. A Unified Approach to Interpreting Model Predictions. In *Proc. of the 31st Intern. Conf. on Neural Information Processing Systems (NeurIPS)*, pages 4768–4777, Red Hook, NY, USA, 2017. Curran Associates Inc.

92 Donald A. Martin. The Determinacy of Blackwell Games. *The Journal of Symbolic Logic*, 63(4):1565–1581, 1998. URL: http://www.jstor.org/stable/2586667.

93 Corto Mascle, Christel Baier, Florian Funke, Simon Jantsch, and Stefan Kiefer. Responsibility and verification: Importance value in temporal logics. In *Proc. of the 36th Annual Symp. on Logic in Computer Science (LICS)* (to appear), 2021.

94 Pavel Naumov and Jia Tao. An epistemic logic of blameworthiness. *Artificial Intelligence*, 283:103269, 2020. doi:10.1016/j.artint.2020.103269.

95 Judea Pearl. Causal diagrams for empirical research. *Biometrika*, 82(4):669–688, 1995. URL: http://www.jstor.org/stable/2337329.

96 Judea Pearl. On the Definition of Actual Cause. *Technical Report R-259, Computer Science Dept., UCLA*, 1998. doi:10.1.1.53.9540.

97 Judea Pearl. *Causality*. Cambridge University Press, 2 edition, 2009. doi:10.1017/CBO9780511803161.

98 Jonas Peters, Dominik Janzing, and Bernhard Schölkopf. *Elements of Causal Inference: Foundations and Learning Algorithms*. MIT Press, Cambridge, MA, USA, 2017.

99 Amir Pnueli. The Temporal Logic of Programs. In *Proc. of the 18th Annual Symp. on Foundations of Computer Science (FOCS)*, pages 46–57, 1977. doi:10.1109/SFCS.1977.32.

100 Duncan Pritchard. Knowledge and understanding. In *Virtue Epistemology Naturalized: Bridges between Virtue Epistemology and Philosophy of Science*, Synthese Library 366, pages 315–327. Springer, 2014.

101 Duncan Pritchard. *Epistemology*. Palgrave Macmillan, 2016.

102 Mitra Purandare and Fabio Somenzi. Vacuum cleaning CTL formulae. In *Proc. of the 14th Intern. Conf. on Computer Aided Verification (CAV)*, pages 485–499, 2002. doi:10.1007/3-540-45657-0_39.

103 Hans Reichenbach. Die Kausalstruktur der Welt und der Unterschied von Vergangenheit und Zukunft. *Sitzungsberichte der Bayerische Akademie der Wissenschaft*, 2:81–119, 1925.

104 Hans Reichenbach. *The Direction of Time*. Berkeley and Los Angeles: University of California Press, 1956.

105 Manos Renieres and Steven P. Reiss. Fault localization with nearest neighbor queries. In *Proc. of the 18th IEEE Intern. Conf. on Automated Software Engineering (ASE)*, pages 30–39, 2003. doi:10.1109/ASE.2003.1240292.

106 Lloyd S. Shapley. A value for n-person games. In *Contributions to the Theory of Games. Vol. II*, pages 307–317. Princeton University Press, 1953.

107 Brian Skyrms. Causal Necessity. *Philosophy of Science*, 48(2):329–335, 1981. doi:10.1086/289003.

108 Jeremy Sproston. Decidable Model Checking of Probabilistic Hybrid Automata. In Mathai Joseph, editor, *Formal Techniques in Real-Time and Fault-Tolerant Systems*, pages 31–45, Berlin, Heidelberg, 2000. Springer Berlin Heidelberg.

109 Yi Sun and Mukund Sundararajan. Axiomatic attribution for multilinear functions. *Proc. of the 12th ACM Conf. on Electronic Commerce (EC)*, 2011. doi:10.1145/1993574.1993601.

110 Mukund Sundararajan and Amir Najmi. The Many Shapley Values for Model Explanation. In *Proc. of the 37th Intern. Conf. on Machine Learning (ICML)*, volume 119 of *Machine Learning Research*, pages 9269–9278. PMLR, 2020.

111 Patrick Suppes. *A Probabilistic Theory of Causality*. Amsterdam: North-Holland Pub. Co., 1970.

112 Thomas Thüm, Sven Apel, Christian Kästner, Ina Schaefer, and Gunter Saake. A Classification and Survey of Analysis Strategies for Software Product Lines. *ACM Comput. Surv.*, 47(1s):6:1–6:45, 2014.

113 Ibo van de Poel. The Relation Between Forward-Looking and Backward-Looking Responsibility. In *Moral Responsibility: Beyond Free Will and Determinism*, pages 37–52, Dordrecht, 2011. Springer. doi:10.1007/978-94-007-1878-4_3.

114 Moshe Y. Vardi. Automatic verification of probabilistic concurrent finite-state programs. In *Proc. of the 26th IEEE Symp. on Foundations of Computer Science (FOCS)*, pages 327–338. IEEE Computer Society, 1985.

115 Joost Vennekens, Marc Denecker, and Maurice Bruynooghe. CP-logic: A language of causal probabilistic events and its relation to logic programming. *Theory and Practice of Logic Programming*, 9(3):245–308, 2009. doi:10.1017/S1471068409003767.

116 John von Neumann. Zur Theorie der Gesellschaftsspiele. *Mathematische Annalen*, 100:295–320, 1928.

117 Erik Štrumbelj and Igor Kononenko. Explaining Prediction Models and Individual Predictions with Feature Contributions. *Knowl. Inf. Syst.*, 41(3):647–665, 2014. doi:10.1007/s10115-013-0679-x.

118 Chao Wang, Zijiang Yang, Franjo Ivancic, and Aarti Gupta. Whodunit? Causal Analysis for Counterexamples. In *Proc. of the 4th Intern. Symp. on Automated Technology for Verification and Analysis (ATVA)*, pages 82–95, 2006. doi:10.1007/11901914_9.

119 Vahid Yazdanpanah, Mehdi Dastani, Wojciech Jamroga, Natasha Alechina, and Brian Logan. Strategic Responsibility Under Imperfect Information. In *Proc. of the 18th Intern. Conf. on Autonomous Agents and MultiAgent Systems (AAMAS)*, pages 592–600. AAMAS Foundation, 2019.

120 Andreas Zeller. Isolating Cause-Effect Chains from Computer Programs. In *Proc. of the 10th ACM SIGSOFT Symp. on Foundations of Software Engineering (FSE)*, pages 1–10, New York, NY, USA, 2002. ACM. doi:10.1145/587051.587053.

121 Danfeng Zhang, Andrew C. Myers, Dimitrios Vytiniotis, and Simon L. Peyton Jones. SHErrLoc: A Static Holistic Error Locator. *ACM Trans. Program. Lang. Syst.*, 39(4):18:1–18:47, 2017. doi:10.1145/3121137.

122 Min Zheng and Samantha Kleinberg. A method for automating token causal explanation and discovery. In *Proc. of the 13th Florida AI Research Society Conf. (FLAIRS)*, 2017.

Symmetries and Complexity

Andrei A. Bulatov
School of Computing Science, Simon Fraser University, Burnaby, Canada

Abstract

The Constraint Satisfaction Problem (CSP) and a number of problems related to it have seen major advances during the past three decades. In many cases the leading driving force that made these advances possible has been the so-called algebraic approach that uses symmetries of constraint problems and tools from algebra to determine the complexity of problems and design solution algorithms. In this presentation we give a high level overview of the main ideas behind the algebraic approach illustrated by examples ranging from the regular CSP, to counting problems, to optimization and promise problems, to graph isomorphism.

2012 ACM Subject Classification Theory of computation → Complexity classes; Theory of computation → Design and analysis of algorithms

Keywords and phrases constraint problems, algebraic approach, dichotomy theorems

Digital Object Identifier 10.4230/LIPIcs.ICALP.2021.2

Category Invited Talk

Funding *Andrei A. Bulatov*: This research was supported by an NSERC Discovery grant.

As Jeavons et al. [39] discovered in the mid 90s, symmetries or lack thereof of combinatorial structures in many cases determine the complexity of the corresponding computational problems. This was the starting point of the so-called algebraic approach that has been initially developed for certain kinds of Constraint Satisfaction Problems (CSPs), and over the course of 20 years has been instrumental in resolving a number of long standing open problems, most important of which is the CSP Dichotomy Conjecture by Feder and Vardi [31]. These techniques also spread out and found applications in multiple areas somewhat related to the CSP. The types of research problems the algebraic approach has been most useful include complexity classifications and design of algorithms. It clearly does not apply to every single kind of CSP-related problems, but whenever the algebraic approach is possible, it has led to a significant progress in the area.

In this presentation we take a bird's-eye view on the main ideas behind the algebraic approach. We do not go into deep technical details, although we give links that can be used by an interested reader, but give a collection of simple examples showing how algebraic techniques can be used in various types of computational problems. In the first part of the presentation, Sections 1,2 we introduce several common types of constraint problems and lay out the basics of the algebraic approach. Then in Sections 3,4 we show how these ideas apply to the CSP Dichotomy Conjecture, and also briefly outline how the algebraic approach works for Counting, Promise, and Valued CSPs. We also mention a somewhat unexpected use of the method in Graph Isomorphism. Although there has been developed a rich and beautiful theory of CSPs on infinite domains [6, 7], we only focus here on finite domains. Also, we leave out many areas were the approach has been successfully used: Quantified CSPs [45], homomorphism lower bounds [41], robust approximation [3], proof complexity [1], global cardinality constraints [20, 21], Subalgebra [12] and Ideal Membership Problems [46, 22], solvability of equations [42], learnability [27], property testing [26], and many others.

1 Constraint Problems

1.1 The problem

There are multiple ways to define the CSP. For our purpose here we give two equivalent definitions [39, 43]. We use the terms *predicate* and *relation* interchangeably.

The first one, we refer to it as the *logic* definition, goes as follows. Fix a set A (always finite in this paper), a *domain*. The input of the CSP is a collection $\mathcal{I}$ of constraints, $R(x_1, \ldots, x_k)$, expressed as predicates over A, where the variables x_i are from some finite set X of variables. The goal is to decide whether a *solution* exists, that is, a mapping $\varphi : X \to A$ that satisfies all the constraints. Alternatively, the input can be thought of as the conjunction of all the constraints, and we want to know whether this formula is satisfiable.

The second definition of the CSP, which we will refer to as the *homomorphic* definition, is given in terms of relational structures. Recall that a relational *signature* is a collection of *relational symbols*, that is, names of relations we are going to use, each symbol R is assigned a natural number k_R, the *arity* of R. For instance, the relational signature of a graph or digraph is $\{E\}$, only one relational symbol, whose arity is 2. For a signature σ a *relational structure* $\mathcal{A}$ with signature σ is a set A along with a predicate $R^{\mathcal{A}} \subseteq A^{k_R}$ for each $R \in \sigma$, called an *interpretation* of R. The set A is called the *base set* of $\mathcal{A}$. Structures with signature σ are often referred to as *σ-structures* and structures with the same signature are called *similar*. Let $\mathcal{A}, \mathcal{B}$ be two σ-structures with base sets A, B, respectively. A *homomorphism* from $\mathcal{A}$ to $\mathcal{B}$ is a mapping $\varphi : A \to B$ such that $R^{\mathcal{B}}(\varphi(a_1), \ldots, \varphi(a_{k_R})) = 1$ for every $R \in \sigma$ and any $a_1, \ldots, a_{k_R} \in A$ with $R^{\mathcal{A}}(a_1, \ldots, a_{k_R}) = 1$. If there is a homomorphism from $\mathcal{A}$ to $\mathcal{B}$, we write $\mathcal{A} \to \mathcal{B}$.

In the homomorphic version of the CSP the input is a pair of similar relational structures $\mathcal{A}, \mathcal{B}$ (always finite in this paper). The question is to decide whether there exists a homomorphism from $\mathcal{A}$ to $\mathcal{B}$. A translation between the two versions of the CSP is straightforward, it will be illustrated in Example 2 below.

▶ **Example 1.** In the 3-SAT problem the question is to decide the satisfiability of a 3-CNF. 3-SAT is readily a CSP in the logic form, it is a conjunction of clauses, each of which represents a ternary predicate on $\{0, 1\}$.

▶ **Example 2.** In the 3-Coloring problem we need to decide the existence of a proper 3-coloring of a given graph G. It can be stated as a CSP by treating the vertices of G as variables that need to be assigned one of the three colors, and edges of G as constraints requiring that if $uv \in E(G)$ then the values of u, v satisfy the predicate $\neq_3$ (u, v), which is the disequality relation on the set of colors.

Note that 3-Coloring can also be naturally represented in the homomorphic form: Any proper 3-coloring of G is a homomorphism from G to K_3. Using this approach one can generalize 3-Coloring to H-Coloring, where H is a fixed graph. The goal in this problem is to decide the existence of a homomorphism from a given graph G to H.

This transition between the two forms of the CSP can be extended to more general case: The structure $\mathcal{A}$ in the homomorphic form encodes the interaction between constraints, while the structure $\mathcal{B}$ encodes the constraints themselves.

▶ **Example 3.** A system of linear (as well as any other type of equations) naturally provides a conjunction of constraints, in which every constraint is represented by an equation.

▶ **Example 4.** The Clique problem, in which we are given a number k and a graph G, asks whether or not G contains a clique of size at least k. This problem can be reformulated as the question about the existence of a homomorphism from K_k to G, that is, a CSP in the homomorphic form.

▶ **Example 5.** The Perfect Matching problem asks whether a graph G has a perfect matching It is representable as a CSP in the logic form, although in a slightly more sophisticated way. The edges of G represent the variables of our CSP instance $\mathcal{I}(G)$ that take values 1 or 0 (in a matching or not), and the vertices correspond to constraints. For each $v \in V(G)$ of degree d we introduce a constraint of the form $R_{1\text{-in-}d}(x_1, \dots, x_d)$, where $x_1, \dots, x_d$ are the edges incident to v, which is true if and only if exactly one of $x_1, \dots, x_d$ equals 1 and the rest equal 0. As is easily seen, any solution of $\mathcal{I}(G)$ corresponds to a perfect matching in G and vice versa.

In order to capture specific computational problems the general CSP is often restricted in a certain way. It can be easily done for the homomorphic version of the CSP [43]. Let $\mathfrak{A}, \mathfrak{B}$ be classes of similar structures with signature σ. Then $\mathrm{CSP}(\mathfrak{A}, \mathfrak{B})$ denotes the class of CSP instances $\mathcal{A}, \mathcal{B}$, in which $\mathcal{A} \in \mathfrak{A}$ and $\mathcal{B} \in \mathfrak{B}$. If $\mathfrak{A}$ or $\mathfrak{B}$ is the class of all σ-structures, we use $\mathrm{CSP}(-, \mathfrak{B})$ and $\mathrm{CSP}(\mathfrak{A}, -)$, respectively. If one of the classes contains only one structure, we write $\mathrm{CSP}(-, \mathcal{B}), \mathrm{CSP}(\mathcal{A}, -)$ instead of $\mathrm{CSP}(-, \{\mathcal{B}\}), \mathrm{CSP}(\{\mathcal{A}\}, -)$. The general CSP is NP-complete as the examples above show, and assuming the Exponential Time Hypothesis it cannot be solved faster than $|\mathcal{B}|^{O(|\mathcal{A}|)}$ [33]. However, restricted problems may have much lower complexity, and this is the drive to understand the complexity of restricted problems that has been guiding the study of the CSP.

▶ **Example 6.** All the problems from Examples 1–5 can be viewed as $\mathrm{CSP}(\mathfrak{A}, \mathfrak{B})$ for appropriate classes $\mathfrak{A}, \mathfrak{B}$.

- 3-SAT is the problem $\mathrm{CSP}(-, \mathcal{B}_{3\text{-SAT}})$ where $\mathcal{B}_{3-SAT}$ is the relational structure with base set $\{0, 1\}$ and 8 predicates that are defined by the 8 possible 3-clauses.
- 3-Coloring is the problem $\mathrm{CSP}(-, K_3)$. More generally, the H-Coloring problem for a graph or digraph H can be represented as $\mathrm{CSP}(-, H)$.
- Representing Linear Equations requires a relational structure with an infinite signature: a predicate symbol for each possible linear equation. It is therefore a common practice to first observe that every system of linear equations is equivalent to one in which every equation contains at most 3 variables; it may require introducing new variables. Then in the case of a finite field $\mathbb{F}$ such a problem is the same as $\mathrm{CSP}(-, \mathcal{B}_{3\text{-Lin}})$, where $\mathcal{B}_{3\text{-Lin}}$ is the relational structure with the base set $\mathbb{F}$ whose predicates are given by all the possible linear equations over $\mathbb{F}$ containing at most 3 variables.
- The Clique problem is $\mathrm{CSP}(\mathfrak{K}, -)$, where $\mathfrak{K}$ is the class of all cliques.
- The Perfect Matching problem is a bit more difficult to represent. Let σ be an (infinite) signature that contains one symbol $R_{1\text{-in-}d}$ for every natural d. Then Perfect Matching is equivalent to $\mathrm{CSP}(\mathfrak{A}_2, \mathcal{B}_{1\text{-in}})$, where $\mathfrak{A}_2$ is the class of σ-structures $\mathcal{A}$ in which every element of the base set appears in exactly two tuples from the predicates of $\mathcal{A}$. Then $\mathcal{B}_{1\text{-in}}$ is the σ-structure with the base set $\{0, 1\}$ and whose predicates are interpreted as in Example 5. Note that the Perfect Matching problem can be expressed more naturally as a *holant* problem [24].

In this paper we focus on the problems of the form $\mathrm{CSP}(-, \mathcal{B})$, which are often referred to as *nonuniform* CSPs. We will shorten the notation to $\mathrm{CSP}(\mathcal{B})$. In this case it is often convenient to replace the structure $\mathcal{B}$ with a *constraint language*, the set of relations given by the predicates of $\mathcal{B}$. For any collection Γ of relations over a set A such a problem is denoted by $\mathrm{CSP}(\Gamma)$. The base set of $\mathcal{B}$ or the set on which Γ is defined on will be called the *domain* of $\mathrm{CSP}(\mathcal{B})$ or $\mathrm{CSP}(\Gamma)$.

1.2 Friends of the CSP

The CSP is not limited to just the decision problem from the previous section. It can also be modified to include other types of problems, some of which we will consider next. Most of the variations below were considered in [28] in the case of a 2-element domain.

Quantified CSP

The CSP in the logic form can also be thought of as checking the validity of an existentially quantified sentence $\exists x_1, \ldots, x_n(R_1 \wedge \cdots \wedge R_k)$. The problem that allows for an arbitrary quantifier prefix is known as the Quantified CSP or QCSP for short. Examples of QCSPs are the Quantified Satisfiability problem (QSAT) as well as a number of standard problems in PSPACE. For a structure $\mathcal{B}$ or a constraint language Γ, $\mathrm{QCSP}(\mathcal{B}), \mathrm{QCSP}(\Gamma)$ denote the Quantified CSPs restricted in the same way as the regular CSP [28, 45].

Counting CSPs

In the Counting CSP, or #CSP, the goal is to find the number of solutions of a CSP instance. In general, such problems belong to the class #P and many of them are #P-complete. Counting CSPs restricted by specifying a relational structure or a constraint language are denoted by $\#\mathrm{CSP}(\mathcal{B}), \#\mathrm{CSP}(\Gamma)$.

Optimization problems

There are several ways to convert a CSP into an optimization problem, see [28] for some examples. The most natural one is, given a CSP instance that may have no solution, find an assignment of variables that maximizes the number of satisfied constraints. This optimization problem is called the Max-CSP. Clearly, even when $\mathrm{CSP}(\Gamma)$ can be solved efficiently, the optimization problem may be hard. Linear Equations provides a well known example of such problem. More examples will be mentioned in Section 4.2 and can be found in [28]. The book [28] also presents the range of possible complexities of Max-CSPs on the domain $\{0, 1\}$.

Another way to optimize a CSP instance is to look for a solution that assigns a specific value to a maximal number of variables. For instance, Max-Ones is the version of SAT that seeks for an assignment in which as many propositional variables as possible are assigned 1.

Valued CSPs

The Valued CSP, or the VCSPs for short, is a generalization of optimization constraint problems such as Max-CSP and Max-Ones. In order to introduce them we need to replace predicates in the regular CSP with more general functions. Let A be a domain and $\mathbb{K}$ an ordered semiring, e.g. the ring of natural, integer, rational, or real numbers. Instead of predicates we now consider functions $R : A^k \to \mathbb{K}$ for some k. An instance of the Valued CSP consists of a set X of variables and a collection $\mathcal{C}$ of *function applications* of the form $R(x_1, \ldots, x_k)$, where each R is as above. For a mapping $\varphi : X \to A$ define its *weight* to be $w(\varphi) = \sum_{R(x_1,\ldots,x_k)\in\mathcal{C}} R(\varphi(x_1), \ldots, \varphi(x_k))$. The goal now is to find a mapping φ that yields the maximal (or minimal) weight possible. Similar to regular CSPs, VCSPs can also be parametrized by a constraint language, which in this case is a set of functions that are allowed in VCSP instances.

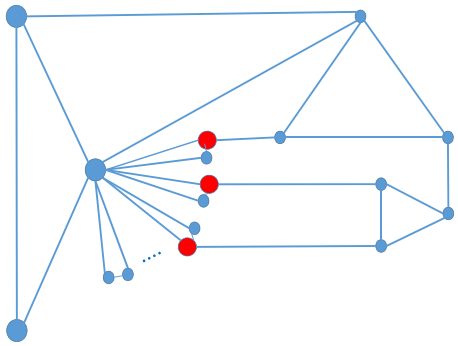


Figure 1 The gadget used to reduce 3-SAT to 3-Coloring.

▶ **Example 7** (Max-CSP as a VCSP)**.** Let Γ be a constraint language, that is, a set of predicates on some set A. We view the predicates from Γ as functions from Cartesian powers of A to $\mathbb{N}$. They take values $0, 1$. This way every instance of CSP(Γ) is treated as an instance of VCSP(Γ). As is easily seen, a mapping that maximizes the number of satisfied constraints in a CSP instance also has the maximal weight in the corresponding instance of the VCSP.

▶ **Example 8** (Max-Ones as a VCSP)**.** Transforming the Max-Ones problem into a VCSP is less straightforward, because we need to make sure that optimization is only happening over solutions of a Max-Ones instance. This is achieved by adding the *infinity* elements $-\infty, \infty$ to $\mathbb{N}$. In terms of order and arithmetic operations they relate to other elements of $\mathbb{N}$ in the natural way. Let Γ be a constraint language on $\{0,1\}$, and let $\Gamma' = \{R' \mid R \in \Gamma\} \cup \{O\}$, where $O : \{0,1\} \to \mathbb{N}$ with $O(0) = 0, O(1) = 1$, and $R'(a_1, \dots, a_k) = 1$ when $R(a_1, \dots, a_k) = 1$, $a_1, \dots, a_k \in \{0,1\}$, and $R'(a_1, \dots, a_k) = -\infty$ otherwise. Take an instance $\mathcal{I}$ of Max-Ones(Γ), replace every predicate $R(x_1, \dots, x_k)$ with a function application $R'(x_1, \dots, x_k)$, and for each variable $x \in X$ of $\mathcal{I}$ add the function application $O(x)$. Denote the resulting instance by $\mathcal{I}'$, it is an instance of VCSP(Γ'). The weight of a mapping $\varphi : X \to \{0,1\}$ in $\mathcal{I}'$ is not negative infinite if and only if φ is a solution of $\mathcal{I}$. Moreover, if w is a solution of $\mathcal{I}$ we have $w(\varphi) = m + \ell$, where m is the number of constraints in $\mathcal{I}$ that does not depend on φ, and ℓ is the number of variables that are assigned 1 by φ. Thus φ maximizes the number of ones if and only if it maximizes the weight in $\mathcal{I}'$.

2 Reductions and symmetries

In this section we look at the formalism that includes primitive-positive definitions and interpretations, and that captures one of the oldest tools in complexity theory, gadget reductions. We then introduce higher level symmetries of problems, polymorphisms, that set boundaries of what can be done using gadget reductions. Finally, we give some examples showing that polymorphisms can also be useful when designing solution algorithms. A more detailed and technical exposition can be found in [4].

2.1 Gadget reductions and primitive-positive definitions

A usual gadget reduction known from a basic course in complexity looks somewhat like what is shown in Fig. 1. They may be complicated and often difficult to come up with. Can we make the process of constructing such gadgets more orderly?

Let Γ be a set of relations (predicates) over a set A. A predicate R over A is said to be *primitive-positive (pp-) definable* in Γ if $R(\mathbf{x}) = \exists \mathbf{y}\ \Phi(\mathbf{x}, \mathbf{y})$, where Φ is a conjunction that involves predicates from Γ and equality relations. The formula above is then called a *pp-definition* of R in Γ. A constraint language Δ is pp-definable in Γ if so is every relation from Δ. In a similar way pp-definability can be introduced for relational structures.

▶ **Example 9.** Let $K_3 = (\{0, 1, 2\}, E)$ be a 3-element complete graph. Its edge relation is the binary disequality relation $\neq_3$ on $\{0, 1, 2\}$. Then the pp-formula

$$\begin{aligned} Q(x, y, z) \quad = \quad & \exists t, u, v, w (E(t, x) \wedge E(t, y) \wedge E(t, z) \wedge E(u, v) \wedge E(v, w) \\ & \wedge E(w, u) \wedge E(u, x) \wedge E(v, y) \wedge E(w, z)) \end{aligned}$$

defines the predicate Q that is true on all triples containing exactly 2 different elements from $\{0, 1, 2\}$.

A link between pp-definitions and reducibility of nonuniform CSPs was first observed in [39].

▶ **Theorem 10** ([39]). *Let Γ and Δ be constraint languages and Δ finite. If Δ is pp-definable in Γ then* $\mathrm{CSP}(\Delta)$ *is polynomial time reducible*[1] *to* $\mathrm{CSP}(\Gamma)$.

The gadget in Fig. 1 can be represented by a pp-formula, in which every variable corresponds to a vertex in the graph, the large red vertices are the free variables, and the edges are the constraints $\neq_3$. In general it seems plausible that pp-definitions and pp-interpretations discussed later capture what we think of "gadgets" and "gadget reductions".

2.2 Polymorphisms

While pp-definitions is a convenient and uniform way of representing gadgets, it is polymorphisms that are at the core of the algebraic approach.

Primitive positive definability can be concisely characterized using polymorphisms. An operation $f : A^k \to A$ is said to be a *polymorphism* of a relation $R \subseteq A^n$ if for any $\mathbf{a}_1, \ldots, \mathbf{a}_k \in R$, $\mathbf{a}_i = (a_{i1}, \ldots, a_{in})$, the tuple $f(\mathbf{a}_1, \ldots, \mathbf{a}_k)$ also belongs to R, where $f(\mathbf{a}_1, \ldots, \mathbf{a}_k)$ stands for $(f(a_{11}, \ldots, a_{k1}), \ldots, f(a_{1n}, \ldots, a_{kn}))$. The parameter k above is called the *arity* of f. Relation R is said to be *invariant* under f. Operation f is a polymorphism of a constraint language Γ if it is a polymorphism of every relation from Γ. Similarly, operation f is a polymorphism of a relational structure $\mathcal{B}$ if it is a polymorphism of every relation of $\mathcal{B}$. The set of all polymorphisms of language Γ or relational structure $\mathcal{B}$ is denoted by $\mathsf{Pol}(\Gamma)$, $\mathsf{Pol}(\mathcal{B})$, respectively.

▶ **Example 11.** Let R be an affine relation, that is, R is the solution space of a system of linear equations over a field $\mathbb{F}$. Then the operation $f(x, y, z) = x - y + z$, where $+, -$ are operations of $\mathbb{F}$, is a polymorphism of R. Indeed, let $A \cdot \mathbf{x} = \mathbf{b}$ be the system defining R, and $\mathbf{x}, \mathbf{y}, \mathbf{z} \in R$. Then

$$A \cdot f(\mathbf{x}, \mathbf{y}, \mathbf{z}) = A \cdot (\mathbf{x} - \mathbf{y} + \mathbf{z}) = A \cdot \mathbf{x} - A \cdot \mathbf{y} + A \cdot \mathbf{z} = \mathbf{b}.$$

In fact, the converse can also be shown: if R is invariant under f, where f is defined in a certain finite field $\mathbb{F}$, then R is the solution space of some system of linear equations over $\mathbb{F}$.

[1] In fact, due to the result of [47] this reduction can be made log-space.

Polymorphisms can be viewed as a generalization of homomorphisms of relational structures. We use $[k]$ to denote the set $\{1,\dots,k\}$. For a structure $\mathcal{B}$ with signature σ and $k \geq 1$, let $\mathcal{B}^k$ denote the following relational structure. The base set of $\mathcal{B}^k$ is B^k, where B is the base set of $\mathcal{B}$. For every symbol $R \in \sigma$, say, ℓ-ary, the predicate $R^{\mathcal{B}^k}$ is given by $(\mathbf{a}_1,\dots,\mathbf{a}_\ell) \in R^{\mathcal{B}^k}$ if and only if $(a_{1i},\dots,a_{\ell i}) \in R^{\mathcal{B}}$ for each $i \in [k]$, where $\mathbf{a}_j = (a_{j1},\dots,a_{jk})$. Then a mapping $f : B^k \to B$ is a polymorphism of $\mathcal{B}$ if and only if f is a homomorphism of $\mathcal{B}^k$ to $\mathcal{B}$.

A link between polymorphisms and pp-definability of relations is given by *Galois connection*.

▶ **Theorem 12** (Galois connection, [8, 34]). *Let Γ be a constraint language on A, and let $R \subseteq A^n$ be a non-empty relation. Then R is preserved by all polymorphisms of Γ if and only if R is pp-definable in Γ.*

Every relation on a set A has projections as polymorphisms. A *projection* is an operation f, say, k-ary, such that for some $i \in [k]$, $f(x_1,\dots,x_k) = x_i$. Theorem 12 means, in particular, that if a constraint language Γ on A does not have polymorphisms other than the projections, every relation is pp-definable in Γ.

The following statement is what makes the algebraic approach possible.

▶ **Corollary 13.** *Let Γ and Δ be constraint languages and Δ finite. If $\mathsf{Pol}(\Gamma) \subseteq \mathsf{Pol}(\Delta)$ then $\mathrm{CSP}(\Delta)$ is polynomial time reducible to $\mathrm{CSP}(\Gamma)$.*

For our next example we need another more technical result.

▶ **Proposition 14** ([19]). *Let Γ be a constraint language on a set A. Then either*

1. *there is a non-injective unary polymorphism f of Γ and $\mathrm{CSP}(\Gamma)$ is polynomial time interreducible with $\mathrm{CSP}(f(\Gamma))$, where $f(\Gamma) = \{f(R) \mid R \in \Gamma\}$, $f(R) = \{f(\mathbf{a}) \mid \mathbf{a} \in R\}$; or*
2. *all unary polymorphisms of Γ are injective and $\mathrm{CSP}(\Gamma)$ is polynomial time interreducible with $\mathrm{CSP}(\Gamma^*)$, where $\Gamma^* = \Gamma \cup \{R_a \mid a \in A\}$, $R_a = \{(a)\}$ is a* constant *relation, i.e. a unary relation that contains only one tuple.*

Constraint languages that contain all the constant relations are called *idempotent*.

▶ **Example 15.** We revisit the reducibility of 3-SAT to 3-Coloring. Let $\Gamma = \{\neq_3\}$ be the constraint language consisting of just the disequality relation $\neq_3$ on the 3-element set $\{0,1,2\}$. As we noted in Example 2, $\mathrm{CSP}(\Gamma)$ is equivalent to 3-Coloring. Let also Γ_{3-SAT} denote the constraint language consisting of the 8 ternary relations represented by 3-clauses. Then $\mathrm{CSP}(\Gamma_{3-SAT})$ is equivalent to 3-SAT. We show that Γ_{3-SAT} is pp-definable in Γ^*. By Theorem 10 and Proposition 14 it implies that 3-SAT is reducible to 3-Coloring.

First, we prove that every polymorphism of $\neq_3$ has only one *essential variable*, that is, a variable such that the value of the function can be changed by changing the value of this variable only. Suppose there is $f \in \mathsf{Pol}(\neq_3)$ that has at least two essential variables. To save on notation we assume that $f(x,y)$ is binary. Assume first that for some $a,a',b,b' \in \{0,1,2\}$ we have $f(a,b) \neq f(a',b), f(a,b) \neq f(a,b')$. Without loss of generality, let $a = b = 0, a' = b' = 1$. Then since $(0,1),(1,0) \in \neq_3$ and f is a polymorphism of $\neq_3$, we also have $f(0,1) \neq f(1,0)$. For a similar reason $f(2,2)$ is not equal to any of $f(0,0), f(0,1), f(1,0)$, which is impossible. Next, as y is an essential variable in $f(x,y)$ there are $a,b,b' \in \{0,1,2\}$ such that $f(a,b) \neq f(a,b')$. Again, let us assume $a = b = 0, b' = 1$. If $f(0,0) \neq f(1,0)$, or $f(0,0) \neq f(2,0)$, or $f(0,1) \neq f(1,1)$, or $f(0,1) \neq f(2,1)$, we have the previous case. Otherwise $f(0,2) \neq f(1,0) = f(0,0), f(0,2) \neq f(1,1) = f(0,1)$, because $(0,1),(2,0),(2,1) \in \neq_3$ and f is a polymorphism of $\neq_3$. By the same argument, either we have the previous case, or $f(0,2) = f(1,2) = f(2,2)$ implying that x is not essential variable in $f(x,y)$, a contradiction.

Second, if $f \in \mathsf{Pol}(\neq_3)$ has only one essential variable, we have $f(x_1, \dots, x_k) = g(x_i)$ for some operation g. Since f is a polymorphism of each constant relation R_a we get $g(a) = f(a, \dots, a) = a$, that is, f is a projection. Thus, the only polymorphisms of Γ^* are projections. By Theorem 12 every relation on $\{0, 1, 2\}$ is pp-definable in Γ^*. This includes the relations from Γ_{3-SAT}. By Theorem 10 $\mathrm{CSP}(\Gamma_{3-SAT})$ is polynomial time reducible to $\mathrm{CSP}(\Gamma)$.

Example 15 gives a reduction and a hardness proof only in one very special case. However, we shall see soon that together with pp-interpretability introduced next this technique gives a hardness proof for all NP-complete CSPs of the form $\mathrm{CSP}(\Gamma)$.

Let Γ, Δ be constraint languages over sets A, B, respectively. We say that Γ *pp-interprets* Δ if there exists a natural number ℓ, a set $F \subseteq A^\ell$, and an onto mapping $\pi : F \to B$ such that Γ pp-defines the following relations

1. the relation F,
2. the π-preimage of the equality relation on B, and
3. the π-preimage of every relation in Δ,

where by the π-preimage of a k-ary relation R on B we mean the ℓk-ary relation $\pi^{-1}(R)$ on A defined by

$$\pi^{-1}(R)(x_{11}, \dots, x_{1k}, x_{21}, \dots, x_{2k}, \dots, x_{\ell 1}, \dots, x_{\ell k}) \qquad \text{is true}$$

if and only if

$$R(\pi(x_{11}, \dots, x_{\ell 1}), \dots, \pi(x_{1k}, \dots, x_{\ell k})) \qquad \text{is true.}$$

If $\ell = 1$ in this definition, we say that Γ *1-pp-interprets* Δ.

▶ **Theorem 16** ([19, 4]). *Let Γ and Δ be constraint languages and Δ finite. If Δ is pp-interpretable in Γ then $\mathrm{CSP}(\Delta)$ is polynomial time reducible to $\mathrm{CSP}(\Gamma)$.*

▶ **Example 17.** We demonstrate how to use pp-interpretability in a small special case of the H-Coloring problem, see Example 2. Consider graph H on the left hand side of Fig. 2, we show that H-Coloring for this graph is NP-complete. In order to do that we will 1-pp-interpret $\neq_3$ in the constraint language $\Gamma = \{E = E(H)\}$. In the definition of pp-interpretation above we set $F = A = \{0, 0', 1, 2\}$, $B = \{0, 1, 2\}$, $\Delta = \{\neq_3\}$, $\ell = 1$, and $\pi : A \to B$ as shown in Fig. 2. The only thing we need to check is that $\pi^{-1}(B)$, $\pi^{-1}(=)$, and $\pi^{-1}(\neq_3)$ are pp-definable in H^2. The first and the last requirements are straightforward, because $\pi^{-1}(B) = A$ and $\pi^{-1}(\neq_3) = E$. For the preimage of the equality relation, the relation $\pi^{-1}(=)$ is the equivalence relation on A with classes $\{0, 0'\}, \{1\}, \{2\}$. This relation is defined by the pp-formula

$$\exists u, w(E(u, w) \wedge E(x, u) \wedge E(x, w) \wedge E(y, v) \wedge E(y, w)).$$

[2] Note that although π is a homomorphism from H to K_3, this does not yet give rise to a reduction from H-Coloring to 3-Coloring. Indeed, any graph is a homomorphic image of a collection of disconnected edges. Homomorphism π is a very special kind of homomorphism that gets along pp-definitions well.

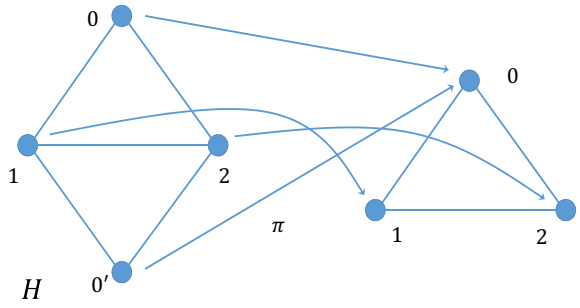


Figure 2 Pp-interpretations of 3-Coloring.

2.3 Polymorphisms and algorithms

So far we have only seen applications of polymorphisms and pp-interpretations for discovering polynomial time reductions between CSPs. In this section we mention two examples when polymorphisms also help developing efficient solution algorithms.

▶ **Example 18.** As we saw in Example 11, a relation can be represented as the set of solutions of a system of linear equations over a finite field $\mathbb{F}$ if and only if it is invariant under the affine operation $f(x, y, z) = x - y + z$, where $+, -$ are the operations of the same field $\mathbb{F}$. This means that if a constraint language Γ is invariant with respect to f then any instance of CSP(Γ) can be thought of as a system of linear equations over $\mathbb{F}$ and so can be solved by Gaussian elimination. In particular, it is possible not only to decide the existence of a solution in polynomial time, but also to construct a concise representation of the set of all solutions – a basis of the solution space.

Affine operations are a special kind of more general Maltsev operations. A ternary operation f is said to be *Maltsev* if it satisfies the equations $f(x, y, y) = f(y, y, x) = x$. The affine operation clearly satisfies them. It was shown in [16] that if a constraint language Γ has a Maltsev operation as a polymorphism, there is a polynomial time algorithm that constructs a concise representation of the set of solutions of any instance of CSP(Γ). Note that when applied to systems of linear equations this algorithm provides an alternative to Gaussian elimination. This result was further generalized in [38].

▶ **Example 19.** For this example it will be convenient to use infix notation for binary operations, that is, to write $x \cdot y$ rather than $f(x, y)$. A binary operation $\cdot$ on a set A is said to be *semilattice* if it is idempotent ($x \cdot x = x$), commutative ($x \cdot y = y \cdot x$), and associative ($x \cdot (y \cdot z) = (x \cdot y) \cdot z$), where the equations hold for all $x, y, z \in A$. A semilattice operation defines a partial order on A: $a \leq b$ if and only if $b = a \cdot b$.

Let $R \subseteq A^k$ be a relation invariant with respect to $\cdot$. Let also $R_i \subseteq A$ denote its *ith projection*, the set $R_i = \{a_i \mid (a_1, \dots, a_k) \in R\}$. As is easily seen, R_i is invariant under $\cdot$, that is, $a \cdot b \in R_i$ for any $a, b \in R_i$. Every set R_i has a greatest element in terms of the order $\leq$, it is the product of all the elements of R_i. Let us denote this element by m_i. Now, if $\mathbf{a}_1, \dots, \mathbf{a}_r$ is a list of all tuples from R_i, we have $\mathbf{a}_1 \cdot \mathbf{a}_2 \cdot \dots \cdot \mathbf{a}_r = (m_1, \dots, m_k)$. In other words, the tuple, whose entries are the greatest elements of the corresponding projections, belongs to R.

The following algorithm known as the *arc-consistency* algorithm [40] uses the discussed above properties of relations invariant under $\cdot$ to solve CSP(Γ) where Γ is any constraint language invariant under $\cdot$. Let $\mathcal{I}$ be an instance of CSP(Γ) with the set of variables X. The algorithm works in rounds until the instance cannot be further improved. Suppose that two constraints $R^1(x, \mathbf{y})$ and $R^2(x, \mathbf{z})$ from $\mathcal{I}$ share a variable. Here $\mathbf{y}, \mathbf{z}$ stand for some variables involved in R^1, R^2, and clearly x does not have to be in the same position in R^1, R^2. Let $R_1 = R^1_1 \cap R^2_1$. If $R_1 \neq R^1_1$, we remove from R^1 all tuples $(a, \mathbf{b})$ such that $a \notin R_1$. Then we repeat the procedure for R^2. After the process converges, every variable $x \in X$ has a domain $A_x \subseteq A$ such that the projection of every constraint containing x on the corresponding coordinate position equals A_x. Finally, that $\cdot$ is a polymorphism of Γ implies that the mapping $\varphi : X \to A$, $\varphi(x) = m_x$, where m_x is the greatest element of A_x is a solution of $\mathcal{I}$.

3 Dichotomies

3.1 The CSP dichotomy

The first attempt for a broad classification of problems of the form CSP(Γ) was made in [48]. When translated into the language of polymorphisms, the main result is

▶ **Theorem 20** ([48]). *Let Γ be a constraint language over $\{0, 1\}$. Then* CSP(Γ) *is solvable in polynomial time if and only if Γ has one of the following polymorphisms: a constant operation, the affine operation $x - y + z \pmod 2$, one of the semilattice operations $x \vee y$ or $x \wedge y$, or the* majority *operation $(x \wedge y) \vee (y \wedge z) \vee (z \wedge x)$. Otherwise it is NP-complete.*

The hardness part of the theorem follows by an argument similar to that in Example 15. The tractability part follows from Proposition 14 and Examples 18 and 19. The case of the majority operation can be solved as 2-SAT or by a slightly stronger consistency algorithm, see [40].

An important feature of Theorem 20 is that it leaves no room for CSPs of intermediate complexity: every CSP(Γ) is either solvable in polynomial time or is NP-complete. Results of this kind are also known as *complexity dichotomies*, and believed to be true for the majority of "natural" problems, even though problems of intermediate complexity provably exist provided P≠NP, [44].

A dichotomy for CSPs over arbitrary finite sets was conjectured in [31, 32]. This conjecture has been referred to as the CSP Dichotomy Conjecture. The conjecture was refined by providing a specific criterion of polynomial time solvability in [19], and settled in [14, 15, 51].

We now state the Dichotomy Theorem from [14, 15, 51]. Let Γ, Δ be constraint languages on sets A, B respectively and such that Γ 1-pp-interprets Δ. Suppose $F \subseteq A$ and $\pi : F \to B$ are the parameters of the pp-interpretation, and θ the equivalence relation on F that is the π-preimage of the equality relation on B, or the kernel of π. For $a \in F$ let $a/_\theta$ denote the θ-class containing a. Take a polymorphism $f(x_1, \dots, x_k)$ of Γ. By $f_{F,\pi}$ we denote the k-ary operation on B given by $f_{F,\pi}(x_1, \dots, x_k) = \pi(f(\pi^{-1}(x_1), \dots, \pi^{-1}(x_k)))$. Since θ is invariant under f, the operation $f_{F,\pi}(x_1, \dots, x_k)$ is properly defined, that is, its value does not depend on the choices of preimages $\pi^{-1}(x_i)$. It is straightforward to verify that $f_{F,\pi}$ is a polymorphism of Δ. Note that in order to define $f_{F,\pi}$ we do not need to specify constraint language Δ, we only need F and π.

Also, by Proposition 14 it suffices to consider idempotent constraint languages.

▶ **Theorem 21** ([14, 15, 51]). *Let Γ be an idempotent constraint language on a finite set A. Then* $\mathrm{CSP}(\Gamma)$ *is NP-complete if and only if there is a pp-interpretation with parameters F, π such that $f_{F,\pi}$ is a projection for any $f \in \mathsf{Pol}(\Gamma)$.*

The criterion in Theorem 21 can be elevated to a higher level of abstraction that involves considering sets of polymorphisms as a new algebraic structure and using the properties of mappings between these structures, *clone homomorphisms*. We touch upon such approach in Section 4.1.

3.2 Counting CSPs and polymorphisms

In this section we give some examples that hint at how a dichotomy result can be obtained for the counting version of the CSP of the form $\#\mathrm{CSP}(\Gamma)$.

We first remark that the algebraic approach works for counting CSPs as well as for decision CSPs. Recall that Γ^* for a constraint language Γ denotes Γ with added constant relations.

▶ **Theorem 22** ([17]). *Let Γ be a constraint language on a finite set A.*

1. $\#\mathrm{CSP}(\Gamma)$ *and* $\#\mathrm{CSP}(\Gamma^*)$ *are polynomial time interreducible.*
2. *For any constraint language Δ on A such that* $\mathsf{Pol}(\Gamma) \subseteq \mathsf{Pol}(\Delta)$ *the problem* $\#\mathrm{CSP}(\Delta)$ *is polynomial time reducible to* $\mathrm{CSP}(\Gamma)$.
3. *For any constraint language Δ, if Γ pp-interprets Δ then* $\#\mathrm{CSP}(\Delta)$ *is polynomial time reducible to* $\#\mathrm{CSP}(\Gamma)$.

Next we use the algebraic approach to prove a dichotomy theorem for $\#H$-Coloring, the counting version of H-Coloring, first proved in [29].

▶ **Example 23.** It was proved in [29] that $\#H$-Coloring is #P-complete, unless every connected component of H is either an isolated vertex, or a complete graph with all loops present, or a complete bipartite graph. That $\#H$-Coloring can be solved in polynomial time for graphs of this kind is straightforward. We show how to prove the hardness part through the algebraic approach in 3 easy steps.

First, using some algebraic machinery ([36] or Theorem 9.13 from [37]) it can be shown that if a constraint language Γ does not have a Maltsev polymorphism, Γ pp-interprets a binary reflexive but not symmetric relation R. Second, using the standard interpolation technique (see e.g. [50]) [17] proves that $\#\mathrm{CSP}(R)$ is #P-complete. This shows that $\#\mathrm{CSP}(\Gamma)$, not only $\#H$-Coloring, is #P-complete for any Γ without a Maltsev polymorphism.

Finally, if a connected graph H is not a single vertex, a complete graph with all loops present, or a complete bipartite graph, it contains the N-graph shown in Fig. 3 as an induced subgraph (some of a, b, c, d may be equal). It remains to observe that no Maltsev operation can be a polymorphism of such graph. Indeed, if f is a Maltsev operation on H then

$$f\left(\binom{a}{c}, \binom{a}{d}, \binom{b}{d}\right) = \binom{b}{c} \notin E(H).$$

It turns out that a generalized version of avoiding N-graphs is key in the case of general counting CSPs, $\#\mathrm{CSP}(\Gamma)$. A constraint language is said to be *singular* if for any pp-interpretable equivalence relations θ, η a certain condition holds on the number of elements in θ- and η-classes. For more details, see [18, 13, 30].

▶ **Theorem 24** ([13, 30]). *For a constraint language Γ on a finite set,* $\#\mathrm{CSP}(\Gamma)$ *is solvable in polynomial time if and only if Γ has a Maltsev polymorphism and is singular. Otherwise it is #P-complete.*

Theorem 24 has been generalized to the weighted version of the CSP in [25].

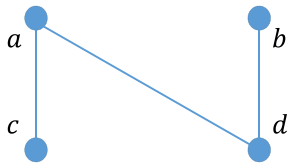


Figure 3 The N-graph.

4 Beyond CSP

In this section we consider several examples in which some sort of algebraic approach is used, although it requires significant modifications. In the first two examples, the Promise CSP and the optimization version, the Valued CSP, while preserving the main motifs of the algebraic approach, that is, rich families of polymorphisms give rise to easy problems, and poor ones give rise to hard problems, new types of "polymorphism-like" objects need to be introduced. In the third case, Graph Isomorphism, the algebraic approach does not (probably) provide a general technique, but is useful in some special cases.

4.1 Promise CSP

Unlike the regular CSP that is often parametrized by a single relational structure, $\mathrm{CSP}(\mathcal{B})$, a Promise CSP or PCSP for short is parametrized by a pair of similar relational structures, $\mathrm{PCSP}(\mathcal{A},\mathcal{B})$, such that $\mathcal{A}\to\mathcal{B}$. An instance of $\mathrm{PCSP}(\mathcal{A},\mathcal{B})$ is a relational structure $\mathcal{C}$ similar to $\mathcal{A},\mathcal{B}$, and the question is whether $\mathcal{C}\to\mathcal{A}$ or $\mathcal{C}\not\to\mathcal{B}$; no specific answer is required if $\mathcal{C}\not\to\mathcal{A}$ but $\mathcal{C}\to\mathcal{B}$. It is also often formulated in a slightly different way: the input is a structure $\mathcal{C}$ such that $\mathcal{C}\to\mathcal{A}$, and the goal is to find a homomorphism from $\mathcal{C}$ to $\mathcal{B}$, which explains the "promise" in the name of the problem, [10, 23].

▶ **Example 25.** The Approximate Graph Coloring problem is parametrized by two numbers k,c, $k\le c$, and the goal is to find a c-coloring of a given k-colorable graph. As is easily seen, this problem is equivalent to $\mathrm{PCSP}(K_k,K_c)$.

Polymorphisms of a single relational structure $\mathcal{A}$ were defined as homomorphisms from $\mathcal{A}^k$ to $\mathcal{A}$. In the case of PCSPs such mappings do not make much sense. Instead, a polymorphism of a pair $\mathcal{A},\mathcal{B}$ is a homomorphism from $\mathcal{A}^k$ to $\mathcal{B}$. Let $\mathsf{Pol}(\mathcal{A},\mathcal{B})$ denote the set of all such homomorphisms for all natural k. Note that as there is a homomorphism $\varphi:\mathcal{A}\to\mathcal{B}$, there also exist some homomorphisms in $\mathsf{Pol}(\mathcal{A},\mathcal{B})$: Fix $i\in[k]$ the mapping $\varphi_i:\mathcal{A}^k\to\mathcal{B}$ given by $\varphi_i(a_1,\dots,a_k)=\varphi(a_i)$ is a homomorphism. Homomorphisms φ_i are analogous to projections in the realm of polymorphisms of a single structure.

We describe the algebraic approach to the PCSP at a higher level of abstraction than that for the CSP. An operation $f:A^n\to B$ is a *minor* of operation $g:A^m\to B$ if there is a mapping $\pi:[m]\to[n]$ such that $f(x_1,\dots,x_n)=g(x_{\pi(1)},\dots,x_{\pi(m)})$. In other words, f is obtained from g by permuting and identifying variables. For example, $f(x_1,x_2,x_3)=g(x_2,x_3,x_2,x_1,x_1,x_3)$ is a minor of $g(x_1,x_2,x_3,x_4,x_5,x_6)$. Let $\mathcal{A}_1,\mathcal{B}_1$, and $\mathcal{A}_2,\mathcal{B}_2$, $\mathcal{A}_1\to\mathcal{B}_1$, $\mathcal{A}_2\to\mathcal{B}_2$, be two pairs of relational structures such that $\mathcal{A}_1,\mathcal{B}_1$ are similar and so are $\mathcal{A}_2,\mathcal{B}_2$. A mapping $\Psi:\mathsf{Pol}(\mathcal{A}_1,\mathcal{B}_1)\to\mathsf{Pol}(\mathcal{A}_2,\mathcal{B}_2)$ (i.e., polymorphisms are mapped to polymorphisms) is called a *minion homomorphism* if the following two conditions hold.

1. Ψ preserves arity, that is, for an operation $f : A_1^k \to B_1$, it holds $\Psi f : A_2^k \to B_2$.
2. Ψ preserves minors, that is, for any m-ary $g \in \mathsf{Pol}(\mathcal{A}_1, \mathcal{B}_1)$ and any $\pi : [m] \to [n]$, if $f(x_1, \dots, x_n) = g(x_{\pi(1)}, \dots, x_{\pi(m)})$ is a minor of g then $\Psi f(x_1, \dots, x_n) = \Psi g(x_{\pi(1)}, \dots, x_{\pi(m)})$.

The core of the algebraic approach to the PCSP is the following

▶ **Theorem 26** ([23]). *Let* $\mathrm{PCSP}(\mathcal{A}_1, \mathcal{B}_1), \mathrm{PCSP}(\mathcal{A}_2, \mathcal{B}_2)$ *be two Promise CSPs such that there is a minion homomorphism from* $\mathsf{Pol}(\mathcal{A}_1, \mathcal{B}_1)$ *to* $\mathsf{Pol}(\mathcal{A}_2, \mathcal{B}_2)$*. Then* $\mathrm{PCSP}(\mathcal{A}_2, \mathcal{B}_2)$ *is polynomial time reducible to* $\mathrm{PCSP}(\mathcal{A}_1, \mathcal{B}_1)$*.*

Note that by setting $\mathcal{A}_1 = \mathcal{B}_1$ and $\mathcal{A}_2 = \mathcal{B}_2$ Theorem 26 specializes to a statement about sets of polymorphisms of a single relational structure. Thus, Theorem 26 is a generalized and more abstract version of the combination of Corollary 13, Proposition 14, and Theorem 16.

▶ **Example 27** ([23]). In this example we use Theorem 26 to prove that $\mathrm{PCSP}(K_3, K_4)$ is NP-complete. In other words we outline a proof that given a 3-colorable graph it is in general NP-hard to find its 4-coloring. In order to do this we present a minion homomorphism from $\mathsf{Pol}(K_3, K_4)$ to $\mathsf{Pol}(\mathcal{A}, \mathcal{A})$, where $\mathcal{A}$ is any 2-element relational structure that only has projections as polymorphisms. Note that $\mathsf{Pol}(\mathcal{A}, \mathcal{A}) = \mathsf{Pol}(\mathcal{A})$, and by Corollary 13 what specific structure we choose is completely irrelevant. For instance, the structure with the base set $\{0, 1\}$ and the only predicate, which is $R_{1\text{-in-}3}$, see Example 6, fits this purpose. Since $\mathrm{CSP}(\mathcal{A}) = \mathrm{PCSP}(\mathcal{A}, \mathcal{A})$ is NP-complete in this case, it implies the result.

The structure of polymorphisms from $\mathsf{Pol}(K_3, K_4)$ is described in [23, 9]. For each (say, k-ary) $f \in \mathsf{Pol}(K_3, K_4)$, there exist $t \in K_4$, $i \in [k]$, and a mapping $\varphi : K_3 \to K_4$ such that $f(a_1, \dots, a_k) \in \{t, \varphi(a_i)\}$ for all $a_1, \dots, a_n \in K_3$. In other words, if the value of f is not t, it depends only on x_i. A mapping $\Psi : \mathsf{Pol}(K_3, K_4) \to \mathsf{Pol}(\mathcal{A}, \mathcal{A})$ is defined as follows: For every k-ary $f \in \mathsf{Pol}(K_3, K_4)$, Ψf is the k-ary projection $p(x_1, \dots, x_k) = x_i$, where i is the parameter associated with f. It is straightforward to verify that Ψ is a minion homomorphism.

4.2 Valued CSP, optimization

The VCSP and optimization problems admit some sort of algebraic approach, however the concept of a polymorphism is substantially different. In this section we briefly outline how the approach works when we want to find the exact optimum of a VCSP [49], and also for approximation algorithms [11]. The functions we consider here are real-valued.

To introduce the notion of polymorphisms needed for VCSPs, we need to take into account all the operations of certain arity on a set, rather than a single operation. Let $\mathcal{O}_A^{(k)}$ denote the set of all k-ary operations on a set A. Let $R : A^m \to \mathbb{R}$ be a function on A. A k-ary *fractional polymorphism* of R is a probability distribution μ on $\mathcal{O}_A^{(k)}$ that for any $\mathbf{a}_1, \dots, \mathbf{a}_k \in A^m$, $\mathbf{a}_i = (a_{i1}, \dots, a_{im})$ satisfies (in the case of minimization problems) the following condition

$$\mathbb{E}_{f \sim \mu}[R(f(\mathbf{a}_1, \dots, \mathbf{a}_k))] \le \mathsf{avg}(R(\mathbf{a}_1), \dots, R(\mathbf{a}_k)),$$

where $f(\mathbf{a}_1, \dots, \mathbf{a}_k) = (f(a_{11}, \dots, a_{k1}), \dots, f(a_{1m}, \dots, a_{km}))$. In the case of maximization problems the inequality should be reversed. Distribution μ is a fractional polymorphism of a valued constraint language Γ if it is a fractional polymorphism of every function from Γ.

▶ **Example 28.** Let $A = \{0, 1\}$, and let μ be the distribution on $\mathcal{O}_A^{(2)}$ that assigns probability $1/2$ to $\vee$ (disjunction) and $\wedge$ (conjunction), which are binary operations on $\{0, 1\}$, and probability 0 to all other operations. For a binary function R the inequality above is transformed into

$$\frac{1}{2}(R(x_1 \vee x_2, y_1 \vee y_2) + R(x_1 \wedge x_2, y_1 \wedge y_2) \le \frac{1}{2}(R(x_1, y_1) + R(x_2, y_2)),$$

which is the well known condition of submodularity.

A binary fractional polymorphism μ (i.e., a distribution on $\mathcal{O}_A^{(2)}$) is said to be *symmetric* if it assigns nonzero probabilities only to operations $f(x,y)$ satisfying $f(x,y) = f(y,x)$.

▶ **Theorem 29** ([49]). *For a valued constraint language* Γ *with real values the problem* VCSP(Γ) *is solvable in polynomial time if and only if* Γ *has a binary symmetric fractional polymorphism.*

Observe that the condition of submodularity is an example of a symmetric fractional polymorphism.

Many VCSPs that cannot be solved exactly can be approximated within a constant factor. Assuming the Unique Games Conjecture (UGC) [11] determines the best approximation factor, so-called *approximation threshold*, for arbitrary VCSPs. The main tool in this result is a variation of fractional polymorphisms. For a set A we again consider probability distributions over $\mathcal{O}_A^{(k)}$. Let $\alpha \in [0,1]$. Distribution μ is said to be an *α-approximation polymorphism* of $R : A^m \to \mathbb{R}$ if for any $\mathbf{a}_1, \dots, \mathbf{a}_k \in A^m$ it satisfies the following condition

$$\alpha \cdot \mathbb{E}_{f\sim\mu}[R(f(\mathbf{a}_1, \dots, \mathbf{a}_k))] \geq \mathsf{avg}(R(\mathbf{a}_1), \dots, R(\mathbf{a}_k)).$$

For a definition of pseudorandom approximation polymorphisms the reader is referred to [11].

▶ **Theorem 30** ([11]). *Let* Γ *be a valued constraint language, and let* α_Γ *be the greatest constant such that there is a pseudorandom* α_Γ*-approximation polymorphism of* Γ*. Then (assuming the UGC)* α_Γ *is the approximation threshold for* VCSP(Γ).

4.3 Graph Isomorphism: bounded color classes

A somewhat surprising application of the algebraic approach in the Graph Isomorphism problem was observed in [5]. In the Graph Isomorphism problem [35] the question is to decide whether two given graphs are isomorphic. The equivalent formulation we use here asks to find a generating set for the *automorphism group* $\mathsf{Aut}(G)$ of G. Often considering the structure of G, e.g. the degrees of its vertices, it is possible to identify some restrictions on the *orbits* of $\mathsf{Aut}(G)$, that is, which vertices can be mapped to each other by automorphisms. For instance, a vertex can only be mapped to a vertex of the same degree. Usually such restrictions are a result of some sort of *color refinement* process, but this is not important here. Suppose that we can identify a partition of $V(G)$ into classes $V_1, \dots, V_k$ such that any automorphism φ maps V_i to V_i. Assume also that the sizes of the V_is are bounded by $\ell \in \mathbb{N}$. Let us further assume for simplicity that $|V_i| = \ell$ for all $i \in [k]$. Then any $\varphi \in \mathsf{Aut}(G)$ is a union of permutations φ_i of V_i, $i \in [k]$, and the question is which combinations of such local permutations give rise to an automorphism of G.

We describe how this problem can be reduced to CSP(Γ) where Γ is a constraint language with a Maltsev polymorphism, see Example 18. Let $V_i = \{v_i^1, \dots, v_i^\ell\}$ be some enumeration of V_i, and let the symmetric group S_ℓ of permutations of $[\ell]$ acts on V_i as follows: for $\pi \in S_\ell$, $\pi(v_i^j) = v_i^{\pi(j)}$. The domain for our CSP is S_ℓ, and the variables are $V_1, \dots, V_k$. For every pair V_i, V_j we introduce a constraint $R_{ij}(V_i, V_j)$ as follows. Suppose $v_i^s v_j^r$ is an edge of G. Then if φ_i, φ_j are a part of an automorphism of G, $\varphi_i(v_i^s)\varphi_j(v_j^r) \in E(G)$. Permutations φ_i, φ_j correspond to some $\pi_i, \pi_j \in S_\ell$. Thus, we define R_{ij} to be $\{(\pi_i, \pi_j) \mid$ for any $v_i^s v_j^r \in E(G)$, $v_i^{\pi_i(s)} v_j^{\pi_j(r)} \in E(G)\}$. Let $\Gamma = \{R_{ij} \mid i, j \in [k]\}$. Observe that R_{ij} is invariant under composition, i.e. if $(\pi_i, \pi_j), (\tau_i, \tau_j) \in R_{ij}$ then $(\pi_i \circ \tau_i, \pi_j \circ \tau_j) \in R_{ij}$. In particular, it is invariant under the operation $x \circ y^{-1} \circ z$ of S_ℓ, which is a Maltsev operation on S_ℓ.

The algorithm from [16] finds a concise representation of the set of solutions of the CSP above, which then can be used to construct a generating set for $\mathsf{Aut}(G)$. The running time of this algorithm is polynomial in k and $|S_\ell| = \ell!$.

5 Conclusion

As we have seen, algebraic methods in CSPs have found their applications in numerous areas of computer science. In some of those areas such as decision, counting, and valued CSPs comprehensive and broad results have been obtained. In some other areas active research is ongoing. In particular the algebraic structure of the Promise CSP and CSPs over infinite domains have received much attention recently, although major questions remain open. There have been attempts to introduce the algebraic approach to the study of the holant problem [2] and certain proof systems suitable for approximation (such as Sum-of-Squares) [22], but these studies are in their infancy. Finally, from our perspective one of the most interesting areas where the algebraic approach is yet to be developed is approximate counting. Apart from the CSP itself it also has strong connections to other fields such as statistical physics, and it would be very interesting to see what kind of algebraic structure is possible here.

References

1 Albert Atserias and Joanna Ochremiak. Proof complexity meets algebra. *ACM Trans. Comput. Log.*, 20(1):1:1–1:46, 2019.

2 Miriam Backens and Leslie Ann Goldberg. Holant clones and the approximability of conservative holant problems. *ACM Trans. Algorithms*, 16(2):23:1–23:55, 2020.

3 Libor Barto and Marcin Kozik. Robustly solvable constraint satisfaction problems. *SIAM J. Comput.*, 45(4):1646–1669, 2016.

4 Libor Barto, Andrei A. Krokhin, and Ross Willard. Polymorphisms, and how to use them. In Andrei A. Krokhin and Stanislav Zivný, editors, *The Constraint Satisfaction Problem: Complexity and Approximability*, volume 7 of *Dagstuhl Follow-Ups*, pages 1–44. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2017.

5 Christoph Berkholz and Martin Grohe. Limitations of algebraic approaches to graph isomorphism testing. In Magnús M. Halldórsson, Kazuo Iwama, Naoki Kobayashi, and Bettina Speckmann, editors, *Automata, Languages, and Programming - 42nd International Colloquium, ICALP 2015, Kyoto, Japan, July 6-10, 2015, Proceedings, Part I*, volume 9134 of *Lecture Notes in Computer Science*, pages 155–166. Springer, 2015.

6 Manuel Bodirsky, Martin Hils, and Barnaby Martin. On the scope of the universal-algebraic approach to constraint satisfaction. *Log. Methods Comput. Sci.*, 8(3), 2009.

7 Manuel Bodirsky and Marcello Mamino. Constraint satisfaction problems over numeric domains. In Andrei A. Krokhin and Stanislav Zivný, editors, *The Constraint Satisfaction Problem: Complexity and Approximability*, volume 7 of *Dagstuhl Follow-Ups*, pages 79–111. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2017.

8 V.G. Bodnarchuk, L.A. Kaluzhnin, V.N. Kotov, and B.A. Romov. Galois theory for Post algebras. I. *Kibernetika*, 3:1–10, 1969.

9 Joshua Brakensiek and Venkatesan Guruswami. New hardness results for graph and hypergraph colorings. In Ran Raz, editor, *31st Conference on Computational Complexity, CCC 2016, May 29 to June 1, 2016, Tokyo, Japan*, volume 50 of *LIPIcs*, pages 14:1–14:27. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2016.

10 Joshua Brakensiek and Venkatesan Guruswami. Promise constraint satisfaction: Structure theory and a symmetric Boolean dichotomy. In Artur Czumaj, editor, *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2018, New Orleans, LA, USA, January 7-10, 2018*, pages 1782–1801. SIAM, 2018.

11 Jonah Brown-Cohen and Prasad Raghavendra. Combinatorial optimization algorithms via polymorphisms. *CoRR*, abs/1501.01598, 2015. `arXiv:1501.01598`.

12 Andrei Bulatov, Marcin Kozik, Peter Mayr, and Markus Steindl. The subpower membership problem for semigroups. *Int. J. Algebra Comput.*, 26(7):1435–1451, 2016.

13 Andrei A. Bulatov. The complexity of the Counting Constraint Satisfaction Problem. *J. ACM*, 60(5):34:1–34:41, 2013.

14 Andrei A. Bulatov. A dichotomy theorem for nonuniform CSPs. In *58th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2017, Berkeley, CA, USA, October 15-17, 2017*, pages 319–330, 2017.

15 Andrei A. Bulatov. A dichotomy theorem for nonuniform CSPs simplified. *CoRR*, abs/2007.09099, 2020. `arXiv:2007.09099`.

16 Andrei A. Bulatov and Víctor Dalmau. A simple algorithm for Mal'tsev constraints. *SIAM J. Comput.*, 36(1):16–27, 2006.

17 Andrei A. Bulatov and Víctor Dalmau. Towards a dichotomy theorem for the Counting Constraint Satisfaction Problem. *Inf. Comput.*, 205(5):651–678, 2007.

18 Andrei A. Bulatov and Martin Grohe. The complexity of partition functions. *Theor. Comput. Sci.*, 348(2-3):148–186, 2005.

19 Andrei A. Bulatov, Peter Jeavons, and Andrei A. Krokhin. Classifying the complexity of constraints using finite algebras. *SIAM J. Comput.*, 34(3):720–742, 2005.

20 Andrei A. Bulatov and Dániel Marx. Constraint Satisfaction Problems and global cardinality constraints. *Commun. ACM*, 53(9):99–106, 2010.

21 Andrei A. Bulatov and Dániel Marx. Constraint Satisfaction parameterized by solution size. *SIAM J. Comput.*, 43(2):573–616, 2014.

22 Andrei A. Bulatov and Akbar Rafiey. On the complexity of CSP-based ideal membership problems. *CoRR*, abs/2011.03700, 2020. `arXiv:2011.03700`.

23 Jakub Bulín, Andrei A. Krokhin, and Jakub Oprsal. Algebraic approach to promise constraint satisfaction. In Moses Charikar and Edith Cohen, editors, *Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing, STOC 2019, Phoenix, AZ, USA, June 23-26, 2019*, pages 602–613. ACM, 2019.

24 Jin-Yi Cai and Xi Chen. *Complexity dichotomies for counting problems. Volume 1: Boolean domain.* Cambridge University Press, 2017.

25 Jin-Yi Cai and Xi Chen. Complexity of counting CSP with complex weights. *J. ACM*, 64(3):19:1–19:39, 2017.

26 Hubie Chen, Matt Valeriote, and Yuichi Yoshida. Constant-query testability of assignments to Constraint Satisfaction Problems. *SIAM J. Comput.*, 48(3):1022–1045, 2019.

27 Hubie Chen and Matthew Valeriote. Learnability of solutions to conjunctive queries. *J. Mach. Learn. Res.*, 20:67:1–67:28, 2019.

28 Nadia Creignou, Sanjeev Khanna, and Madhu Sudan. *Complexity Classifications of Boolean Constraint Satisfaction Problems*, volume 7 of *SIAM Monographs on Discrete Mathematics and Applications*. SIAM, 2001.

29 Martin E. Dyer and Catherine S. Greenhill. The complexity of counting graph homomorphisms. *Random Struct. Algorithms*, 17(3-4):260–289, 2000.

30 Martin E. Dyer and David Richerby. An effective dichotomy for the Counting Constraint Satisfaction Problem. *SIAM J. Comput.*, 42(3):1245–1274, 2013.

31 Tomas Feder and Moshe Y. Vardi. Monotone Monadic SNP and constraint satisfaction. In *Proceedings of 25th ACM Symposium on the Theory of Computing (STOC)*, pages 612–622, 1993.

32 Tomas Feder and Moshe Y. Vardi. The computational structure of Monotone Monadic SNP and constraint satisfaction: A study through Datalog and group theory. *SIAM Journal of Computing*, 28:57–104, 1998.

33 Fedor V. Fomin, Alexander Golovnev, Alexander S. Kulikov, and Ivan Mihajlin. Lower bounds for the graph homomorphism problem. In Magnús M. Halldórsson, Kazuo Iwama, Naoki Kobayashi, and Bettina Speckmann, editors, *Automata, Languages, and Programming - 42nd International Colloquium, ICALP 2015, Kyoto, Japan, July 6-10, 2015, Proceedings, Part I*, volume 9134 of *Lecture Notes in Computer Science*, pages 481–493. Springer, 2015.

34 D. Geiger. Closed systems of function and predicates. *Pacific Journal of Mathematics*, pages 95–100, 1968.

35 Martin Grohe and Pascal Schweitzer. The graph isomorphism problem. *Commun. ACM*, 63(11):128–134, 2020.

36 J. Hagemann and A. Mitschke. On n-permutable congruences. *Algebra Universalis*, pages 8–12, 1973.

37 D. Hobby and R.N. McKenzie. *The Structure of Finite Algebras*, volume 76 of *Contemporary Mathematics*. American Mathematical Society, Providence, R.I., 1988.

38 Pawel M. Idziak, Petar Markovic, Ralph McKenzie, Matthew Valeriote, and Ross Willard. Tractability and learnability arising from algebras with few subpowers. *SIAM J. Comput.*, 39(7):3023–3037, 2010.

39 Peter Jeavons, David A. Cohen, and Marc Gyssens. Closure properties of constraints. *J. ACM*, 44(4):527–548, 1997.

40 Peter G. Jeavons, David A. Cohen, and Martin C. Cooper. Constraints, consistency and closure. *Artificial Intelligence*, 101(1-2):251–265, 1998.

41 Peter Jonsson, Victor Lagerkvist, and Biman Roy. Fine-grained time complexity of Constraint Satisfaction Problems. *ACM Trans. Comput. Theory*, 13(1):2:1–2:32, 2021.

42 Ondrej Klíma, Pascal Tesson, and Denis Thérien. Dichotomies in the complexity of solving systems of equations over finite semigroups. *Theory Comput. Syst.*, 40(3):263–297, 2007.

43 Phokion G. Kolaitis and Moshe Y. Vardi. Conjunctive-query containment and constraint satisfaction. In Alberto O. Mendelzon and Jan Paredaens, editors, *Proceedings of the Seventeenth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, June 1-3, 1998, Seattle, Washington, USA*, pages 205–213. ACM Press, 1998.

44 Richard Ladner. On the structure of polynomial time reducibility. *Journal of the ACM*, 22:155–171, 1975.

45 Barnaby Martin. Quantified constraints in twenty seventeen. In Andrei A. Krokhin and Stanislav Zivný, editors, *The Constraint Satisfaction Problem: Complexity and Approximability*, volume 7 of *Dagstuhl Follow-Ups*, pages 327–346. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2017.

46 Monaldo Mastrolilli. The complexity of the ideal membership problem for constrained problems over the Boolean domain. In Timothy M. Chan, editor, *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2019, San Diego, California, USA, January 6-9, 2019*, pages 456–475. SIAM, 2019.

47 Omer Reingold. Undirected connectivity in log-space. *J. ACM*, 55(4):17:1–17:24, 2008.

48 Thomas J. Schaefer. The complexity of satisfiability problems. In *Proceedings of the 10th ACM Symposium on Theory of Computing (STOC'78)*, pages 216–226, 1978.

49 Johan Thapper and Stanislav Zivný. The complexity of finite-valued CSPs. *J. ACM*, 63(4):37:1–37:33, 2016.

50 Leslie G. Valiant. The complexity of enumeration and reliability problems. *SIAM J. Comput.*, 8(3):410–421, 1979.

51 Dmitriy Zhuk. A proof of the CSP dichotomy conjecture. *J. ACM*, 67(5):30:1–30:78, 2020.

Distributed Subgraph Finding: Progress and Challenges

Keren Censor-Hillel
Department of Computer Science, Technion, Haifa, Israel

Abstract

This is a survey of the exciting recent progress made in understanding the complexity of distributed subgraph finding problems. It overviews the results and techniques for assorted variants of subgraph finding problems in various models of distributed computing, and states intriguing open questions.

2012 ACM Subject Classification Networks → Network algorithms; Theory of computation → Distributed algorithms

Keywords and phrases distributed algorithms, subgraph finding, limited bandwidth

Digital Object Identifier 10.4230/LIPIcs.ICALP.2021.3

Category Invited Talk

Funding This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement no. 755839.

Acknowledgements The author thanks Francois Le Gall for clarifications and Dean Leitersdorf for useful feedback on a preliminary version of this survey.

1 Introduction

Distributed subgraph finding is motivated by considering a user in a network whose connections are unknown to its users. Typical examples could be friends or followers in social networks, or computing devices in large networks. A prime question is what is the local structure of the network, e.g., do two connected users share common connections, or are they part of a slightly larger cycle or clique? While some social networks provide the user with information about common connections they have with their connections, the general fundamental question that arises is that of finding small subgraphs in such distributed settings. This type of questions also emerges when processing huge graphs by distributed systems. For example, Hirvonen, Rybicki, Schmid, and Suomela [33] study distributed algorithms for finding large cuts in triangle-free graphs, and Pettie and Su [46] study coloring in triangle-free graphs.

This article surveys the state-of-the-art in distributed subgraph finding. We focus on synchronous settings, in which the main complexity measure is the number of communication rounds that is required in order to find a subgraph H. We will elaborate on the computational models and on the possible interpretations of what it means to *find* a subgraph, but let us start with a warm-up.

Warm-up: locality. The first simple observation is that in a synchronous network with no additional restrictions, the round complexity of finding a subgraph H by the devices in the network is $\Theta(k)$, where k is the diameter of H. The reason for this is that in a single round of communication, all nodes of the network can learn the neighbors of their neighbors,[1] and by induction, within t rounds each node can learn the topology within its t-neighborhood.

[1] This assumes that all nodes begin with knowledge of their neighbors. Removing this assumption incurs another round of communication to this argument.

To be more precise, what we get is that within $O(k)$ rounds, each node is able to *list* all the instances of H to which it belongs. This is the most powerful variant of subgraph finding.

The clear drawback of the above example is that it sweeps the under the carpet the complexity of sending potentially very large messages in a single round. Therefore, while the above classic LOCAL model [42] is suitable for studying many other tasks, it misrepresents the complexity of subgraph finding. We will thus focus mainly on two distributed settings that capture limited bandwidth, and in which the complexity of some subgraph finding problems is related. Additional settings will be discussed towards the end of the survey.

Computational models. The standard model that imposes restrictions on the bandwidth over the above setting is the CONGEST model [45]. This is a synchronous model, in which each of n nodes can send a message to each of its neighbors in every round, where the size of messages is bounded by $O(\log n)$ bits. This choice of this bound on the size of messages allows sending a node identifier within a single message. Typically, the graph that is processed is the graph that underlies the communication network.

Another important model is the CONGESTED CLIQUE model [43], which we will abbreviate as the CLIQUE model in this document, in which the n nodes are part of a fully connected network, with the same message bound of $O(\log n)$ bits as in the CONGEST model. Here, the input graph is an arbitrary graph G on n nodes, which is typically assigned through a bijection to the nodes of the CLIQUE, such that each node receives as input the edges in G that are adjacent to its assigned node.

Subgraph finding. The extreme case mentioned above, where each node in the system lists all instances of H to which it belongs is referred to as *membership listing* (or sometimes *local listing* or *local enumeration*). In settings with bounded bandwidth, this is typically a difficult variant that is known to require many rounds of computation for many subgraphs (see, e.g., discussion about triangle finding in Section 2). A weaker variant is that of *listing* or *enumeration*, in which it is required that every instance of the subgraph H is output by some node, but not necessarily a node which belongs to that instance.

Apart from these two listing variants, it is in many cases important to merely *detect* whether the input contains a copy of H or not. The standard formalization of distributed detection is that if there is no copy of H then all nodes output `false`, and otherwise at least one node outputs `true`. The reason for not demanding a unanimous output is to avoid incurring an overhead just for propagating the information of existence of H, for example if a graph contains a single copy of H and has many nodes that are very far from that instance.

Similarly to the case of listing, the detection problem also has a membership variant, which requires each node to output `true` or `false` based on whether it is a part of a copy of H or not.

Outline. Section 2 overviews the case in which H is a triangle. It covers both the CLIQUE and the CONGEST models. Sections 3 and 4 address larger cliques and larger cycles in both models, respectively. Finally, Section 5 briefly mentions additional subgraphs and additional settings.

2 Triangle Finding

A naïve simulation of the warm-up algorithm for membership listing in the CONGEST or CLIQUE models, in which all neighbors are sent to each other neighbor one by one, gives a trivial $O(\Delta)$-round algorithm for triangle membership listing, where Δ is the maximum degree in the graph. However, it is possible to do better, as we overview in this section.

2.1 Triangle Finding in the CLIQUE Model

We begin with the CLIQUE model.

Triangle listing in the CLIQUE model. The first non-trivial algorithm for triangle finding is due to Dolev, Lenzen, and Peled [19]. This is a deterministic triangle listing algorithm for the CLIQUE model, which has a complexity of $O(n^{1/3}/\log n)$ rounds. The simplicity of this algorithm turned out to be a huge advantage for later additional results, as we will see. The algorithm works as follows: The vertices of the graph are partitioned into $n^{1/3}$ subsets $S_1, \ldots, S_{n^{1/3}}$, each of $n^{2/3}$ nodes. Each of the n nodes receives a different tuple of three of these subsets. A node that receives $S_{i_1}, S_{i_2}, S_{i_3}$ for indices $1 \leq i_1, i_2, i_3 \leq n^{1/3}$ (that are not necessarily different) collects all edges with one endpoint in one of the three subsets and one endpoint in another, that is, this node collects all edges in $E(S_{i_1}, S_{i_2}) \cup E(S_{i_1}, S_{i_3}) \cup E(S_{i_2}, S_{i_3})$, and reports all triangles that it finds. It is straightforward to see that all triangles are listed by this algorithm since the number of 3-tuples of subsets is n and so each is handled by some node.

The round complexity of the algorithm follows by proving that each node needs to send and receive $O(n^{4/3})$ edges in total, which are to and from locations that are known to all nodes (we will discuss this knowledge property later), since the partition to subsets is hardcoded and so it is known to all nodes. Sending: Take a node v and assume that it is in the subset S_i. There can be at most $n^{2/3}$ edges between v and nodes in S_j and these edges need to be sent to all nodes that have S_i and S_j in their 3-tuple. Since there are $n^{1/3}$ such 3-tuples, these $n^{2/3}$ edges need to be sent to $n^{1/3}$ nodes. Repeating this for all $n^{1/3}$ possibilities for j gives a total of $n^{2/3+1/3+1/3} = n^{4/3}$ edges that v has to send. Receiving: Each node needs to learn 3 subsets of edges, each containing at most $n^{2/3} \cdot n^{2/3} = n^{4/3}$ edges. To conclude the complexity analysis, one can use the simple claim that [19] proves, which states that a routing task in which each node needs to send and receive n messages in a known pattern can be done in 2 rounds. This means that the $O(n^{4/3})$ sent and received messages per node are divided by n, yielding a complexity of $O(n^{1/3})$ rounds. Noticing that the partition and routing are fixed, one can refrain from sending actual edge identifiers and replace them with a bit mask, which saves a logarithmic factor and results in a complexity of $O(n^{1/3}/\log n)$ rounds.

This complexity turns out to be optimal. The first lower bound for this task was of $\Omega(n^{1/3}/\log^3 n)$ rounds given by Pandurangan, Robinson and Scquizzato [44], and it was followed by a tight lower bound of $\Omega(n^{1/3}/\log n)$ rounds given by Izumi and Le Gall [36]. The lower bound is proven using an information-theoretic argument, which bounds by $\Omega(n^{4/3})$ the entropy of the transcript that a certain node sees on a random graph in which each edge appears independently at random with probability 1/2. As the entropy of the transcript is a lower bound on its length, and since each node can receive at most $O(n \log n)$ bits per round, the lower bound on the round complexity follows. This approach also allowed [36] to obtain a lower bound of $\Omega(n/\log n)$ rounds for *local* listing of triangles, in which each node needs to output a list of all the triangles which include it.

In addition to the aforementioned algorithm, [19] also gives a triangle-listing algorithm whose complexity improves upon the above in the case that the graph contains *many* triangles. Specifically, they provide a randomized algorithm that completes in $O(n^{1/3}/(t^{2/3}+1)+1)$ rounds in expectation, where t is the number of triangles in the graph, and in $O(\min\{n^{1/3}\log^{2/3} n/(t^{2/3}+1), n^{1/3}\})$ rounds with high probability.

Triangle detection in the CLIQUE model. The approach of [19] inherently lists all triangles. A natural question is whether the decision problem of triangle detection is easier than listing, for which the answer turns out to be affirmative. Censor-Hillel, Kaski, Korhonen, Lenzen, Paz, and Suomela [12] show how to perform matrix multiplication over a ring in the CLIQUE model within $O(n^{0.158})$ rounds. More precisely, the complexity is $O(n^{1-2/\omega})$ rounds, where ω is the matrix multiplication exponent, currently known to be bounded by 2.3728596 due to Alman and Vassilevska-Williams [2]. Ring matrix multiplication directly carries over to triangle detection with the same complexity. The main approach of [12] is to simulate matrix multiplication algorithms for parallel settings in the CLIQUE model. This includes the so-called parallel 3D matrix multiplication, as well as bilinear Strassen-like algorithms.

Elaborating upon the matrix multiplication algorithms is beyond the scope of this survey. Yet, we must mention a crucial component that underlies these CLIQUE algorithms, as well as numerous additional CLIQUE algorithms for various tasks, which is the ingenious routing technique of Lenzen [41]. In a nutshell, this technique provides a way to route in $O(1)$ rounds any set of messages in which each node needs to send and receive at most n messages.

The above complexity of $O(n^{0.158})$ rounds for triangle detection in the CLIQUE model is the current state-of-the-art. For reasons that will be explained shortly, unlike the $\Omega(n^{1/3})$ lower bound for triangle listing in this model, there is no known lower bound for triangle detection. We thus have the following major open problem in distributed triangle finding.

▶ **Open Problem 2.1.** *What is the complexity of triangle detection in the* CLIQUE *model? In particular, is there an algorithm that is faster than $O(n^{0.158})$ rounds? Can any lower bound be proven?*

The second part of the above question which asks for a lower bound for triangle detection in the CLIQUE model is considered hard: Very roughly speaking, Drucker, Kuhn, and Oshman [20] show that the CLIQUE model is strong enough to simulate certain circuits, which implies that for a wide range of problems, any super-constant lower bound in the CLIQUE model would result in a major breakthrough in circuit complexity. It is noteworthy that the problem of triangle listing does not fall into this category of problems due to its large outputs, which explains why this does not contradict the aforementioned lower bound of $\Omega(n^{1/3})$ rounds for triangle listing.

For the broadcast version of this model, a.k.a. the BROADCAST CONGESTED CLIQUE model, in which the messages sent by a node in a certain round must all be identical, an $\Omega(n/e^{O(\sqrt{\log n})} \log n)$-round lower bound on deterministic triangle detection is given by [20], through a reduction from 3-party number-on-forehead set disjointness.

Triangle listing in sparse graphs in the CLIQUE model. The first algorithms for triangle finding in sparse graphs were given by Dolev, Lenzen, and Peled [19]. One algorithm has a round complexity of $O(\Delta^2/n + 1)$ and another has a round complexity of $O(A^2/n + \log_{2+n/A^2} n)$, where A is the arboricity of the graph.[2] The latter implies a round complexity of $O(m^2/n^3)$, in terms of the number of edges, m.

Pandurangan, Robinson, and Scquizzato [44] give a randomized triangle listing algorithm that completes within $\tilde{O}(m/n^{5/3} + 1)$ rounds, w.h.p. At the heart of the algorithm lies the partitioning approach of [19], but more is needed in order to exploit sparsity. In [44], the

[2] The arboricity of a graph is the minimal number of forests that contain all of its edges. While bounded by the maximum degree Δ, the arboricity can in some cases be much smaller, e.g., a star has a linear maximum degree but its arboricity is 1.

partition is random, and the routing of edges to the nodes to which they are assigned is done in a randomized load balanced manner. This approach is what handles load balancing of the information that needs to be routed in the system in a way which is sensitive to the sparsity of the graph, rather than optimizing only for the worst case. Moreover, the aforementioned $\tilde{\Omega}(n^{1/3})$ lower bound of [44] for triangle listing in general graphs follows from a $\tilde{\Omega}(m/n^{5/3})$ lower bound for graphs with m edges.[3]

Censor-Hillel, Leitersdorf, and Turner [14] obtained an algorithm for sparse graphs, with a complexity of $O(m/n^{5/3}+1)$ which is similar to, but slightly improves upon, the complexity of the algorithm of [44], by polylogarithmic factors. The algorithm of [14] is deterministic and is based on an algorithm for sparse matrix multiplication, which completes in $O((\rho_S\rho_T n)^{1/3}+1)$ rounds, where S and T are the input matrices and ρ_A is the average number of non-zero elements per row of a matrix A (i.e., it is the number of non-zero elements of A, divided by n).[4] While the aforementioned semi-ring matrix multiplication algorithm of [12] assigns the n^3 element-wise multiplication to the nodes in an optimal manner, it applies to the worst case. The general approach used by [14] for sparse matrix multiplication is to assign the element-wise multiplications to nodes in a way which is optimal given the input sparsity. The way this is done is by load balancing the number of non-zero elements that each node needs to send and receive in order for all element-wise multiplications to be computed.

Notes on matrix multiplication in the CLIQUE model. The first algorithm for matrix multiplication in this model is due to Drucker, Kuhn, and Oshman [20], who showed a randomized complexity of $O(n^{\omega-2}) \approx O(n^{0.373})$ rounds with high probability, for semirings. A deterministic $O(n^{1/3})$-round algorithm for matrix multiplication over a semiring is given in [12].

Censor-Hillel, Dory, Korhonen, and Leitersdorf [8] provide two additional sparse matrix multiplication algorithms, used for distance computations. Compared to [14], these algorithms also benefit from sparsity of the output matrix $P = ST$. More concretely, their first algorithm completes in $O((\rho_S\rho_T\rho_P)^{1/3}/n^{2/3}+1)$ rounds. This matches the complexity algorithm of [14] for a general P, but improves upon it for sparse output matrices (recall that the multiplication of two sparse matrices need not be sparse in general). The second algorithm of [8] pays an additive $\log n$ over the first one, i.e., has a round complexity of $O((\rho_S\rho_T\rho)^{1/3}/n^{2/3} + \log n)$, but here ρ_P is replaced by ρ, which stands for the number of elements per row that are needed. That is, there is no need to compute any element in the output matrix P that is not among the ρ smallest ones in its row (given an appropriate definition of the total order on matrix elements). In other words, the complexity of this algorithm does not depend on the sparsity of $P = ST$, but on some sparsity parameter that is given as input.

Additional algebraic algorithms in the CLIQUE model with important applications were given by Le Gall [27]. These include fast algorithms for rectangular matrix multiplications, as well as for multiple instances of matrix multiplication.

As explained earlier, we do not expect a lower bound for matrix multiplication in the CLIQUE model. However, it is shown in [12] that a near-linear number of rounds is required in the BROADCAST CONGESTED CLIQUE model.

Finally, we mention that matrix multiplication based algorithms also give results for *counting* for some small subgraphs [12, 14].

[3] In fact, the results of [44] hold for the k-machine model (see Klauck, Nanongkai, Pandurangan, and Robinson [39]), which is similar to the CLIQUE model, but has k nodes rather than n. For a general k, the complexity of the algorithm is $\tilde{O}(m/n^{5/3} + n/k^{4/3})$, and the lower bound is $\tilde{\Omega}(m/k^{5/3})$.

[4] The sparse matrix multiplication algorithm of [14] can also be converted to the k-machine model, in which it has a complexity of $O(n^{4/3}(\rho_S\rho_T)^{1/3}/k^{5/3} + 1)$ rounds.

2.2 Triangle Finding in the CONGEST Model

We now move to triangle finding problems in the CONGEST model.

The first breakthroughs in this model are the first non-trivial (sublinears) algorithms due to Izumi and Le Gall [36]. These are randomized algorithms for triangle listing and detection in $O(n^{3/4}\log n)$ and $O(n^{2/3}\log^{2/3} n)$ rounds w.h.p., respectively. The approach of these algorithms is to split the task of finding triangles into two parts: one which looks for triangles that are ϵ-heavy and another which looks for other triangles, where an ϵ-heavy triangle is one in which at least one of its edges appears in at least n^ϵ triangles. Very roughly speaking, heavy triangles can be detected within $O(n^{1-\epsilon})$ rounds by randomly sampling which edges to send, and can be listed in $O(n^{1-\epsilon/2})$ rounds by randomly hashing the edges sent to different neighbors. A more involved argument shows that non-heavy triangles can be listed in $O(n^{1-\epsilon} + n^{(1+\epsilon)/2}\log n)$ rounds, w.h.p. Carefully plugging in the right values of $n^\epsilon = \tilde{O}(n^{1/3})$ and $n^\epsilon = \tilde{O}(n^{1/2})$ then gives the round complexities for triangle detection and listing.

The breakthrough that came after [36] is due to Chang, Pettie, and Zhang [15], who showed triangle listing (and thus also detection) in the CONGEST model in $\tilde{O}(n^{1/2})$ rounds, w.h.p. In a nutshell, the main methodology of this algorithm has two elements: One element is an algorithm for decomposing the graph into well-connected components which could behave somewhat similarly to the CLIQUE model. The second element is to have the nodes of each component search for triangles that have an edge in the component.

In more detail, [15] showed how to compute the following expander decomposition in $O(n^{1/2})$ rounds. This decomposition is a partition of the set of edges of the graph into three sets. In the first set E_m, each connected component has minimum degree $n^{1/2}$ and conductance $\Omega(1/\operatorname{polylog} n)$. The graph induced by the edges in the second set, E_s, is such that its arboricity is at most $n^{1/2}$. The third set of remaining edges, E_r is at most a constant fraction of the total number of edges, and the triangle listing algorithm which we discuss in what follows recurses over this remaining set of edges. The algorithm for the decomposition itself is beyond the scope of this survey, and here we only mention that it is rather far from being merely a distributed implementation of its centralized counterpart.[5]

The triangle listing algorithm over the two first sets above then works as follows. Since the arboricity in E_s is bounded by $n^{1/2}$, triangles with at least one edge in this set are listed in a straightforward manner, using an orientation that is deduced by the decomposition algorithm. Then, triangles with an edge in any high-conductance component (i.e., in E_m) are listed by having the nodes of each component mimic a variant of the triangle listing algorithm in the CLIQUE model of Dolev, Lenzen, and Peleg [19]. This mimicking has three aspects. First, it replaces the deterministic partition of [19] with a randomized partition, in order to have a good probability for a good balance of information within the component. Second, the fact that a component has large conductance indeed implies that it has a small mixing time, but this alone is insufficient for efficiently exchanging large amounts of information within the component. To this end, the algorithm makes use of the random-walk based routing techniques of Ghaffari, Kuhn, and Su [28] and Ghaffari and Li [29], whose complexities improve as the mixing time decreases. Finally, even with these ingredients, some nodes of a component may have too many edges that touch them that are not inside the component, preventing them from efficiently using their inner-component edges for routing this large amount of information. Thus, some additional edges are added to the set E_r of remaining edges that the decomposition left for recursing over.

[5] The actual decomposition result is more general, with a parameter δ which is tuned here to be $\delta = 1/2$ in order to optimize the running time of the triangle listing algorithm that uses it.

The decomposition approach was refined by Chang and Saranurak [16], allowing the complexity of triangle listing to drop to $\tilde{O}(n^{1/3})$ rounds, w.h.p. Since the $\tilde{\Omega}(n^{1/3})$ lower bound for triangle listing in the CLIQUE model directly applies also in CONGEST, this complexity is near-optimal (up to polylog n factors). At a high level, the decomposition obtained in [16] improves upon the one in [15] in that it does not have E_s at all (the bounded arboricity part) and in the complexity of obtaining it. In addition, a somewhat modified version of the routing algorithms of [28, 29] is introduced, for the triangle listing usage of the decomposition.

Chang and Saranurak [17] then show deterministic algorithms for expander decomposition and for expander routing. These yield round complexities of $O(n^{0.58})$ and $n^{2/3} + o(1)$ for deterministic triangle detection and triangle listing, respectively.

An additional algorithm for deterministic triangle listing is given by Huang, Pettie, Zhang, and Zhang [34]. The complexity of this algorithm, given in terms of the maximum degree Δ, is $O(\Delta/\log n + \log\log\Delta)$ rounds, w.h.p. For $\Delta = \tilde{O}(n^{1/3})$, this is faster compared with the algorithm of [17], while the latter is faster for the larger range of Δ.

Since the complexities of both [17] and [34] do not reach yet the lower bound of $\tilde{\Omega}(n^{1/3})$ rounds, we note the following open question.

▶ **Open Problem 2.2.** *What is the complexity of deterministic triangle listing in the* CONGEST *model? Does randomization help for this problem?*

The above are triangle listing algorithms so they clearly also solve the detection variant. However, as opposed to triangle listing, for triangle detection we currently do not have good lower bounds. What we do know is the following. Abboud, Censor-Hillel, Khoury, and Lenzen [1] showed that triangle detection cannot be solved in the CONGEST model within a single round by a deterministic algorithm. Specifically, they showed that any single-round algorithm requires a bandwidth of $\Delta \log n$ bits. Fischer, Gonen, Kuhn, and Oshman [24] showed that this also holds for randomized algorithms, by showing that randomized single-round algorithms require a bandwidth of Δ bits. Both works also addressed the round complexity of 1-bit bandwidth algorithms, with a lower bound of $\Omega(\log^* n)$ rounds given in [1], which was improved to $\Omega(\log n)$ rounds in [24].[6]

We will later (Section 4) describe some lower bounds for finding other subgraphs in the CONGEST model which are based on reductions from 2-party communication complexity problems, and there we will see why these techniques do not give any meaningful lower bound for triangles. Moreover, in the spirit of the aforementioned argument of Drucker, Kuhn, and Oshman [20], Eden, Fiat, Fischer, Kuhn, and Oshman [21] showed that a lower bound of $\omega(\log n)$ for triangle detection in CONGEST would imply major breakthroughs in circuit complexity. This leaves us with another curious gap in our knowledge of triangle finding in this model.

▶ **Open Problem 2.3.** *What is the complexity of triangle detection in the* CONGEST *model (randomized and deterministic)?*

3 Finding Larger Cliques

The CLIQUE model. The deterministic triangle listing algorithm of [19] for the CLIQUE model can be easily generalized to list larger cliques. To find cliques of size p for some integer $p \geq 3$, the set of nodes is partitioned into $n^{1/p}$ sets. Each node is assigned a p-tuple of sets

[6] The function $\log^* n$ counts the number of times the logarithm function needs to be applied starting from n until the value drops to at most 1.

and learns all edges between any two of its sets. A similar argument to that of triangles shows that indeed all p-cliques are listed by this algorithm, and that its round complexity is $O(n^{1-2/p}/\log n)$. In fact, it is easy to see that this algorithm lists all instances of any subgraph H of p nodes.

This complexity is optimal, due to a lower bound of $\tilde{\Omega}(n^{1-2/p})$ rounds by Fischer, Gonen, Kuhn, and Oshman [24], which generalizes the aforementioned lower bound for triangle listing of [44] and [36], using additional machinery. In the BROADCAST CONGESTED CLIQUE model, Drucker, Kuhn, and Oshman [20] show that $\Omega(n/\log n)$ rounds are needed for p-cliques, even for the stronger detection variant, for almost all values of $p \geq 4$ (as long as $p \leq (1-\epsilon)n$ for some constant $\epsilon > 0$).

For sparse graphs, Censor-Hillel, Le Gall, and Leitersdorf [11] show that listing can be complete within $\tilde{O}(m/n^{1+2/p}+1)$ rounds, for $p \geq 3$. This follows from their CONGEST approach which is discussed below, and is tight up to polylogarithmic factors using the lower bound technique of [24, 36, 44].

As opposed to the listing variant, for p-clique detection the aforementioned hardness of obtaining lower bounds in the CLIQUE model by [20] kicks in, and we do not know any non-constant lower bound (or any larger than 1 lower bound, for that matter), leaving the complexity of p-clique detection open for $p > 4$.

▶ **Open Problem 3.1.** *What is the complexity of p-clique detection in the* CLIQUE *model (randomized and deterministic), for $p \geq 4$?*

The CONGEST model. Algorithms for finding larger cliques in the CONGEST model, as in the case of triangles, are also based on the conductance decomposition algorithms of Chang, Pettie, and Zhang [15] and of Chang and Saranurak [16]. The first sublinear algorithms for larger cliques in CONGEST are due to Eden, Fiat, Fischer, Kuhn, and Oshman [21]. They show that 4-cliques can be listed in $\tilde{O}(n^{5/6+o(1)})$ rounds and that 5-cliques can be listed in $\tilde{O}(n^{21/22+o(1)})$ rounds, w.h.p. These listing algorithms are more involved compared with the triangle listing algorithm, due to the need to handle, for example, a 4-clique with one edge within a certain component and another edge within a different component, which imposes a complex challenge that becomes even worse as p grows.

Following [21], the work of Censor-Hillel, Le Gall, and Leitersdorf [11] provided algorithms for listing p-cliques in $\tilde{O}(n^{p/(p+2)})$ rounds, w.h.p., for all $p \geq 4$ except for $p = 5$. For $p = 5$, this work gave an algorithm which completes in $\tilde{O}(n^{3/4+o(1)})$ rounds, w.h.p. The main approach of these algorithms is iterating over the decomposition in a way that balances the minimum degree and the arboricity thresholds. Within each cluster, sparsity-aware listing helps speeding up the computation. While these algorithms improved upon the state-of-the-art for $p = 4, 5$ and were the first sublinear algorithms for $p \geq 6$, they fell short of the [24] lower bound of $\tilde{\Omega}(n^{1-2/p})$ rounds. The question of whether clique listing in the CONGEST model is as easy as, or harder than, its CLIQUE counterpart, was recently answered by Censor-Hillel, Chang, Le Gall, and Leitersdorf [7], using the newer conductance decomposition of Chang and Saranurak [16] and additional mechanisms for optimal sparsity-aware listing and for efficient transmission of edges in the clutsers.

For $p = 4$, the tight listing of [7] also implies that 4-clique detection is not easier than its listing counterpart in the CONGEST model. This is due to a lower bound of $\tilde{\Omega}(n^{1/2})$ for 4-clique detection in CONGEST, by Czumaj and Konrad [18]. The lower bound of [18] is more general, and says that detecting p-cliques in the CONGEST model requires $\tilde{\Omega}(n^{1/2}/p)$ rounds for $p \leq n^{1/2}$, and $\tilde{\Omega}(n/p)$ rounds for $p \geq n^{1/2}$. Thus, for values of p larger than 4, there is still a gap between detection and listing.

▶ **Open Problem 3.2.** *What is the complexity of p-clique detection in the* CONGEST *model (randomized and deterministic), for $p > 4$?*

The lower bound of [18] uses a reduction from 2-party set disjointness. The same work also shows that listing can be done sufficiently fast by the two parties, which shows that if the detection problems are harder, a different lower bound technique must be used.

4 Finding Larger Cycles

The CLIQUE model. Since the algorithm of [19] easily lists any subgraph of p nodes, we have that, in particular, p-cycles can be listed in the CLIQUE model within $O(n^{1-2/p})$ rounds (deterministically).

The detection variant was then addressed by Censor-Hillel, Kaski, Korhonen, Lenzen, Paz, and Suomela [12], who show a deterministic constant-round algorithm for detecting 4-cycles. In addition, they show an $O(n^{0.158})$-round algorithm for detecting p-cycles, for any constant p. More accurately, the complexity is $2^{O(p)}n^{0.158}$ rounds. This is a randomized algorithm that relies on the color coding technique of Alon, Yuster, and Zwick [3], in order to avoid too much congestion that would be caused by collecting all possible candidates for cycle nodes. More recently, Censor-Hillel, Fischer, Gonen, Le Gall, Leitersdorf, and Oshman [10] showed that $2p$-cycles can be detected in $O(1)$ rounds for any p, using a technique which also allows fast girth approximation. This is, notably, a deterministic algorithm. For odd values of p, it is still not known whether there is a constant-round p-cycle detection algorithm.

▶ **Open Problem 4.1.** *What is the complexity of p-cycle detection in the* CLIQUE *model (randomized and deterministic), for odd values of p?*

The CONGEST model. For 4-cycle detection, Drucker, Kuhn, and Oshman [20] provide a tight complexity of $\tilde{\Theta}(n^{1/2})$, by providing both an algorithm and a lower bound. They also show that for any odd value of p, detecting p-cycles requires $\tilde{\Omega}(n)$ rounds. The upper bound is obtained by setting a threshold of $T = 2n^{1/2}$, and defining heavy nodes as nodes with at least T neighbors. First, all non-heavy nodes send all their neighbors to all their neighbors, in T rounds. This detects 4-cycles which have at least 2 non-neighboring non-heavy nodes. Then, a heavy node v with more than T heavy neighbors reports that there exists a 4-cycle. Finally, a heavy node v with at most T heavy neighbors sends its heavy neighbors to all of its neighbors, again in T rounds. The reason for which too many heavy neighbors imply a 4-cycle is because this means that the total number of neighbors of heavy neighbors is at least T^2, which is more than $2n$, and so there must be a node other than v which is connected to two of the neighbors of v, and hence we have a 4-cycle. Otherwise, if v sends its heavy neighbors to all of its neighbors, then any 4-cycle with two neighboring nodes that are heavy is detected by one of its nodes (by a simple case analysis).

The matching lower bound for 4-cycles is a good point of reference for understanding lower bounds for the CONGEST model that rely on reductions from 2-party communication complexity. It relies on the fact that there is a 4-cycle-free graph G on n nodes with $\Theta(n^{3/2})$ edges, due to Erdös [22], whose work implies that this is the Turán Number of 4-cycles [48]. The setting is as follows. Each of two players, Alice and Bob, has an input string of k bits, $x = (x_1, \ldots, x_k)$ and $y = (y_1, \ldots, y_k)$, respectively. By communicating with each other, the players need to compute the set disjointness function $Disj(x, y)$, whose value is 1 if and only if the input strings represent disjoint subsets of the set $\{1, \ldots, k\}$, that is, if and only if there is no index $1 \leq i \leq k$ such that $x_i = y_i = 1$. The 2-party set disjointness problem is known

to require exchanging $\Omega(k)$ bits, even by randomized protocols, due to Kalyanasundaram and Schnitger [38], Razborov [47], and Bar-Yossef, Jayram, Kumar, and Sivakumar [4]. The reduction to distributed detection of 4-cycles is as follows. Each of the two players takes a subgraph of the 4-cycle free graph G according to their respective input. That is, the edges of G are mapped to the set $\{1, \ldots, k\}$, with a value of n for which $k = \Theta(n^{3/2})$. Each player imagines a subgraph of G that has an edge for any index in which their input is 1. The players then imagine that their two subgraphs are connect by a perfect matching: each node in Alice's subgraph is connected to the respective node in Bob's subgraph (the nodes of both subgraphs are the nodes of G). Now, it is easy to see that the combined graph contains a 4-cycle if and only if the input strings x, y are disjoint, as the only 4-cycles that can occur are those that have two edges from the perfect matching, and two edges that represent the same index in the bit strings of the players. Thus, if Alice and Bob can simulate a distributed algorithm for 4-cycle detection, they solve set disjointness. They simulate a given algorithm as follows. Any message that the algorithm sends between two nodes that are in the same subgraph (either that of Alice of that of Bob) can be internally simulated by the respective player. The only messages that need to be explicitly sent between the two players are those that are sent along the edges of the perfect matching. There are n such edges, and so simulating a round of the detection algorithms costs $O(n \log(n))$ bits of communication between the two players. Since $\Omega(k) = \Omega(n^{3/2})$ is a lower bound on the total number of bits that need to be exchanged for solving set disjointness, we get that the distributed 4-cycle detection algorithm must consist of at least $\Omega(k/n \log(n)) = \Omega(n^{3/2}/n \log(n)) = \tilde{\Omega}(n^{1/2})$ rounds.

A remark on triangle detection. While this approach for reductions which imply CONGEST lower bounds is heavily used in the literature, it is doomed to fail for triangle detection. The reason is that any triangle in a graph has at least one player which knows at least two of its nodes and thus all of its edges, which nullifies any attempt for a lower bound constructions, regardless of the 2-party problem or the graph construction.

Korhonen and Rybicki [40] then showed that the $\tilde{\Omega}(n^{1/2})$ lower bound for 4-cycles can be extended and applies to p-cycles for all even values of p. Notably, in the spirit of the results of [20], it is shown in Censor-Hillel, Fischer, Gonen, Oshman, Le Gall, and Leitersdorf [10] that going above this lower bound for $p = 6$ would imply new lower bounds in circuit complexity, which are considered hard to obtain. The reason that this barrier for lower bounds works in the CONGEST model is because it is shown that it is sufficient to consider high-conductance clusters, and that these can simulate circuits in a similar manner to the CLIQUE model.

On the upper bound side, Korhonen and Rybicki [40] showed that p-cycle detection for any constant p can be done in a linear in n number of rounds. For odd values of p, this is optimal due to the above lower bound. They also showed that this can be done faster for degenerate graphs. Fischer, Gonen, Kuhn, and Oshman [24] showed how to detect $2p$-cycles within $O(n^{1-1/p(p-1)})$ rounds, for $p \geq 2$. This was subsequently improved by Eden, Fiat, Fischer, Kuhn, and Oshman [21], who showed how to detect $2p$-cycles in $\tilde{O}_p(n^{1-2/(p^2-p+2)})$ rounds for odd $p \geq 3$, and in $\tilde{O}_p(n^{1-2/(p^2-2p+4)})$ rounds for even $p \geq 4$. They also show that as opposed to the case of 4-cliques, the listing variant of 4-cycles is harder than its detection counterpart, requiring $\tilde{\Omega}(n)$ rounds. For $p = 3, 4, 5$, Censor-Hillel, Fischer, Gonen, Oshman, Le Gall, and Leitersdorf [10] then showed improved algorithms (randomized) for detecting $2p$-cycles, which completes within $\tilde{O}(n^{1-1/p})$ rounds. Still, the upper bounds here are above the respective $\tilde{\Omega}(n^{1/2})$ lower bound.

▶ **Open Problem 4.2.** *What is the complexity of p-cycle detection in the* CONGEST *model (randomized and deterministic), for even value of $p \geq 6$?*

5 Additional Variants of Distributed Subgraph Finding

Finding Other Subgraphs. The literature has also been studying additional subgraphs, apart from cliques and cycles.

Drucker, Kuhn, and Oshman [20] study the complexity of finding trees and complete bipartite subgraphs in the BROADCAST CONGESTED CLIQUE model. Korhonen and Rybicki [40] show that trees can be detected in $O(1)$ rounds in the BROADCAST CONGEST model.

On the lower bound front, Gonen and Oshman [31] showed lower bounds for finding subgraphs that are created from smaller ones using certain allowed operations. Fischer, Gonen, Kuhn, and Oshman [24] showed that for any $p \geq 4$, there exists a subgraph H of size p such that H detection requires $\tilde{\Omega}(n^{2-\Theta(1/p)})$ rounds. Eden, Fiat, Fischer, Kuhn, and Oshman [21] showed that this complexity is roughly the right one, by providing an upper bound of $\tilde{O}(n^{2-2/(3p+1)} + o(1))$ rounds. In particular, their result shows that there does not exist a constant-sized subgraph for which detection requires a truly quadratic number of rounds.

Subgraph Freeness Testing. The variant of testing for H-freeness has also been studied in the distributed setting, initially introduced by Brakerski and Patt-Shamir [6].

The definition of distributed testing for H-freeness requires that if there is no instance of H in the graph then all the nodes output `true`, but instead of requiring at least one node to output `false` in case there is an instance of H as would be an analog to the detection problem, testing only requires at least one node to output `false` in case the graph is far from being H-free. Here, being ϵ-far from having a property is the same as its standard definition by Goldreich, Goldwasser, and Ron [30], and means that no matter which ϵ-fraction of the graph is changed (removed, in the case of H-freeness), the resulting graph must still have a copy of H. For triangles and other small subgraphs, this typically means that there are many instances of H in a graph that is far from being H-free.

For testing triangle-freeness, Censor-Hillel, Fischer, Schwartzman, and Vasudev [9] showed a $O(1/\epsilon^2)$-round algorithm. This was later improved by Even, Fischer, Fraigniaud, Gonen, Levi, Medina, Montealegre, Olivetti, Oshman, Rapaport, and Todinca [23] and Fraigniaud and Olivetti [25] to a complexity of $O(1/\epsilon)$ rounds. These, along with the work of Fraigniaud, Rapaport, Salo, and Todinca [26] also show fast testing for larger cliques, cycles, and additional subgraphs.

Subgraph Finding in Dynamic Networks. Subgraph finding has also been studied from the perspective of dynamic networks. Bonne and Censor-Hillel [5] characterize the bandwidth that is required for various clique finding problems by algorithms that work in a dynamic setting and must produce the correct answer immediately at the end of the round in which a topology change occurs.

Subgraph finding in a harsh dynamic setting in which the number of topology changes per round is unlimited, was studied by Censor-Hillel, Kolobov, and Schwartzman [13], who showed upper and lower bounds for small subgraphs.

Subgraph Finding in Quantum Networks. Izumi, Le Gall, and Magniez [37] have shown that in a quantum CONGEST model, triangle detection can be solved within $\tilde{O}(n^{1/4})$ rounds, using a distributed version developed by Izumi and Le Gall in [35] of a *Grover Search* [32]. We note that the listing variant for triangles cannot be improved using quantum tools, since it is obtained through information theoretic arguments, which apply to this setting as well.

References

1 Amir Abboud, Keren Censor-Hillel, Seri Khoury, and Christoph Lenzen. Fooling views: a new lower bound technique for distributed computations under congestion. *Distributed Computing*, 33:545—559, 2020. doi:10.1007/s00446-020-00373-4.

2 Josh Alman and Virginia Vassilevska Williams. A refined laser method and faster matrix multiplication. In *Proceedings of the 32nd Annual ACM-SIAM Symposium on Discrete Algorithms, (SODA)*, 2021.

3 Noga Alon, Raphael Yuster, and Uri Zwick. Color-coding. *J. ACM*, 42(4):844–856, 1995. doi:10.1145/210332.210337.

4 Ziv Bar-Yossef, T. S. Jayram, Ravi Kumar, and D. Sivakumar. An information statistics approach to data stream and communication complexity. *J. Comput. Syst. Sci.*, 68(4):702–732, 2004. doi:10.1016/j.jcss.2003.11.006.

5 Matthias Bonne and Keren Censor-Hillel. Distributed detection of cliques in dynamic networks. In *Proceedings of the 46th International Colloquium on Automata, Languages, and Programming, (ICALP)*, pages 132:1–132:15, 2019. doi:10.4230/LIPIcs.ICALP.2019.132.

6 Zvika Brakerski and Boaz Patt-Shamir. Distributed discovery of large near-cliques. *Distributed Comput.*, 24(2):79–89, 2011. doi:10.1007/s00446-011-0132-x.

7 Keren Censor-Hillel, Yi-Jun Chang, François Le Gall, and Dean Leitersdorf. Tight distributed listing of cliques. In *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, 2021.

8 Keren Censor-Hillel, Michal Dory, Janne H. Korhonen, and Dean Leitersdorf. Fast approximate shortest paths in the congested clique. *Distributed Computing*, 2020. doi:10.1007/s00446-020-00380-5.

9 Keren Censor-Hillel, Eldar Fischer, Gregory Schwartzman, and Yadu Vasudev. Fast distributed algorithms for testing graph properties. *Distributed Computing*, 32(1):41–57, 2019. doi:10.1007/s00446-018-0324-8.

10 Keren Censor-Hillel, Orr Fischer, Tzlil Gonen, François Le Gall, Dean Leitersdorf, and Rotem Oshman. Fast distributed algorithms for girth, cycles and small subgraphs. In *Proceedings of the 34th International Symposium on Distributed Computing (DISC)*, pages 33:1–33:17, 2020. doi:10.4230/LIPIcs.DISC.2020.33.

11 Keren Censor-Hillel, François Le Gall, and Dean Leitersdorf. On distributed listing of cliques. In *Proceedings of the ACM Symposium on Principles of Distributed Computing (PODC)*, pages 474–482, 2020. doi:10.1145/3382734.3405742.

12 Keren Censor-Hillel, Petteri Kaski, Janne H. Korhonen, Christoph Lenzen, Ami Paz, and Jukka Suomela. Algebraic methods in the congested clique. *Distributed Computing*, 32(6):461–478, 2019. doi:10.1007/s00446-016-0270-2.

13 Keren Censor-Hillel, Victor I. Kolobov, and Gregory Schwartzman. Finding subgraphs in highly dynamic networks. *CoRR*, abs/2009.08208, 2020, To appear in SPAA. arXiv:2009.08208.

14 Keren Censor-Hillel, Dean Leitersdorf, and Elia Turner. Sparse matrix multiplication and triangle listing in the congested clique model. *Theor. Comput. Sci.*, 809:45–60, 2020. doi:10.1016/j.tcs.2019.11.006.

15 Yi-Jun Chang, Seth Pettie, and Hengjie Zhang. Distributed triangle detection via expander decomposition. In *Proceedings of the 30th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 821–840, 2019. doi:10.5555/3310435.3310486.

16 Yi-Jun Chang and Thatchaphol Saranurak. Improved distributed expander decomposition and nearly optimal triangle enumeration. In *Proceedings of the ACM Symposium on Principles of Distributed Computing (PODC)*, pages 66–73, 2019. doi:10.1145/3293611.3331618.

17 Yi-Jun Chang and Thatchaphol Saranurak. Deterministic distributed expander decomposition and routing with applications in distributed derandomization. In *Proceedings of the 61st IEEE Annual Symposium on Foundations of Computer Science (FOCS)*, pages 377–388, 2020. doi:10.1109/FOCS46700.2020.00043.

18 Artur Czumaj and Christian Konrad. Detecting cliques in CONGEST networks. In *Proceedings of the 32nd International Symposium on Distributed Computing (DISC)*, pages 16:1–16:15, 2018. doi:10.4230/LIPIcs.DISC.2018.16.

19 Danny Dolev, Christoph Lenzen, and Shir Peled. "tri, tri again": Finding triangles and small subgraphs in a distributed setting - (extended abstract). In *Proceedings of the 26th International Symposium on Distributed Computing (DISC)*, pages 195–209, 2012. doi:10.1007/978-3-642-33651-5_14.

20 Andrew Drucker, Fabian Kuhn, and Rotem Oshman. On the power of the congested clique model. In *Proceedings of the ACM Symposium on Principles of Distributed Computing (PODC)*, pages 367–376, 2014. doi:10.1145/2611462.2611493.

21 Talya Eden, Nimrod Fiat, Orr Fischer, Fabian Kuhn, and Rotem Oshman. Sublinear-time distributed algorithms for detecting small cliques and even cycles. In *Proceedings of the 33rd International Symposium on Distributed Computing (DISC)*, pages 15:1–15:16, 2019. doi:10.4230/LIPIcs.DISC.2019.15.

22 Paul Erdös. On sequences of integers no one of which divides the product of two others and on some related problems. *Inst. Math. Mech. Univ. Tomsk*, 2:74–82, 1938.

23 Guy Even, Orr Fischer, Pierre Fraigniaud, Tzlil Gonen, Reut Levi, Moti Medina, Pedro Montealegre, Dennis Olivetti, Rotem Oshman, Ivan Rapaport, and Ioan Todinca. Three notes on distributed property testing. In *Proceedings of the 31st International Symposium on Distributed Computing (DISC)*, pages 15:1–15:30, 2017. doi:10.4230/LIPIcs.DISC.2017.15.

24 Orr Fischer, Tzlil Gonen, Fabian Kuhn, and Rotem Oshman. Possibilities and impossibilities for distributed subgraph detection. In *Proceedings of the 30th Symposium on Parallelism in Algorithms and Architectures (SPAA)*, pages 153–162, 2018. doi:10.1145/3210377.3210401.

25 Pierre Fraigniaud and Dennis Olivetti. Distributed detection of cycles. In Christian Scheideler and Mohammad Taghi Hajiaghayi, editors, *Proceedings of the 29th ACM Symposium on Parallelism in Algorithms and Architectures, SPAA 2017, Washington DC, USA, July 24-26, 2017*, pages 153–162. ACM, 2017. doi:10.1145/3087556.3087571.

26 Pierre Fraigniaud, Ivan Rapaport, Ville Salo, and Ioan Todinca. Distributed testing of excluded subgraphs. In Cyril Gavoille and David Ilcinkas, editors, *Distributed Computing - 30th International Symposium, DISC 2016, Paris, France, September 27-29, 2016. Proceedings*, volume 9888 of *Lecture Notes in Computer Science*, pages 342–356. Springer, 2016. doi:10.1007/978-3-662-53426-7_25.

27 François Le Gall. Further algebraic algorithms in the congested clique model and applications to graph-theoretic problems. In *Proceedings of the 30th International Symposium on Distributed Computing (DISC)*, pages 57–70, 2016. doi:10.1007/978-3-662-53426-7_5.

28 Mohsen Ghaffari, Fabian Kuhn, and Hsin-Hao Su. Distributed MST and routing in almost mixing time. In *Proceedings of the ACM Symposium on Principles of Distributed Computing (PODC)*, pages 131–140, 2017. doi:10.1145/3087801.3087827.

29 Mohsen Ghaffari and Jason Li. New distributed algorithms in almost mixing time via transformations from parallel algorithms. In *Proceedings of the 32nd International Symposium on Distributed Computing (DISC)*, pages 31:1–31:16, 2018. doi:10.4230/LIPIcs.DISC.2018.31.

30 Oded Goldreich, Shafi Goldwasser, and Dana Ron. Property testing and its connection to learning and approximation. *J. ACM*, 45(4):653–750, 1998. doi:10.1145/285055.285060.

31 Tzlil Gonen and Rotem Oshman. Lower bounds for subgraph detection in the CONGEST model. In *Proceedings of the 21st International Conference on Principles of Distributed Systems (OPODIS)*, pages 6:1–6:16, 2017. doi:10.4230/LIPIcs.OPODIS.2017.6.

32 Lov K. Grover. A fast quantum mechanical algorithm for database search. In *Proceedings of the Twenty-Eighth Annual ACM Symposium on the Theory of Computing (STOC 1996)*, pages 212–219, 1996. doi:10.1145/237814.237866.

33 Juho Hirvonen, Joel Rybicki, Stefan Schmid, and Jukka Suomela. Large cuts with local algorithms on triangle-free graphs. *Electron. J. Comb.*, 24(4):P4.21, 2017. URL: http://www.combinatorics.org/ojs/index.php/eljc/article/view/v24i4p21.

34 Dawei Huang, Seth Pettie, Yixiang Zhang, and Zhijun Zhang. The communication complexity of set intersection and multiple equality testing. In *Proceedings of the 2020 ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1715–1732, 2020. doi:10.1137/1.9781611975994.105.

35 Taisuke Izumi and François Le Gall. Quantum distributed algorithm for the all-pairs shortest path problem in the CONGEST-CLIQUE model. In *Proceedings of the 2019 ACM Symposium on Principles of Distributed Computing (PODC 2019)*, pages 84–93, 2019. doi:10.1145/3293611.3331628.

36 Taisuke Izumi and François Le Gall. Triangle finding and listing in CONGEST networks. In *Proceedings of the ACM Symposium on Principles of Distributed Computing (PODC)*, pages 381–389, 2017. doi:10.1145/3087801.3087811.

37 Taisuke Izumi, François Le Gall, and Frédéric Magniez. Quantum distributed algorithm for triangle finding in the CONGEST model. In *Proceedings of the 37th International Symposium on Theoretical Aspects of Computer Science (STACS)*, pages 23:1–23:13, 2020. doi:10.4230/LIPIcs.STACS.2019.49.

38 Bala Kalyanasundaram and Georg Schnitger. The probabilistic communication complexity of set intersection. *SIAM J. Discret. Math.*, 5(4):545–557, 1992. doi:10.1137/0405044.

39 Hartmut Klauck, Danupon Nanongkai, Gopal Pandurangan, and Peter Robinson. Distributed computation of large-scale graph problems. In *Proceedings of the Twenty-Sixth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 391–410, 2015. doi:10.1137/1.9781611973730.28.

40 Janne H. Korhonen and Joel Rybicki. Deterministic subgraph detection in broadcast CONGEST. In James Aspnes, Alysson Bessani, Pascal Felber, and João Leitão, editors, *21st International Conference on Principles of Distributed Systems, OPODIS 2017, Lisbon, Portugal, December 18-20, 2017*, volume 95 of *LIPIcs*, pages 4:1–4:16. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2017. doi:10.4230/LIPIcs.OPODIS.2017.4.

41 Christoph Lenzen. Optimal deterministic routing and sorting on the congested clique. In *Proceedings of the ACM Symposium on Principles of Distributed Computing (PODC)*, pages 42–50, 2013. doi:10.1145/2484239.2501983.

42 Nathan Linial. Locality in distributed graph algorithms. *SIAM J. Comput.*, 21(1):193–201, 1992. doi:10.1137/0221015.

43 Zvi Lotker, Boaz Patt-Shamir, Elan Pavlov, and David Peleg. Minimum-weight spanning tree construction in $O(\log \log n)$ communication rounds. *SIAM J. Comput.*, 35(1):120–131, 2005. doi:10.1137/S0097539704441848.

44 Gopal Pandurangan, Peter Robinson, and Michele Scquizzato. On the distributed complexity of large-scale graph computations. In *Proceedings of the 30th on Symposium on Parallelism in Algorithms and Architectures (SPAA)*, pages 405–414, 2018.

45 David Peleg. *Distributed Computing: A Locality-Sensitive Approach*. Society for Industrial and Applied Mathematics, USA, 2000.

46 Seth Pettie and Hsin-Hao Su. Fast distributed coloring algorithms for triangle-free graphs. In Fedor V. Fomin, Rusins Freivalds, Marta Z. Kwiatkowska, and David Peleg, editors, *Automata, Languages, and Programming - 40th International Colloquium, ICALP 2013, Riga, Latvia, July 8-12, 2013, Proceedings, Part II*, volume 7966 of *Lecture Notes in Computer Science*, pages 681–693. Springer, 2013. doi:10.1007/978-3-642-39212-2_59.

47 Alexander A. Razborov. On the distributional complexity of disjontness. In Mike Paterson, editor, *Automata, Languages and Programming, 17th International Colloquium, ICALP90, Warwick University, England, UK, July 16-20, 1990, Proceedings*, volume 443 of *Lecture Notes in Computer Science*, pages 249–253. Springer, 1990. doi:10.1007/BFb0032036.

48 Paul Turán. On an extremal problem in graph theory. *Mat. Fiz. Lapok*, 48:436–452, 1941.

Error Resilient Space Partitioning

Orr Dunkelman
Computer Science Department,
University of Haifa, Israel

Zeev Geyzel
Mobileye, an Intel company,
Jerusalem, Israel

Chaya Keller
Department of Computer Science,
Ariel University, Israel

Nathan Keller
Department of Mathematics,
Bar Ilan University, Ramat Gan, Israel

Eyal Ronen
School of Computer Science,
Tel Aviv University, Israel

Adi Shamir
Department of Computer Science,
Weizmann Institute of Science, Rehovot, Israel

Ran J. Tessler
Department of Mathematics,
Weizmann Institute of Science, Rehovot, Israel

Abstract

In this paper we consider a new type of space partitioning which bridges the gap between continuous and discrete spaces in an *error resilient* way. It is motivated by the problem of rounding noisy measurements from some continuous space such as $\mathbb{R}^d$ to a discrete subset of representative values, in which each tile in the partition is defined as the preimage of one of the output points. Standard rounding schemes seem to be inherently discontinuous across tile boundaries, but in this paper we show how to make it perfectly consistent (with *error resilience* ϵ) by guaranteeing that any pair of consecutive measurements X_1 and X_2 whose L_2 distance is bounded by ϵ will be rounded to the same nearby representative point in the discrete output space. We achieve this resilience by allowing a few bits of information about the first measurement X_1 to be unidirectionally communicated to and used by the rounding process of the second measurement X_2. Minimizing this revealed information can be particularly important in privacy-sensitive applications such as COVID-19 contact tracing, in which we want to find out all the cases in which two persons were at roughly the same place at roughly the same time, by comparing cryptographically hashed versions of their itineraries in an error resilient way.

The main problem we study in this paper is characterizing the achievable tradeoffs between the amount of information provided and the error resilience for various dimensions. We analyze the problem by considering the possible colored tilings of the space with k available colors, and use the color of the tile in which X_1 resides as the side information. We obtain our upper and lower bounds with a variety of techniques including isoperimetric inequalities, the Brunn-Minkowski theorem, sphere packing bounds, Sperner's lemma, and Čech cohomology. In particular, we show that when $X_i \in \mathbb{R}^d$, communicating $\log_2(d+1)$ bits of information is both sufficient and necessary (in the worst case) to achieve positive resilience, and when d=3 we obtain a tight upper and lower asymptotic bound of $(0.561\ldots)k^{1/3}$ on the achievable error resilience when we provide $\log_2(k)$ bits of information about X_1's color.

2012 ACM Subject Classification Theory of computation → Randomness, geometry and discrete structures; Theory of computation → Computational geometry; Theory of computation → Error-correcting codes

Keywords and phrases space partition, high-dimensional rounding, error resilience, sphere packing, Sperner's lemma, Brunn-Minkowski theorem, Čech cohomology

Digital Object Identifier 10.4230/LIPIcs.ICALP.2021.4

Category Invited Talk

Related Version *Full Version*: `https://arxiv.org/abs/2008.03675`

Acknowledgements We thank Stephen D. Miller for inspiring discussions on sphere packing.

1 Introduction

Studying various types of space partitioning of a continuous space such as $\mathbb{R}^d$ is a central topic in computational geometry (see, e.g., [11, Chapters 6,12] and the references therein), and each type of partition has different properties and applications within computer science, electrical engineering, and applied mathematics. For example, in *error-correcting codes* (which are extensively used in data communication) we try to squeeze the largest possible number of equal sized disjoint balls into the input space, while in *vector quantization* [16] (which is used extensively in data compression) we try to completely cover the input space with a small number of tiles whose volumes are as similar as possible. In this paper we investigate a new variant which can be viewed as "continuous error correction over the reals". Our main motivation is the problem of rounding noisy analog measurements in $\mathbb{R}^d$, in order to digitally process or store them (see, e.g., [2, 10]). This rounding process seems to be inherently discontinuous across tile boundaries, and this problem is compounded by the fact that for large d almost all input vectors are near boundaries. One natural solution to this discontinuity problem is to try to minimize the fraction of pairs X_1, X_2 with $distance(X_1, X_2) < \epsilon$ which are rounded differently by considering *foam tilings* that minimize the total surface area of unit volume tiles (such a tiling is called "foam" since it emerges in physical collections of soap bubbles). In a beautiful FOCS paper [22] (which was highlighted at CACM [23]), Kindler et al. introduced a clever new construction of such tiles which they called *spherical cubes*. What makes these tiles special is that they have the $O(\sqrt{d})$ surface area of a ball and yet they can tile the whole $\mathbb{R}^d$ space without gaps, which solved an open problem posed by Lord Kelvin in 1887.

In this paper we consider the more ambitious goal of achieving *error resilience* which completely eliminates all the discontinuities in the rounding process rather than reducing their fraction. We call such a rounding scheme *consistent rounding*[1], and make it possible by thinking about X_1 and X_2 as two consecutive noisy measurements of the same X. When the first measurement X_1 is rounded, we allow it to produce a few bits of side information about how it was rounded, and to provide them as an auxiliary input to the process that decides how to round X_2. Note that both X_1 and X_2 are assumed to be real valued vectors which require an infinite number of bits to fully specify them.

To demonstrate the basic idea, consider the one dimensional case in which X_1 and X_2 are real values which have to be consistently rounded to the same nearby integer whenever they are close enough. X_1 is always rounded to the nearest integer, and it produces a single bit of side information which is whether it was rounded to an even or an odd integer P. When X_2 is measured, it is rounded to the nearest integer which has the same parity as P. To demonstrate this process, consider the problematic inputs $X_1 = 0.4999$ and $X_2 = 0.5001$: X_1 is rounded to 0, and X_2 is also rounded to 0 since it is the closest even integer. In fact, X_2 could be anywhere between -1 and 1 and still be consistently rounded to 0, and thus the rounding scheme is resilient to additive errors of up to 0.5. In fact (see Sec. 4.1), this is the highest possible error resilience of any one dimensional consistent rounding scheme; other natural schemes (such as providing one bit of side information about whether X_1 was rounded up or down) provide lower resilience.

The way we think about the problem is to consider a colored tiling of the real line with two colors: All the values in $[-0.5, 0.5)$, $[1.5, 2.5)$, etc. are colored by 1, and all the values in $[0.5, 1.5)$, $[2.5, 3.5)$, etc. are colored by 2. The side information provided about X_1 is the

[1] We note that in statistics, the term "consistent rounding" is used to denote a rounding that is consistent with some external constraints; see [26, p. 237].

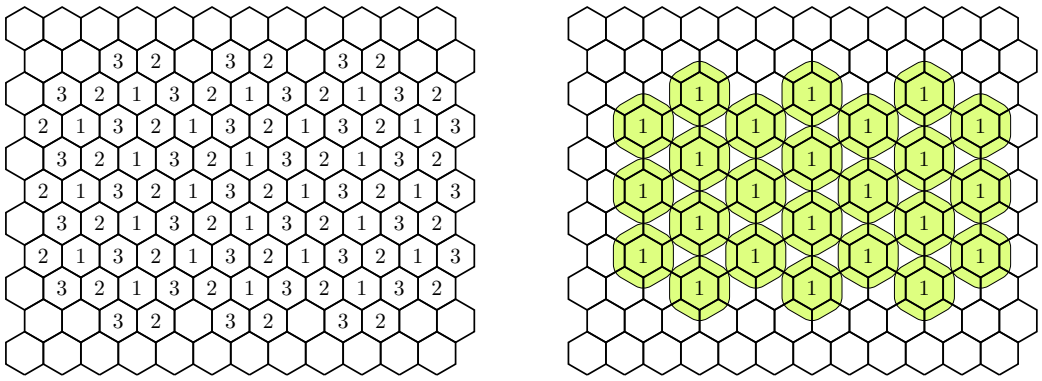


Figure 1 A 3-colored hexagonal tiling of the plane, and a maximal non-intersecting inflation of the tiles colored 1.

color of the tile in which it is located, and the way we process X_2 is to round it to the center of the closest tile which has the same color as that of X_1. The essential property of our partition is that the minimum distance between any two tiles with the same color is 1, and thus we can "inflate" all the tiles of any particular color to their outer parallel body in order to include any erroneously measured value X_2 up to a distance of 0.5 away from the original tile, and still get nonoverlapping tiles which make it possible to uniquely associate such points with original tiles.

To make this perspective clearer, consider the two dimensional plane. In the obvious checkerboard tiling by unit squares, we need at least 4 colors (and thus 2 bits of side information) to color the tiles if we do not allow equi-colored tiles to touch. We can reduce the number of colors to 3 (and thus, provide only $\log_2(3) = 1.58$ bits of side information) by considering the hexagonal partition of the plane depicted in the left part of Figure 1. Given a two dimensional point X_1, we always round it to the center of the hexagon in which it is located, and given X_2 we round it to the center of the nearest hexagon which has X_1's color. To determine the error resilience of this scheme, we inflate all the hexagonal tiles of a particular color by the same amount until they touch each other, as depicted in the right part of Figure 1. As it turns out, this natural scheme is not optimal since the inflated hexagons' corners touch prematurely, leaving large gaps between them. A 3-colored tiling with a higher error resilience will be described in Section 5.1.1, and an asymptotically optimal tiling for a large number of colors will be described in Section 5.2.1.

For inputs $X \in \mathbb{R}^d$, we can provide one bit of side information about each one of its d entries separately, but for large d this is very inefficient. In our colored tiling formulation, it suffices to reveal the color of X_1 in order to consistently round X_2, and thus if we can tile the space with k colors, we need only $\log_2(k)$ bits to specify this color. This naturally leads to the question of what is the minimum number of colors needed to tile $\mathbb{R}^d$ by bounded sized tiles so that no two tiles of the same color will touch (even at a corner). Surprisingly, we could not find any reference to this natural question. As we show in Section 3, there can be no such colored tiling with d colors, and as we show in Section 5.3, $d+1$ colors are sufficient. Consequently, $\log_2(d+1)$ bits of side information about X_1 are necessary and sufficient (in the worst case) to obtain a consistent rounding scheme with positive error resilience. To prove the negative result, we use techniques borrowed from algebraic topology (namely, either a generalization of Sperner's lemma or the Čech cohomology and other cohomology theories), and to prove the positive result we provide an explicit construction of such a colored tiling.

Table 1 Summary of our lower and upper bounds on the error resilience, for different d and k.

Scenario	Lower Bound (LB) on ER	Upper Bound (UB) on ER	Techniques	Source
3 colors in $\mathbb{R}^2$	0.354	0.413	Brunn-Minkowski ineq. (UB), Brick wall tiling (LB)	Sec. 4.1 (UB), Sec. 5.1.1 (LB)
4 colors in $\mathbb{R}^2$	0.5	0.564	Brunn-Minkowski ineq. (UB), Brick wall tiling (LB)	Sec. 4.1 (UB), Sec. 5.1.1 (LB)
k colors in $\mathbb{R}^2$	$0.537\sqrt{k} - O(1)$	$0.537\sqrt{k}$	Circle packing (UB), HCR tiling (LB)	Sec. 4.2 (UB), Sec. 5.2.1 (LB)
4 colors in $\mathbb{R}^3$	0.25	0.365	Brunn-Minkowski ineq. (UB), 3-dim Brick wall (LB)	Sec. 4.1 (UB), Sec. 5.1.2 (LB)
k colors in $\mathbb{R}^3$	$(0.561 - o(1))k^{1/3}$	$0.561k^{1/3}$	Sphere packing (UB), CPB tiling (LB)	Sec. 4.2 (UB), Sec. 5.2.2 (LB)
k colors in $\mathbb{R}^8$	$(0.707 - o(1))k^{1/8}$	$0.707k^{1/8}$	Sphere packing (UB), CPB tiling (LB)	Sec. 4.2 (UB), Sec. 5.2.2 (LB)
k colors in $\mathbb{R}^{24}$	$(1 - o(1))k^{1/24}$	$k^{1/24}$	Sphere packing (UB), CPB tiling (LB)	Sec. 4.2 (UB), Sec. 5.2.2 (LB)
$d+1$ colors in $\mathbb{R}^d$	$\Omega(1/d)$	$O(\log d/\sqrt{d})$	Brunn-Minkowski ineq. (UB), Dimension reducing tiling (LB)	Sec. 4.1 (UB), Sec. 5.3 (LB)

ER – error resilience, LB – lower bound, UB – upper bound,
HCR – honeycomb of rectangles, CPB – close packing of boxes

In addition to minimizing the amount of side information, we study the question of maximizing the error resilience for a given d and k. In the negative direction, in Section 4 we obtain several upper bounds on the achievable error resilience, using different techniques from geometry and analysis (including isoperimetry, the Brunn-Minkowski inequality and results on the sphere packing problem). In the positive direction, we construct in Section 5 a variety of tiling schemes. In particular, while for $d = 2$ and $k = 3$ the hexagonal tiling scheme described above is resilient to additive errors of up to 0.31, we present a tiling with resilience of 0.354, and show that no 3-color tiling can have resilience higher than 0.413. We also show that the maximal resilience achieved by a $(d+1)$-coloring of $\mathbb{R}^d$ is between $\Omega(1/d)$ and $O(\log d/\sqrt{d})$, and use the recent breakthrough results on sphere packing [6, 18, 25] to obtain tight asymptotic lower and upper bounds on the resilience in dimensions $2, 3, 8$, and 24. Our bounds are summarized in Table 1.

Applications. The problem of "continuous error correction over the reals" has numerous applications. For example, in biometric identification, multiple measurements of the same fingerprint are similar but not identical. It would be very helpful if all these slight variants could be represented by the same rounded point P, and we can achieve this by storing a small amount of side information in the biometric database during the initial registration of a new employee.

Another example is the problem of developing a contact tracing app for the COVID-19 pandemic, where we want to record all the cases in which two smart phones were at roughly the same place at roughly the same time. We can do this by measuring in each phone the GPS location, the local time, and perhaps other parameters such as the ambient noise level (in order to rule out the case of people living in different apartments which are separated by a common wall). When someone tests positive for COVID-19, the health authority wants

to reveal a list of his measurements, but in order to keep the patient's privacy, it wants to cryptographically hash each measurement before publishing it. Since the measurements are likely to be slightly different for the infected and exposed persons, the health authority can publish the small amount of side information along with the consistently rounded and then hashed measurements.

Finally, the problem may also be relevant to the construction of quantum error correction codes, since the state of a quantum computer is a complex-valued vector in a Hilbert space with exponentially many dimensions which can be perturbed by external noise. Note that the logarithmic number of side information bits we need for error correction can be stored and processed classically. However, describing such potential applications is beyond the scope of this paper.

Related work. A line of study that seems related to our work is *fuzzy* constructions that were widely studied in the cryptographic literature, such as the *fuzzy commitment scheme* of Juels and Wattenberg [20]). Dodis et al. [12] introduced the notions of *fuzzy extractors* and *secure sketches*, which enable two parties to secretly reach a consensus value from multiple noisy measurements of some high entropy source (a recent survey of such techniques can be found in [14]). However, such schemes concentrate on the aspects of cryptographic security (which we do not consider), and produce sketches whose size depends on the number of possible inputs (which is meaningless for real valued inputs). In this sense our consistent rounding scheme can be viewed as an exceptionally efficient reconciliation process, since it can produce for each million entry vector of arbitrarily large real numbers a 20 bit "sketch" (in the form of its color side information), and process this information with trivial point location algorithms.

Open problems. While we fully solved the question of minimizing the amount of side information required for error resilience, several questions remain open regarding the maximal resilience rate that can be achieved for a given amount of side information. In particular, for dimensions $2, 3, 8, 24$ we determined the exact asymptotic resilience when $\log_2 k$ bits of information are allowed, using a connection to the *densest sphere packing* problem. When only very few bits of information are allowed, the situation is much less clear. For example, we do not even know whether the brick wall constructions we present in Section 5.1.1 have the highest error resilience among rounding schemes in $\mathbb{R}^2$ with $\log_2 3$ and $\log_2 4$ bits of side information. It will be interesting to obtain new upper bounds via different techniques or new lower bound constructions.

2 Our Setting

In this section we present the basic setting that will be assumed throughout the paper.

Colored tiling. We study tilings of $\mathbb{R}^d$, where each tile is *connected*, *closed* and *bounded*, and the tiles intersect only in their boundaries. In some of the results we make additional assumptions on the tiles or drop some of the assumptions; such changes are stated explicitly. Each tile is colored in one of k colors.

Error resilience and inflation. In order to compute the error resilience of a given tiling (with respect to the L_2 distance), we consider all tiles of the same color and inflate them (i.e., replace the tile T by the set $T' = \{y : \exists x \in T, |x - y| < r\}$ for some $r > 0$) until they touch each other. Clearly, the error resilience is the maximal r for which such a non-intersecting

inflation is possible. We note that in convex geometry, such an inflation T' is called *the outer parallel body of radius r* of T (see [15, p. 943]). The minimal distance between two same-colored points in different tiles is denoted by t, and so, the error resilience is $t/2$.

Breaking ties. A fine point about consistent rounding schemes is how to break ties, and here we deal differently with X_1 and X_2. We want to be able to deal with any X_1, and thus we think about the tiles as being closed sets which include their boundaries. Therefore, points X_1 which are on the boundary between tiles can have more than one possible color. We allow such ties to be broken arbitrarily in the sense that X_1 can be rounded to the center of any one of the tiles that it belong to. However, when we think about X_2 we allow it to be at a distance of strictly less than some bound, and thus the inflated tiles (that contain all the possible X_2's we are interested in) are open sets which have no intersections. Consequently, each X_2 can belong to at most one inflated tile, and is rounded to the center of that tile with no possible ties.

Normalization. The d-dimensional volume of a figure $T \subset \mathbb{R}^d$ is denoted by $\lambda(T)$. We normalize the tiling by assuming that *the volume of each tile is bounded by* 1 (like in the 1-dimensional case presented in the introduction, where all tiles are segments of length 1). We make the natural assumption that any inflated tile T' satisfies $\lambda(\bar{T}') = \lambda(T')$, where $\bar{T}'$ is the topological closure of T'. Normalization with respect to other natural metrics, as well as alternative distance metrics, are discussed in the full version of the paper.

3 The Minimal Number of Colors Required for Error Resilience

In this section we prove that the minimal number of colors required for achieving any positive error resilience in a tiling of $\mathbb{R}^d$ is $d+1$. We provide two proofs, under different additional natural assumptions on the tiles. The first assumes that all tiles are uniformly bounded and relies on a generalization of Sperner's lemma. The second proof assumes that the tiles and their non-empty intersections are contractible (while not having to be uniformly bounded) and uses a more advanced algebraic-topologic argument. The lower bound $d+1$ on the number of required colors is tight; a matching construction for any d is presented in Section 5.3.

3.1 Lower bound for bounded tiles, using Sperner's Lemma

The main result of this subsection is the following.

▶ **Proposition 1.** *For any $m > 0$, the following holds. Let $T_1, T_2, \ldots$ be a colored tiling of $\mathbb{R}^d$ in $d+1$ colors, in which each tile is contained in a box with side length m. If the error resilience of the tiling is $\delta > 0$, then there exist tiles in all $d+1$ colors that intersect at a point.*
Consequently, any tiling with positive error resilience uses at least $d+1$ colors.

We use a generalization of the classical Sperner's lemma [24], called *Bapat's connector-free lemma*. As Bapat's lemma was originally proved only in $\mathbb{R}^2$ and in a discrete setting, we first provide a proof of a continuous version in $\mathbb{R}^d$, and then derive Proposition 1 from it.

3.1.1 Bapat's connector-free lemma – continuous version

Let Δ be a d-simplex in a Euclidean space, i.e., the convex hull of $d+1$ points $x_0, \ldots, x_d$ which do not lie in a d-space. The i'th face of Δ is the span of $\{x_j\}_{j \neq i}$. A *connector* in a d-simplex is a connected set which intersects all its $(d-1)$-dimensional faces.

Bapat's connector-free lemma asserts the following:

▶ **Theorem 2.** *Let $C_0, \ldots, C_d$ be a cover of a d-simplex Δ by closed sets such that the minimal distance between two connected components of the same set is $\delta > 0$. Suppose that the interiors of the sets are disjoint and that no C_i contains a connector. Then $\bigcap_{i=0}^{d} C_i \neq \emptyset$.*

In order to prove the theorem we will reduce it to an analogous discrete claim. The reduction is simple, but requires some more terminology.

A *triangulation* T of a simplex $\Delta \subset \mathbb{R}^d$ is a cover of it by simplices whose interiors are disjoint, such that the intersection of any set of simplices is either empty or the convex hull of some vertices. Note that the vertices of Δ are, in particular, vertices of the triangulation, and that the faces of Δ are endowed by an induced triangulation. The *diameter* of T is the supremum of distances between vertices which share an edge. The 1-*skeleton* of T is the graph formed by the vertices and the edges. A *discrete connector* is a connected subset of the 1-skeleton of T which contains vertices from each facet of Δ. We can now state the discrete version of Theorem 2 (which is the actual statement proved by Bapat, for $d = 2$).

▶ **Theorem 3** (Bapat). *Suppose that the vertices of T are partitioned into disjoint sets $A_0, A_1, \ldots, A_d$ such that no A_i contains a discrete connector. Then there exists a simplex in T whose $d+1$ vertices belong to different sets A_i.*

Proof of Theorem 2, assuming Theorem 3. Assume towards contradiction that there exist sets $C_0, \ldots, C_d$ which cover Δ, such that no C_i contains a connector, but $\bigcap_{i=0}^{d} C_i = \emptyset$. Define the function $f : C_0 \times C_1 \cdots \times C_d \to \mathbb{R}_+$ by setting $f(p_0, \ldots, p_d)$ to be the diameter of the convex hull of $(p_0, \ldots, p_d)$, which is the maximal distance between two p_i's. The domain of the function f compact (here we use the assumption that the minimal distance between two connected components of the same C_i is at least δ) and its range is $\mathbb{R}_+$ (since we assumed $\bigcap_{i=0}^{d} C_i = \emptyset$). Hence, f attains a minimum $\epsilon > 0$.

Let $\eta = \min(\delta, \epsilon)/3$. Let $\tilde{C}_i$ be the η-thickening of C_i in Δ, i.e., $\tilde{C}_i = \{p \in \Delta : \operatorname{dist}(p, C_i) < \eta\}$. Then $\tilde{C}_i$ is an open cover of Δ. By the choice of η, neither $\tilde{C}_i$ contains a connector,[2] and $\bigcap_{i \in \{0,\ldots,d\}} \tilde{C}_i = \emptyset$.

Let T be a triangulation of Δ whose diameter is less than η and all whose vertices lie in the interiors of the sets C_i. (Clearly, such a triangulation exists.) Define A_i as the subset of vertices which lie in $\operatorname{int}(C_i)$. These sets are well defined since the interiors of the different C_i's are disjoint. Since the triangulation is of diameter less than η, each edge between two vertices which belong to the same A_i lies in $\tilde{C}_i$. Indeed, its endpoints are in C_i and any point on the edge is of distance less than η to any endpoint, hence it is in $\tilde{C}_i$. Thus, since $\tilde{C}_i$ contains no connector, A_i contains no discrete connector.

We can now apply Theorem 3 to deduce that there exists a simplex $t = \{v_0, \ldots, v_d\} \in T$ such that $\forall i : v_i \in A_i$. But since $v_i \in C_i$ for all i, this implies $f(v_0, \ldots, v_d) < \eta < \epsilon$, a contradiction to the definition of ϵ. This completes the proof. ◀

[2] To be precise, this relies on the slightly stronger assumption that each connected component of C_i is at least δ-far from one of the facets of Δ. While this extra assumption can be avoided in the proof, it clearly holds in our setting so we make it for simplicity.

To prove Theorem 3, we use the classical Sperner's lemma [24]. To present it, a few more definitions are due.

A *(d+1)-labelling* of a triangulation T is of the simplex $\Delta = \text{conv}(e_0, \ldots, e_d)$ is a function $\ell : V(T) \to \{0, 1, \ldots, d\}$, that is, an assignment of one of $d+1$ colors to each vertex of the triangulation. A $(d+1)$-labelling ℓ is called *proper* if $\ell(e_i) = i$, and for each $v \in T$ that belongs to a lower-dimensional face $\text{conv}(e_{i_1}, \ldots, e_{i_r})$, we have $\ell(v) \in \{i_1, \ldots, i_r\}$.

▶ **Theorem 4** (Sperner's lemma)**.** *For any triangulation T of Δ, any proper labelling of T contains a simplex all whose vertices have different labels.*

Proof of Theorem 3. We define, using the sets $A_0, \ldots, A_d$, a proper labelling ℓ of T. For any j and any $v \in A_j$, $\ell(v)$ is defined as the minimal $i \in \{0, \ldots, d\}$ such that the connected component of v in A_j does not intersect the i'th face of Δ. Note that ℓ is well-defined, since each v belongs to a single A_j and no A_j contains a connector. Clearly, $\ell(v) \neq i$, whenever v belongs to the i'th face of Δ. Furthermore, this implies that if $v \in \text{conv}(e_{i_1}, \ldots, e_{i_r})$, then $\ell(v) \in \{i_1, \ldots, i_r\}$ (as all other colors are forbidden). Hence, ℓ is proper.

By Sperner's lemma, applied to the labelling ℓ, there exists a simplex $\{v_0, \ldots, v_d\} \in T$ all whose vertices have different labels. Assume w.l.o.g. that $\ell(v_i) = i$. We want to show that each v_i belongs to a different A_j, which will complete the proof. Assume towards contradiction $v_i, v_k \in A_j$ for $i \neq k$. On the one hand, $\ell(v_i) = i \neq k = \ell(v_k)$. On the other hand, $v_i,\ v_k$ belong to the same connected component in A_j, hence, by the definition of ℓ they must map to the same value. A contradiction, and Theorem 3 follows. ◀

3.1.2 Proof of Proposition 1

We are now ready to prove Proposition 1.

Let $T_1, T_2, \ldots$ be a tiling of $\mathbb{R}^d$ that satisfies the assumptions of the proposition. Consider the restriction of the tiling to a large simplex Δ (say, of side length $100m$).

For $i = 0, \ldots, d$, denote by $C_i \subset \Delta$ the union of all tiles colored i, restricted to Δ. Clearly, C_i is a closed set and the distance between any two connected components of C_i is at least 2δ. (Indeed, each tile is connected, and as the error resilience of the tiling is δ, the distance betwen two same-colored tiles is at least 2δ).

We claim that no C_i contains a connector. Indeed, a connector cannot include points from different tiles. A single tile is included in a box with side length m, and thus, cannot touch all facets of a simplex with side length $100m$. Hence, there is no single-colored connector.

Therefore, we can apply Theorem 2 to deduce that $\bigcap_{i=0}^{d} C_i \neq \emptyset$, which is exactly the assertion of Proposition 1.

3.2 Lower bound for contractible tiles, using Čech cohomology

Recall that a set in $\mathbb{R}^d$ is called *contractible* if it can be continuously shrunk to a point within the set. (The formal definition is that the identity is homotopic to a constant map.)

Informally, in this section we prove that if the tiles and their non-empty intersections are finite unions of contractible sets (that do not have to be uniformly bounded), then at least $d+1$ colors are required for error resilience. We note that the proof uses a somewhat heavier algebraic-topologic machinery, and so, a reader might prefer to skip it in first reading.
Due to the possibility of pathologies, the formal statement is a bit more cumbersome:

▶ **Proposition 5.** *Let $T_1, T_2, \ldots$ be a colored tiling of $\mathbb{R}^d$ with positive error resilience, in which the tiles and all their non-empty intersections are disjoint unions of finitely many closed contractible sets. Assume that the tiling is locally finite (meaning that the number*

of tiles that intersect any bounded ball $B(0,r)$ is finite) and that all T_i's are bounded (not necessarily uniformly). In addition, assume that each T_i has an open neighborhood U_i such that for any I,

$$\bigcap_{i\in I} U_i \neq \emptyset \Leftrightarrow \bigcap_{i\in I} T_i \neq \emptyset,$$

and the U_i's and their non-empty intersections are disjoint unions of finitely many contractibles. Then the number of colors is at least $d+1$.

A similar method proves an analogous statement for colored tilings of the sphere $\mathbb{S}^d$ (i.e., the unit sphere in $\mathbb{R}^{d+1}$):

▶ **Proposition 6.** *Let $T_1, T_2, \ldots, T_N$ be a colored tiling of $\mathbb{S}^d$ with positive error resilience, in which the tiles and all their non-empty intersections are disjoint unions of finitely many closed contractible sets. Assume that each T_i has an open neighborhood U_i such that for any set of indices I,*

$$\bigcap_{i\in I} U_i \neq \emptyset \Leftrightarrow \bigcap_{i\in I} T_i \neq \emptyset,$$

and the U_i's and their non-empty intersections are disjoint unions of finitely many contractibles. Then the number of colors is at least $d+1$.

▶ Remark 7. We stress that for most natural tilings the additional assumption on the existence of the neighborhoods U_i follows from the existence of T_i's with the corresponding properties. However, there are topological pathologies in which this is not the case.

The proof of Propositions 5 and 6 uses the notion of Čech cohomology and classical results regarding its properties. For the ease of reading, we begin with an intuitive explanation of the proof ideas, and then present the formal proof.

Intuitive proof. The d'th (singular) cohomology group is a topological invariant of a manifold which roughly counts "non trivial holes" of dimension d. A classical result asserts that the d'th cohomology group of a d-dimensional compact oriented manifold like $\mathbb{S}^d$ is $\mathbb{R}$. (This corresponds to the intuitive understanding that $\mathbb{S}^d$ has one d-dimensional hole.) The de-Rham cohomology and the Čech cohomology are analytic and algebro-geometric/combinatorial invariants, that in many cases agree with their topological cousin. In particular, the d'th de-Rham and Čech cohomologies of $\mathbb{S}^d$ are equal to $\mathbb{R}$ as well.

The d'th Čech cohomology with respect to an open cover of the manifold depends on properties of intersections of $d+1$ sets in that cover. In general, it depends on the sets which form the cover, however, it is known that if these sets and their non-empty intersections are finite disjoint unions of contractibles, then the cohomology groups remain the same, independently of the cover. In particular, if the d'th Čech cohomology with respect to such a cover is non trivial, then there must be $d+1$ sets with a non-empty intersection.

Hence, for our cover $U_1, U_2, \ldots$, we know that its d'th Čech cohomology is $\mathbb{R}$. This readily completes the proof of the proposition for $\mathbb{S}^d$, as this implies that there must be a point that belongs to at least $d+1$ of the U_i's. The proof in $\mathbb{R}^d$ works in essentially the same way, with cohomology groups replaced by cohomology groups with compact support.

Formal proof. For the proof we recall the notion of Čech cohomology with values in the constant sheaf $\underline{\mathbb{R}}$, and describe the slightly less standard concept of Čech cohomology with compact support.

Definitions. Let S be either $\mathbb{R}^d$ or a compact manifold such as $\mathbb{S}^d$. Let $\mathcal{U} = \{U_1, U_2, \ldots\}$ be an open cover of S. If S is compact, we assume the collection to be finite. If S is $\mathbb{R}^d$, we assume it to be locally finite and assume in addition that each U_i is bounded.

- A $q-$*simplex* $\sigma = (U_{i_0}, \ldots, U_{i_q})$ of $\mathcal{U}$ is an ordered collection of $q+1$ different sets chosen from $\mathcal{U}$, such that
$$\bigcap_{k=0}^{q} U_{i_k} \neq \emptyset.$$
- For a $q-$simplex $\sigma = (U_{i_k})_{k\in\{0,\ldots,q\}}$, the j'th *partial boundary* is the $(q-1)$-simplex
$$\partial_j \sigma := (U_{i_k})_{k\in\{0,\ldots,q\}\setminus\{j\}},$$
obtained by removing the j'th set from σ.
- A $q-$*cochain* of $\mathcal{U}$ is a function which associates to any $q-$simplex a real number. The $q-$cochains form a vector space denoted by $C^q(\mathcal{U}, \underline{\mathbb{R}})$, with operations
$$(\lambda f + \mu g)(\sigma) = \lambda f(\sigma) + \mu g(\sigma), \quad \text{where} \quad \lambda, \mu \in \mathbb{R},\ f, g \in C^q(\mathcal{U}, \underline{\mathbb{R}}),\ \sigma \text{ is a } q-\text{simplex}.$$
Similarly, we define $C^q_c(\mathcal{U}, \underline{\mathbb{R}})$, as the vector space of $q-$cochains *with compact support*, meaning those cochains which assign 0 to all $q-$simplices, except for finitely many.
- There is a *differential map* $\delta_q : C^q(\mathcal{U}, \underline{\mathbb{R}}) \to C^{q+1}(\mathcal{U}, \underline{\mathbb{R}})$ whose application to $f \in C^q(\mathcal{U}, \underline{\mathbb{R}})$ is the $(q+1)-$cochain $\delta_q(f)$ whose value at a $(q+1)-$simplex σ is
$$(\delta_q f)(\sigma) = \sum_{j=0}^{q+1} (-1)^j f(\partial_j \sigma).$$
The restriction of δ_q to $C^q_c(\mathcal{U}, \underline{\mathbb{R}})$ maps it to $C^{q+1}_c(\mathcal{U}, \underline{\mathbb{R}})$.
- It can be easily seen that $\delta_{q+1} \circ \delta_q = 0$.
- The *q'th Čech cohomology group (with compact support)* of S with respect to the cover $\mathcal{U}$ and values in $\underline{\mathbb{R}}$ is
$$\check{H}^q(\mathcal{U}, \underline{\mathbb{R}}) := \mathrm{Ker}(\delta_q)/\mathrm{Image}(\delta_{q-1}),$$
$$\check{H}^q_c(\mathcal{U}, \underline{\mathbb{R}}) := \mathrm{Ker}(\delta_q|_{C^q_c(\mathcal{U},\underline{\mathbb{R}})})/\mathrm{Image}(\delta_{q-1}|_{C^{q-1}_c(\mathcal{U},\underline{\mathbb{R}})}).$$
- A cover (by open sets) is *good* if all its sets as well as their multiple intersections are either empty or contractible. It is *almost good* if all non empty intersections are unions of finitely many disjoint contractible components.

Classical results we use. The first result we use is the following:

▶ **Theorem 8.** *If S is a compact smooth orientable manifold (such as $\mathbb{S}^d$), and $\mathcal{U}$ is a good or an almost good finite cover, then*
$$\check{H}^i(\mathcal{U}, \underline{\mathbb{R}}) \simeq H^i_{dR}(S),$$
where the right hand side is the standard de-Rham cohomology group.
Similarly, if $S = \mathbb{R}^d$ and $\mathcal{U}$ is a locally finite good or almost good cover whose sets are bounded, then
$$\check{H}^i_c(\mathcal{U}, \underline{\mathbb{R}}) \simeq H^i_{dR,c}(S),$$
where the right hand side is the i'th de-Rham cohomology group with compact support.

For further reading about de-Rham cohomology, with or without compact support, we refer the reader to [4, Sec. 1]. For further reading about the Čech cohomology, we refer to [4, Sec. 8]. In particular, Theorem 8, for the compact case and good covers is Theorem 8.9 there. The passage to almost good covers is straightforward: In the paragraph which precedes the proof, it is explained that the obstructions to the isomorphism between Čech and de-Rham cohomologies are given by products of the i'th de-Rham cohomology groups, for $i \geq 1$, of the different intersections $\bigcap_{k=0}^{q} U_{i_k}$. Since those intersections are disjoint unions of contractibles, their higher cohomology groups vanish, hence there is no obstruction to the isomorphism.

Regarding the case $S = \mathbb{R}^d$, the proof in [4, Sec. 8] requires a few small changes: In the statement of Proposition 8.5 there, one needs to replace the de-Rham complex of the manifold with the de-Rham complex with compact support, and the direct product with direct sum. The maps r, δ which appear there will still be well defined by our local finiteness assumption on the cover, and the assumption that U_i's are bounded. The proof requires no change. Then, the double complex in the definition of Proposition 8.8 should also be defined using direct sum rather than direct product, but again there is no change in the proof. Given these changes in definitions, the proof of Theorem 8.9 (also for the almost good case) is unchanged.

The second standard result, which is a consequence of Poincaré duality, is the following:

▶ **Theorem 9.** *For a compact smooth oriented manifold S of dimension d (such as $\mathbb{S}^d$),*

$$H^d_{dR}(S) \simeq \mathbb{R}.$$

Similarly, for $S = \mathbb{R}^d$, we have $H^d_{dR,c}(\mathbb{R}^d) \simeq \mathbb{R}$.

See, for example, [4, Sec. 7] for the compact case, and [4, Sec. 4] for $\mathbb{R}^d$.

Theorems 8 and 9 yield:

▶ **Corollary 10.** *If S is a compact smooth orientable manifold (such as $\mathbb{S}^d$), and $\mathcal{U}$ is a good or an almost good finite cover, then*

$$\check{H}^d(\mathcal{U}, \underline{\mathbb{R}}) = \mathbb{R}.$$

Similarly, if $S = \mathbb{R}^d$ and $\mathcal{U}$ is a locally finite good or almost good cover whose sets are bounded, then $\check{H}^d_c(\mathcal{U}, \underline{\mathbb{R}}) = \mathbb{R}$.

Proof of Propositions 5 and 6. We show that there must exist $d+1$ T_i's whose intersection is non-empty. This clearly implies that for achieving any positive error resilience, at least $d+1$ colors are needed.

Assume on the contrary that any $(d+1)$-intersection of the T_i's is empty. Let U_i be as in the statement of the Propositions. Then by definition, they form an almost good cover. All intersections of at least $d+1$ U_i's are empty by our assumptions. Therefore, there are no $d-$simplices, and so $C^d(\mathcal{U}, \underline{\mathbb{R}}) = 0$. Thus, in the compact case, $\check{H}^d(\mathcal{U}, \underline{\mathbb{R}}) = 0$. But on the other hand, by Corollary 10,

$$\check{H}^d(\mathcal{U}, \underline{\mathbb{R}}) \simeq \mathbb{R},$$

a contradiction. For $\mathbb{R}^d$ the same argument works, with $\check{H}^d_c(\mathcal{U}, \underline{\mathbb{R}})$ in place of $\check{H}^d(\mathcal{U}, \underline{\mathbb{R}})$.

4 Upper Bounds on the Error Resilience

In this section we consider tilings of $\mathbb{R}^d$ by tiles $T_1, T_2, \ldots$ of volume at most 1. Each point in $\mathbb{R}^d$ is colored in one of $k \geq d+1$ colors, and our goal is to maximize the minimal distance t between two points of the same color that belong to different tiles. (The maximum is taken over all possible tilings that satisfy the mild regularity conditions stated in Section 2 and over all possible colorings.) Clearly, the error resilience of a rounding scheme based on such a colored tiling is $t/2$.

We present two upper bounds on t, using the Brunn-Minkowski inequality and results on sphere packing. Another bound, using the Minkowski-Steiner formula, is presented in the full version of the paper (see [13]).

The basic idea behind our upper bound proofs is as follows. Assume we have a colored tiling of $\mathbb{R}^d$, with minimal distance t. Pick a single color – say, black – and consider all black tiles inside a large cube S. We obtain a new collection of tiles $T'_1, T'_2, \ldots, T'_m$ that covers part of S. The assumption that the minimal distance between two same-colored points in different tiles is t implies that if we inflate each black tile T'_i into its open parallel outer body of radius $t/2$,

$$T''_i = \{x : \exists y \in T'_i, |x-y| < t/2\}, \tag{1}$$

then the inflations T''_i are pairwise disjoint. Hence, the sum of their volumes essentially cannot exceed the volume of the large cube, and this allows bounding t from above.

4.1 An upper bound using the Brunn-Minkowski inequality

The inflations T''_i can be represented in terms of the *Minkowski sum* of sets in $\mathbb{R}^d$.

▶ **Definition 11.** *For $A, B \subset \mathbb{R}^d$, the Minkowski sum of A, B is $A+B = \{a+b : a \in A, b \in B\}$.*

In terms of this definition, we have

$$T''_i = T'_i + B(0, t/2), \tag{2}$$

where $B(0, t/2)$ is an open ball of radius $t/2$ around the origin. This allows us to lower bound the volume of each T''_i, using the classical Brunn-Minkowski (BM) inequality (see, e.g., [5]). Recall the inequality asserts the following.

▶ **Theorem 12** (Brunn-Minkowski)**.** *Let A, B be compact sets in $\mathbb{R}^d$. Then*

$$\lambda(A+B)^{1/d} \geq \lambda(A)^{1/d} + \lambda(B)^{1/d},$$

where $\lambda(X)$ is the volume of X (formally, the d-dimensional Lebesgue measure of X).

▶ **Proposition 13.** *Let $T_1, T_2, \ldots$ be a k-colored tiling of $\mathbb{R}^d$, with tiles of volume ≤ 1 and minimal distance t. Then*

$$t \leq \left(\frac{2\Gamma(d/2+1)^{1/d}}{\sqrt{\pi}} \right) \cdot (k^{1/d} - 1),$$

where $\Gamma(\cdot)$ is the Gamma function.

Proof. Consider a cube S such that $\lambda(S) = n^d$ (for some "large" n). By the pigeonhole principle, there exists a color (say, black) that covers at least n^d/k of the volume of S. Look at the black tiles whose intersection with S is non-empty, and denote their intersections with S by $T'_1, T'_2, \ldots, T'_m$. Hence, we have m "black" subsets of S, each of volume at most 1, whose total volume is at least n^d/k.

For each T'_i, define $T''_i = T'_i + B(0, t/2)$. By assumption, the regions T''_i are disjoint. Furthermore, they are included in $S + B(0, t/2)$ whose volume is less than $(n+t)^d$. Hence,

$$\sum_i \lambda(T''_i) \leq (n+t)^d. \tag{3}$$

By the Brunn-Minkowski inequality, we have

$$\forall i : \lambda(T''_i)^{1/d} \geq \lambda(T'_i)^{1/d} + (b_{t/2})^{1/d},$$

where $b_{t/2}$ is the volume of the d-dimensional ball $B(0, t/2)$. Thus,

$$\forall i : \lambda(T''_i) \geq \sum_{j=0}^{d} \binom{d}{j} \lambda(T'_i)^{j/d} (b_{t/2})^{1-j/d}.$$

Summing over i and using (3), we get

$$(n+t)^d \geq \sum_{i=1}^{m} \sum_{j=0}^{d} \binom{d}{j} \lambda(T'_i)^{j/d} (b_{t/2})^{1-j/d}. \tag{4}$$

As $0 \leq \lambda(T'_i) \leq 1$, for any $0 \leq j \leq d$ we have $\sum_i \lambda(T'_i)^{j/d} \geq \sum_i \lambda(T'_i) \geq n^d/k$, and hence we obtain

$$(n+t)^d \geq \frac{n^d}{k} \cdot \left(1 + b_{t/2}^{1/d}\right)^d.$$

This implies

$$(1 + \frac{t}{n}) k^{1/d} - 1 \geq b_{t/2}^{1/d} = \frac{\pi^{1/2}}{\Gamma(d/2+1)^{1/d}} \cdot \frac{t}{2}.$$

Letting $n \to \infty$ and rearranging, we obtain

$$t \leq \left(\frac{2\Gamma(d/2+1)^{1/d}}{\sqrt{\pi}} \right) \cdot (k^{1/d} - 1),$$

as asserted. ◀

Asymptotic upper bound. For a large number $k \gg d$ of colors, Proposition 13 gives the upper bound

$$t \leq (\sqrt{\frac{2}{\pi e}} + o_d(1)) \sqrt{d} k^{1/d},$$

as follows from (5). This bound is not far from being tight. Indeed, its dependence on k is correct, as it can be easily matched by a periodic cubic tiling, in which each tile is a cube with side length 1 and the basic unit is a large cube with side length $k^{1/d}$ that contains each color in exactly one tile (in the same order). Moreover, even regarding the "coefficient" of $k^{1/d}$, the optimal asymptotic upper bounds for $d = 3, 8, 24$ which we obtain below via the sphere packing problem, improve over this bound by only a small factor.

Upper bound for $k = d + 1$ colors. To estimate the upper bound we obtain in this case, note that

$$(d+1)^{1/d} - 1 = (1 + o_d(1))\frac{\ln(d)}{d}, \qquad \text{and} \qquad \Gamma(d/2+1)^{1/d} = \left(\frac{1}{\sqrt{2e}} + o_d(1)\right)\sqrt{d}. \quad (5)$$

Therefore, the bound we obtain in this case is

$$t \leq \left(\sqrt{\frac{2}{\pi e}} + o_d(1)\right)\frac{\ln d}{\sqrt{d}},$$

which implies that the error resilience decreases to zero as d tends to infinity. For comparison, the lower bound we obtain in Section 5.3 is $t \geq \Omega(1/d)$.

Upper bounds for small values of d, k. For $d = 3, k = 4$, the bound is

$$t \leq \frac{2\Gamma(2.5)^{1/3}}{\sqrt{\pi}} \cdot (4^{1/3} - 1) \approx 0.729.$$

For $d = 2$ and $k = 3, 4$, the upper bounds we obtain are $t \leq 0.826$ and $t \leq 1.128$, respectively. For comparison, the best constructions we have in these settings satisfy $t = 0.5$, $t = 1/\sqrt{2}$, and $t = 1$, respectively.

Discussion. The upper bound given by Proposition 13 is loose in two ways. One source of loss is the application of the Brunn-Minkowski inequality. Here, the inequality is tight if the tiles are *balls*, and the farther they are from balls, the larger is the loss. Another source of loss is the space left between the inflations, that is not taken into account in the proof.

Interestingly, there is a dichotomy between these two sources of loss. As follows from the sphere packing problem, when the tiles are balls (and so, there is no loss in the BM inequality), the space between the inflations (and so, the loss of the second type) is relatively large. The space between the inflations can be made smaller if the tiles are taken to be polytopes with a few vertices. However, this comes at the expense of increased loss in the BM inequality, as is demonstrated in the full version of the paper.

Optimality of our 1-dimensional rounding scheme. The argument described above gives an easy proof of the optimality of the 1-dimensional rounding scheme presented in the introduction. Indeed, consider a 2-colored tiling of the line and look at the segment $I = [-n, n]$ for some large n. By the pigeonhole principle, we may assume that black tiles cover at least half of I. By the 1-dimensional Brunn-Minkowski inequality, for each black tile $T'_i \subset I$ and the corresponding inflation $T''_i = T'_i + (-t/2, t/2)$, we have $\lambda(T''_i) \geq \lambda(T'_i) + t$. As $\forall i : \lambda(T'_i) \leq 1$, there are at least n tiles. Since the T''_i's are pairwise disjoint and included in $[-n-1, n+1]$, we obtain

$$2n + 2 \geq \sum_i \lambda(T''_i) \geq \sum_i \lambda(T'_i) + \sum_i t \geq n + nt,$$

and thus, $t \leq (n+2)/n$. By letting n tend to infinity, we obtain $t \leq 1$, implying that the error resilience of any two-colored tiling of the line is at most $1/2$.

4.2 An upper bound using the Sphere Packing problem

Our second upper bound uses reduction to the classical *sphere packing* problem, which asks for the maximal possible *density* of a set of non-intersecting congruent spheres in $\mathbb{R}^d$.

▶ **Definition 14.** *The density of a sphere packing (i.e., collection of pairwise disjoint congruent spheres) $P = \cup P_i$ in $\mathbb{R}^d$ is*

$$\limsup_{r\to\infty} \frac{\lambda\left(B(0,r) \cap \bigcup P\right)}{\lambda(B(0,r))}.$$

Intuitively, this measures the fraction of the volume of a large ball covered by the packing.

▶ **Notation 15.** *Denote the maximal density of a sphere packing in $\mathbb{R}^d$ by δ_d, and the volume of the unit ball $B(0,1) \subset \mathbb{R}^d$ by $v_d = \pi^{d/2}/\Gamma(d/2+1)$.*

▶ **Proposition 16.** *Let $T_1, T_2, \ldots$ be a tiling of $\mathbb{R}^d$ in k colors, with tiles of volume ≤ 1 and minimal distance t. Then*

$$t \leq \left(2(\delta_d/v_d)^{1/d}\right) \cdot k^{1/d} = \left(\frac{2\Gamma(d/2+1)^{1/d} \cdot \delta_d^{1/d}}{\sqrt{\pi}}\right) \cdot k^{1/d}.$$

Note that the asymptotic upper bound of Proposition 16 is stronger than the asymptotic upper bound that follows from Proposition 13 by the constant factor $(\delta_d)^{1/d}$. For small values of k, the upper bound given by Proposition 13 is stronger.

Proof. Let T be a k-colored tiling of $\mathbb{R}^d$ that satisfies the assumptions of the proposition, and consider the sequence of balls $\{B(0,n)\}_{n=1,2,3,\ldots}$. By the pigeonhole principle, there exists a color (say, black) such that for each n_ℓ in an infinite subsequence $\{n_\ell\}_{\ell=1,2,\ldots}$, the intersection of the black tiles with the ball $B(0,n_\ell)$ has volume of at least

$$\lambda(B(0,n_\ell))/k = \frac{n_\ell^d \cdot v_d}{k}.$$

As the volume of each tile is at most 1, we know that for each n_ℓ, the number of black tiles that intersect $B(0,n_\ell)$ is at least $n_\ell^d v_d/k$.
Pick some value n_ℓ, denote the intersections of black tiles with $B(0,n_\ell)$ by $T_1', T_2', \ldots$, and take one point x_i from each tile T_i'. As the minimal distance between two black points in different tiles is t, balls of radius $t/2$ around the points x_i are pairwise disjoint. Hence, their total volume is at least

$$\frac{n_\ell^d \cdot v_d}{k} \cdot (t/2)^d \cdot v_d.$$

On the other hand, each such ball is contained in the ball $B(0,(n_\ell+t))$ (since its radius is $t/2$, and it contains a point in $B(0,n_\ell)$). This implies that for any $\epsilon > 0$ and for a sufficiently large $\ell = \ell(\epsilon)$, the total volume of these balls must be smaller than $(1+\epsilon)\delta_d \cdot \lambda(B(0,n_\ell+t))$, as otherwise, the infinite collection of the balls $B(x_i,t/2)$ (where for each ball $B(0,n_\ell)$ we select x_i's in the way described above, respecting the x_i's selected for smaller values of n_ℓ) would be a sphere packing of $\mathbb{R}^d$ whose density is larger than δ_d. Therefore, for a sufficiently large n_ℓ, we have

$$\frac{n_\ell^d \cdot v_d}{k} \cdot (t/2)^d \cdot v_d \leq (1+\epsilon)(1+\frac{t}{n_\ell})^d \delta_d \cdot n_\ell^d v_d,$$

and letting $\epsilon \to 0$ and $n_\ell \to \infty$, we obtain $t \leq 2(\delta_d/v_d)^{1/d} \cdot k^{1/d}$, as asserted. ◀

Discussion. In the two last decades, there has been a tremendous progress in the research of the sphere packing problem. In 2005, Hales ([18], see also [17]) solved the problem for $d = 3$, proving a 17'th century conjecture of Kepler. Three years ago, in a beautiful short paper, Viazovska [25] solved the problem for $d = 8$, and shortly after, Cohn, Kumar, Miller, Radchenko, and Viazovska [6] used Viazovska's method along with other tools to solve the problem for $d = 24$. For other dimensions, the problem is still open. We can use the results of [6, 18, 25], along with the value of δ_2 that was obtained already by Lagrange, to obtain tight upper bounds on t in dimensions $2, 3, 8$, and 24.

- For $d = 2$, Lagrange (1773) showed that $\delta_2 = \frac{\pi}{2\sqrt{3}} \approx 0.907$. Hence, we obtain the bound $t \leq \frac{2^{1/2}}{3^{1/4}} \cdot k^{1/2} \approx 1.074k^{1/2}$.
- For $d = 3$, Hales [18, 17] showed that $\delta_3 = \frac{\pi}{3\sqrt{2}} \approx 0.740$. Hence, we obtain the bound $t \leq 2^{1/6} \cdot k^{1/3} \approx 1.122k^{1/3}$.
- For $d = 8$, Viazovska [25] showed that $\delta_8 = \frac{\pi^4}{2^4 4!} \approx 0.254$. Hence, we obtain the bound $t \leq \sqrt{2}k^{1/8} \approx 1.414k^{1/8}$.
- For $d = 24$, Cohn et al. [6] showed that $\delta_{24} = \frac{\pi^{12}}{12!} \approx 0.0019$. Hence, we obtain the bound $t \leq 2k^{1/24}$.

As we show in Section 5, all these bounds are asymptotically tight.

Using the same method, we can leverage any upper bound for the sphere packing problem (namely, upper bound on δ_d) into an upper bound on the error resilience of a rounding scheme in the corresponding dimension. The best currently known bound on δ_d for a large d is by Cohn and Zhao [7], who obtained a constant-factor improvement over the classical Kabatiansky-Levenshtein [21] bound $\delta_d \leq 2^{-(0.5990+o(1))d}$. A list of conjectured bounds for $d \leq 10$ can be found in [8].

5 Lower Bounds on the Error Resilience

In this section (like in Section 4), we consider tilings of $\mathbb{R}^d$ by tiles $T_1, T_2, \ldots$ of volume at most 1. Each point in $\mathbb{R}^d$ is colored in one of $k \geq d + 1$ colors, and our goal is to maximize the minimal distance t between two points of the same color that belong to different tiles.

We obtain lower bounds on t for various values of d, k, by constructing explicit tilings. First, we present *brick-wall* tilings, which provide lower bounds for small values of d, k. Then we present tilings based on *close sphere packing*, which show the tightness of our asymptotic bounds for $d = 2, 3, 8, 24$. Finally, we inductively construct a *dimension-reducing* tiling which shows that for any dimension d, positive error resilience can be obtained with $d + 1$ colors.

5.1 Brick-wall tilings

We begin with a tiling of the plane, and then use it to construct a tiling of $\mathbb{R}^3$.

5.1.1 2-dimensional brick wall

In the 2-dimensional brick wall tiling with k colors, demonstrated in Figure 2, each tile is a rectangle with side lengths $\sqrt{(k-2)/2}$ and $\sqrt{2/(k-2)}$ (and so, the area of each tile is 1). The tiling is periodic, where the basic unit is two rows of adjacent rectangles, colored in a round robin fashion. For an even k, the second row is placed exactly below the first row, and the sequence of colors is shifted by $k/2$. For an odd k, the second row is indented by half a brick (making the tiling look like a brick wall), and the sequence is shifted by $(k + 1)/2$.

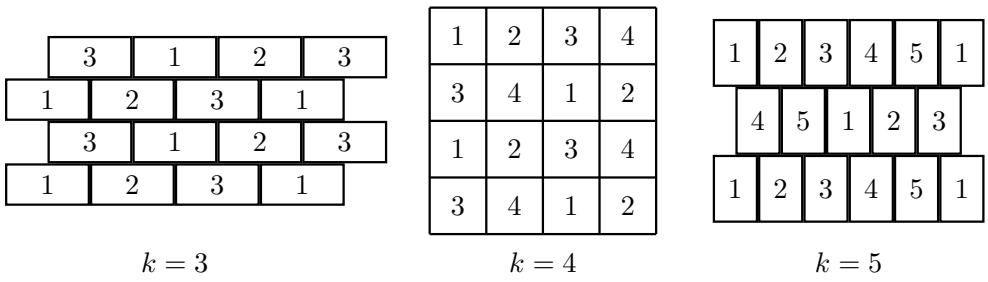


Figure 2 The 2-dimensional brick wall tiling for $k = 3, 4, 5$ colors. The ratio between the width and the height of each tile is $2 : k - 2$.

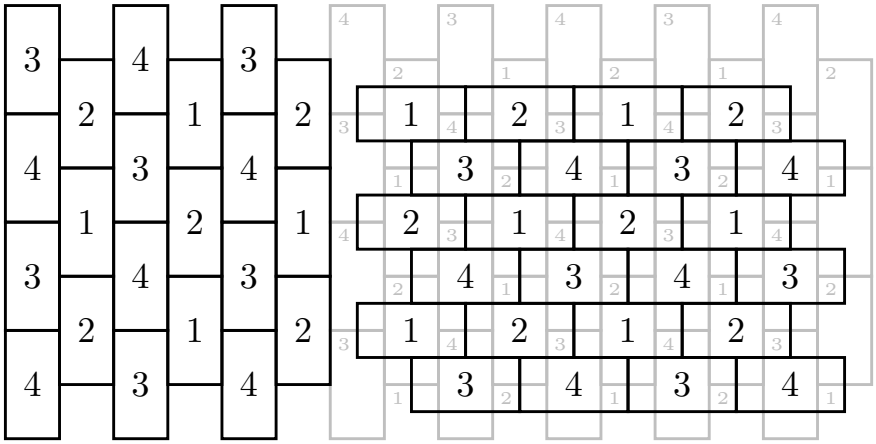


Figure 3 The 3-dimensional brick wall tiling.

It is easy to see that the minimal distance between two same-colored points in different tiles is $\sqrt{(k-2)/2}$. (This distance is attained both in the vertical and in the horizontal directions. Having the same minimal distance in both directions is the optimization that dictates the side lengths of the bricks.) In particular, we obtain the lower bounds $t \geq 1/\sqrt{2}$ for 3 colors, $t \geq 1$ for 4 colors, and $t \geq \sqrt{3/2}$ for 5 colors.

5.1.2 3-dimensional brick wall

The 3-dimensional brick wall (3BW) tiling, demonstrated in Figure 3, is a periodic tiling of $\mathbb{R}^3$, colored in 4 colors. In order to present the tiling, we need an auxiliary notation.

Notation. Consider the slab $D = \mathbb{R} \times \mathbb{R} \times [z_1, z_2] \subset \mathbb{R}^3$. We say that a tiling $T_1, T_2, \ldots$ of D is a *fattened plane tiling* if there exists a tiling $T'_1, T'_2, \ldots$ of the plane such that $\forall i : T_i = T'_i \times [z_1, z_2]$.

The structure of 3BW. The 3BW tiling is periodic, where the basic unit consists of two brick wall layers, placed one on top of the other in the way presented in Figure 3. Each brick wall layer is a fattening of a brick wall tiling of the plane. The underlying plane tiling is a

15	16	13	14	15	16	13	14
1	2	3	4	1	2	3	4
5	6	7	8	5	6	7	8
9	10	11	12	9	10	11	12
13	14	15	16	13	14	15	16
3	4	1	2	3	4	1	2
7	8	5	6	7	8	5	6
11	12	9	10	11	12	9	10
15	16	13	14	15	16	13	14
1	2	3	4	1	2	3	4

Figure 4 The honeycomb of rectangles tiling for $d = 2$ and $k = 16$ colors. The boundaries of the basic "large rectangles" are depicted in bold. The placement of the tiles in each single color corresponds to the honeycomb lattice.

periodic tiling, in which the basic unit consists of four columns of adjacent rectangles, where the even columns are indented by half a brick, making the tiling look like a brick wall. In the lower layer, in odd columns, the colors 1,2 are used alternately, and in even columns, the colors 3,4 are used alternately. Furthermore, the colors in the third and fourth columns are shifted by one, see Figure 3. In the upper layer columns are replaced by rows. Note that once the layers are placed, the coloring of one layer fully determines the coloring of the other.

Analysis given in the full version of the paper shows that by taking $a = 1$ and fixing the height of each layer to be $1/2$, we obtain $t = 1/2$, and thus, error resilience of 0.25.

5.2 Tilings based on close sphere packing

We present the tiling in the case of $\mathbb{R}^2$, where it is easier to describe and analyze, and then we generalize it to higher dimensions.

5.2.1 Honeycomb of rectangles

In the honeycomb of rectangles tiling of the plane with $k = m^2$ colors, each tile is a rectangle with side lengths a and $1/a$, where

$$a = \left(\frac{m^2 - 2m + 1}{\frac{3}{4}m^2 - m}\right)^{1/4} \geq \left(\frac{4}{3} - \frac{8}{3m}\right)^{1/4}. \tag{6}$$

The tiling is periodic, where the basic unit is composed as follows. First, we construct a basic "large rectangle", which is an m-by-m square block of tiles, using all the $k = m^2$ colors (in arbitrary order). Then, the basic unit of the tiling is two "fat rows" of adjacent large rectangles, where the second row is indented by half a large rectangle. The coloring of each large rectangle is the same. The tiling, for $k = 16$, is demonstrated in Figure 4. Note that the tiles colored in some single color form the shape of a honeycomb lattice (including the centers of the hexagons). This is why we call the tiling "honeycomb of rectangles".

It is easy to see that the minimal horizontal and diagonal distances between two same-colored tiles are

$$(m-1)a \qquad \text{and} \qquad \sqrt{\left(\frac{m-1}{a}\right)^2 + \left(\left(\frac{m}{2}-1\right)a\right)^2},$$

respectively. By choosing a such that the two distances are equal, we obtain (6), and the asymptotic lower bound

$$t \geq \left(\frac{4}{3} - \frac{8}{3m}\right)^{1/4} \cdot (m-1) \geq (4/3)^{1/4} \cdot \sqrt{k} - O(1) \approx 1.074\sqrt{k} - O(1),$$

that matches the upper bound obtained above up to an additive $O(1)$ term.

5.2.2 Close packing of boxes

Motivation. This construction, a k-colored tiling of $\mathbb{R}^3$ where $k \gg 3$, is a natural generalization to $\mathbb{R}^3$ of the "honeycomb of rectangles" tiling presented in Section 5.2.1. The idea behind the construction is to choose an optimal sphere packing in $\mathbb{R}^3$, and construct a fattened plane tiling in which the tiles in each color are placed at the centers of the spheres of the packing. (Recall that as the number of colors is large, the size of each tile is negligible with respect to the size of its inflation, and hence, we can treat the tiles as single points.)

We use the classical HCP lattice (one of the most common *closed packings*, see [9]), which corresponds to a periodic sphere packing in which the basic unit is two hexagonal layers of spheres, where in the top layer, each sphere is placed on top in the hollow between three spheres in the bottom layer. The coordinates of the centers of these spheres are:

$$(r,r,r), (3r,r,r), (5r,r,r), \ldots, (2r, r+\sqrt{3}r, r), (4r, r+\sqrt{3}r, r), (6r, r+\sqrt{3}r, r), \ldots$$

for the bottom layer, and

$$(2r, r+\frac{\sqrt{3}}{3}r, r+\frac{2\sqrt{6}}{3}r), (4r, r+\frac{\sqrt{3}}{3}r, r+\frac{2\sqrt{6}}{3}r), \ldots, (r, r+\frac{4\sqrt{3}}{3}r, r+\frac{2\sqrt{6}}{3}r), (3r, r+\frac{4\sqrt{3}}{3}r, r+\frac{2\sqrt{6}}{3}r), \ldots$$

for the top layer.

The structure of the tiling. Assume that the number of colors is $k = m^3$. Each tile is a box with side lengths (a, b, c) to be determined below, and the basic unit is a "large box", which is an $m \times m \times m$ cubic block of tiles, using all the $k = m^3$ colors (in arbitrary order). Then, the basic unit of the tiling is a two-layer fattened plane tiling, in which each layer is a fattened copy of the "honeycomb of rectangles" tiling. The upper layer is shifted by $\frac{m}{2}a$ in the x-coordinate and by $\frac{\sqrt{3}m}{6}a$ in the y-coordinate, so that the corners of the large boxes lie in the coordinates of the sphere centers described above (for $r = \frac{m}{2}a$). A quick calculation shows that in order to make this possible, the proportion $(a : b : c)$ should be *approximately* $(2 : \sqrt{3} : 2\sqrt{6}/3)$ (where we neglect the size of each tile with respect to the size of the inflation, which can be absorbed in an $1 - o(1)$ multiplicative factor in the final value of t). The volume of each tile is clearly abc. In order to make the volumes of all tiles equal to 1, we need

$$a \cdot \frac{\sqrt{3}}{2}a \cdot \frac{\sqrt{6}}{3}a = 1,$$

and thus, $a = (6/\sqrt{18})^{1/3} = 2^{1/6} \approx 1.122$. Hence, the side lengths of each tile are

$$(2^{1/6}, 3^{1/2}/2^{5/6}, 2^{2/3}/3^{1/2}) \approx (1.122, 0.972, 0.916),$$

and the minimal distance between two same-colored points in different tiles is

$$(m-1)a = (2^{1/6} - o(1))k^{1/d},$$

which matches the upper bound proved in Section 4.1.

Generalization to higher dimensions. A similar tiling can be constructed to match any *lattice sphere packing*, assuming the number of colors k is sufficiently large with respect to d. Hence, any dense lattice sphere packing can be translated into a lower bound on the asymptotic error resilience of rounding schemes in the corresponding dimension. In particular, as the E_8 *lattice* and the *Leech lattice* which attain the maximal possible density of sphere packings in dimension 8 and 24 (respectively) are lattice packings, they can be translated to box tilings showing that the asymptotic upper bounds on the error resilience in $\mathbb{R}^8$ and $\mathbb{R}^{24}$ proved in Section 4.1 are tight.

5.3 The dimension reducing tiling

We exemplify the tiling in $\mathbb{R}^3$, but it will be apparent how to generalize it to higher dimensions.

Informal description of the tiling. Informally, the tiling is constructed as follows.

Step 1: We begin with dividing $\mathbb{R}^3$ into basic cubes with side length a (to be determined below), depicted in Figure 5(a). Then, we make the "interior" of each basic cube into a tile and give all these tiles the color 1. In order to keep a minimal distance of t between two points colored 1 in different tiles, we must leave a neighborhood of width $t/2$ in each side of each facet of the basic cube. Hence, we are left with "fattened" walls of total width t. The part of such a wall included in a basic cube is shown in Figure 5(b).

Step 2: We make the "interior" of each wall (i.e., fattened facet) into a tile and give all these tiles the color 2, as is demonstrated in Figure 5(b). (Note that each tile contains points from two adjacent basic cubes.) In order to keep a minimal distance of t between two points colored 2 in different tiles, we must leave a neighborhood of $t/\sqrt{2}$ near each edge (i.e., intersection of facets), as is shown in Figure 5(c). Hence, we are left with a "fattened skeleton".

Step 3: We make the "interior" of each edge of the skeleton into a tile and give all these tiles the color 3, as is shown in Figure 5(d). (Note that each tile contains points from four adjacent basic cubes.) In order to keep a minimal distance of t between two points colored 3 in different tiles, we must leave an additional neighborhood of $t/\sqrt{2}$ near each vertex (i.e., intersection of edges), as is demonstrated in Figure 5(d). Hence, we are left with neighborhoods of the corners (a.k.a. vertices).

Step 4: We make the neighborhoods of the corners into tiles and give them the color 4. (Note that each tile contains points from 8 adjacent basic cubes.) We have to make sure that the distance between each two such "fattened corners" is at least t, and this requirement dictates the choice of t.

The analysis of the tiling (including its formal definition and explanation of the choice of parameters) is presented in the full version of the paper. It is clear that the resulting 4-colored tiling can be generalized into a $(d+1)$-colored tiling of $\mathbb{R}^d$.

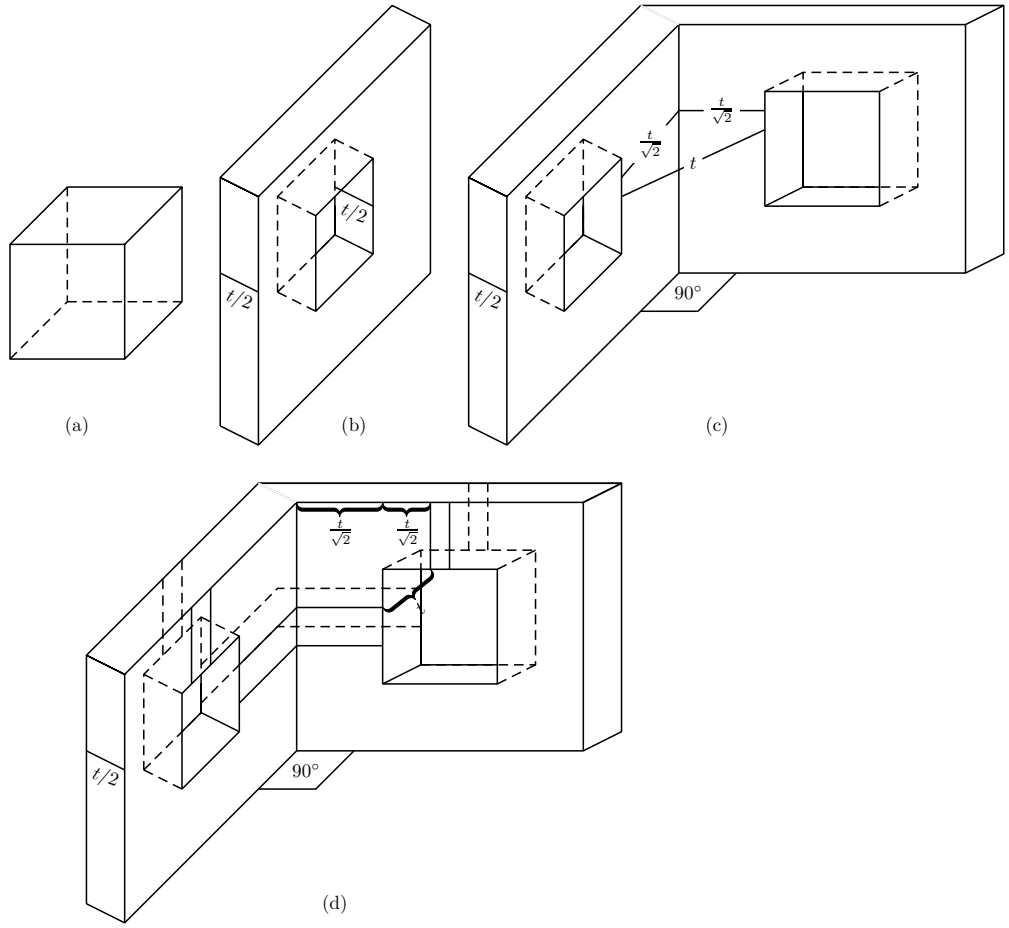


Figure 5 The dimension reducing tiling. Part (a) shows the basic cubes we start with. Part (b) shows a fattened wall of width $t/2$, and its interior that gets the color 2. Part (c) shows where the minimal distance between two tiles colored 2 is attained. Part (d) presents (in full lines) the interiors of the fattened edges that get the color 3 and shows where the minimal distance between two such tiles is attained.

References

1. Erdem Alkim, Léo Ducas, Thomas Pöppelmann, and Peter Schwabe. Post-quantum Key Exchange – A New Hope. In *25th USENIX Security Symposium, USENIX Security 16, Austin, TX, USA, August 10-12, 2016*, pages 327–343, 2016. URL: `https://www.usenix.org/conference/usenixsecurity16/technical-sessions/presentation/alkim`.
2. Michel L. Balinksi. How should data be rounded? In *Lecture Notes – Monograph Series, Vol. 28: Distributions with Fixed Marginals and Related Topics*, pages 33–44. Institute of Mathematical Statistics, 1996.
3. R. B. Bapat. Sperner's lemma with multiple labels. In *Modeling, Computation and Optimization, Chapter 16*, pages 257–262. World Scientific, 2009.
4. Raoul Bott and Loring W. Tu. *Differential Forms in Algebraic Topology*. Springer, 1982.
5. Herbert Busemann. *Convex surfaces*. Interscience publishers, 1958.
6. Henry Cohn, Abhinav Kumar, Stephen D. Miller, Danylo Radchenko, and Maryna Viazovska. The sphere packing problem in dimension 24. *Ann. of Math.*, 185:1017–1033, 2017.

7 Henry Cohn and Yufei Zhao. Sphere packing bounds via spherical codes. *Duke Mathematical Journal*, 163(10):1965–2002, 2014.

8 John H. Conway and Neil J. A. Sloane. What are all the best sphere packings in low dimensions? *Discrete Comput. Geom.*, 13:383–403, 1995.

9 John H. Conway and Neil J. A. Sloane. *Sphere packing, lattices and groups (3rd edition)*. Springer, 2013.

10 Lawrence Cox and Lawrence Ernst. Controlled rounding. *INFOR: Information Systems and Operational Research*, 20(4):423–432, 1982.

11 Mark de Berg, Otfried Cheong, Marc van Kreveld, and Mark Overmars. *Computational Geometry (3rd edition)*. Springer, 2008.

12 Yevgeniy Dodis, Rafail Ostrovsky, Leonid Reyzin, and Adam D. Smith. Fuzzy extractors: How to generate strong keys from biometrics and other noisy data. *SIAM J. Comput.*, 38(1):97–139, 2008. doi:10.1137/060651380.

13 Orr Dunkelman, Zeev Geyzel, Chaya Keller, Nathan Keller, Eyal Ronen, Adi Shamir, and Ran J. Tessler. Error resilient space partitioning (full version). *CoRR*, 2020. arXiv:2008.03675.

14 Benjamin Fuller, Leonid Reyzin, and Adam D. Smith. When are fuzzy extractors possible? *IEEE Trans. Inf. Theory*, 66(8):5282–5298, 2020. doi:10.1109/TIT.2020.2984751.

15 Jacob E. Goodman, Joseph O'Rourke, and Csaba D. Tóth. *Computational Geometry (3rd edition)*. CRC Press, 2017.

16 Robert M. Gray. Vector Quantization. *IEEE ASSP*, 1(2):4–29, 1984.

17 Thomas Hales, Mark Adams, and et al. A formal proof of the kepler conjecture. *Forum of Mathematics, Pi*, 5:e2, 2017.

18 Thomas C. Hales. A proof of the Kepler conjecture. *Ann. of Math.*, 162:1065–1185, 2005.

19 Piotr Indyk, Rajeev Motwani, Prabhakar Raghavan, and Santosh S. Vempala. Locality-preserving hashing in multidimensional spaces. In *Proceedings of the Twenty-Ninth Annual ACM Symposium on the Theory of Computing, El Paso, Texas, USA, May 4-6, 1997*, pages 618–625, 1997. doi:10.1145/258533.258656.

20 Ari Juels and Martin Wattenberg. A fuzzy commitment scheme. In *CCS '99, Proceedings of the 6th ACM Conference on Computer and Communications Security, Singapore, November 1-4, 1999*, pages 28–36, 1999. doi:10.1145/319709.319714.

21 G. A. Kabatyanskii and V. I. Levenshtein. Bounds for packings on a sphere and in space (Russian). *Problemy Peredaci Informacii*, 14:3–25, 1978. English translation in *Prob. Inform. Transmission* **14** (1978), pp. 1—17.

22 Guy Kindler, Ryan O'Donnell, Anup Rao, and Avi Wigderson. Spherical cubes and rounding in high dimensions. In *49th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2008, October 25-28, 2008, Philadelphia, PA, USA*, pages 189–198, 2008. doi:10.1109/FOCS.2008.50.

23 Guy Kindler, Anup Rao, Ryan O'Donnell, and Avi Wigderson. Spherical cubes: optimal foams from computational hardness amplification. *Commun. ACM*, 55(10):90–97, 2012. doi:10.1145/2347736.2347757.

24 E. Sperner. Neuer beweis für die invarianz der dimensionszahl und des gebietes (German). *Abh. Math. Seminar Univ. Hamburg*, 6:265–272, 1928.

25 Maryna S. Viazovska. The sphere packing problem in dimension 8. *Ann. of Math.*, 185:991–1015, 2017.

26 Leon Willenborg and Ton de Waal. *Elements of Statistical Disclosure Control*, volume 155 of *Lecture Notes in Statistics*. Springer, 2001.

27 Günter M. Ziegler. *Lectures on Polytopes*. Graduate Texts in Mathematics. Springer, 1995.

Algebraic Proof Systems

Toniann Pitassi ✉
University of Toronto, Canada

Abstract

Given a set of polynomial equations over a field F, how hard is it to prove that they are simultaneously unsolvable? In the last twenty years, algebraic proof systems for refuting such systems of equations have been extensively studied, revealing close connections to both upper bounds (connections between short refutations and efficient approximation algorithms) and lower bounds (connections to fundamental questions in circuit complexity.)

The Ideal Proof System (IPS) is a simple yet powerful algebraic proof system, with very close connections to circuit lower bounds: [2] proved that lower bounds for IPS imply $VNP \neq VP$, and very recently connections in the other direction have been made, showing that circuit lower bounds imply IPS lower bounds [3, 1].

In this talk I will survey the landscape of algebraic proof systems, focusing on their connections to complexity theory, derandomization, and standard proposional proof complexity. I will discuss the state-of-the-art lower bounds, as well as the relationship between algebraic systems and textbook style propositional proof systems. Finally we end with open problems, and some recent progress towards proving superpolynomial lower bounds for bounded-depth Frege systems with modular gates (a major open problem in propositional proof complexity).

2012 ACM Subject Classification Theory of computation → Proof complexity; Theory of computation → Algebraic complexity theory; Theory of computation → Circuit complexity

Keywords and phrases complexity theory, proof complexity, algebraic circuits

Digital Object Identifier 10.4230/LIPIcs.ICALP.2021.5

Category Invited Talk

References

1 Yaroslav Alekseev, Dima Grigoriev, Edward A. Hirsch, and Iddo Tzameret. Semi-algebraic proofs, IPS lower bounds, and the tau-conjecture: can a natural number be negative? In Konstantin Makarychev, Yury Makarychev, Madhur Tulsiani, Gautam Kamath, and Julia Chuzhoy, editors, *Proccedings of the 52nd Symposium on Theory of Computing, (STOC)*, pages 54–67. ACM, 2020.

2 Joshua A. Grochow and Toniann Pitassi. Circuit complexity, proof complexity, and polynomial identity testing: The ideal proof system. *J. ACM*, 65(6):37:1–37:59, 2018.

3 Rahul Santhanam and Iddo Tzameret. Iterated lower bound formulas: A diagonalization approach to proof co mplexity. In *Proceedings of the 53rd Symposium on Theory of Computing (STOC)*. ACM, 2021.

A Very Sketchy Talk

David P. Woodruff ✉
Carnegie Mellon University, Pittsburgh, PA, USA

Abstract

We give an overview of dimensionality reduction methods, or sketching, for a number of problems in optimization, first surveying work using these methods for classical problems, which gives near optimal algorithms for regression, low rank approximation, and natural variants. We then survey recent work applying sketching to column subset selection, kernel methods, sublinear algorithms for structured matrices, tensors, trace estimation, and so on. The focus is on fast algorithms. This is a short survey accompanying an invited talk at ICALP, 2021.

2012 ACM Subject Classification Theory of computation → Streaming, sublinear and near linear time algorithms

Keywords and phrases dimensionality reduction, optimization, randomized numerical linear algebra, sketching

Digital Object Identifier 10.4230/LIPIcs.ICALP.2021.6

Category Invited Talk

Funding *David P. Woodruff*: Supported by the Office of Naval Research (ONR) grant N00014-18-1-2562, the National Science Foundation (NSF) under Grant No. CCF-1815840, and a Simons Investigator Award.

1 Introduction

Sketching, or data dimensionality reduction, is a popular tool for speeding up algorithms in machine learning, optimization, and randomized numerical linear algebra.

In the overconstrained least squares regression problem one is given an $n \times d$ matrix A, $n > d$, together with an $n \times 1$ vector b, and one is tasked with finding an $x \in \mathbb{R}^d$ for which $\|Ax - b\|_2^2 = \sum_{i=1}^n (\langle A_i, x\rangle - b_i)^2$ is as small as possible, where A_i is the i-th row of A, b_i is the i-th entry of b, and $\langle A_i, x\rangle$ denotes the inner product between A_i and x. Geometrically, in $\mathbb{R}^n$ one can view this as finding the vector Ax which is closest to b in the column span of A (which is a d-dimensional subspace) in terms of Euclidean distance, that is, Ax is just the projection of b onto the column span of A. Alternatively, in $\mathbb{R}^{d+1}$ one can think of having n points, the i-th of which is (A_i, b_i), and one is trying to find a hyperplane defined by x so as to minimize the sum of squares of distances between the points $(A_i, \langle A_i, x\rangle)$ and the points (A_i, b_i). There is a closed-form solution to this problem of the form $x = A^- b$, where A^- is the Moore-Penrose pseudoinverse of A, and the optimal x can be computed in $O(nd^2)$ time by computing the singular value decomposition (SVD) [1] of A. While this is an exact solution, the $O(nd^2)$ running time is prohibitive for large values of n and moderate values of d.

The sketch-and-solve paradigm instead solves this problem by first choosing a random matrix $S \in \mathbb{R}^{k \times n}$, where $k \ll n$. One then computes $S \cdot A$, which is a small $k \times n$ matrix, as well as $S \cdot b$, which is a small $k \times 1$ vector. One then solves the much smaller regression problem $\min_x \|SAx - Sb\|_2$ by computing its minimizer $x' = (SA)^- Sb$, and the hope is that $\|Ax' - b\|_2^2 \leq (1+\epsilon)\|Ax^* - b\|_2$, where $x^* = A^- b$ is the minimizer of $\|Ax - b\|_2^2$.

[1] This can be sped up using theoretical algorithms for fast matrix multiplication, giving $O(n \cdot d^{\omega-1})$ time, where $\omega \approx 2.376$ is the exponent of fast matrix multiplication.

A natural question is how to choose the sketching matrix S. One could choose S to be a $k \times n$ matrix of independent and identically distributed $N(0, 1/k)$ random variables (normal with mean 0 and variance $1/k$), where $k = O(d/\epsilon^2)$, which turns out to work (see, e.g., discussion in [39]), but then computing $S \cdot A$ would take at least nd^2 time naïvely, which although it can be sped up with fast matrix multiplication, is slower than just computing the exact solution $x^* = A^- b$. Sárlos [32] pioneered the sketch-and-solve paradigm and observed that one could instead choose S to be a so-called Subsampled Randomized Hadamard Transform, that is, $S = P \cdot H \cdot D$, where D is an $n \times n$ diagonal matrix with independent diagonal entries each chosen uniformly in $\{-1, 1\}$, H is the $n \times n$ Hadamard matrix (assuming n is a power of 2), and P uniformly samples $d \operatorname{poly}(\log d)/\epsilon^2$ entries of whichever vector it is applied to (see [20] for optimizations to the logarithmic factors). Then $S \cdot A$ and $S \cdot b$ can now be computed in $O(nd \log n)$ time; indeed, this follows since D and P can be applied to a vector in $O(n)$ time, and using the recursive structure defining H the matrix H can be applied to a vector in $O(n \log n)$ time. Consequently, $SA = PHDA$ can be computed in $O(nd \log n)$ time. One can then solve $\min_x \|SAx - Sb\|_2$ in $d^3 \operatorname{poly}(\log d)/\epsilon^2$ time, and the solution x' can be shown to, with large probability, satisfy $\|SAx' - Sb\|_2 \leq (1+\epsilon)\|Ax^* - b\|_2$. This gives a runtime of $\tilde{O}(nd \log n + d^3/\epsilon^2)$, where the notation $\tilde{O}(f)$ denotes $f \cdot \operatorname{poly}(\log f)$. The additive d^3/ϵ^2 term can be further improved using fast matrix multiplication algorithms.

This was later improved by Clarkson and Woodruff [14] who showed that one could instead choose the so-called CountSketch matrix S [12] as one's sketching matrix; the analysis was later simplified and improved in [24, 27, 8, 17]. Here S is $k \times n$, where $k = O(d^2/\epsilon^2)$ is again independent of the large dimension n. The key property is that S has a single randomly chosen non-zero entry per column, which is chosen at a uniformly random position and is uniform in $\{-1, 1\}$, and chosen independently across the columns. The key property is that now SA and Sb can be computed in $\operatorname{nnz}(A)$ time, where $\operatorname{nnz}(A)$ denotes the number of non-zero entries of A, and that the solution x' to $\min_x \|SAx - Sb\|_2$ is such that $\|Ax' - b\|_2 \leq (1+\epsilon)\|Ax^* - b\|_2$. The proof in [14] departed from previous proofs which first argued that any fixed vector y has its norm preserved up to a $(1 \pm \epsilon)$-multiplicative factor with probability $1 - 2^{-O(d)}$; after this, a standard net argument could then be used. In fact CountSketch does not have this property and [14] instead observed that the $2^{O(d)}$ vectors one is interested in preserving the norms of, all live in a low-dimensional subspace, and consequently, the number of "heavy coordinates" as one ranges over all vectors in the subspace is a small subset of all possible n coordinates. This then enables CountSketch to work in the sketch-and-solve paradigm, and gives an overall algorithm for solving regression in $\operatorname{nnz}(A) + \operatorname{poly}(d/\epsilon)$ time.

While we have the property that $\|Ax' - b\|_2^2 \leq (1+\epsilon)\|Ax^* - b\|_2^2$, this guarantee is often not sufficient in machine learning and optimization tasks, and one would instead like to bound $\|x^* - x'\|_2^2$. Indeed, one could hope x' is close to a "ground truth" hyperplane and therefore give good generalization error. To do so, note that

$$\begin{aligned}
\|x^* - x'\|_2^2 &\leq \frac{\|Ax^* - Ax'\|_2^2}{\sigma_{min}^2(A)} \\
&\leq \frac{\|Ax' - b\|_2^2 - \|Ax^* - b\|_2^2}{\sigma_{min}^2(A)} \\
&\leq \frac{((1+\epsilon)^2 - 1)\|Ax^* - b\|_2^2}{\sigma_{min}^2(A)} \\
&\leq \frac{O(\epsilon)\|Ax^* - b\|_2^2}{\sigma_{min}^2(A)},
\end{aligned}$$

where the first inequality follows from the definition of the minimum singular value, the second inequality follows since $Ax^* - b$ and $Ax^* - Ax'$ are orthogonal, and the third inequality follows from the objective function guarantee discussed above. We note that this simple guarantee was significantly improved in [30], where the authors bounded $\|x^* - x\|_\infty^2$, i.e., the difference on every single coordinate; this improved analysis holds for the Subsampled Randomized Hadamard Transform as well as Gaussian sketches, but not CountSketch.

Another unsatisfactory aspect of the above is that the dependence on the approximation factor ϵ is polynomial, rather than polylogarithmic. To achieve the latter, one can combine sketching and optimization techniques in a somewhat different way. One can first run the sketch-and-solve paradigm with CountSketch with a constant $\epsilon_0 = 1/2$ to find an x' with $\|Ax' - b\|_2^2 \leq 3/2 \cdot \|Ax^* - b\|_2^2$. This step takes $\text{poly}(d)$ time independent of the value of ϵ. Let $x_0 = x'$. In parallel, one could compute $S \cdot A$ for a CountSketch matrix S with $\epsilon_0 = 1/2$. Then write $SA = QR$, where Q is a matrix with orthonormal columns; one could find QR by letting $SA = U\Sigma V^T$ be its SVD and setting $Q = U$. This step takes $\text{poly}(d)$ time independent of the value of ϵ. Let

$$\kappa(AR^{-1}) = \frac{\sigma_{max}^2(AR^{-1})}{\sigma_{min}^2(AR^{-1})}.$$

It is not hard to see that $\kappa(AR^{-1}) \leq 3$. Indeed, by the so-called subspace embedding property of S, we have for all x:

$$\frac{1}{2}\|Ax\|_2^2 \leq \|SAx\|_2^2 \leq \frac{3}{2}\|Ax\|_2^2.$$

This means for all unit vectors x, $\|AR^{-1}x\|_2^2 \leq (3/2)\|SAR^{-1}x\|_2^2 = 3/2$, and similarly for all unit vectors x, $\|AR^{-1}x\|_2^2 \geq (1/2)\|SAR^{-1}x\|_2^2 = 1/2$. Here the equalities follow from the fact that $SAR^{-1} = Q$, which has orthonormal columns, so $\|Qx\|_2^2 = 1$ for all unit vectors x. Using that $\sigma_{max}(B) = \sup_{\text{unit } x} \|Bx\|_2$ and $\sigma_{min}(B) = \inf_{\text{unit } x} \|Bx\|_2$ for a matrix B with more rows than columns, we have:

$$\sigma_{max}^2(AR^{-1}) \leq \frac{3}{2} \cdot \sigma_{max}^2(SAR^{-1}) = \frac{3}{2} \cdot 1 = \frac{3}{2}.$$

Here the equality follows from the fact that $SAR^{-1} = Q$, and Q has orthonormal columns, and thus $\sigma_{max}(Q) = \sigma_{min}(Q) = 1$. Similarly,

$$\sigma_{min}^2(AR^{-1}) \geq \frac{1}{2} \cdot \sigma_{min}^2(SAR^{-1}) = \frac{1}{2}.$$

Consequently, $\kappa(AR^{-1}) \leq 3$. At this point, one can simply run gradient descent on the function $f(x) = \frac{1}{2}\|AR^{-1}x - b\|_2^2$ with initial solution Rx_0. By standard arguments, the number of iterations required to get ϵ error is $O(\kappa \log(1/\epsilon)) = O(\log(1/\epsilon))$. Moreover, one never needs to explicitly compute $A \cdot R^{-1}$. Indeed, given an iterate x_t in some iteration t, one can compute $R^{-1}x_t$ and then $AR^{-1}x_t$, in $O(d^2 + \text{nnz}(A))$ time per iteration, and thus $O((\text{nnz}(A) + d^2)\log(1/\epsilon))$ time overall. This, together with the additive $O(\text{nnz}(A) + \text{poly}(d))$ time needed to find R^{-1} and x_0, gives an overall running time of $O((\text{nnz}(A) + d^2)\log(1/\epsilon) + \text{poly}(d))$. We refer the reader to [14] for further details.

2 Extensions

There are many related problems to regression (and other problems!) for which sketching can be applied, and we outline only a few here.

2.1 Ridge Regression

There are regularized variants of regression, such as ridge regression, where one instead seeks to minimize $\|Ax-b\|_2^2+\lambda\|x\|_2^2$, for a parameter $\lambda>0$. Here the $\lambda\|x\|_2^2$ term is known as the ridge or regularization, and encourages low-norm solutions. This formulation is useful both when $n>d$ as well as when $n<d$; in the latter case A is underconstrained, i.e., has more columns than rows, and without the regularization there are multiple solutions possible and one is often interested in a low-norm solution. We remark that low-norm solutions often have better properties for applications, e.g., may not overfit the data as much, and by setting λ to be large one encourages low norm solutions. For ridge regression, one can sketch A with a dimension that depends on the so-called statistical dimension $sd_\lambda(A)=\sum_i \frac{\sigma_i^2}{\lambda+\sigma_i^2}$, which is always bounded by the rank of A though can be much smaller if a few values σ_i are very large and λ is set appropriately. Sketching is thus immediately useful in the overconstrained case when $n>d$, since $sd_\lambda(A)$ may be much less than d, and so the sketching dimension is much smaller. In the underconstrained case, one often instead *sketches on the right*, setting up the problem $\min_y\|ARy-b\|_2^2+\lambda\|Ry\|_2^2$, for a sketching matrix R. In this case sketching ultimately allows for solving the problem in $\mathrm{nnz}(A)+\mathrm{poly}(sd_\lambda\sigma_1(A)/(\epsilon\lambda))$ time, which although depends on $\sigma_1(A)$, can still be useful. We note that one can combine sketching on both the left and the right for ridge regression; we refer the reader to [2, 3] and the references therein for further details and the history of sketching as applied to this problem.

2.2 Kernel Regression

Another application is kernel regression. In the kernel setting one is given n points $x^1,\ldots,x^n\in\mathbb{R}^d$ and one would like to apply an often non-linear mapping ϕ to "lift" them to a feature space. A notable example is the polynomial kernel of degree q, where $\phi:\mathbb{R}^d\to\mathbb{R}^{d^q}$ where $\phi(x)_{i_1,i_2,\ldots,i_q}=x_{i_1}\cdot x_{i_2}\cdots x_{i_q}$. One reason the polynomial kernel is so important is that one can often Taylor-expand other kernels, such as the Gaussian kernel, and approximate them by a polynomial kernel of large enough degree. Define the $d^q\times n$ matrix A with i-th column equal to $\phi(x^i)$. One would never want to compute this matrix, as the number d^q of rows is prohibitively large. Nevertheless, one would like to be able to solve optimization problems with respect to this matrix.

In particular, in the kernel ridge regression problem one seeks to find a vector $y\in\mathbb{R}^d$ so as to minimize $\|A^TAy-b\|_2^2+\lambda\|y\|_2^2$. Initial work [28] showed how, given vectors $x^1,\ldots,x^q$, each in $\mathbb{R}^d$, to compute a sketch $S(x^1\otimes x^2\otimes\cdots\otimes x^q)$ without first having to compute the tensor product $x^1\otimes x^2\otimes\cdots\otimes x^q$, which would require an unreasonable d^q amount of time. The rough idea is to apply separate sketches $S^1,\ldots,S^q$ to each of the q "modes", obtaining $S^1x^1,S^2x^2,\ldots,S^qx^q$, where each S^i is a CountSketch matrix. If S^i has k rows, then the coordinates of each S^ix^i are associated with the coefficients of a degree-$(k-1)$ polynomial in a formal variable z. One then multiplies these polynomial modulo z^k-1 using the Fast Fourier Transform to improve efficiency. Interestingly, one can show this corresponds to applying another CountSketch S (which is a function of $S^1,\ldots,S^q$) to the vector $x^1\otimes x^2\otimes\cdots\otimes x^q$, with certain structural properties (so S is not a truly random CountSketch matrix, but nevertheless is good enough). Applying this to each column of A separately, which has the form $(x^i)^{\otimes q}$, one can then solve the sketched kernel regression problem:

$$\min_y\|A^TS^TSAy-b\|_2^2+\lambda\|Ay\|_2^2,$$

where now one has SA without ever having to materialize the matrix A.

While the initial work was well-suited for low degree polynomial kernels (small q), their dependence on the sketching dimension is exponential in q, making them less suitable for tasks such as approximating the Gaussian kernel, where q is chosen to be at least logarithmic in n. In recent work [1], a binary tree scheme was used, together with additional sketching at each internal node, to design a linear and oblivious (not dependent on the input) sketch which reduces the dependence on the degree q to polynomial. This was then successfully applied to sketching the Gaussian kernel.

2.3 Structured Matrix Regression

In a number of real-world instances of regression, the design matrix A is structured, e.g., it might be a Hankel, Toeplitz, Vandermonde, or more generally a low-displacement rank matrix. Such matrices A come with fast matrix multiplication algorithms, meaning one can compute $A \cdot x$ in $O(n \log n)$ or $n \cdot \text{polylog}(n)$ time, which is often significantly faster than the nd time needed to multiply an arbitrary $n \times d$ matrix A with a vector x. Notice that the running time is sublinear in the time to write down the matrix A and this leads to the quest for obtaining sublinear time algorithms for a number of optimization problems. Using dimensionality reduction-based methods [19, 33], if $T(A)$ is the time to multiply a given matrix A by an arbitrary vector x, it is possible to $(1+\epsilon)$-approximate least squares regression in $T(A) \log n + \text{poly}(d \log n/\epsilon)$ time, yielding sublinear time (in nd) for a number of structured regression problem.

3 Wrapping Up

While our focus in this short survey was on variants of regression, there is also a large body of work on applying sketching to other optimization problems. A very small set of examples includes the following:

- low rank approximation [14], where one seeks to approximate an $n \times d$ matrix A by a product of an $n \times k$ matrix L and a $k \times d$ matrix R, where $k \ll \min(n, d)$, and thus one can store A with only $(n + d)k$ parameters as opposed to nd parameters
- CUR decomposition [9], which is a special kind of low rank approximation where one seeks to approximate A by CUR, where C is an $n \times c$ matrix and consists of an actual subset of columns of A, R is an $r \times d$ matrix and consists of an actual subset of rows of A, and U is a small $c \times r$ arbitrary matrix. Here the hope is that r, c are small and that this provides a more "interpretable" low rank approximation
- clustering, for which low-rank approximation can quickly provide an initial dimensionality reduction, which can then be used to create a coreset for problems such as k-means [18], or k-median [35, 21]
- distributed, sliding window, and streaming computation (see, e.g., [13, 10, 11], and references therein): since sketches provide a form of compression and can be easily updated due to their linearity, they are naturally useful for providing communication-efficient distributed algorithms as well as space-efficient algorithms in the sliding window and streaming models
- optimization, where sketching can be used for example to compress gradients in first order optimization [23], as well as inside of each iteration in second order methods [29], which often involve solving a least squares regression problem
- finding a latent simplex is an important problem in topic models and community detection, for which one is given n data points that are formed by randomly perturbing some points that come from a latent simplex. Sketching was recently used to obtain truly input sparsity time algorithms in [4]

- trace estimation, where one is given an $n \times n$ positive semidefinite matrix A and would like to estimate $\sum_{i=1}^{n} A_{i,i}$ up to a multiplicative factor of $1+\epsilon$. Recently, fast sketching algorithms for low rank approximation were used in part to do this with constant probability using $O(1/\epsilon)$ matrix-vector products [25], improving the long-standing Hutchinson's algorithm which uses $\Omega(1/\epsilon^2)$ matrix-vector products.
- tensor low rank approximation [37] and weighted low rank approximation [31, 7], where one is given a tensor and would like to find a low rank approximation in some norm; such problems are often NP-hard and bicriteria algorithms, as well as fixed parameter tractable algorithms, have been proposed to obtain provable guarantees.
- robust variants of regression and low rank approximation, where the standard sum of squares error measure is replaced with more robust loss functions such as sum of absolute values [34, 15, 16, 36, 38]
- sublinear time low rank approximation, where one uses the structure of the input matrix to devise algorithms that achieve relative error low rank approximation in sublinear time, e.g., for positive semidefinite matrices [26, 5] or distance matrices [6, 22].

Sketching and dimensionality reduction are rapidly expanding areas. Some of these topics are covered in my older monograph [39]. See also the course notes for the "Algorithms for Big Data" class I teach at CMU, which contains much of this material[2]. I would like to thank the ICALP program committee for giving me the opportunity to write this, and my apologies for only covering a small part of the vast body of work on sketching and for focusing on work that I am most familiar with, and even unfortunately omitting many of those references as well. Hopefully though, if the reader has not seen sketching before, this document can serve as a short and simple introduction to the area.

References

1 Thomas D. Ahle, Michael Kapralov, Jakob Bæk Tejs Knudsen, Rasmus Pagh, Ameya Velingker, David P. Woodruff, and Amir Zandieh. Oblivious sketching of high-degree polynomial kernels. In *Proceedings of the 2020 ACM-SIAM Symposium on Discrete Algorithms, SODA 2020, Salt Lake City, UT, USA, January 5-8, 2020*, pages 141–160, 2020.

2 Haim Avron, Kenneth L. Clarkson, and David P. Woodruff. Faster kernel ridge regression using sketching and preconditioning. *SIAM J. Matrix Anal. Appl.*, 38(4):1116–1138, 2017.

3 Haim Avron, Kenneth L. Clarkson, and David P. Woodruff. Sharper bounds for regularized data fitting. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques, APPROX/RANDOM 2017, August 16-18, 2017, Berkeley, CA, USA*, pages 27:1–27:22, 2017.

4 Ainesh Bakshi, Chiranjib Bhattacharyya, Ravi Kannan, David P. Woodruff, and Samson Zhou. Learning a latent simplex in input-sparsity time, 2021.

5 Ainesh Bakshi, Nadiia Chepurko, and David P. Woodruff. Robust and sample optimal algorithms for PSD low rank approximation. In *61st IEEE Annual Symposium on Foundations of Computer Science, FOCS 2020, Durham, NC, USA, November 16-19, 2020*, pages 506–516, 2020.

6 Ainesh Bakshi and David P. Woodruff. Sublinear time low-rank approximation of distance matrices. In *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, December 3-8, 2018, Montréal, Canada*, pages 3786–3796, 2018.

[2] `http://www.cs.cmu.edu/~dwoodruf/teaching/15859-fall17/index.html`
`http://www.cs.cmu.edu/~dwoodruf/teaching/15859-fall19/index.html`
`http://www.cs.cmu.edu/~dwoodruf/teaching/15859-fall20/index.html`

7 Frank Ban, David P. Woodruff, and Richard Zhang. Regularized weighted low rank approximation. In *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*, pages 4061–4071, 2019.

8 Jean Bourgain, Sjoerd Dirksen, and Jelani Nelson. Toward a unified theory of sparse dimensionality reduction in euclidean space. In *Proceedings of the Forty-Seventh Annual ACM on Symposium on Theory of Computing, STOC 2015, Portland, OR, USA, June 14-17, 2015*, pages 499–508, 2015.

9 Christos Boutsidis and David P. Woodruff. Optimal CUR matrix decompositions. *SIAM J. Comput.*, 46(2):543–589, 2017.

10 Christos Boutsidis, David P. Woodruff, and Peilin Zhong. Optimal principal component analysis in distributed and streaming models. In *Proceedings of the 48th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2016, Cambridge, MA, USA, June 18-21, 2016*, pages 236–249, 2016.

11 Vladimir Braverman, Petros Drineas, Cameron Musco, Christopher Musco, Jalaj Upadhyay, David P. Woodruff, and Samson Zhou. Near optimal linear algebra in the online and sliding window models, 2020. `arXiv:1805.03765`.

12 Moses Charikar, Kevin C. Chen, and Martin Farach-Colton. Finding frequent items in data streams. *Theor. Comput. Sci.*, 312(1):3–15, 2004.

13 Kenneth L. Clarkson and David P. Woodruff. Numerical linear algebra in the streaming model. In *Proceedings of the 41st Annual ACM Symposium on Theory of Computing, STOC 2009, Bethesda, MD, USA, May 31 - June 2, 2009*, pages 205–214, 2009.

14 Kenneth L. Clarkson and David P. Woodruff. Low rank approximation and regression in input sparsity time. In *Symposium on Theory of Computing Conference, STOC'13, Palo Alto, CA, USA, June 1-4, 2013*, pages 81–90, 2013.

15 Kenneth L. Clarkson and David P. Woodruff. Input sparsity and hardness for robust subspace approximation, 2015. `arXiv:1510.06073`.

16 Kenneth L. Clarkson and David P. Woodruff. Sketching for M-estimators: A unified approach to robust regression. In *Proceedings of the Twenty-Sixth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2015, San Diego, CA, USA, January 4-6, 2015*, pages 921–939, 2015.

17 Michael B. Cohen. Nearly tight oblivious subspace embeddings by trace inequalities. In *Proceedings of the Twenty-Seventh Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2016, Arlington, VA, USA, January 10-12, 2016*, pages 278–287, 2016.

18 Michael B. Cohen, Sam Elder, Cameron Musco, Christopher Musco, and Madalina Persu. Dimensionality reduction for k-means clustering and low rank approximation. In *Proceedings of the Forty-Seventh Annual ACM on Symposium on Theory of Computing, STOC 2015, Portland, OR, USA, June 14-17, 2015*, pages 163–172, 2015.

19 Michael B. Cohen, Yin Tat Lee, Cameron Musco, Christopher Musco, Richard Peng, and Aaron Sidford. Uniform sampling for matrix approximation. In *Proceedings of the 2015 Conference on Innovations in Theoretical Computer Science, ITCS 2015, Rehovot, Israel, January 11-13, 2015*, pages 181–190, 2015.

20 Michael B. Cohen, Jelani Nelson, and David P. Woodruff. Optimal approximate matrix product in terms of stable rank, 2016. `arXiv:1507.02268`.

21 Zhili Feng, Praneeth Kacham, and David P. Woodruff. Strong coresets for subspace approximation and k-median in nearly linear time. *CoRR*, abs/1912.12003, 2019.

22 Piotr Indyk, Ali Vakilian, Tal Wagner, and David P. Woodruff. Sample-optimal low-rank approximation of distance matrices. In *Conference on Learning Theory, COLT 2019, 25-28 June 2019, Phoenix, AZ, USA*, pages 1723–1751, 2019.

23 Nikita Ivkin, Daniel Rothchild, Enayat Ullah, Vladimir Braverman, Ion Stoica, and Raman Arora. Communication-efficient distributed sgd with sketching. *arXiv preprint*, 2019. `arXiv:1903.04488`.

24 Xiangrui Meng and Michael W. Mahoney. Low-distortion subspace embeddings in input-sparsity time and applications to robust linear regression. In *Symposium on Theory of Computing Conference, STOC'13, Palo Alto, CA, USA, June 1-4, 2013*, pages 91–100, 2013.

25 Raphael A. Meyer, Cameron Musco, Christopher Musco, and David P. Woodruff. Hutch++: Optimal stochastic trace estimation. In *4th Symposium on Simplicity in Algorithms, SOSA 2021, Virtual Conference, January 11-12, 2021*, pages 142–155, 2021.

26 Cameron Musco and David P. Woodruff. Sublinear time low-rank approximation of positive semidefinite matrices. In *58th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2017, Berkeley, CA, USA, October 15-17, 2017*, pages 672–683, 2017.

27 Jelani Nelson and Huy L. Nguyen. OSNAP: faster numerical linear algebra algorithms via sparser subspace embeddings. In *54th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2013, 26-29 October, 2013, Berkeley, CA, USA*, pages 117–126, 2013.

28 Ninh Pham and Rasmus Pagh. Fast and scalable polynomial kernels via explicit feature maps. In *The 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD 2013, Chicago, IL, USA, August 11-14, 2013*, pages 239–247, 2013.

29 Mert Pilanci and Martin J. Wainwright. Iterative hessian sketch: Fast and accurate solution approximation for constrained least-squares, 2014. `arXiv:1411.0347`.

30 Eric Price, Zhao Song, and David P. Woodruff. Fast regression with an ell_infty guarantee. In *44th International Colloquium on Automata, Languages, and Programming, ICALP 2017, July 10-14, 2017, Warsaw, Poland*, pages 59:1–59:14, 2017.

31 Ilya P. Razenshteyn, Zhao Song, and David P. Woodruff. Weighted low rank approximations with provable guarantees. In *Proceedings of the 48th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2016, Cambridge, MA, USA, June 18-21, 2016*, pages 250–263, 2016.

32 Tamás Sarlós. Improved approximation algorithms for large matrices via random projections. In *47th Annual IEEE Symposium on Foundations of Computer Science (FOCS 2006), 21-24 October 2006, Berkeley, California, USA, Proceedings*, pages 143–152, 2006.

33 Xiaofei Shi and David P. Woodruff. Sublinear time numerical linear algebra for structured matrices. In *The Thirty-Third AAAI Conference on Artificial Intelligence, AAAI 2019, The Thirty-First Innovative Applications of Artificial Intelligence Conference, IAAI 2019, The Ninth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2019, Honolulu, Hawaii, USA, January 27 - February 1, 2019*, pages 4918–4925, 2019.

34 Christian Sohler and David P. Woodruff. Subspace embeddings for the l_1-norm with applications. In *Proceedings of the 43rd ACM Symposium on Theory of Computing, STOC 2011, San Jose, CA, USA, 6-8 June 2011*, pages 755–764, 2011.

35 Christian Sohler and David P. Woodruff. Strong coresets for k-median and subspace approximation: Goodbye dimension. In *59th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2018, Paris, France, October 7-9, 2018*, pages 802–813, 2018.

36 Zhao Song, David P. Woodruff, and Peilin Zhong. Towards a zero-one law for entrywise low rank approximation. *CoRR*, abs/1811.01442, 2018. `arXiv:1811.01442`.

37 Zhao Song, David P. Woodruff, and Peilin Zhong. Relative error tensor low rank approximation. In *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2019, San Diego, California, USA, January 6-9, 2019*, pages 2772–2789, 2019.

38 Ruosong Wang and David P. Woodruff. Tight bounds for p oblivious subspace embeddings. In *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2019, San Diego, California, USA, January 6-9, 2019*, pages 1825–1843, 2019.

39 David P. Woodruff. Sketching as a tool for numerical linear algebra. *Found. Trends Theor. Comput. Sci.*, 10(1-2):1–157, 2014.

Fine-Grained Hardness for Edit Distance to a Fixed Sequence

Amir Abboud
Weizmann Institute of Science, Rehovot, Israel

Virginia Vassilevska Williams
MIT, Cambridge, MA, US

Abstract

Nearly all quadratic lower bounds conditioned on the Strong Exponential Time Hypothesis (SETH) start by reducing k-SAT to the Orthogonal Vectors (OV) problem: Given two sets A,B of n binary vectors, decide if there is an orthogonal pair $a \in A, b \in B$. In this paper, we give an alternative reduction in which the set A does not depend on the input to k-SAT; thus, the quadratic lower bound for OV holds even if one of the sets is fixed in advance.

Using the reductions in the literature from OV to other problems such as computing similarity measures on strings, we get hardness results of a stronger kind: there is a family of sequences $\{S_n\}_{n=1}^{\infty}$, $|S_n| = n$ such that computing the Edit Distance between an input sequence X of length n and the (fixed) sequence S_n requires $n^{2-o(1)}$ time under SETH.

2012 ACM Subject Classification Theory of computation → Design and analysis of algorithms

Keywords and phrases SAT, edit distance, fine-grained complexity, conditional lower bound, sequence alignment

Digital Object Identifier 10.4230/LIPIcs.ICALP.2021.7

Category Track A: Algorithms, Complexity and Games

Funding *Amir Abboud*: Work partly done at the IBM Almaden Research Center.
Virginia Vassilevska Williams: Supported by an NSF CAREER Award, NSF Grants CCF-1528078, CCF-1514339 and CCF-1909429, a BSF Grant BSF:2012338, a Google Research Fellowship and a Sloan Research Fellowship.

1 Introduction

The first step in nearly all hardness proofs for polynomial time problems, that are conditioned on the Strong Exponential Time Hypothesis (SETH), is a seminal reduction of Williams [42] from k-SAT to the Orthogonal Vectors problem.

▶ **Definition 1** (The OV Problem). *Given two sets $A, B \subseteq \{0,1\}^{d(n)}$ of n binary vectors of dimension $d(n)$, decide if there is a pair $a \in A, b \in B$ that are orthogonal, i.e. $\forall i \in [d(n)]$: $a[i] = 0 \vee b[i] = 0$.*

The SETH states that k-SAT cannot be solved in $(2-\varepsilon)^n$ time, where $\varepsilon > 0$ is independent of k. The aforementioned reduction takes such a k-CNF formula on n variables and produces two sets of $N = 2^{n/2}$ binary vectors in $d(N) = O(\log N)$ dimensions. Thus, a subquadratic $O(N^{2-\varepsilon})$ algorithm for OV gives a $(2-\varepsilon')^n$ algorithm for k-SAT and refutes SETH. The current best algorithms for OV are mildly subquadratic $O(N^2/f(N))$ where $f(N) = N^{o(1)}$ [11, 25].

It turns out that OV is at the core of so many other problems, making their complexity quadratic. Dozens of fine-grained reductions from OV to various important problems from diverse fields have been designed in the last decade, resulting in a long list of quadratic SETH-based lower bounds [40]. For example, to prove their $n^{2-o(1)}$ lower bound for the Edit

Distance problem, Backurs and Indyk [14] encode each set of vectors A, B with a sequence S_A, S_B of length $O(nd(n))$, such that the edit distance between them reveals the existence of an orthogonal pair.

1.1 Results

In this paper we revisit the simple reduction of Williams from k-SAT to OV, that has been presented in countless lectures on fine-grained complexity, and expose a surprising room for improvement: there is an alternative reduction (with vectors of a mildly larger dimension) that encodes the k-CNF formula with *only one* of the sets. That is, the set of vectors A is *fixed* in all the instances produced by the new reduction. The implications for the SETH-hard problems may sound bizarre: it takes n^2 time to compute the Edit Distance even if one of the two sequences is always the same (for all inputs of length n). Before discussing the implications further, let us try to clarify the result with a high level technical overview.

1.1.1 Main idea

The difference between the two reductions is simple to explain. Assume we are given a k-CNF formula ϕ on n variables $x_1, \ldots, x_n$ and m clauses $C_1, \ldots, C_m$ such that $\phi = (C_1 \wedge \cdots \wedge C_m)$. In both reductions, we enumerate all partial assignments $\alpha \in \{\texttt{false}, \texttt{true}\}^{n/2}$ to the first half of the variables $x_1, \ldots, x_{n/2}$, and also all partial assignments $\beta \in \{\texttt{false}, \texttt{true}\}^{n/2}$ to the other half of the variables $x_{n/2+1}, \ldots, x_n$, and the goal is to find a satisfying pair, i.e. a pair α, β that when put together make a full satisfying assignment $(\alpha\beta)$.

To do this, Williams encodes each partial assignment, α or β, with a binary vector in m dimensions that has a 0 at the j^{th} coordinate if the partial assignment satisfies the clause C_j and 1 otherwise; it follows that a pair of vectors is orthogonal iff $(\alpha\beta)$ is a satisfying assignment. The vectors corresponding to the α's (the partial assignment to the first half of variables) go to the set A and the vectors corresponding to the β's go to the set B; notice that the vectors in both sets depend on the clauses of ϕ.

In the new reduction, the vectors corresponding to the α's are defined *as if ϕ has all possible clauses of size k*, i.e. as if ϕ is the complete k-CNF formula. The j^{th} coordinate depends on whether α satisfies the j^{th} clause in a certain canonical ordering of all k-CNF clauses over n variables. Thus, the set A does not depend on the (real) input formula ϕ. Then, the definition of the vectors for the β's is similar but with an extra condition: the j^{th} coordinate is automatically set to 0 if the j^{th} clause in the canonical ordering does not exist in ϕ, regardless of whether β satisfies it or not. As a result, all clauses that do not appear in ϕ cannot affect the orthogonality of any pair in $A \times B$, and the correctness of the reduction is maintained.

The only downside of the new reduction is that the size of the vectors grows from $m = O(n)$ (because of the sparsification lemma) to $O(n^{2k})$. That is, from $O(\log N)$ to $\log^{O(1)} N$. However, the dimension is still $N^{o(1)}$ and this is sufficient to deduce many $n^{2-o(1)}$ SETH-based lower bounds. Indeed, as often observed, these lower bounds can be based on the (safer than SETH) hypothesis that OV is hard when the dimension is $N^{o(1)}$.[1] Some exceptions, where the reductions crucially rely on the dimension being logarithmic, are [7, 27, 31].

To formalize the new result as a theorem, in Section 2 we introduce the $\mathrm{OV}_{\mathcal{A}}$ problem in which we are given as input only one set B of n vectors and are asked if there is an orthogonal pair in $A_n \times B$ where A_n is the n^{th} set in an (efficiently producible) family of sets $\mathcal{A} = \{A_n\}_{n=1}^{\infty}$.

[1] Recent works [33, 5] have shown that refuting this "moderate dimension OV" hypothesis has consequences that are potentially more remarkable than refuting SETH.

1.1.2 Implications for other problems

Combining the new reduction with the existing reductions from OV in a black-box way leads to some interesting consequences. For example, the reductions from OV to the computation of certain distance measures on pairs of sequences, curves, or time-series, such as Edit Distance [14], Longest Common Subsequence [3, 22, 19], Local Alignment [9], Fréchet Distance [20], and Dynamic Time Warping Distance [22, 3], all proceed by encoding each of the sets A, B *independently* with a sequence S_A, S_B. Therefore, the quadratic $n^{2-o(1)}$ lower bounds implied by SETH hold even if only one sequence S_B is given as input, while S_A only depends on $n = |S_B|$. Another example is the regular expression matching problem [15, 21] of deciding whether a string x can be generated from a regular expression y. Again, the new reduction shows a quadratic lower bound even if the string (or the expression) are fixed. We find it surprising that such severe-looking restrictions of the problems do not reduce the complexity.

The corollaries go beyond sequence problems. The lower bounds for Bichromatic Closest Pair [12, 28] now also hold when one of the two sets is fixed; note that this is incomparable to the recent lower bound for Monochromatic Closest Pair [30]. In the Subtree Isomorphism problem we are given two rooted, unordered, unlabelled trees and are asked if one is contained in the other (a pattern and a host). The quadratic lower bound [2] can now be shown even for a fixed pattern or a fixed host. The implications are less clear-cut for many other graph problems; for example, the basic reduction from OV to diameter in sparse graphs [39] may now produce slightly simpler graphs but it is hard to characterize in what way. It is likely that a white-box usage of the new reduction will lead to interesting results; we leave this for future work.

1.1.3 Generalizing to k-OV

The SETH lower bound for OV generalizes to an $n^{k-o(1)}$ lower bound, for all $k \geq 2$, for the k-OV problem: given k sets of binary vectors $A_1, \ldots, A_k$ in d dimensions, decide if there are k vectors $a_i \in A_i$ that are orthogonal in the sense that $\forall j \in [d] : (a_1[j] \wedge \cdots \wedge a_k[j]) = 0$. The hardness of k-OV has been used to prove the hardness of several other problems (where a reduction from 2-OV is not known) [8, 10, 33, 38, 1, 16, 32, 29, 23]; e.g. an $n^{k-o(1)}$ lower bound for k-LCS, the problem of computing the longest common subsequence of k strings.

The reduction to OV extends to k-OV in the same way: instead of partitioning the variables into two sets of size $n/2$, we split them into k sets of n/k, resulting in k sets of $N = 2^{n/k}$ vectors. Applying the new idea, in Section 3, we get a reduction to instances where the first $k-1$ sets are fixed, and only one set is given as input. Consequently, the k-LCS problem has an $n^{k-o(1)}$ lower bound even if $k-1$ of the sequences are fixed.

1.1.4 Hardness for compressible instances

A surprising feature of SETH-based hardness results for problems in P is that they are proved for highly compressible instances. The reductions produce OV instances of size $N = 2^{n/2}$ that are fully determined by a k-CNF formula of size $O(n^k)$ and can therefore be losslessly encoded with $O(\log N)$ bits. This is surprising because a priori one might expect the worst case instances to require $\Omega(N)$ bits to specify. The reductions can even be adapted (with major modifications) [1] to prove the hardness for instances where the data is compressible with standard grammar-compression schemes such as the Lempel-Ziv family. The new results go a step further: the hardness holds even if one of the inputs is fully determined.

The preprocessing model

A few recent papers study the complexity of the central problems of fine-grained complexity in models with preprocessing [17, 24, 35, 36, 34]. For instance, it was shown that if we can preprocess one of the strings of an Edit Distance instance in near-linear time, then we can obtain a better approximation in sublinear time [34].

It is not difficult to get strong lower bounds for OV (and Edit Distance) in these models by combining the old reduction with a partitioning trick (similar to [41]): an algorithm that solves OV in subquadratic time $n^{2-\varepsilon}$ after preprocessing each set (separately) in arbitrary polynomial time, say n^{100}, also refutes SETH, because we can split A and B into $n^{2-1/200}$ sets of size $n^{1/200}$, preprocess each in a total of $n^{1-1/200} \cdot n^{100/200} < n^{1.5}$ and then solve OV for each pair of parts in a total of $n^{2(1-1/200)} \cdot n^{1/200(2-\varepsilon)} = n^{2-\varepsilon'}$.

The new reduction gives even more power since it implies the hardness with a fixed set, that intuitively, can be preprocessed indefinitely; formally, we get conditional lower bounds even against algorithms with preprocessing using arbitrary polynomial *space*, rather than time (see Section 5).[2] This also has implications for dynamic algorithms where a preprocessing phase is often allowed [8, 37], strengthening the lower bounds from polynomial time to space. Observe that if we relax the requirements further to allow exponential space, a linear time algorithm becomes possible: using $2^{O(n)}$ space we can construct a look-up table storing the answers to all possible inputs.

1.1.5 Generalization to Formula-SAT

Hansen, Williams, and the authors [6] have observed that Edit Distance and other problems in P are even harder than OV; if we solve them in subqadratic time, not only do we solve SAT faster on CNF formulas (the simplest kind of formulas), but we also solve it on arbitrary formulas, small depth circuits, and branching programs. Thus, the quadratic lower bounds for Edit Distance and LCS (but not OV) can be based on the safer Formula-SETH (or BP-SETH or NC-SETH): the hypothesis that SAT on arbitrary formulas of size $2^{o(n)}$ cannot be solved in $(2-\varepsilon)^n$ time [6, 4, 26]. These reductions start by reducing Formula-SAT to a problem similar in spirit to OV, called Formula-Satisfying-Pair, that can then be reduced to Edit Distance. It turns out that the new idea of encoding the formula only in one set can be applied in this case as well (see Section 6), and so all the Formula-SETH lower bounds still hold if one of the sequences is fixed.

1.1.6 Roadmap and preliminaries

We start with the new reduction from k-SAT to OV in Section 2. Then, in Section 3, we generalize it to OV on $k \geq 2$ sets. In Section 4 we explain the implication for Edit Distance. Then, we discuss conditional lower bounds for the preprocessing model in Section 5. And finally, in Section 6 we extend the new reduction to formulas beyond CNF's.

We use the notation $[n] = \{1, \ldots, n\}$ and `false`, `true` for boolean truth values.

[2] These results were mentioned in [34] and credited to this work as a personal communication.

2 The new reduction: k-SAT to OV with a fixed set

In this section we give the main result of the paper: a reduction from k-SAT to the Orthogonal Vectors problem (Definition 1) where one of the two sets is fixed for all inputs of size n.

To formalize this, we define the $\text{OV}_{\mathcal{A}}$ problem, which is the same as OV but where the input set A is not given as input. Instead, if the input set B has size n, then we will always choose A to be the set $A_n \in \mathcal{A}$ where $\mathcal{A} = \{A_n\}_{n=1}^{\infty}$ is a family of vector sets, containing one set of each size n. The formal definition below also incorporates the dimension of the vectors $d(n)$ which is taken to be a function of the number of vectors n, as is standard when studying OV.

▶ **Definition 2** (The $\text{OV}_{\mathcal{A}}$ Problem). *For a family $\mathcal{A} = \{A_n\}_{n=1}^{\infty}$ of vector-sets, such that $A_n \subseteq \{0,1\}^{d(n)}$ is a set of n binary vectors of dimension $d(n)$, we define the $OV_{\mathcal{A}}$ problem as: Given a set B of n binary vectors of dimension $d(n)$ decide if there is a pair $a \in A_n, b \in B$ that are orthogonal, i.e. $\forall i \in [d(n)] : a[i] = 0 \vee b[i] = 0$.*

We can now give the main theorem of this paper, giving a quadratic lower bound for $\text{OV}_{\mathcal{A}}$ under SETH. The family $\mathcal{A}$ for which the result holds will be clarified in the proof; we remark that it is quite natural and that is easy to produce each set A_n algorithmically in linear time.

▶ **Theorem 3** (Main). *For any function $d(n) = \log^{\omega(1)} n$, there is a family of vector-sets $\mathcal{A} = \{A_n\}_{n=1}^{\infty}$ of dimension $d(n)$ such that the $OV_{\mathcal{A}}$ problem requires $n^{2-o(1)}$ under SETH. Moreover, each set A_n can be produced in $O(n \cdot d(n))$ time and log space.*

Proof. Fix integers $n, k \geq 1$, and let $\mathcal{C}_{n,k}$ be the set of width k clauses over n variables,

$$\mathcal{C}_{n,k} = \{(\ell_1 \vee \cdots \vee \ell_k) \mid \forall i \in [k] : \ell_i \in \{x_1, \ldots, x_n\} \cup \{x_1, \ldots, x_n\}\},$$

and pick an arbitrary ordering over the clauses such that $\mathcal{C}_{n,k} = \{C_1, \ldots, C_M\}$ where $M = \binom{2n}{k} = O(n^{2k})$.

We now define the set $A_N \in \mathcal{A}$ for $N = 2^{n/2}$ as follows.[3] For each truth assignment α to the variables $x_1, \ldots, x_{n/2}$, i.e. a partial assignment, we create a vector v_α. That is, $\forall \alpha \in \{\texttt{false}, \texttt{true}\}^{n/2}$ representing the assignment $x_1 = \alpha(1), \ldots, x_{n/2} = \alpha(n/2)$, we define the vector v_α as follows:

$$\forall j \in [M] : v_\alpha[j] = \begin{cases} 0, & \text{if } \alpha \text{ satisfies the clause } C_j \\ 1, & \text{otherwise.} \end{cases}$$

That is, to set the j^{th} coordinate of v_α we check if the partial assignment α already satisfies the clause C_j (the j^{th} clause in $\mathcal{C}$) and choose 0 if so, and 1 otherwise.

Notice that the dimension of the vectors is $O(n^{2k}) = \log^{O(1)} N$ and it is asymptotically dominated by $d(N) = \log^{\omega(1)} N$. Moreover, A_N can be produced in $2^{n/2} \cdot n^{2k} = O(Nd(N))$ time, and $O(n + k \log n) = O(\log N)$ space.

We are now ready to reduce k-SAT to the $\text{OV}_{\mathcal{A}}$ problem. Given a k-CNF formula ϕ on n variables $x_1, \ldots, x_n$ as input, we define a set B of $N = 2^{n/2}$ vectors as follows. For each truth assignment $\beta \in \{\texttt{false}, \texttt{true}\}^{n/2}$ to the variables $x_{n/2+1}, \ldots, x_n$, i.e. a partial assignment to the second half of the variables, we create a vector v_β such that:

$$\forall j \in [M] : v_\beta[j] = \begin{cases} 0, & \text{if either } \beta \text{ satisfies the clause } C_j, \text{ or } C_j \notin \phi \\ 1, & \text{otherwise, i.e. } \mathcal{C}_j \in \phi \text{ but } \beta \text{ does not satisfy it.} \end{cases}$$

[3] While this only defines A_N for values of N that are equal to $2^{n/2}$ with integer n, it is easy to extend these A_N's into a family $\mathcal{A}$, e.g. by padding with dummy vectors that are all 1.

Thus, while v_α considered all clauses $C_j \in \mathcal{C}$ similarly, the vectors $v_\beta \in B$ depend on the input formula ϕ and automatically set to 0 all coordinates corresponding to clauses that do not appear in ϕ.

Finally, we claim that B is a "yes" $\text{OV}_{\mathcal{A}}$ instance iff ϕ is satisfiable, and therefore if $\text{OV}_{\mathcal{A}}$ can be solved in truly subquadratic time then for all k, k-SAT can be solved in $(2^{n/2})^{2-\varepsilon} = 2^{(1-\varepsilon/2)n}$ for some $\varepsilon > 0$, refuting SETH. The correctness follows from the following claim proved below.

▷ Claim 4. There is an orthogonal pair $a \in A_N, b \in B$ iff ϕ is satisfiable.

There exist a pair $a \in A_N, b \in B$ that are orthogonal iff there are partial assignments α, β to the first and second half of the variables, respectively, such that $v_\alpha \in A_N$ and $v_\beta \in B$ are orthogonal, meaning that for all $j \in [M]$ either $v_\alpha[j] = 0$ or $v_\beta = 0$, i.e. either α satisfies C_j or β satisfies C_j or C_j is not a clause in ϕ at all. The latter is equivalent to saying that the truth assignment $(\alpha\beta)$ composed of α and β satisfies all clauses that are in ϕ, and we conclude that there is an orthogonal pair iff ϕ has a satisfying assignment. ◀

3 Extension to k-OV

In this section we extend the result of Section 2 to the k-OV problem.

▶ **Definition 5** (The k-OV Problem). *Given k sets $A_1, \ldots, A_k \subseteq \{0,1\}^{d(n)}$ of n binary vectors of dimension $d(n)$, decide if there are k vectors $a_i \in A_i$ that are orthogonal, i.e. $\forall j \in [d(n)] : a_1[j] = 0 \vee \cdots \vee a_k[j] = 0$.*

As before, we define a version of the problem in which only one of the sets is given as input and the other $k-1$ are fixed.

▶ **Definition 6** (The k-$\text{OV}_{\mathcal{A}_1,\ldots,\mathcal{A}_k}$ Problem). *For any $k-1$ families $\mathcal{A}_i = \{A_{i,n}\}_{n=1}^{\infty}, i \in [k-1]$ of vector-sets, such that $\forall i \in [k-1] : A_{i,n} \subseteq \{0,1\}^{d(n)}$ is a set of n binary vectors of dimension $d(n)$, we define the k-$OV_{\mathcal{A}_1,\ldots,\mathcal{A}_k}$ problem as: Given a set A_k of n binary vectors of dimension $d(n)$ decide if there are k vectors $\forall i \in [k-1] : a_i \in A_{i,n}$ and $a_k \in A_k$ that are orthogonal, i.e. $\forall j \in [d(n)] : a_1[j] = 0 \vee \cdots \vee a_k[j] = 0$.*

We are now ready to extend Theorem 3.

▶ **Theorem 7.** *For all $k \geq 2$ and any function $d(n) = \log^{\omega(1)} n$, there exist $k-1$ families of vector-sets $\mathcal{A}_i = \{A_{i,n}\}_{n=1}^{\infty}, i \in [k-1]$ of dimension $d(n)$ such that the k-$OV_{\mathcal{A}_1,\ldots,\mathcal{A}_k}$ problem requires $n^{k-o(1)}$ under SETH. Moreover, each set $A_{i,n}$ can be produced in $O(n \cdot d(n))$ time and log space.*

Proof. Recall from the proof of Theorem 3, the definition of $\mathcal{C}_{n,w}$ the set of all width w clauses over n variables.[4]

For all $i \in [k-1]$, we define the set $A_{i,N} \in \mathcal{A}_i$, where $N = 2^{n/k}$ as follows. For each partial assignment $\alpha^{(i)}$ to the variables $x_{(i-1)\cdot n/k+1}, \ldots, x_{i\cdot(n/k)}$, we create a vector $v_{\alpha^{(i)}}$ and add it to $A_{i,N}$. Let C_j denote the j^{th} clause in a canonical ordering of the clauses in $\mathcal{C}_{n,w}$ and define $v_{\alpha^{(i)}}$ as:

[4] Note that we changed the notation of the clause size from k to w to avoid conflict with k the number of sets. That is, we are now reducing w-SAT to k-OV.

$$\forall j \in [|\mathcal{C}_{n,w}|] : v_{\alpha^{(i)}}[j] = \begin{cases} 0, & \text{if } \alpha^{(i)} \text{ satisfies the clause } C_j \\ 1, & \text{otherwise.} \end{cases}$$

Notice that the dimension of the vectors is $O(n^{2w}) = \log^{O(1)} N$ and it is asymptotically dominated by $d(N) = \log^{\omega(1)} N$. Moreover, $A_{i,N}$ can be produced in $2^{n/k} \cdot n^{2k} = O(Nd(N))$ time, and $O(n + w \log n) = O(\log N)$ space.

We are now ready to reduce w-SAT to the k-OV$_{\mathcal{A}_1,\ldots,\mathcal{A}_k}$ problem. Given a w-CNF formula ϕ on n variables $x_1, \ldots, x_n$ as input, we define a set A_k of $N = 2^{n/k}$ vectors as follows. For each partial assignment $\alpha^{(k)}$ to the variables $x_{(k-1)\cdot n/k+1}, \ldots, x_n$, we create a vector $v_{\alpha^{(k)}}$ such that:

$$\forall j \in [|\mathcal{C}_{n,w}|] : v_{\alpha^{(i)}}[j] = \begin{cases} 0, & \text{if either } \alpha^{(k)} \text{ satisfies the clause } C_j, \text{ or } C_j \notin \phi \\ 1, & \text{otherwise, i.e. } \mathcal{C}_j \in \phi \text{ but } \alpha^{(k)} \text{ does not satisfy it.} \end{cases}$$

Thus, while $v_{\alpha^{(i)}} \in A_{i,N}$ for $i \in [k-1]$ considered all clauses $C_j \in \mathcal{C}_{n,w}$ similarly, the vectors $v_{\alpha^{(k)}} \in A_k$ depend on the input formula ϕ and automatically set to 0 all coordinates corresponding to clauses that do not appear in ϕ.

Finally, we claim that A_k is a "yes" k-OV$_{\mathcal{A}_1,\ldots,\mathcal{A}_k}$ instance iff ϕ is satisfiable, and therefore if k-OV$_{\mathcal{A}_1,\ldots,\mathcal{A}_k}$ can be solved faster than $n^{k-o(1)}$ time then for all w, w-SAT can be solved in $(2^{n/k})^{k-\varepsilon} = 2^{(1-\varepsilon/k)n}$ for some $\varepsilon > 0$, refuting SETH. The correctness follows from the following claim, which can be proved similarly to Claim 4.

▷ Claim 8. There exist k-orthogonal vectors $a_i \in A_{i,N}$ for $i \in [k-1]$ and $a_k \in A_k$ iff ϕ is satisfiable. ◀

4 Corollaries for Edit Distance and other problems

The hardness of OV$_{\mathcal{A}}$ has immediate consequences to all the many OV-hard problems, establishing their hardness even in restricted settings. Let us formalize the implication for Edit Distance, and briefly remark on how it applies to the other problems.

▶ **Definition 9** (The Edit Distance Problem). *Given two sequences X, Y of length n, return the minimum number operations that can transform X into Y. The allowed edit-operations are insertions, deletions, and substitutions of single characters.*

In a similar way to our definition of OV$_{\mathcal{A}}$ in Section 2, we formalize a restricted problem where only one of the two sequences is given as input.

▶ **Definition 10** (The ED$_{\mathcal{X}}$ Problem). *For a family $\mathcal{X} = \{X_n\}_{n=1}^{\infty}$ of sequences, such that X_n is a sequence of length n, we define the ED$_{\mathcal{X}}$ problem as: Given a sequence Y of length n, return the minimum number edit-operations that can transform X_n into Y.*

We can now prove the statement in the title of the paper, showing a quadratic SETH lower bound for ED$_{\mathcal{X}}$.

▶ **Theorem 11.** *There is a family of sequences $\mathcal{X} = \{X_n\}_{n=1}^{\infty}, |X_n| = n$ such that the ED$_{\mathcal{X}}$ problem requires $n^{2-o(1)}$ under SETH. Moreover, each sequence X_n can be produced in $O(n)$ time and log space.*

Proof. By Theorem 3 it is enough to reduce the $\mathrm{OV}_{\mathcal{A}}$ problem to $\mathrm{ED}_{\mathcal{X}}$. To do that, we start with an $\mathrm{OV}_{\mathcal{A}}$ instance B of size n and dimension $d(n) = n^{o(1)}$, for the family $\mathcal{A}$ constructed by Theorem 3, and we apply the reduction of Backurs and Indyk [14] (or the later one which uses a smaller alphabet [22]) to B and A_n. We get two sequences $X = S_X(A_n)$ and $Y = S_Y(B)$ of length $N = f(n) = O(n \cdot d(n))$, for some specific function $f : \mathbb{N} \to \mathbb{N}$, such that the Edit Distance between X and Y is less than a certain value τ_n iff A, B contains an orthogonal pair. Moreover, the encodings S_X and S_Y have the property that they take linear time and log space to compute, and most importantly, that S_X only depends on A_n and n but not on B. Therefore, we can construct the family of sequences $\mathcal{X}$ by setting $X_N = S_X(A_n)$ where $N = f(n)$. We get that solving $\mathrm{ED}_{\mathcal{X}}$ in $O(N^{2-\varepsilon})$ time, for some $\varepsilon > 0$, leads to an $O(n^{2-\varepsilon'})$ time algorithm for solving $\mathrm{OV}_{\mathcal{A}}$, for some $\varepsilon' > 0$, refuting SETH. ◀

To get the corollaries for LCS, k-LCS, Subtree Isomorphism, and the other problems mentioned in Section 1.1, the same arguments work: we simply have to check that the reductions from OV operate on each set separately.

5 Lower bounds for the Preprocessing model

In this section we present the implications of the new reductions for the limitations of preprocessing. As discussed in Section 1.1, any quadratic lower bound for OV already implies a quadratic lower bound even if the algorithm is allowed to preprocess one of the sets in arbitrary polynomial time. However, having a reduction from k-SAT to OV with a fixed set leads to stronger conditional lower bounds that address algorithms with arbitrary polynomial *space* preprocessing. These lower bounds are no longer based on SETH but on a plausible strengthening of it to *nonuniform* algorithms.

Recall [13] that $TIME[T(n)]/A(n)$ is the class of problems that can be solved by an $O(T(n))$ time algorithm that is given an advice string X_n of length $O(A(n))$ for all inputs of size n.

▶ **Hypothesis 12** (Nonuniform-SETH). *There is no $\varepsilon > 0$ such that for all $k \geq 3$ the k-SAT problem is in $TIME[(2-\varepsilon)^n]/2^{(1+o(1))\cdot n/2}$.*

This is a strong hypothesis, but it is not at all clear how the nonuniformity can help in solving SAT faster, and therefore it might be as plausible as SETH. Moreover, even the extreme version of this hypothesis, where the size of the advice is increased from $2^{n/2\cdot(1+o(1))}$ all the way to $2^{n\cdot(1-\varepsilon)}$ remains plausible. In fact, for our conditional lower bound below, we can weaken the hardness hypothesis to allow significantly smaller advice length: $O(2^{\varepsilon n})$ for any constant $\varepsilon > 0$, and the same conclusion for OV would follow. To obtain this stronger result it suffices to give a modification to Theorem 3 so that it reduces to *asymmetric* $\mathrm{OV}_{\mathcal{A}}$ where the sets A and B have different sizes; the details of this standard modification are omitted from this paper. In any case, the main message of the following theorem is to highlight a connection between algorithms in the preprocessing model and a breakthrough in the nonuniform complexity of SAT.

▶ **Theorem 13.** *Suppose there is an algorithm that, given two sets A, B of n binary vectors in $d(n) = \log^{\omega(1)} n$ dimensions, can preprocess the set A using $S(n) = O(n^c)$ bits of space, and subsequently solve OV on A, B in $O(n^{2-\varepsilon})$ time, for some $\varepsilon > 0, c \geq 1$. Then, Nonuniform-SETH is false.*

Proof. Fix $\varepsilon > 0, c \geq 1$ and suppose that an algorithm ALG can solve OV in subquadratic $O(n^{2-\varepsilon})$ time after preprocessing the set A using $S(n) = n^c$ space. We start by reducing k-SAT on N variables to $\mathrm{OV}_{\mathcal{A}}$ on sets of size $n = 2^{N/2}$ using Theorem 3. Then, to solve

$\mathrm{OV}_{\mathcal{A}}$ instances of size n using ALG we can take the set A_n and split it into $n^{1-1/tc}$ instances of size $n^{1/tc}$ each, for some $t \geq 2$ to be specified later. We use ALG to preprocess each of these parts in an unknown amount of time but only $O((n^{1/tc})^c)$ space; the total space that it uses for all the parts is $s_n = n^{1-1/tc} \cdot O(n^{c/tc}) = O(n^{1+1/t-1/tc})$. Let X_n be the bit-string of length s_n that encodes the state of the memory after the algorithm is done preprocessing these sets, i.e. the concatenation of the $n^{1-1/tc}$ memory tapes. Observe, crucially, that the string X_n only depends on n, even though generating it might have take an unknown amount of time. We will choose it to be the advice string for our algorithm for all inputs of size n. Then, using the string X_n our algorithm can solve any $\mathrm{OV}_{\mathcal{A}}$ instance B of size n in truly subquadratic time: we split B into $n^{1-1/tc}$ parts of size $n^{1/tc}$ each, and then solve each pair of parts using ALG and the string X_n in subquadratic $(n^{1/tc})^{2-\varepsilon}$ time, giving a total of $(n^{1-1/tc})^2 \cdot (n^{1/tc})^{2-\varepsilon} = n^{2-\varepsilon'}$ time, for $\varepsilon' = \varepsilon/tc > 0$. Going back to k-SAT, our algorithm runs in time $2^{N/2\cdot(2-\varepsilon')}$ time and has used $O(n^{1+1/t}) = O(2^{N/2\cdot(1+1/t)}) = 2^{N/2\cdot(1+o(1))}$ bits of advice, where the latter equality holds because we can choose t to be arbitrarily large, refuting Nonuniform-SETH. ◀

6 Extension to Formula-SAT and Formula-Satisfying-Pair

In this section, we extend the results of Section 2 so that the starting point is Formula-SAT rather than CNF-SAT, and the end problem is Formula-Satisfying-Pair (with a fixed formula and set A) rather than OV (with a fixed set A). Throughout, we consider deMorgan formulas that have AND/OR gates of fan-in two, and we assume that all the NOT gates are at the bottom, meaning that the leaves of each formula are either variables or their negation. The size of a formula is the total number of gates and the depth is the maximum number of levels from root to leaf.

▶ **Definition 14** (Formula-SAT). *Given a formula F over n variables, decide if it is satisfiable.*

Notice that CNF formulas are a special kind of depth 2 formulas. The following hypothesis is a more believable version of SETH.

▶ **Hypothesis 15** (Formula-SETH [6]). *There is no $\varepsilon > 0$ such that Formula-SAT on formulas of size $2^{o(n)}$ can be solved in $O((2-\varepsilon)^n)$ time.*

This hypothesis is sometimes referred to as Branching-Program-SETH (BP-SETH) since it is equivalent to a hypothesis about SAT on branching programs, or as NC-SETH which is a similar assumption about SAT on polylog depth circuits, which are equivalent to formulas of $2^{\mathrm{poly}\log n}$ size.

Just like the SETH-based lower bounds go via the OV problem, the Formula-SETH lower bounds often go via a problem such as the Formula-Satisfying-Pair problem.

▶ **Definition 16** (Formula-Satisfying-Pair [4]). *Given a formula $F = F(x_1, \ldots, x_m, y_1, \ldots, y_m)$ of size $2m$ where each variable is used exactly once, and two sets $A, B \subseteq \{0,1\}^m$ of size n, decide whether there is a pair $a \in A, b \in B$ such that $F(a_1, \ldots, a_m, b_1, \ldots, b_m) =$ `true`.*

A simple reduction, similar to the one by Williams [42], shows an $n^{2-o(1)}$ lower bound for Formula-Satisfying-Pair with $m = n^{o(1)}$ under the Formula-SETH. And with intricate gadgetry, Formula-Satisfying-Pair can be reduced to Edit Distance, LCS, Fréchet, and other problems establishing Formula-SETH lower bounds for them as well [6, 4]. The main result of this section is to prove a Formula-SETH lower bound for Formula-Satisfying-Pair with a fixed formula F and a fixed set A. As a result, the Formula-SETH lower bounds for Edit Distance and the other problems also hold when one sequence is fixed.

▶ **Definition 17** ($\mathsf{FSP}_{\mathcal{F},\mathcal{A}}$). *For a family $\mathcal{A} = \{A_n\}_{n=1}^{\infty}$ of vector-sets, such that $A_n \subseteq \{0,1\}^{d(n)}$ is a set of n binary vectors of dimension $d(n)$, and a family $\mathcal{F} = \{F_n\}_{n=1}^{\infty}$ of formulas, such that F_n is over $d(n)$ variables and has size $2d(n)$ we define the $FSP_{\mathcal{F},\mathcal{A}}$ problem as: Given a set B of n binary vectors of dimension $d(n)$ decide if there is a pair $a \in A_n, b \in B$ such that $F_n(a,b) =$ `true`.*

After formalizing the problem with fixed formula and set A we are ready to state the theorem. The rest of this section is dedicated to the proof.

▶ **Theorem 18.** *There is a family of vector-sets $\mathcal{A} = \{A_n\}_{n=1}^{\infty}$ of dimension $d(n) = n^{o(1)}$ and a family of formulas $\mathcal{F} = \{F_n\}_{n=1}^{\infty}$ over $d(n)$ variables and of size $2d(n)$ such that the $FSP_{\mathcal{F},\mathcal{A}}$ problem requires $n^{2-o(1)}$ under Formula-SETH. Moreover, each set A_n and formula F_n can be produced in $O(n \cdot d(n))$ time.*

Our approach is to imitate the main idea in the proof of Theorem 2. There, we looked at the set of all clauses, which is similar to thinking about the super-set of all k-CNF formulas. Now, we are faced with arbitrary formulas, and it is not so clear what the corresponding super-set would be: the set of all gates does not make much sense. To make this work, we go through the following intermediate problem, which is similar to Formula-SAT but has a structure that is easier to work with when constructing a fixed formula for $\mathrm{FSP}_{\mathcal{F},\mathcal{A}}$.

For a depth bound $depth(n)$ we define the *Canonical-Depth-$d(n)$* formula to be the formula over n variables defined by a full binary tree in which all the gates are (?) indicating a gate that could either be AND or OR. Moreover, each leaf pointing to a variable x_i is also labelled with a (?) indicating that it could either be x_i or $\bar{x_i}$. We also fix a canonical numbering of the $s(n) = 2^{d(n)}$ gates of this formula. To get a "real" formula, we must specify $s(n)$ bits indicating for each (?) gate whether it is AND, OR, or if it is a leaf whether it is a negation or not. A natural way to make the specification is to make the j^{th} gate AND if the j^{th} bit is 1 and OR otherwise, and to make a leaf-gate a negation iff the corresponding bit is 1.

▶ **Definition 19** (Canonical-Formula-SAT)**.** *Given $s(n)$ bits for specifying the gates of a canonical-depth-$d(n)$ formula over n variables, decide if the resulting formula is satisfiable.*

The reduction has two steps, described in Sections 6.1 and 6.2.

6.1 From Formula-SAT to Canonical-Formula-SAT

This step is not immediate only because an arbitrary $s(n)$-sized formula could have a structure that is very far from a full-binary tree. Nonetheless, we can transform it into such using standard techniques without blowing up the size by more than polynomial factors. And since our interest is in $s(n) = 2^{o(n)}$, polynomial blowups do not matter.

Given a formula F of size $s(n)$ we begin by applying the depth-reduction of Bonet-Buss [18] to get an equivalent formula F' of depth $depth(n) = O(\log s(n))$. Then, we enforce that all paths from root to leaves have length exactly $d(n)$: if a leaf is higher, we add an equivalent subtree, e.g. by replacing x_i with $(x_i \wedge \mathtt{true}) \wedge (\mathtt{true} \wedge \mathtt{true})$ and so on. The total size of the final formula F'' is $2^{depth(n)} = s^{O(1)}$.

Then, to complete the reduction, we simply go over the gates of F'' and generate a string g of $s(n)$ bits that encodes it with the above natural representation. Thus, g is a "yes" instance for Canonical-Formula-SAT iff F is satisfiable.

6.2 From Canonical-Formula-SAT to $\text{FSP}_{\mathcal{F},\mathcal{A}}$

The next idea is to transform the canonical formula $F^{(?)}$ that is full of (?) gates along with a bit-string g specifying the gates, into a formula F with real gates that takes the bits of g as inputs. In more details, we take the j^{th} gate in $F^{(?)}$, that takes input from the two gates G_1 and G_2 and we replace it with the following subformula (a similar but different formula is used for leaf-gates):

$$G_j = (g_j \wedge (G_1 \wedge G_2)) \vee (\bar{g_j} \wedge (G_1 \vee G_2))$$

This subformula encodes our natural representation where $g_j = 1$ iff the gate is AND. After these transformations, the size of the formula blows up by $2^{depth(n)}$ since we have to make two copies of each subformula at every level, but this is still $s^{O(1)}$. Notice that the formula F is now fixed, and the formula we started from is only encoded with the g_j's which can be thought of as inputs to F (note that, as opposed to the x_i's these inputs are not free, and so they do not increase the complexity of SAT).

We are now ready to define the $\text{FSP}_{\mathcal{F},\mathcal{A}}$ instances that encode the satisfiability of F. Let $N = 2^{n/2}$. We define the formula F_N, i.e. the N^{th} member of the family of formulas $\mathcal{F}$, to be equal to F after we duplicate each variable so that it is used once in the formula. The vectors $a \in A_N$ have dimension $O(s(n)) = N^{o(1)}$ and are defined as follows. For each partial assignment $\alpha \in \{\texttt{true}, \texttt{false}\}^{n/2}$ to the variables $x_1, \ldots, x_{n/2}$ we define the vector a such that for all $j \in [s(n)]$ a_j is set to 1 iff the j^{th} gate in F is a leaf gate marked with a literal x_h or $\bar{x_h}$ and α makes this literal true. Note that the vectors in A do not depend on the g_j's at all.

Finally, the vectors $b \in B$ do depend on the g_j's. For each partial assignment β to the variables $x_{n/2+1}, \ldots, x_n$, we construct a vector b. For all $j \in [s(n)]$ we set b_j as follows. If the j^{th} gate in F is a leaf gate marked with a literal x_h or $\bar{x_h}$ and β makes this literal true, then we set $b_j = 1$. If the j^{th} gate is a leaf gate marked with a variable g_h (or its negation) then we set $b_j = g_h$ (or its negation). Notice that the g's affect all vectors in B in the same way (as is the case in the proof of Theorem 3).

To conclude the correctness of this reduction, observe that an evaluation of F_N on a pair of vectors a, b is equivalent to the evaluation of F on the corresponding partial assignments α, β and the given g.

References

1. Amir Abboud, Arturs Backurs, Karl Bringmann, and Marvin Künnemann. Fine-grained complexity of analyzing compressed data: Quantifying improvements over decompress-and-solve. In *58th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2017, Berkeley, CA, USA, October 15-17, 2017*, pages 192–203, 2017.
2. Amir Abboud, Arturs Backurs, Thomas Dueholm Hansen, Virginia Vassilevska Williams, and Or Zamir. Subtree isomorphism revisited. In *Proceedings of the Twenty-Seventh Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2016, Arlington, VA, USA, January 10-12, 2016*, pages 1256–1271, 2016.
3. Amir Abboud, Arturs Backurs, and Virginia Vassilevska Williams. Tight hardness results for lcs and other sequence similarity measures. In *2015 IEEE 56th Annual Symposium on Foundations of Computer Science*, pages 59–78. IEEE, 2015.
4. Amir Abboud and Karl Bringmann. Tighter connections between formula-sat and shaving logs. *arXiv preprint*, 2018. `arXiv:1804.08978`.
5. Amir Abboud, Karl Bringmann, Holger Dell, and Jesper Nederlof. More consequences of falsifying SETH and the orthogonal vectors conjecture. In *Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2018, Los Angeles, CA, USA, June 25-29, 2018*, pages 253–266, 2018.

6 Amir Abboud, Thomas Dueholm Hansen, Virginia Vassilevska Williams, and Ryan Williams. Simulating branching programs with edit distance and friends: or: a polylog shaved is a lower bound made. In *Proceedings of the forty-eighth annual ACM symposium on Theory of Computing*, pages 375–388, 2016.

7 Amir Abboud, Aviad Rubinstein, and Ryan Williams. Distributed pcp theorems for hardness of approximation in p. In *2017 IEEE 58th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 25–36. IEEE, 2017.

8 Amir Abboud and Virginia Vassilevska Williams. Popular conjectures imply strong lower bounds for dynamic problems. In *55th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2014, Philadelphia, PA, USA, October 18-21, 2014*, pages 434–443, 2014.

9 Amir Abboud, Virginia Vassilevska Williams, and Oren Weimann. Consequences of faster alignment of sequences. In *Automata, Languages, and Programming - 41st International Colloquium, ICALP 2014, Copenhagen, Denmark, July 8-11, 2014, Proceedings, Part I*, pages 39–51, 2014.

10 Amir Abboud, Virginia Vassilevska Williams, and Huacheng Yu. Matching triangles and basing hardness on an extremely popular conjecture. *SIAM Journal on Computing*, 47(3):1098–1122, 2018.

11 Amir Abboud, Ryan Williams, and Huacheng Yu. More applications of the polynomial method to algorithm design. In *Proceedings of the twenty-sixth annual ACM-SIAM symposium on Discrete algorithms*, pages 218–230. SIAM, 2014.

12 Josh Alman and Ryan Williams. Probabilistic polynomials and hamming nearest neighbors. In *2015 IEEE 56th Annual Symposium on Foundations of Computer Science*, pages 136–150. IEEE, 2015.

13 Sanjeev Arora and Boaz Barak. *Computational complexity: a modern approach*. Cambridge University Press, 2009.

14 Arturs Backurs and Piotr Indyk. Edit distance cannot be computed in strongly subquadratic time (unless SETH is false). In *Proceedings of the Forty-Seventh Annual ACM on Symposium on Theory of Computing, STOC 2015, Portland, OR, USA, June 14-17, 2015*, pages 51–58, 2015.

15 Arturs Backurs and Piotr Indyk. Which regular expression patterns are hard to match? In *2016 IEEE 57th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 457–466. IEEE, 2016.

16 Arturs Backurs, Liam Roditty, Gilad Segal, Virginia Vassilevska Williams, and Nicole Wein. Towards tight approximation bounds for graph diameter and eccentricities. In *Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing*, pages 267–280, 2018.

17 Nikhil Bansal and Ryan Williams. Regularity lemmas and combinatorial algorithms. In *2009 50th Annual IEEE Symposium on Foundations of Computer Science*, pages 745–754. IEEE, 2009.

18 Maria Luisa Bonet and Samuel R Buss. Size-depth tradeoffs for boolean formulae. *Information Processing Letters*, 49(3):151–155, 1994.

19 Karl Bringman and Marvin Künnemann. Multivariate fine-grained complexity of longest common subsequence. In *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1216–1235. SIAM, 2018.

20 Karl Bringmann. Why walking the dog takes time: Frechet distance has no strongly subquadratic algorithms unless SETH fails. In *55th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2014, Philadelphia, PA, USA, October 18-21, 2014*, pages 661–670, 2014.

21 Karl Bringmann, Allan Grønlund, and Kasper Green Larsen. A dichotomy for regular expression membership testing. In *2017 IEEE 58th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 307–318. IEEE, 2017.

22 Karl Bringmann and Marvin Künnemann. Quadratic conditional lower bounds for string problems and dynamic time warping. In Venkatesan Guruswami, editor, *IEEE 56th Annual Symposium on Foundations of Computer Science, FOCS 2015, Berkeley, CA, USA, 17-20 October, 2015*, pages 79–97. IEEE Computer Society, 2015. doi:10.1109/FOCS.2015.15.

23 Karl Bringmann, Marvin Künnemann, and André Nusser. Fréchet distance under translation: conditional hardness and an algorithm via offline dynamic grid reachability. In *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 2902–2921. SIAM, 2019.

24 Timothy M Chan and Moshe Lewenstein. Clustered integer 3sum via additive combinatorics. In *Proceedings of the forty-seventh annual ACM symposium on Theory of computing*, pages 31–40, 2015.

25 Timothy M Chan and Ryan Williams. Deterministic apsp, orthogonal vectors, and more: Quickly derandomizing razborov-smolensky. In *Proceedings of the twenty-seventh annual ACM-SIAM symposium on Discrete algorithms*, pages 1246–1255. SIAM, 2016.

26 Lijie Chen, Shafi Goldwasser, Kaifeng Lyu, Guy N Rothblum, and Aviad Rubinstein. Fine-grained complexity meets ip= pspace. In *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1–20. SIAM, 2019.

27 Lijie Chen and Ryan Williams. An equivalence class for orthogonal vectors. In Timothy M. Chan, editor, *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2019, San Diego, California, USA, January 6-9, 2019*, pages 21–40. SIAM, 2019. doi:10.1137/1.9781611975482.2.

28 Karthik CS, Roee David, and Bundit Laekhanukit. On the complexity of closest pair via polar-pair of point-sets. *SIAM Journal on Discrete Mathematics*, 33(1):509–527, 2019.

29 Karthik CS, Bundit Laekhanukit, and P Manurangsi. On the parameterized complexity of approximating dominating set. In *Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing, STOC*, pages 1283–1296, 2018.

30 Karthik CS and Pasin Manurangsi. On closest pair in euclidean metric: Monochromatic is as hard as bichromatic. *ITCS*, 2019.

31 Mina Dalirrooyfard, Andrea Lincoln, and Virginia Vassilevska Williams. New techniques for proving fine-grained average-case hardness. In *61st IEEE Annual Symposium on Foundations of Computer Science, FOCS 2020, Durham, NC, USA, November 16-19, 2020*, pages 774–785. IEEE, 2020. doi:10.1109/FOCS46700.2020.00077.

32 Lech Duraj, Marvin Künnemann, and Adam Polak. Tight conditional lower bounds for longest common increasing subsequence. *Algorithmica*, 81(10):3968–3992, 2019.

33 Jiawei Gao, Russell Impagliazzo, Antonina Kolokolova, and Ryan Williams. Completeness for first-order properties on sparse structures with algorithmic applications. *ACM Trans. Algorithms*, 15(2):23:1–23:35, 2019. doi:10.1145/3196275.

34 Elazar Goldenberg, Aviad Rubinstein, and Barna Saha. Does preprocessing help in fast sequence comparisons? In *Proceedings of the 52nd Annual ACM SIGACT Symposium on Theory of Computing*, pages 657–670, 2020.

35 Isaac Goldstein, Tsvi Kopelowitz, Moshe Lewenstein, and Ely Porat. Conditional lower bounds for space/time tradeoffs. In *Workshop on Algorithms and Data Structures*, pages 421–436. Springer, 2017.

36 Alexander Golovnev, Siyao Guo, Thibaut Horel, Sunoo Park, and Vinod Vaikuntanathan. Data structures meet cryptography: 3sum with preprocessing. In *Proceedings of the 52nd Annual ACM SIGACT Symposium on Theory of Computing*, pages 294–307, 2020.

37 Monika Henzinger, Sebastian Krinninger, Danupon Nanongkai, and Thatchaphol Saranurak. Unifying and strengthening hardness for dynamic problems via the online matrix-vector multiplication conjecture. In *Proceedings of the Forty-Seventh Annual ACM on Symposium on Theory of Computing, STOC 2015, Portland, OR, USA, June 14-17, 2015*, pages 21–30, 2015.

38 Robert Krauthgamer and Ohad Trabelsi. Conditional lower bounds for all-pairs max-flow. *ACM Transactions on Algorithms (TALG)*, 14(4):1–15, 2018.

39 Liam Roditty and Virginia Vassilevska Williams. Fast approximation algorithms for the diameter and radius of sparse graphs. In *Symposium on Theory of Computing Conference, STOC'13, Palo Alto, CA, USA, June 1-4, 2013*, pages 515–524, 2013.

40 Virginia Vassilevska Williams. On some fine-grained questions in algorithms and complexity. In *Proceedings of the ICM*, 2018.

41 Virginia Vassilevska Williams and R. Ryan Williams. Subcubic equivalences between path, matrix, and triangle problems. *J. ACM*, 65(5):27:1–27:38, 2018. doi:10.1145/3186893.

42 Ryan Williams. A new algorithm for optimal 2-constraint satisfaction and its implications. *Theor. Comput. Sci.*, 348(2-3):357–365, 2005. doi:10.1016/j.tcs.2005.09.023.

Local Approximations of the Independent Set Polynomial

Dimitris Achlioptas
Department of Informatics & Telecommunications, University of Athens, Greece

Kostas Zampetakis
Department of Computer Science & Engineering, University of California Santa Cruz, CA, USA

Abstract

The independent set polynomial of a graph has one variable for each vertex and one monomial for each independent set, comprising the product of the corresponding variables. Given a graph G on n vertices and a vector $\boldsymbol{p} \in [0,1)^n$, a central problem in statistical mechanics is determining whether the independent set polynomial of G is non-vanishing in the polydisk of $\boldsymbol{p}$, i.e., whether $|Z_G(\boldsymbol{x})| > 0$ for every $\boldsymbol{x} \in \mathbb{C}^n$ such that $|x_i| \leq p_i$. Remarkably, when this holds, $Z_G(-\boldsymbol{p})$ is a lower bound for the avoidance probability when G is a dependency graph for n events whose probabilities form vector $\boldsymbol{p}$. A *local* sufficient condition for $|Z_G| > 0$ in the polydisk of $\boldsymbol{p}$ is the Lovász Local Lemma (LLL).

In this work we derive several new results on the efficient evaluation and bounding of Z_G. Our starting point is a monotone mapping from subgraphs of G to truncations of the tree of self-avoiding walks of G. Using this mapping our first result is a local *upper* bound for $Z(-\boldsymbol{p})$, similar in spirit to the local lower bound for $Z(-\boldsymbol{p})$ provided by the LLL. Next, using this mapping, we show that when G is chordal, Z_G can be computed exactly and in linear time on the entire complex plane, implying *perfect* sampling for the hard-core model on chordal graphs. We also revisit the task of bounding $Z(-\boldsymbol{p})$ from below, i.e., the LLL setting, and derive four new lower bounds of increasing sophistication. Already our simplest (and weakest) bound yields a strict improvement of the famous asymmetric LLL, i.e., a strict relaxation of the inequalities of the asymmetric LLL without any further assumptions. This new asymmetric local lemma is sharp enough to recover Shearer's *optimal* bound in terms of the maximum degree $\Delta(G)$. We also apply our more sophisticated bounds to estimate the zero-free region of the hard-core model on the triangular lattice (hard hexagons model).

2012 ACM Subject Classification Mathematics of computing → Combinatorics; Theory of computation → Dynamic programming

Keywords and phrases Independent Set Polynomial, Lovász Local Lemma, Self-avoiding Walks

Digital Object Identifier 10.4230/LIPIcs.ICALP.2021.8

Category Track A: Algorithms, Complexity and Games

Acknowledgements We are deeply grateful to the anonymous reviewers for their extremely thorough review of our submission and for an abundance of insightful comments and suggestions that greatly clarified the presentation and motivated the improvement of some of the results.

1 The Independent Set Polynomial

We write $[n]$ to denote the set $\{1, 2, \ldots, n\}$, with the convention $[0] = \emptyset$. Throughout, G is a graph on $[n]$ and $\mathrm{Ind}(G)$ denotes the set of all independent sets of G.

▶ **Definition 1.** *The* independent set polynomial *of a graph G with variables $x_1, \ldots, x_n$ is*

$$Z_G(\boldsymbol{x}) = Z(\boldsymbol{x}) := \sum_{\substack{I \subseteq S \\ I \in \mathrm{Ind}(G)}} \prod_{i \in I} x_i \ . \tag{1}$$

For arbitrary $S \subseteq [n]$ we denote the independent set polynomial of the subgraph of G induced by S by $Z_G(\boldsymbol{x}; S) = Z(S)$, the latter notation relying on $\boldsymbol{x}$ being clear from context.

▶ Remark 2. We will often refer to the components of the variable vector $\boldsymbol{x}$ as *activities*.

▶ **Definition 3.** *The* polydisk *of* $\boldsymbol{p} \in [0,\infty)^n$ *is the set* $\{\boldsymbol{x} \in \mathbb{C}^n : |x_i| \le p_i \text{ for all } i \in [n]\}$.

The complexity of computing and approximating the independent set polynomial is an extensively studied subject. This is because there are important instantiations of the polynomial when the activities are positive reals, negative reals, and even complex numbers.

1.1 Positive Reals: The Hard-Core Model

In many natural computational problems in combinatorics, statistics, and statistical physics we are given as input a graph G that defines a set $\Omega = \Omega(G)$ of objects (configurations) of interest, e.g., matchings in G. A weight function $w : \Omega \to (0, +\infty)$ assigns a positive weight to each element $\sigma \in \Omega$, giving rise to a probability distribution $\pi(\sigma) = w(\sigma)/Z$, where the normalizing factor $Z := \sum_{\sigma\in\Omega} w(\sigma)$ is called the *partition function*. When $\Omega = \mathrm{Ind}(G)$ and each $I \in \mathrm{Ind}(G)$ has weight $w(I) = \prod_{i\in I} x_i$, where $\boldsymbol{x} \in (0,+\infty)^n$, the distribution is the *hard-core* model of statistical physics, and the independent set polynomial when $S = [n]$ equals its partition function. Observe that in the *univariate* case where all vertex activities equal $x > 0$, i.e., $\boldsymbol{x} = x\mathbf{1}$, as $x \to \infty$ the polynomial is increasingly dominated by the contribution of the largest independent sets, readily suggesting the intractability of evaluating the polynomial for arbitrarily large values of x. A celebrated achievement in this area is the characterization of the computational tractability of approximating the univariate partition function. Let $\Delta = \Delta(G)$ denote the maximum degree of G, let $\boldsymbol{x} = x\mathbf{1}$, and let

$$x_c = x_c(\Delta) := \frac{(\Delta-1)^{\Delta-1}}{(\Delta-2)^{\Delta}} \searrow \frac{\mathrm{e}}{\Delta} \ ,$$

where $\searrow$ denotes convergence from above. Weitz [22] proved the partition function can be approximated arbitrary well (FPTAS) for $x < x_c$, while Sly and Sun [19] proved that approximating the partition function is NP-hard for $x > x_c$.

1.2 Complex Numbers: Phase Transitions

The study of partition functions when the arguments of the corresponding polynomial are complex numbers dates back to the 1952 work of Lee and Yang [23] who established a connection between the location of zeros of the partition function on the complex plane and the presence of phase transitions on the real axis. The high-level idea is that since we identify phase transitions as discontinuities in the derivatives of free energy, i.e., of $\log Z$, such a transition can only occur at a point of the complex plane if there is at least one nearby zero of the partition function. Specifically, in the follow-up paper [12], Lee and Yang instantiated this connection for the ferromagnetic Ising model by proving that the zeros of the partition function always lie on the unit circle in the complex plane, and used this fact to conclude that the ferromagnetic Ising model can have at most one phase transition. The Lee-Yang approach has since become a cornerstone of the study of phase transitions, and has been used extensively in the statistical physics literature: see, e.g., [2, 9, 13, 21] for specific examples, and Ruelle's book [16] for background. There have also been attempts to relate the Lee-Yang program to the Riemann hypothesis [14].

Zeros of partition functions when the variables take complex values have also been studied in a purely combinatorial setting without reference to the physical interpretation: see, for example, Choe et al. [6]. Another important example is the work of Chudnovsky and Seymour [7], who show that the zeros of the univariate independent set polynomial of

claw-free graphs lie on the real line. Finally, in a seminal work, Scott and Sokal [17] proved that the independent set polynomial is non-zero in the polydisk of $\boldsymbol{p} \in [0,1)^n$, if and only if $Z_G(-\lambda \boldsymbol{p}) > 0$ for every $\lambda \in [0,1]$.

1.3 The Probabilistic Method and the Lovász Local Lemma

The *Probabilistic Method* [1] amounts to establishing the existence of mathematical objects with a property of interest by demonstrating a probability distribution under which they have positive probability. The power of the method stems from the fact that if the probability of the objects under the distribution is indeed positive, then *any* multiplicative underestimation of it is enough to imply existence. Typically, the property of interest, $\mathcal{P}$, is the intersection of the complements of several simpler properties, each property expressing some particular "flaw," so that $\mathcal{P}$ coincides with flawlessness. Thus, if we endow a universe of candidate objects Ω, where sets $\{F_i\}_{i=1}^n \subseteq \Omega$ correspond to the different flaws, with a probability measure μ, the goal is to prove that the *avoidance probability*, $\mu(\bigcap_{i\in[n]} \overline{F}_i)$, is strictly positive.

Given only the marginals $p_i := \mu(F_i)$, the best lower bound we can give for the avoidance probability is $1 - \sum_i p_i$ since, for all we know, the flaws could be disjoint. To improve upon the union bound, we need to constrain the flaw overlaps. A natural and extremely successful way to do this is in terms of a graph G on $[n]$. Concretely, let $\Gamma_i(G) = \Gamma_i$ denote the neighborhood of vertex i in G and let $\Gamma_i^+ = \Gamma_i \cup \{i\}$. We say that G is a *dependency graph* for $\{F_i\}_{i=1}^n$ with respect to μ if for every $i \in [n]$ and every set $\{j_1, j_2, \ldots\} \subseteq [n] \setminus \Gamma_i^+$,

$$\mu(F_i \mid \overline{F_{j_1}} \cap \overline{F_{j_2}} \cap \cdots) = \mu(F_i) = p_i \ . \tag{2}$$

Note that the presence of an edge in G does not prescribe any specific kind of dependency between its two corresponding events, only a lack of constraint thereof. Thus, a complete dependency graph (clique) conveys no information at all about how the n events overlap, while an empty dependency graph implies that the n events are mutually independent.

In applications, given the measure μ and the sets of flaws $\{F_i\}_{i=1}^n$ it is typically difficult to derive much more than a vector $\boldsymbol{p} = (p_1, p_2, \ldots, p_n)$ of (upper bounds for) the flaw probabilities and a (possibly pessimistic) dependency graph G. As a result, it is desirable to have sufficient conditions for a pair $\boldsymbol{p}, G$ to have the property that *every* probability measure compatible with it has strictly positive avoidance probability. Remarkably, Shearer [18] gave a sufficient *and necessary* condition for a pair to have this property.

▶ **Definition 4.** *Given a graph G, let $\mathcal{S}(G) = \{\boldsymbol{p} \in [0,1)^n : Z_G(-\boldsymbol{p}; S) > 0$, for all $S \subseteq [n]\}$.*

Shearer showed that membership in $\mathcal{S}(G)$ characterizes the vectors $\boldsymbol{p}$ for which every probability measure compatible with $\boldsymbol{p}, G$ has strictly positive avoidance probability. For this he showed that given $\boldsymbol{p}, G$, in order to minimize the avoidance probability, one should try to realize the (unique) measure μ^* under which events adjacent in G are disjoint. He then showed that if $Z(-\boldsymbol{p}; S) \leq 0$ for some $S \subseteq [n]$, then μ^* can not be realized and the avoidance probability is 0, while, otherwise, $\mu^*(\bigcap_{i\in S} \overline{F}_i) = Z(-\boldsymbol{p}; S) \geq 0$ for every $S \subseteq [n]$ and, therefore, the avoidance probability is at least $Z(-\boldsymbol{p}; [n])$, i.e., the value of the independent set polynomial of G at $-\boldsymbol{p}$. Unfortunately, performing this evaluation is generally intractable, as it involves a summation over $\mathrm{Ind}(G)$.

The Lovász Local Lemma is a *sufficient* condition for membership in $\mathcal{S}(G)$, along with a lower bound for $Z(-\boldsymbol{p})$. Below is a general formulation (the so-called *asymmetric*).

▶ **Theorem 5** (Lovász [20]). *Let μ be a probability measure on set Ω and let G be a dependency graph for $\{F_i\}_{i\in[n]} \subseteq \Omega$. If there exist $r_1, r_2, \ldots, r_n \in [0,1)$ such that for every $i \in [n]$,*

$$p_i \leq r_i \prod_{j\in\Gamma_i} (1-r_j) \ , \tag{3}$$

then $\mu\left(\bigcap_{i=1}^{n} \overline{F_i}\right) \geq \prod_{i\in[n]} (1-r_i) > 0.$

▶ Remark 6. Theorem 5 holds (and is known as the "Lopsided LLL" of Erdős and Spencer [8]) if condition (2) holds with "$\leq$" instead of "$=$". All our results also hold in that setting.

2 Our Results

2.1 An Improved General / Asymmetric LLL

We strictly improve the asymmetric LLL, and thus all its applications, as follows.

▶ **Theorem 7.** *Theorem 5 holds if* (3) *is replaced by*

$$p_i \leq r_i \prod_{j\in\Gamma_i} \frac{1-r_j}{1-r_i r_j} \ . \tag{4}$$

While our Theorem 7 retains all the flexibility of the asymmetric LLL to adjust to events with different degrees and probabilities, it is sharp enough to recover the *optimal* bound in terms of the maximum degree $\Delta(G)$ (attained as a limit by Δ-regular trees as depth goes to infinity). Specifically, if every event has probability at most p and is mutually independent of all but $\Delta \geq 2$ other events, Theorem 7 implies the optimal condition $p < \frac{(\Delta-1)^{(\Delta-1)}}{\Delta^\Delta}$ originally proven by Shearer [18], whereas the asymmetric LLL requires $p < \frac{\Delta^\Delta}{(\Delta+1)^{(\Delta+1)}}$.

A fairly recent improvement of Theorem 5 is the so-called *cluster expansion* LLL by Bissacot et al. [5], wherein the presence of edges in the neighborhood of a vertex, i.e., the presence of triangles, allow one to relax the condition corresponding to that vertex. While our Theorem 7 is, in general, incomparable with the cluster expansion LLL, the overall trend is that the former wins when neighborhoods are sparse, while the latter when they are dense.

In Sections 2.4, 2.5 we will see four significant improvements of Theorem 7. The weakest of these is already *exact* on *arbitrary* trees (uniform trees being the worst case for given Δ).

2.2 An Upper Bound for the Partition Function on the Negative Reals

Recall that given a vector $\boldsymbol{p}$, the central problem is determining whether $|Z_G| > 0$ on the *polydisk* of $\boldsymbol{p}$, i.e., for every $\boldsymbol{x} \in \mathbb{C}^n$ such that $|x_i| \leq p_i$ for all $i \in [n]$. Since $Z_G(\mathbf{0}) = 1 > 0$, if $\boldsymbol{p} \notin \mathcal{S}(G)$, continuity implies $|Z_G(\lambda \boldsymbol{p}_S)| = 0$ for some $S \subseteq [n]$ and $\lambda \in (0,1]$, where $\boldsymbol{p}_S$ is the vector that results by setting to 0 all coordinates of $\boldsymbol{p}$ outside S. On the other hand, Scott and Sokal [17] showed that for $\boldsymbol{p} \in \mathcal{S}(G)$ and every $\lambda \in [0,1]$, the magnitude of Z_G over the polydisk of $\lambda\boldsymbol{p}$ is minimized at $-\lambda\boldsymbol{p}$. Thus, membership in $\mathcal{S}(G)$ is equivalent to $Z(-\lambda\boldsymbol{p}) > 0$ for every $\lambda \in [0,1]$ and characterizes the vectors on whose polydisks Z_G does not vanish.

The LLL is a local sufficient condition for $\boldsymbol{p} \in \mathcal{S}(G)$, providing a strictly positive lower bound for $Z_G(-\boldsymbol{p})$ for such $\boldsymbol{p}$ (and, thus, for $|Z_G|$ on the polydisk of $\boldsymbol{p}$). We show that $Z_G(-\boldsymbol{p})$ can also be bounded *from above* for $\boldsymbol{p} \in \mathcal{S}(G)$.

▶ **Definition 8.** *Given a permutation π of $[n]$, let $\overleftarrow{\Gamma_i} = \overleftarrow{\Gamma_i}(\pi) = \Gamma_i \cap \{j \in [n] : \pi(j) < \pi(i)\}$ and let $\overrightarrow{\Gamma_i} = \overrightarrow{\Gamma_i}(\pi) = \Gamma_i \cap \{j \in [n] : \pi(j) > \pi(i)\}$.*

▶ **Theorem 9** (Upper Bound). *Given $\boldsymbol{p}, G$ and a permutation π of $[n]$, define $\boldsymbol{r} = \boldsymbol{r}(\pi)$ by*

$$p_i = r_i \prod_{j \in \overleftarrow{\Gamma}_i(\pi)} (1 - r_j) \ , \quad \text{for every } i \in [n] \ . \tag{5}$$

(Note that $\boldsymbol{r}$ is well-defined as $r_1 = p_1$, while for $i > 1$, r_i is determined by $p_i, r_1, \ldots, r_{i-1}$.) If $\boldsymbol{p} \in \mathcal{S}(G)$, then $Z(-\boldsymbol{p}; S) \leq \prod_{j \in S}(1 - r_j)$, for every $S \subseteq [n]$.

▶ Remark 10. If $\boldsymbol{r'}$ satisfies (3) and $\boldsymbol{r}$ is defined by (5), then $1 - r'_i \leq 1 - r_i$ for every $i \in [n]$.

2.3 Exact Partition Function Computation for Chordal Graphs

Recall that a graph is *chordal* if all its induced cycles have length three. We prove that the independent set polynomial of a chordal graph can be evaluated *anywhere* on the complex plane in *linear* time. We conjecture that chordality is closely related to the exact solvability of the hard-core model for certain highly transitive graphs, e.g., triangular lattice (hard hexagons model [3]), and that the hard-core model is not the only statistical mechanics model for which chordality relates to exact solvability. We leave this as future work.

▶ **Fact 11.** *A graph G on $[n]$ is chordal iff it has a* perfect elimination ordering*, i.e., a permutation π of $[n]$ such that $\overrightarrow{\Gamma}_i(\pi)$ is a clique for every $i \in [n]$. If the identity permutation is a perfect elimination ordering for G, we say that G is chordally presented.*

▶ **Theorem 12.** *If G is chordally presented, then $Z_G(\boldsymbol{x}) = \prod_{i \in [n]} (1 + r_i)$, where*

$$x_i = r_i \prod_{j \in \overleftarrow{\Gamma}_i} (1 + r_j) \ , \quad \text{for every } i \in [n] \ . \tag{6}$$

(Note that $\boldsymbol{r}$ is well-defined as $r_1 = x_1$, while for $i > 1$, r_i is determined by $x_i, r_1, \ldots, r_{i-1}$.)

▶ **Corollary 13.** *The independent set polynomial of a chordal graph can be evaluated anywhere on the complex plane in linear time. A perfect sample from the hard-core distribution on a chordal graph can be obtained in linear time.*

Proof. A chordal presentation of chordal graph $G = (V, E)$ can be found in time $O(|V| + |E|)$. Computing each r_i given $r_1, \ldots, r_{i-1}$ requires $O(|\Gamma_i|)$ steps. Thus, $Z_G(\boldsymbol{x}, [n])$ can be evaluated in $O(|V| + |E|)$ steps. Regarding sampling we observe that chordal graphs are closed with respect to vertex deletions. Thus, given $Z_G(\boldsymbol{x}, S)$ for arbitrary $S \subseteq [n]$ and $\{r_i\}_{i \in S}$, computing $Z_G(\boldsymbol{x}, T)$ for $T \subseteq S$ can be done by $O(|S| - |T|)$ divisions. Since $\{r_i\}_{i \in [n]}$ can be computed in $O(|V| + |E|)$ steps via (6), the claim follows. ◀

The previous best result on the independent set polynomial of chordal graphs is due to Okamoto, Uno, and Uehara [15] who showed that it can be evaluated exactly in linear time at $\boldsymbol{x} = \mathbf{1}$, i.e., that the number of independent sets can be counted. Since their algorithm is also capable of counting the number of independent sets of any given size $k = 1, \ldots, n$ in linear time, evaluating the univariate independent set polynomial can be done in polynomial time. However, our algorithm is significantly simpler, runs in linear time, and works also on the multivariate setting. A very recent related work by Heinrich and Müller [10] showed that the independent set polynomial can be evaluated exactly for $\boldsymbol{x} \in \mathbb{R}^n$, when G is strongly orderable. These form a subclass of weakly chordal graphs that contains chordal bipartite graphs. Finally, in terms of (arbitrarily good, randomized) approximate evaluation of the independent set polynomial, Bezakova and Sun [4] showed that a natural Markov chain for the hard core model with positive fugacities, i.e., for the case $\boldsymbol{x} \in \mathbb{R}^n$, mixes in polynomial time on chordal graphs with separators of *bounded* size.

2.4 Local Lemmata on Unordered Vertices (Simpler)

In this section we present four local lemmata providing sufficient conditions for $\boldsymbol{p} \in \mathcal{S}(G)$. As we will see in Section 3.2, determining $Z_G(-\boldsymbol{p})$ exactly amounts to understanding the set of all possible walks on G obeying certain ordering and self-avoidance constraints.

Dropping the ordering restriction and replacing self-avoidance by non-repetitiveness within distance 1 gives Theorem 14. Extending the scope of non-repetitiveness to distance 2 gives Theorem 15. Enforcing the ordering restriction while replacing self-avoidance by non-repetitiveness within distance 1 and 2, yields Theorems 17 and 18, respectively.

2.4.1 Incorporating All Paths of Length at most One

▶ **Theorem 14.** *Given $\boldsymbol{p} \in [0,1)^n$ and G, assume that for every path (i) of length 0 and every path (i,j) of length 1, there exist $r_i, r_{i,j} \in [0,1)$, respectively, such that*

$$p_i \leq r_i \prod_{j \in \Gamma_i} (1 - r_{i,j}) \tag{7}$$

$$p_j \leq r_{i,j} \prod_{k \in \Gamma_j \setminus \{i\}} (1 - r_{j,k}) \ . \tag{8}$$

Then, $Z(-\boldsymbol{p}) \geq \prod_{i \in [n]} (1 - r_i)$.

Theorem 7 follows from Theorem 14, as we show in Section 4.3: given $\{r'_i\}_{i \in [n]}$ satisfying (4) we can compute $r_{i,j}$ for every oriented edge (i,j) to satisfy (7), (8) (with $r_i = r'_i$).

2.4.2 Incorporating All Paths of Length at most Two

▶ **Theorem 15.** *Given $\boldsymbol{p} \in [0,1)^n$ and G, assume that for every path (i) of length 0, every path (i,j) of length 1, and every path (i,j,k) of length 2 such that $i \in \Gamma_k$, there exist $r_i, r_{i,j}, r_{i,j,k} \in [0,1)$, respectively, such that*

$$p_i \leq r_i \prod_{j \in \Gamma_i} (1 - r_{i,j}) \tag{9}$$

$$p_j \leq r_{i,j} \prod_{\substack{k \in \Gamma_j \setminus \{i\} \\ i \notin \Gamma_k}} (1 - r_{j,k}) \prod_{\substack{k \in \Gamma_j \setminus \{i\} \\ i \in \Gamma_k}} (1 - r_{i,j,k}) \tag{10}$$

$$p_k \leq r_{i,j,k} \prod_{\substack{\ell \in \Gamma_k \setminus \{i,j\} \\ j \notin \Gamma_\ell}} (1 - r_{k,\ell}) \prod_{\substack{\ell \in \Gamma_k \setminus \{i,j\} \\ j \in \Gamma_\ell}} (1 - r_{j,k,\ell}) \ . \tag{11}$$

Then, $Z(-\boldsymbol{p}) \geq \prod_{i \in [n]} (1 - r_i)$.

2.5 Local Lemmata on Ordered Vertices (Sharper)

A *walk* starting at vertex i is a sequence of vertices $(v_0, v_1, \ldots, v_\ell)$, such that $v_0 = i$ and for all $k \in [\ell]$, vertex v_{k-1} is adjacent to vertex v_k.

▶ **Definition 16.** *Given a walk $w = (v_0, v_1, \ldots, v_\ell)$, let $\mathcal{F}(v_0) = \emptyset$, while for $k \in [\ell]$ let*

$$\mathcal{F}(v_0, \ldots, v_k) = \mathcal{F}(v_0, \ldots, v_{k-1}) \cup \{v_{k-1}\} \cup \{u \in \Gamma_{v_{k-1}} : u \geq v_k\} \ . \tag{12}$$

Let $\mathcal{N}(v_0, \ldots, v_\ell) = \Gamma_{v_\ell} \cap \mathcal{F}(v_0, \ldots, v_\ell)$.

2.5.1 Incorporating All Paths of Length at most One

▶ **Theorem 17.** *Given $\boldsymbol{p} \in [0,1)^n$ and G, assume that for every vertex $i \in [n]$ and every oriented edge (i,j) there exist $r_i, r_{i,j} \in [0,1)$, respectively, such that*

$$p_i \le r_i \prod_{j \in \Gamma_i} (1 - r_{i,j}) \tag{13}$$

$$p_j \le r_{i,j} \prod_{k \in \Gamma_j \setminus \mathcal{N}(i,j)} (1 - r_{j,k}) \ . \tag{14}$$

Then, $Z(-\boldsymbol{p}) \ge \prod_{i \in [n]} (1 - r_i)$.

Theorem 17 implies Theorem 14, since $\{i\} \subseteq \mathcal{N}(i,j)$ implying that any collection of numbers satisfying (8) also satisfies (14).

2.5.2 Incorporating All Paths of Length at most Two

▶ **Theorem 18.** *Given $\boldsymbol{p} \in [0,1)^n$ and G assume that for every path (i) of length 0, every path (i,j) of length 1, and every path (i,j,k) of length 2 for which $\mathcal{N}(i,j,k) \neq \mathcal{N}(j,k)$, there exist $r_i, r_{i,j}, r_{i,j,k} \in [0,1)$, respectively, such that*

$$p_i \le r_i \prod_{j \in \Gamma_i} (1 - r_{i,j}) \tag{15}$$

$$p_j \le r_{i,j} \prod_{\substack{k \in \Gamma_j \setminus \mathcal{N}(i,j) \\ \mathcal{N}(i,j,k) = \mathcal{N}(j,k)}} (1 - r_{j,k}) \prod_{\substack{k \in \Gamma_j \setminus \mathcal{N}(i,j) \\ \mathcal{N}(i,j,k) \neq \mathcal{N}(j,k)}} (1 - r_{i,j,k}) \tag{16}$$

$$p_k \le r_{i,j,k} \prod_{\substack{\ell \in \Gamma_k \setminus \mathcal{N}(i,j,k) \\ \mathcal{N}(j,k,\ell) = \mathcal{N}(k,\ell)}} (1 - r_{k,\ell}) \prod_{\substack{\ell \in \Gamma_k \setminus \mathcal{N}(i,j,k) \\ \mathcal{N}(j,k,\ell) \neq \mathcal{N}(k,\ell)}} (1 - r_{j,k,\ell}) \ . \tag{17}$$

Then, $Z(-\boldsymbol{p}) \ge \prod_{i \in [n]} (1 - r_i)$.

Theorem 18 implies Theorem 15, since $\{i\} \subseteq \mathcal{N}(i,j)$ and $\Gamma_k \cap \{i,j\} \subseteq \mathcal{N}(i,j,k)$, implying that any collection of numbers satisfying (10), (11) also satisfy (16), (17).

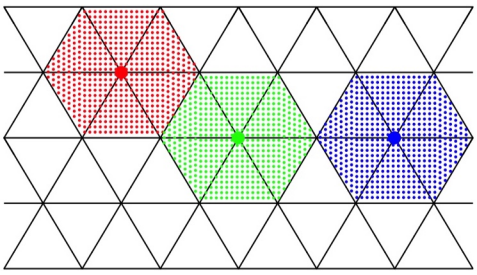
Theorem 18 also yields Theorem 17, by replacing the set $\mathcal{N}(i,j,k)$ in inequality (17) with its subset $\mathcal{N}(i,j)$ and imposing the additional equality constraints $r_{i,j,k} = r_{j,k}$. These modifications increase the number of (shrinking) factors in both (16) and (17), and together with the additional equality constraints make the resulting system of inequalities stricter.

Thus, to prove Theorems 14–18 it suffices to prove Theorem 18, which we do in Section 4.4.

2.6 Benchmarking: the Radii of $\mathcal{S}(G)$

As mentioned earlier, determining the set of activities for which the partition function is non-vanishing in the corresponding polydisk is a central problem in statistical mechanics. This is primarily motivated by the Lee-Yang [23] approach to studying phase transitions. Since phase transitions (non-analyticies of [one or more derivatives of] the log-partition function) can occur only in infinite-size systems, to study them on a locally-finite countable graph G_∞ (typically a regular lattice), we consider an increasing sequence of subgraphs $(G_n)_{n \ge 1}$ converging to G_∞ and study the limiting free energy per vertex $f_{G_\infty} = \lim_{n \to \infty} n^{-1} \log Z_{G_n}(\boldsymbol{x})$. Nonanalyticities of f_{G_∞} for real $\boldsymbol{x}$, arise from singularities of $\log Z_{G_n}(\boldsymbol{x})$ for complex $\boldsymbol{x}$ that approach the real axis in the limit $n \to \infty$. But the singularities of $\log Z_{G_n}(\boldsymbol{x})$ are precisely the zeros of $Z_{G_n}(\boldsymbol{x})$, hence the desire to determine the set $\mathcal{S}(G)$. Of particular interest is the so-called *uniform*, i.e., univariate, case $\boldsymbol{x} = x\mathbf{1}$, where all the activities are the same.

To benchmark our methods, we consider one of the very few exactly solved cases of the hard-core model, namely the case where G_∞ is the triangular lattice. This is known as the "hard hegaxons" model, since its valid configurations amount to placements of (centers of) hexagons in a triangular lattice so that no two hexagons overlap, i.e., to selecting an independent set of the triangular lattice to serve as the set of hexagon centers.



For this model it is known that the critical value is $x_c = \frac{5\sqrt{5}-11}{2} = 0.09016...$ Applying the asymmetric LLL, which only exploits that $\Delta(G_n) = 6$, gives $x_c \geq \Delta^\Delta/(\Delta+1)^{\Delta+1} = 6^6/7^7 = 0.0566$. Our improved asymmetric LLL (Theorem 7), improving the dependence on Δ, yields $x_c \geq (\Delta-1)^{\Delta-1}/\Delta^\Delta = 5^5/6^6 = 0.0669$.

Kolipaka, Szegedy, and Xu [11], introduced a family of sufficient conditions for the avoidance probability to be positive that range between the asymmetric LLL and the exact result of Shearer [18]. To apply their so-called "clique LLL" we color the triangular faces in a chess board pattern and decompose the triangular lattice using the white triangles as the parts of the clique-decomposition. Optimizing the resulting parameters yields $x_c \geq 0.07407$.

Finally, the cluster expansion LLL of Bissacot et al. [5], exploiting the presence of 6 triangles in the neighborhood of each vertex, yields $x_c \geq 0.0776$.

2.6.1 Simpler Bound: $x_c \geq 0.08115$

To apply Theorem 17 in the triangular lattice we order the neighbors of each vertex by taking the eastern neighbor to be the greatest, and then descending counter-clockwise. The translation symmetry of the lattice allows us to capture all possible paths of length up to one using only seven variables. Specifically, r_0 corresponds to vertices (paths of length 0), r_1 corresponds to arcs (paths of length 1) heading east, r_2 to arcs heading northeast, etc. Thus, inequalities (13) and (14) require x to be simultaneously less than all of the following:

$$
\begin{aligned}
&r_0 \cdot (1-r_1)\cdot(1-r_2)\cdot(1-r_3)\cdot(1-r_4)\cdot(1-r_5)\cdot(1-r_6)\\
&r_1 \cdot (1-r_1)\cdot(1-r_2)\cdot(1-r_3)\cdot(1-r_5)\cdot(1-r_6)\\
&r_2 \cdot (1-r_1)\cdot(1-r_2)\cdot(1-r_3)\cdot(1-r_4)\\
&r_3 \cdot (1-r_2)\cdot(1-r_3)\cdot(1-r_4)\cdot(1-r_5)\\
&r_4 \cdot (1-r_3)\cdot(1-r_4)\cdot(1-r_5)\cdot(1-r_6)\\
&r_5 \cdot (1-r_1)\cdot(1-r_4)\cdot(1-r_5)\cdot(1-r_6)\\
&r_5 \cdot (1-r_1)\cdot(1-r_5)\cdot(1-r_6)
\end{aligned}
$$

Taking $r_0 = 0.3055479560$, $r_1 = 0.2499747372$, $r_2 = 0.2063465756$, $r_3 = 0.1924531372$, $r_4 = 0.1818805124$, $r_5 = 0.1958294533$, $r_6 = 0.1602118920$, this is achieved for $x \leq 0.08115$.

2.6.2 Sharper Bound: $x_c \geq 0.08636$

To apply Theorem 18 we need to also consider paths of length 2. To do this we introduce a set of 6 additional variables $r_{1,3}, r_{2,1}, r_{2,4}, r_{3,5}, r_{5,1}, r_{6,1}$ (while, a priori, there are 6^2 "types" of paths of length 2, many of them are infeasible, while for others the type of the first arc implies the type of the second). Specifically, $r_{1,3}$ corresponds to a path first heading east and then heading northwest, $r_{2,1}$ corresponds to a path first heading northeast and then heading east, etc. The resulting 13 inequalities are satisfied for $x \leq 0.08636$ and $r_0 = 0.3939972440$, $r_1 = 0.2956228200$, $r_2 = 0.2271540263$, $r_3 = 0.2187337137$, $r_4 = 0.2144822763$, $r_5 = 0.2060776445$, $r_6 = 0.1736041642$, $r_{1,3} = 0.1820809928$, $r_{2,1} = 0.2347015656$, $r_{2,4} = 0.1772472600$, $r_{3,5} = 0.1675677715$, $r_{5,1} = 0.1868706968$, $r_{6,1} = 0.2417955235$.

3 Relating the Independent Set Polynomial to Walk Trees

3.1 Main Recurrence, Occupation Ratios, and Trees

For $i \in [n]$ and $S \subseteq [n] \setminus \{i\}$, given input $\boldsymbol{x}$, we define

$$Z(\boldsymbol{x}; i \mid S) := \frac{Z(\boldsymbol{x}; S \cup \{i\})}{Z(\boldsymbol{x}; S)} = Z(i \mid S) \ .$$

Trivially, $Z = Z([n]) = \prod_{i \in [n]} Z(i \mid [i-1])$, since $Z(\emptyset) = 1$. To estimate $Z(i \mid S)$ observe that the contribution to $Z(S \cup \{i\})$ of the sets including vertex i equals x_i times the contribution of the sets not including $\Gamma^+(i)$. Therefore,

$$Z(S \cup \{i\}) = Z(S) + x_i Z(S \setminus \Gamma_i) \ . \tag{18}$$

With the above in mind, let $j_1 \geq \ldots \geq j_d$ be the descending ordering of $\Gamma_i \cap S$, and for $\ell \in [d]$ write $S_\ell = S \setminus \{j_1, \ldots, j_\ell\}$. Dividing (18) by $Z(S)$ and writing the ratio $Z(S \setminus \Gamma_i)/Z(S)$ in telescopic form yields

$$Z(i \mid S) = 1 + x_i \frac{1}{\frac{Z(S)}{Z(S \setminus \Gamma_i)}} = 1 + x_i \frac{1}{\prod_{\ell=1}^{d} \frac{Z(S \setminus \{j_1, \ldots, j_{\ell-1}\})}{Z(S \setminus \{j_1, \ldots, j_\ell\})}} = 1 + x_i \prod_{\ell=1}^{d} \frac{1}{Z(j_\ell \mid S_\ell)} \ . \tag{19}$$

It is convenient to introduce the quantity $\text{ratio}_G(\boldsymbol{x}; (i, S)) := Z(i \mid S) - 1 = \text{ratio}(i, S)$ and rewrite (19) as

$$\text{ratio}(i, S) = x_i \prod_{\ell=1}^{d} \frac{1}{1 + \text{ratio}(j_\ell, S_\ell)} \ . \tag{20}$$

Thus,

$$Z = Z([n]) = \prod_{i \in [n]} Z(i \mid [i-1]) = \prod_{i \in [n]} (1 + \text{ratio}(i, [i-1])) \ . \tag{21}$$

We can now characterize the set $\mathcal{S}(G)$ as follows.

▶ **Lemma 19.** *The following are equivalent:*

1. *For every $S \subseteq [n]$, $Z(-\boldsymbol{p}; S) > 0$.*
2. *For every $i \in [n]$, and $S \subseteq [n] \setminus \{i\}$, $\text{ratio}(-\boldsymbol{p}; (i, S)) > -1$.*
3. *For every $i \in [n]$, and $S \subseteq [i-1]$, $\text{ratio}(-\boldsymbol{p}; (i, S)) > -1$.*

Proof.
$(1 \implies 2)$ If $Z(S), Z(S \cup \{i\}) > 0$, then $1 + \mathrm{ratio}(i, S) = Z(S \cup \{i\})/Z(S) > 0$.
$(2 \implies 3)$ Trivial.
$(3 \implies 1)$ For any $S \subseteq [n]$ and $j \in S$, write $S_j = \{k \in S : k < j\}$. By telescoping, $Z(S) = \prod_{j \in S} Z(j|S_j) = \prod_{j \in S} (1 + \mathrm{ratio}(j, S_j))$. Since $S_j \subseteq [i-1]$, the last product is positive. ◀

Say that $S, T \subseteq [n]$ are *separate* if they are disjoint and no edge has one endpoint in each. Clearly, if S, T are separate, then $Z(S \cup T) = Z(S)Z(T)$. Moreover, if $T \subseteq [n]$ is separate from $S \cup \{i\}$, then

$$\begin{aligned} \mathrm{ratio}(i, S \cup T) &= x_i \frac{Z((S \cup T) \setminus \Gamma_i)}{Z(S \cup T)} \\ &= x_i \frac{Z((S \setminus \Gamma_i) \cup T)}{Z(S \cup T)} \\ &= x_i \frac{Z(S \setminus \Gamma_i)Z(T)}{Z(S)Z(T)} = \mathrm{ratio}(i, S) \ . \end{aligned} \tag{22}$$

▶ **Definition 20.** *For a vertex v of a rooted tree T, we use $\widehat{\mathrm{ratio}}_T(v)$ to denote the quantity $\mathrm{ratio}_T(v, T(v))$, where $T(v)$ is the set of vertices other than v in the subtree rooted at v.*

Using (22) we observe that for a rooted tree T, recurrence (20) can be rewritten as

$$\widehat{\mathrm{ratio}}_T(v) = x_v \prod_{\ell=1}^{d} \frac{1}{1 + \widehat{\mathrm{ratio}}_T(v_\ell)} \ , \tag{23}$$

where $\{v_1, \ldots, v_d\}$ are the children of v in T.

3.2 Relating Arbitrary Graphs to Walk-Trees

Let $w = (v_0, v_1, \ldots, v_\ell)$ be an arbitrary walk of length ℓ.

▶ **Definition 21.** *w is* self-avoiding *if its vertices are* distinct.

▶ **Definition 22.** *w is* descending *if $v_k < v_{k-1}$ for all $k \in [\ell]$.*

Recall that, per Definition 16, for a walk $w = (v_0, v_1, \ldots, v_\ell)$, we let $\mathcal{F}(v_0) = \emptyset$, while for $k \in [\ell]$ we let $\mathcal{F}(v_0, \ldots, v_k) = \mathcal{F}(v_0, \ldots, v_{k-1}) \cup \{v_{k-1}\} \cup \{u \in \Gamma_{v_{k-1}} : u \geq v_k\}$.

▶ **Definition 23.** *$w = (v_0, v_1, \ldots, v_\ell)$ is* self-bounding *if $v_{k+1} \notin \mathcal{F}(v_0, \ldots, v_k)$ for all $k \in [\ell]$.*

▶ Remark 24. Self-bounding walks were defined in [17] as "truncated self-avoiding walks." The idea is that the next vertex in a self-bounding walk is subject to the additional requirement, relative to a self-avoiding walk, that it can also not be connected to certain neighbors of previously visited vertices. While a descending walk is self-bounding, the converse need not hold. For instance, in $G = ([4], \{\{1,2\}, \{2,3\}, \{3,4\}, \{4,1\}\})$, the walk $(4, 1, 2)$ is self-bounding but not descending.

For $S \subseteq [n]$, we write G_S for the subgraph of G induced by S.

▶ **Definition 25.** *Let $\mathcal{W}$ be a set of walks on G all starting at i, such that if $w \in \mathcal{W}$, then the same is true for every prefix of w. The* walk-tree *corresponding to the set of walks $\mathcal{W}$ has as its root the walk (i) of length 0, while the children of each vertex (walk) are its extensions by one step. The* activity *of vertex $(i, v_1, \ldots, v_\ell)$ of the walk-tree is x_{v_ℓ}.*

We use $\mathcal{L}_i := \mathcal{L}_i(G)$ to denote the walk-tree of self-bounding walks on $G_{[i]}$ starting at i.

The following theorem reduces the computation of ratios of an arbitrary graph G, to that of the tree ratios of $\mathcal{L}_i(G)$.

▶ **Theorem 26.** *For every vertex $i \in [n]$ and every walk $w = (v_0, v_1, \ldots, v_\ell)$ in $\mathcal{L}_i$,*

$$\widehat{\text{ratio}}_{\mathcal{L}_i}(w) = \text{ratio}_G\left(v_\ell, [i-1] \setminus \mathcal{F}(v_0, \ldots, v_\ell)\right) \quad . \tag{24}$$

In particular, $\widehat{\text{ratio}}_{\mathcal{L}_i}((i)) = \text{ratio}_G\left(i, [i-1]\right)$.

Proof. We proceed by induction on the size of the subtree rooted at w.

If w is a leaf in $\mathcal{L}_i$ then, trivially, $\text{ratio}_{\mathcal{L}_i}(w) = x_{v_\ell}$ and $\mathcal{F}(v_0, \ldots, v_\ell) \supseteq \Gamma_{v_\ell}$, for otherwise w could be extended. Thus, $[i-1] \setminus \mathcal{F}(v_0, \ldots, v_\ell)$ is separate from v_ℓ, which per (22) implies that $\text{ratio}_G\left(v_\ell, [i-1] \setminus \mathcal{F}(v_0, \ldots, v_\ell)\right) = \text{ratio}_G(v_\ell, \emptyset) = x_{v_\ell}$.

If w is not a leaf in $\mathcal{L}_i$, assume the theorem holds for its descendants. Let $\{j_1, \ldots, j_d\} = \Gamma_{v_\ell} \cap ([i-1] \setminus \mathcal{F}(v_0, \ldots, v_\ell))$, with $j_1 \geq \ldots \geq j_d$, and for $t \in [d]$ write $w_t = (v_0, v_1, \ldots, v_\ell, j_t)$. The first equality below follows from (23), the second from the inductive hypothesis, the third from the definition of $\mathcal{F}$, and the last from (20):

$$\widehat{\text{ratio}}_{\mathcal{L}_i}(w) = x_{v_\ell} \prod_{t=1}^{d} \frac{1}{1 + \widehat{\text{ratio}}_{\mathcal{L}_i}(w_t)} \tag{25}$$

$$= x_{v_\ell} \prod_{t=1}^{d} \frac{1}{1 + \text{ratio}_G\left(j_t, [i-1] \setminus \mathcal{F}\left(v_0, \ldots, v_\ell, j_t\right)\right)} \tag{26}$$

$$= x_{v_\ell} \prod_{t=1}^{d} \frac{1}{1 + \text{ratio}_G\left(j_t, ([i-1] \setminus \mathcal{F}\left(v_0, \ldots, v_\ell\right)) \setminus \{j_1, \ldots, j_t\}\right)} \tag{27}$$

$$= \text{ratio}_G\left(v_\ell, [i-1] \setminus \mathcal{F}\left(v_0, \ldots, v_\ell\right)\right) \quad . \tag{28}$$

◀

3.3 Tree Monotonicity

▶ **Definition 27.** *Let T be a tree with root r. The set of* prefixes *of T comprises T itself, and every tree with root r that can be derived by removing a leaf from a prefix of T.*

▶ **Lemma 28.** *Let T be a tree with root r and assume that $\boldsymbol{p}$ is such that $\widehat{\text{ratio}}_T(-\boldsymbol{p}; v) > -1$ for every vertex $v \neq r$ of T. Then the function $f : \boldsymbol{x} \mapsto \widehat{\text{ratio}}_T\left(-\boldsymbol{x}; r\right)$ is smooth and strictly decreasing in each coordinate inside $P = \{(x_1, \ldots, x_n) : 0 \leq x_i \leq p_i\}$. In particular, if T' is a prefix of T, then $\widehat{\text{ratio}}_T\left(-\boldsymbol{p}; r\right) \leq \widehat{\text{ratio}}_{T'}\left(-\boldsymbol{p}; r\right)$.*

Proof. We use induction on the size of T. If T consists of just $\{r\}$, then $\widehat{\text{ratio}}_T\left(-\boldsymbol{x}; r\right) = -x_r$, satisfying the claim trivially. Let now T be a tree of size n, and assume that the lemma holds for every tree of size strictly less than n. If $\Gamma_i = \{j_1, \ldots, j_d\}$, then per (23),

$$\widehat{\text{ratio}}_T\left(-\boldsymbol{x}; r\right) = -x_r \prod_{\ell=1}^{d} \frac{1}{1 + \widehat{\text{ratio}}_T\left(-\boldsymbol{x}; j_\ell\right)} \quad . \tag{29}$$

Since each subtree rooted at j_ℓ has size strictly less than n, the inductive hypothesis implies that $\widehat{\text{ratio}}_T\left(-\boldsymbol{x}; j_\ell\right)$ is strictly decreasing inside P. Therefore, the lemma hypothesis that $\widehat{\text{ratio}}_T\left(-\boldsymbol{p}; j_\ell\right) > -1$ implies that $\widehat{\text{ratio}}_T\left(-\boldsymbol{x}; j_\ell\right) > -1$. Since the function $1/(1+x)$ is smooth and strictly decreasing for $x > -1$, the claim follows by the smoothness and monotonicity of the d factors in (29) implied by the inductive hypothesis.

To see the claim regarding prefixes of T, observe that $\widehat{\text{ratio}}_{T'}\left(-\boldsymbol{p}; r\right) = \widehat{\text{ratio}}_T\left(-\boldsymbol{p'}; r\right)$, where $\boldsymbol{p'}$ is derived by setting to 0 all coordinates of $\boldsymbol{p}$ corresponding to vertices not in T'. ◀

▶ **Theorem 29.** *$\boldsymbol{p} \in \mathcal{S}(G)$ iff $\widehat{\text{ratio}}_{\mathcal{L}_i}(-\boldsymbol{p}; w) > -1$, for all $i \in [n]$ and $w \in \mathcal{L}_i$.*

Proof. If $\boldsymbol{p} \in \mathcal{S}(G)$, then, by Lemma 19, $\text{ratio}_G(i, S) > -1$, for every $i \in [n]$ and $S \subseteq [n] \setminus \{i\}$. Thus, per Theorem 26, $\widehat{\text{ratio}}_{\mathcal{L}_i}(-\boldsymbol{p}; w) = \text{ratio}_G(v_\ell, [i-1] \setminus \mathcal{F}(v_0, \ldots, v_\ell)) > -1$.

For the converse, we will show that if $\widehat{\text{ratio}}_{\mathcal{L}_i}(-\boldsymbol{p}; w) > -1$ for all $i \in [n]$ and $w \in \mathcal{L}_i$, then $\text{ratio}(i, S) > -1$ for every $i \in [n]$ and $S \subseteq [i-1]$, which, by Lemma 19, implies $\boldsymbol{p} \in \mathcal{S}(G)$. Let $i \in [n]$ and $S \subseteq [i-1]$ be arbitrary, write $S^+ := S \cup \{i\}$, and let $\widetilde{\mathcal{L}}_i$ be the prefix of $\mathcal{L}_i$ obtained by deleting all walks intersecting the complement of S^+. It is easy to check that $\widetilde{\mathcal{L}}_i$ coincides with the tree of self-bounding walks starting at i on the subgraph of $G_{[i]}$ induced by S^+, i.e., $\widetilde{\mathcal{L}}_i(G) = \mathcal{L}_i(G_{S^+})$. Thus, Theorem 26 gives the second equality below, while the monotonicity of tree prefixes, per Lemma 28, gives the first inequality:

$$\text{ratio}_G(-\boldsymbol{p}; (i, S)) = \text{ratio}_{G_{S^+}}(-\boldsymbol{p}; (i, S)) = \widehat{\text{ratio}}_{\widetilde{\mathcal{L}}_i}(-\boldsymbol{p}; (i)) \geq \widehat{\text{ratio}}_{\mathcal{L}_i}(-\boldsymbol{p}; (i)) > -1 \ . \quad \blacktriangleleft$$

4 Proofs of Results

4.1 Upper Bound (Theorem 9)

Let $\mathcal{D}_i := \mathcal{D}_i(G)$ denote the tree of descending walks on G starting at i. Due to its highly recursive structure, if two vertices in $\mathcal{D}_i$ correspond to walks that end on the same vertex, then their ratios are equal. As a result, the different root ratios satisfy the following simple system of equations.

▶ **Theorem 30.** *Given $\boldsymbol{x} \in \mathbb{C}^n$, for $i = 1, 2, \ldots, n$ let*

$$r_i = x_i \prod_{j \in \overleftarrow{\Gamma}_i} \frac{1}{1 + r_j} \ . \tag{30}$$

Then, $\widehat{\text{ratio}}_{\mathcal{D}_i}(\boldsymbol{x}; (i)) = r_i$, for every $i \in [n]$.

Proof. We use induction on i. For $i = 1$, trivially, $\widehat{\text{ratio}}_{\mathcal{D}_1}(\boldsymbol{x}; (1)) = x_1 = r_1$. Assume now that (30) holds for all $i < k$. Clearly, the root walk (k) can only be extended by taking a step to a neighbor smaller than k. If $\{j_1, \ldots, j_d\} = \overleftarrow{\Gamma}_k$, then (23) yields (31). For the first equality in (32), note that appending k as a prefix to every vertex of $\mathcal{D}_{j_\ell}$ yields the subtree of $\mathcal{D}_k$ rooted at (k, j_ℓ), while the inductive hypothesis yields the second equality in (32).

$$\widehat{\text{ratio}}_{\mathcal{D}_k}(\boldsymbol{x}; (k)) = x_k \prod_{\ell=1}^{d} \frac{1}{1 + \widehat{\text{ratio}}_{\mathcal{D}_k}(\boldsymbol{x}; (k, j_\ell))} \tag{31}$$

$$= x_k \prod_{\ell=1}^{d} \frac{1}{1 + \widehat{\text{ratio}}_{\mathcal{D}_{j_\ell}}(\boldsymbol{x}; (j_\ell))} = x_i \prod_{j \in \overleftarrow{\Gamma}_i} \frac{1}{1 + r_j} = r_k \ . \tag{32}$$

$\blacktriangleleft$

Proof of Theorem 9. Without loss of generality, we assume that π is the identity. Equation (21) yields (33), while (34) follows from Theorem 26. Recalling that descending walks are self-bounding shows that $\mathcal{D}_i$ is a prefix of $\mathcal{L}_i$ and, thus, per Lemma 28, $\widehat{\text{ratio}}_{\mathcal{D}_i}(-\boldsymbol{p}; (i)) \geq \widehat{\text{ratio}}_{\mathcal{L}_i}(-\boldsymbol{p}; (i))$, yielding (35). Finally, our hypothesis is equivalent to $-r_i$ satisfying (30) for $\boldsymbol{x} = -\boldsymbol{p}$ so that Theorem 30, implies $\widehat{\text{ratio}}_{\mathcal{D}_i}(-\boldsymbol{p}; (i)) = -r_i$ and, thus, (36).

$$Z(-\boldsymbol{p}; S) = \prod_{i\in[n]} (1 + \mathrm{ratio}_G(-\boldsymbol{p}; (i, [i-1]))) \tag{33}$$

$$= \prod_{i\in[n]} \left(1 + \widehat{\mathrm{ratio}}_{\mathcal{L}_i}(-\boldsymbol{p}; (i))\right) \tag{34}$$

$$\leq \prod_{i\in[n]} \left(1 + \widehat{\mathrm{ratio}}_{\mathcal{D}_i}(-\boldsymbol{p}; (i))\right) \tag{35}$$

$$= (1 - r_i) \ . \tag{36}$$

◀

4.2 Chordal Graphs (Theorem 12)

We claim that a graph on $[n]$ is chordally presented iff its set of descending walks equals its set of self-bounding walks. Given this claim, Theorem 12 follows from Theorem 30. Since descending walks are self-bounding, the following suffices to prove our claim.

▶ **Theorem 31.** *G is not chordally presented iff there is a vertex i and a self-bounding walk on $G_{[i]}$ starting at i that is not descending.*

Proof. Let $(i =: v_0, v_1, \ldots, v_\ell)$ be a self-bounding walk on $G_{[i]}$ that is not descending and let $2 \leq k \leq \ell$ be the minimum index such that $v_{k-1} < v_k$. The minimality of k implies $v_{k-1} < v_{k-2}$ and, hence, that $v_k, v_{k-2} \in \overrightarrow{\Gamma}_{v_{k-1}}$. Since the walk is self-bounding, $v_k \notin \Gamma_{v_{k-2}}$, i.e., there is no edge between v_{k-2} and v_k, implying that G is not chordally presented.

If G is not chordally presented, there must be vertices $a < b < c$ such that a is connected to b and c, but b is not connected to c. Clearly, the walk (c, a, b) on $G_{[c]}$ is self-bounding but not descending. ◀

4.3 Proof of Theorem 7 given Theorem 14

Proof. Given $\{r'_i\}_{i\in[n]}$, let $r_i = r'_i$ and $r_{i,j} = r'_i \frac{1-r'_j}{1-r'_i r'_j}$. We show that if (4) is satisfied, then (7) and (8) are satisfied. Indeed,

$$r_i \prod_{j\in\Gamma_i} (1 - r_{j,i}) = r'_i \prod_{j\in\Gamma_i} \left(1 - r'_j \frac{1-r'_i}{1-r'_j r'_i}\right) = r'_i \prod_{j\in\Gamma_i} \left(\frac{1-r'_j}{1-r'_j r'_i}\right) \geq p_i \ ,$$

and

$$r_{i,j} \prod_{k\in\Gamma_i\setminus\{j\}} (1 - r_{k,i}) = r'_i \frac{1-r'_j}{1-r'_i r'_j} \prod_{k\in\Gamma_i\setminus\{j\}} \left(1 - r'_k \frac{1-r'_i}{1-r'_k r'_i}\right) = r'_i \prod_{j\in\Gamma_i} \left(\frac{1-r'_j}{1-r'_j r'_i}\right) \geq p_i.$$

◀

4.4 Proof of Theorem 18

Proof. We claim that (15)–(17) imply $\mathrm{ratio}(-\boldsymbol{p}; (i, S)) \geq -r_i$ for all $i \in [n]$ and $S \subseteq [n] \setminus \{i\}$. This suffices since $Z([n]) = \prod_{i=1}^n Z(i|[i-1]) = \prod_{i=1}^n (1 + \mathrm{ratio}(i, [i-1])) \geq \prod_{i=1}^n (1 - r_i)$.

To prove the claim we prove that if (15)–(17) hold, then (a),(b),(c) below hold (our claim is equivalent to (a); we only prove (b), (c) as aids for proving (a)):

(a) $\text{ratio}(-\boldsymbol{p}; (i, S)) \geq -r_i$, for every (empty) path (i) and every $S \subseteq [n] \setminus \{i\}$.

(b) $\text{ratio}(-\boldsymbol{p}; (j, S)) \geq -r_{i,j}$, for every path (i, j) and every $S \subseteq [n] \setminus \mathcal{N}(i, j)$.

(c) $\text{ratio}(-\boldsymbol{p}; (k, S)) \geq -r_{i,j,k}$, for every path (i, j, k) such that $\mathcal{N}(i, j, k) \neq \mathcal{N}(j, k)$ and every $S \subseteq [n] \setminus \mathcal{N}(i, j, k)$.

To prove (a),(b),(c) we proceed by induction on $|S|$. For $S = \emptyset$, we see that (15)–(17) imply $-p_i \geq \max\{-r_{k,j,i}, -r_{j,i}, -r_i\}$, for any $i, j, k \in [n]$, while $\text{ratio}(-\boldsymbol{p}; (i, S)) = -p_i$.

For $S \neq \emptyset$, assume that (a),(b),(c) hold for all sets of size strictly less than $|S|$.

(a) For any path (i) and any set $S \subseteq [n] \setminus \{i\}$, equation (20) implies the first equality below, while the inductive hypothesis yields the first inequality (since $\mathcal{N}(i, j)$ is non-empty):

$$\begin{aligned}
\text{ratio}(i, S) &= -p_i \prod_{j \in S \cap \Gamma_i} \frac{1}{1 + \text{ratio}(j, S \setminus \mathcal{N}(i, j))} \\
&\geq -p_i \prod_{j \in S \cap \Gamma_i} \frac{1}{1 - r_{i,j}} \\
&\geq -p_i \prod_{j \in \Gamma_i} \frac{1}{1 - r_{i,j}} \ .
\end{aligned}$$

Assumption (15) concludes the argument.

(b) For any path (i, j) and any set $S \subseteq [n] \setminus \mathcal{N}(i, j)$, equation (20) implies the first equality below, the second equality holds since S is devoid of vertices from $\mathcal{N}(i, j)$, while the third equality follows easily from the definition of $\mathcal{N}$:

$$\begin{aligned}
\text{ratio}(j, S) &= -p_j \prod_{k \in S \cap \Gamma_j} \frac{1}{1 + \text{ratio}(k, S \setminus \mathcal{N}(j, k))} \\
&= -p_j \prod_{k \in S \cap \Gamma_j} \frac{1}{1 + \text{ratio}(k, S \setminus (\mathcal{N}(i, j) \cup \mathcal{N}(j, k)))} \\
&= -p_j \prod_{k \in S \cap \Gamma_j} \frac{1}{1 + \text{ratio}(k, S \setminus \mathcal{N}(i, j, k))} \qquad (37)
\end{aligned}$$

Decomposing the product in (37) into two groups of factors yields (38) and invoking the inductive hypothesis yields (39):

$$\prod_{\substack{k \in S \cap \Gamma_j \\ \mathcal{N}(i,j,k) = \mathcal{N}(j,k)}} \left(\frac{1}{1 + \text{ratio}(k, S \setminus \mathcal{N}(j, k))} \right) \prod_{\substack{k \in S \cap \Gamma_j \\ \mathcal{N}(i,j,k) \neq \mathcal{N}(j,k)}} \left(\frac{1}{1 + \text{ratio}(k, S \setminus \mathcal{N}(i, j, k))} \right) \qquad (38)$$

$$\leq \prod_{\substack{k \in S \cap \Gamma_j \\ \mathcal{N}(i,j,k) = \mathcal{N}(j,k)}} \left(\frac{1}{1 - r_{j,k}} \right) \prod_{\substack{k \in S \cap \Gamma_j \\ \mathcal{N}(i,j,k) \neq \mathcal{N}(j,k)}} \left(\frac{1}{1 - r_{i,j,k}} \right) \qquad (39)$$

$$\leq \prod_{\substack{k \in \Gamma_j \setminus \mathcal{N}(i,j) \\ \mathcal{N}(i,j,k) = \mathcal{N}(j,k)}} \left(\frac{1}{1 - r_{j,k}} \right) \prod_{\substack{k \in \Gamma_j \setminus \mathcal{N}(i,j) \\ \mathcal{N}(i,j,k) \neq \mathcal{N}(j,k)}} \left(\frac{1}{1 - r_{i,j,k}} \right) \ .$$

Assumption (16) concludes the argument.

(c) For any path (i,j,k) such that $\mathcal{N}(i,j,k) \neq \mathcal{N}(j,k)$ and any set $S \subseteq [n] \setminus \mathcal{N}(i,j,k)$, equation (20) implies the first equality below, the second equality holds since S is devoid of vertices from $\mathcal{N}(j,k)$, while the third equality follows easily from the definition of $\mathcal{N}$:

$$\begin{aligned}\text{ratio}(k,S) &= -p_k \prod_{\ell \in S \cap \Gamma_k} \frac{1}{1+\text{ratio}(\ell, S \setminus \mathcal{N}(k,\ell))} \\ &= -p_k \prod_{\ell \in S \cap \Gamma_k} \frac{1}{1+\text{ratio}(\ell, S \setminus (\mathcal{N}(j,k) \cup \mathcal{N}(k,\ell)))} \\ &= -p_k \prod_{\ell \in S \cap \Gamma_k} \frac{1}{1+\text{ratio}(\ell, S \setminus \mathcal{N}(j,k,\ell))} \ . \end{aligned} \tag{40}$$

Decomposing the product in (40) into two groups of factors yields (41) and invoking the inductive hypothesis yields (42):

$$\prod_{\substack{\ell \in S \cap \Gamma_k \\ \mathcal{N}(j,k,\ell) = \mathcal{N}(k,\ell)}} \left(\frac{1}{1+\text{ratio}(\ell, S \setminus \mathcal{N}(k,\ell))} \right) \prod_{\substack{\ell \in S \cap \Gamma_j \\ \mathcal{N}(j,k,\ell) \neq \mathcal{N}(k,\ell)}} \left(\frac{1}{1+\text{ratio}(\ell, S \setminus \mathcal{N}(j,k,\ell))} \right) \tag{41}$$

$$\leq \prod_{\substack{\ell \in S \cap \Gamma_k \\ \mathcal{N}(j,k,\ell) = \mathcal{N}(k,\ell)}} \left(\frac{1}{1-r_{k,\ell}} \right) \prod_{\substack{\ell \in S \cap \Gamma_k \\ \mathcal{N}(j,k,\ell) \neq \mathcal{N}(k,\ell)}} \left(\frac{1}{1-r_{j,k,\ell}} \right) \tag{42}$$

$$\leq \prod_{\substack{\ell \in \Gamma_k \setminus \mathcal{N}(i,j,k) \\ \mathcal{N}(j,k,\ell) = \mathcal{N}(k,\ell)}} \left(\frac{1}{1-r_{k,\ell}} \right) \prod_{\substack{\ell \in \Gamma_k \setminus \mathcal{N}(i,j,k) \\ \mathcal{N}(j,k,\ell) \neq \mathcal{N}(k,\ell)}} \left(\frac{1}{1-r_{j,k,\ell}} \right) \ .$$

Assumption (17) concludes the argument. ◀

References

1. Noga Alon and Joel H. Spencer. *The Probabilistic Method*. Wiley Publishing, 4th edition, 2016.
2. Taro Asano. Lee-Yang theorem and the Griffiths inequality for the anisotropic Heisenberg ferromagnet. *Phys. Rev. Lett.*, 24:1409–1411, June 1970. doi:10.1103/PhysRevLett.24.1409.
3. R J Baxter. Hard hexagons: exact solution. *Journal of Physics A: Mathematical and General*, 13(3):L61–L70, 1980. doi:10.1088/0305-4470/13/3/007.
4. Ivona Bezáková and Wenbo Sun. Mixing of markov chains for independent sets on chordal graphs with bounded separators. In Donghyun Kim, R. N. Uma, Zhipeng Cai, and Dong Hoon Lee, editors, *Computing and Combinatorics - 26th International Conference, COCOON 2020, Atlanta, GA, USA, August 29-31, 2020, Proceedings*, volume 12273 of *Lecture Notes in Computer Science*, pages 664–676. Springer, 2020. doi:10.1007/978-3-030-58150-3_54.
5. Rodrigo Bissacot, Roberto FernÁndez, Aldo Procacci, and Benedetto Scoppola. An improvement of the lovász local lemma via cluster expansion. *Comb. Probab. Comput.*, 20(5):709?719, 2011. doi:10.1017/S0963548311000253.
6. Young-Bin Choe, James G. Oxley, Alan D. Sokal, and David G. Wagner. Homogeneous multivariate polynomials with the half-plane property. *Advances in Applied Mathematics*, 32(1):88–187, 2004. Special Issue on the Tutte Polynomial. doi:10.1016/S0196-8858(03)00078-2.
7. M. Chudnovsky and P. Seymour. The roots of the independence polynomial of a claw-free graph. *J. Comb. Theory, Ser. B*, 97:350–357, 2007.
8. Paul Erdös and Joel Spencer. Lopsided lovász local lemma and latin transversals. *Discret. Appl. Math.*, 30(2-3):151–154, 1991. doi:10.1016/0166-218X(91)90040-4.

9 Ole J. Heilmann and Elliott H. Lieb. Theory of monomer-dimer systems. *Comm. Math. Phys.*, 25(3):190–232, 1972. URL: https://projecteuclid.org:443/euclid.cmp/1103857921.

10 Marc Heinrich and Haiko M?ller. Counting independent sets in strongly orderable graphs, 2021. arXiv:2101.01997.

11 Kashyap Babu Rao Kolipaka, Mario Szegedy, and Yixin Xu. A sharper local lemma with improved applications. In Anupam Gupta, Klaus Jansen, José D. P. Rolim, and Rocco A. Servedio, editors, *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques - 15th International Workshop, APPROX 2012, and 16th International Workshop, RANDOM 2012, Cambridge, MA, USA, August 15-17, 2012. Proceedings*, volume 7408 of *Lecture Notes in Computer Science*, pages 603–614. Springer, 2012. doi:10.1007/978-3-642-32512-0_51.

12 T. D. Lee and C. N. Yang. Statistical theory of equations of state and phase transitions. II. Lattice gas and Ising model. *Phys. Rev.*, 87:410–419, August 1952. doi:10.1103/PhysRev.87.410.

13 Charles M. Newman. Zeros of the partition function for generalized Ising systems. *Communications on Pure and Applied Mathematics*, 27(2):143–159, 1974. doi:10.1002/cpa.3160270203.

14 Charles M. Newman. The GHS inequality and the Riemann hypothesis. *Constructive Approximation*, 7(1):389–399, 1991. doi:10.1007/BF01888165.

15 Yoshio Okamoto, Takeaki Uno, and Ryuhei Uehara. Counting the number of independent sets in chordal graphs. *Journal of Discrete Algorithms*, 6(2):229–242, 2008. Selected papers from CompBioNets 2004. doi:10.1016/j.jda.2006.07.006.

16 David Ruelle. *Statistical mechanics: rigorous results*. World Scientific, Singapore, 1999. URL: http://cds.cern.ch/record/392442.

17 Alexander D. Scott and Alan D. Sokal. The repulsive lattice gas, the independent-set polynomial, and the Lovász local lemma. *Journal of Statistical Physics*, 118:1151–1261, 2005.

18 James B. Shearer. On a problem of Spencer. *Combinatorica*, 5(3):241–245, 1985. doi:10.1007/BF02579368.

19 Allan Sly and Nike Sun. The computational hardness of counting in two-spin models on d-regular graphs. In *53rd Annual IEEE Symposium on Foundations of Computer Science, FOCS 2012, New Brunswick, NJ, USA, October 20-23, 2012*, pages 361–369. IEEE Computer Society, 2012. doi:10.1109/FOCS.2012.56.

20 Joel Spencer. Asymptotic lower bounds for ramsey functions. *Discrete Mathematics*, 20:69–76, 1977. doi:10.1016/0012-365X(77)90044-9.

21 Masuo Suzuki and Michael E. Fisher. Zeros of the partition function for the Heisenberg, ferroelectric, and general Ising models. *Journal of Mathematical Physics*, 12(2):235–246, 1971. doi:10.1063/1.1665583.

22 Dror Weitz. Counting independent sets up to the tree threshold. In Jon M. Kleinberg, editor, *Proceedings of the 38th Annual ACM Symposium on Theory of Computing, Seattle, WA, USA, May 21-23, 2006*, pages 140–149. ACM, 2006. doi:10.1145/1132516.1132538.

23 C. N. Yang and T. D. Lee. Statistical theory of equations of state and phase transitions. i. theory of condensation. *Phys. Rev.*, 87:404–409, August 1952. doi:10.1103/PhysRev.87.404.

Almost-Linear-Time Weighted ℓ_p-Norm Solvers in Slightly Dense Graphs via Sparsification

Deeksha Adil
University of Toronto, Canada

Brian Bullins
Toyota Technological Institute at Chicago, IL, USA

Rasmus Kyng
ETH Zurich, Switzerland

Sushant Sachdeva
University of Toronto, Canada

Abstract

We give almost-linear-time algorithms for constructing sparsifiers with $n\,\mathrm{poly}(\log n)$ edges that approximately preserve weighted $(\ell_2^2 + \ell_p^p)$ flow or voltage objectives on graphs. For flow objectives, this is the first sparsifier construction for such mixed objectives beyond unit ℓ_p weights, and is based on expander decompositions. For voltage objectives, we give the first sparsifier construction for these objectives, which we build using graph spanners and leverage score sampling. Together with the iterative refinement framework of [Adil et al, SODA 2019], and a new multiplicative-weights based constant-approximation algorithm for mixed-objective flows or voltages, we show how to find $(1 + 2^{-\mathrm{poly}(\log n)})$ approximations for weighted ℓ_p-norm minimizing flows or voltages in $p(m^{1+o(1)} + n^{4/3+o(1)})$ time for $p = \omega(1)$, which is almost-linear for graphs that are slightly dense $(m \geq n^{4/3+o(1)})$.

2012 ACM Subject Classification Theory of computation → Sparsification and spanners; Theory of computation → Network flows

Keywords and phrases Weighted ℓ_p-norm, Sparsification, Spanners, Iterative Refinement

Digital Object Identifier 10.4230/LIPIcs.ICALP.2021.9

Category Track A: Algorithms, Complexity and Games

Related Version *Full Version*: `https://arxiv.org/abs/2102.06977`

Funding *Deeksha Adil*: Supported by a Post Graduate Doctoral Scholarship awarded by NSERC (Natural Sciences and Engineering Research Council of Canada) and SS's Discovery Grant.
Sushant Sachdeva: Supported by a Discovery Grant awarded by NSERC (Natural Sciences and Engineering Research Council of Canada).

1 Introduction

Network flow problems are some of the most extensively studied problems in optimization (e.g. see [4, 37, 20]). A general network flow problem on a graph $G(V, E)$ with n vertices and m edges can be formulated as

$$\min_{\boldsymbol{B}^\top \boldsymbol{f} = \boldsymbol{d}} \mathtt{cost}(\boldsymbol{f}),$$

where $\boldsymbol{f} \in \mathbb{R}^E$ is a flow vector on edges satisfying net vertex demands $\boldsymbol{d} \in \mathbb{R}^V$, $\boldsymbol{B} \in \mathbb{R}^{E \times V}$ is the signed edge-vertex incidence matrix of the graph, and $\mathtt{cost}(\boldsymbol{f})$ is a cost measure on flows. The weighted ℓ_∞-minimizing flow problem, i.e., $\mathtt{cost}(\boldsymbol{f}) = \|\boldsymbol{S}^{-1}\boldsymbol{f}\|_\infty$, captures

the celebrated maximum-flow problem with capacities $\boldsymbol{S}$; the weighted ℓ_1-minimizing flow problem, $\texttt{cost}(\boldsymbol{f}) = \|\boldsymbol{Sf}\|_1$ captures the transshipment problem generalizing shortest paths with lengths $\boldsymbol{S}$; and $\texttt{cost}(\boldsymbol{f}) = \boldsymbol{f}^\top \boldsymbol{Rf} = \|\boldsymbol{R}^{\frac{1}{2}}\boldsymbol{f}\|_2^2$ captures the electrical flow problem [42].

Dual to flow problems are voltage problems, which can be formulated as

$$\min_{\boldsymbol{d}^\top \boldsymbol{v}=1} \texttt{cost}'(\boldsymbol{Bv}),$$

Analogous to the flow problems, picking $\texttt{cost}'(\boldsymbol{Bv}) = \|\boldsymbol{SBv}\|_1$ captures the capacitated min-cut problem, $\texttt{cost}'(\boldsymbol{Bv}) = \|\boldsymbol{S}^{-1}\boldsymbol{Bv}\|_\infty$ captures vertex-labeling [26], and $\texttt{cost}'(\boldsymbol{Bv}) = (\boldsymbol{Bv})^\top \boldsymbol{R}^{-1}\boldsymbol{Bv} = \|\boldsymbol{R}^{-\frac{1}{2}}\boldsymbol{Bv}\|_2^2$ captures the electrical voltages problem.

The seminal work of Spielman and Teng [42] gave the first nearly-linear-time algorithm for computing $(1 + 1/\operatorname{poly}(n))$-approximate solutions to electrical (weighted ℓ_2-minimizing) flow/voltage problems. This work spurred the "Laplacian Paradigm" for designing faster algorithms for several classic graph optimization problems including maximum flow [11, 38, 22], multi-commodity flow [22], bipartite matching [29], transshipment [39], and graph partitioning [32]; culminating in almost-linear-time or nearly-linear-time low-accuracy algorithms (i.e. $1+\varepsilon$ approximations with $\operatorname{poly}(\frac{1}{\varepsilon})$ running time dependence) for many of these problems.

Progress on high-accuracy algorithms (i.e. algorithms that return $(1 + 1/\operatorname{poly}(n))$-approximate solutions with only a $\operatorname{poly}(\log n)$ factor overhead in time) for solving these problems has been harder to come by, and for many flow problems has been based on interior point methods [18]. E.g. the best running time for maximum flow stands at $\widetilde{O}(\min(m\sqrt{n}, n^\omega + n^{2+1/6}))$ [27, 15] and $\widetilde{O}(m^{4/3})$ for unit-capacity graphs [29, 28, 21]. Other results making progress in this direction include works on shortest paths with small range negative weights [16], and matrix-scaling [13, 5]. Recently, there has been progress on the dense case. In [44], the authors developed an algorithm for weighted bipartite matching and transshipment running in $\widetilde{O}(m + n^{3/2})$ time. This is a nearly-linear-time algorithm in moderately dense graphs.

Bubeck et al. [9] restarted the study of faster high-accuracy algorithms for the weighted ℓ_p-norm objective, $\texttt{cost}(\boldsymbol{f}) = \|\boldsymbol{Sf}\|_p$, a natural intermediate objective between ℓ_2 and ℓ_∞. This result improved the running time significantly over classical interior point methods [31] for p close to 2. Adil et al. [1] gave a high-accuracy algorithm for computing ℓ_p-norm minimizing flows in time $\min\{m^{\frac{4}{3}+o(1)}, n^\omega\}$ for $p \in (2, \sqrt{\log n}]$. Building on their work, Kyng et al. [25] gave an almost-linear-time high-accuracy algorithm for *unit-weight* ℓ_p-norm minimizing flows $\texttt{cost}(\boldsymbol{f}) = \|\boldsymbol{f}\|_p^p$ for large $p \in (\omega(1), \sqrt{\log n}]$. More generally, they give an almost-linear time-high-accuracy algorithm for *mixed* $\ell_2^2 + \ell_p^p$ objectives as long as the ℓ_p^p-norm is unit-weight, i.e.,

$$\texttt{cost}(\boldsymbol{f}) = \|\boldsymbol{R}^{\frac{1}{2}}\boldsymbol{f}\|_2^2 + \|\boldsymbol{f}\|_p^p.$$

Their algorithm for $(\ell_2^2 + \ell_p^p)$-minimizing flows was subsequently used as a key ingredient in recent results improving the running time for high-accuracy/exact maximum flow on unit-capacity graphs to $m^{4/3+o(1)}$ [28, 21].

In this paper, we obtain a nearly-linear running time for weighted $\ell_2^2 + \ell_p^p$ flow/voltage problems on graphs. Our algorithm requires $p(m^{1+o(1)} + n^{4/3+o(1)})$ time for $p = \omega(1)$ which is almost-linear-time for $p \le m^{o(1)}$ in slightly dense graphs, $(m \ge n^{4/3+o(1)})$.

Our running time $m^{1+o(1)} + n^{4/3+o(1)}$ is even better than the $\widetilde{O}(m + n^{3/2})$ time obtained for bipartite matching in [44]. Our result beats the $\Omega(n^{3/2})$ barrier that arises in [44] from the use of interior point methods that maintain a vertex dual solution using dense updates across $\sqrt{n}$ iterations. The progress on bipartite matching relies on highly technical

graph-based inverse maintenance techniques that are tightly interwoven with interior point method analysis. In constrast, our sparsification methods provide a clean interface to iterative refinement, which makes our analysis much more simple and compact.

Graph Sparsification. Various notions of graph sparsification – replacing a dense graph with a sparse one, while approximately preserving some key properties of the dense graph – have been key ingredients in faster low-accuracy algorithms. Benczúr and Karger [8] defined cut-sparsifiers that approximately preserve all cuts, and used them to give faster low-accuracy approximation algorithms for maximum-flow. Since then, several notions of sparsification have been studied extensively and utilized for designing faster algorithms [33, 41, 35, 40, 30, 38, 22, 36, 17, 24, 19, 12].

Sparsification has had a smaller direct impact on the design of faster high-accuracy algorithms for graph problems, limited mostly to the design of linear system solvers [42, 23, 34, 24]. Kyng et al. [25] constructed sparsifiers for weighted ℓ_2^2 + unweighted ℓ_p^p-norm objectives for flows. In this paper, we develop almost-linear time algorithms for building sparsifiers for weighted $\ell_2^2 + \ell_p^p$ norm objectives for flows and voltages,

$$\texttt{cost}(\boldsymbol{f}) = \|\boldsymbol{R}^{\frac{1}{2}}\boldsymbol{f}\|_2^2 + \|\boldsymbol{S}\boldsymbol{f}\|_p^p, \text{ and } \texttt{cost}'(\boldsymbol{B}\boldsymbol{v}) = \|\boldsymbol{W}^{\frac{1}{2}}\boldsymbol{B}\boldsymbol{v}\|_2^2 + \|\boldsymbol{U}\boldsymbol{B}\boldsymbol{v}\|_p^p,$$

and utilize them as key ingredients in our faster high-accuracy algorithms for optimizing such objectives on graphs. Our construction of sparsifiers for flow objectives builds on the machinery from [25], and our construction of sparsifiers for voltage objectives builds on graph spanners [6, 33, 7].

2 Our Results

Our main results concern flow and voltage problems for mixed $(\ell_2^2 + \ell_p^p)$-objectives for $p \geq 2$. Since our algorithms work best for large p, we restrict our attention to $p = \omega(1)$ in this overview. Section 3 provides detailed running times for all $p \geq 2$. We emphasize that by setting the quadratic term to zero in our mixed $(\ell_2^2 + \ell_p^p)$-objectives, we get new state of the art algorithms for ℓ_p-norm miniziming flows and voltages.

Mixed ℓ_2-ℓ_p-norm minimizing flow. Consider a graph $G = (V, E)$ along with non-negative diagonal matrices $\boldsymbol{R}, \boldsymbol{S} \in \mathbb{R}^{E\times E}$, and a gradient vector $\boldsymbol{g} \in \mathbb{R}^E$, as well as demands $\boldsymbol{d} \in \mathbb{R}^V$. We refer to the diagonal entries of $\boldsymbol{R}$ and $\boldsymbol{S}$ as ℓ_2-weights and ℓ_p-weights respectively. Let $\boldsymbol{B}$ denote the signed edge-vertex incidence of G (see Appendix in full version). We wish to solve the following minimization problem with the objective $\mathcal{E}(\boldsymbol{f}) = \boldsymbol{g}^\top\boldsymbol{f} + \|\boldsymbol{R}^{1/2}\boldsymbol{f}\|_2^2 + \|\boldsymbol{S}\boldsymbol{f}\|_p^p$

$$\min_{\boldsymbol{B}^\top\boldsymbol{f}=\boldsymbol{d}} \mathcal{E}(\boldsymbol{f}) \tag{1}$$

We require $\boldsymbol{g} \perp \{\ker(\boldsymbol{R}) \cap \ker(\boldsymbol{S}) \cap \ker(\boldsymbol{B})\}$ so that the problem has bounded minimum value, and $\boldsymbol{d} \perp \boldsymbol{1}$ so a feasible solution exists. These conditions can be checked in linear time and have a simple combinatorial interpretation. Note that the choice of graph edge directions in $\boldsymbol{B}$ matters for the value of $\boldsymbol{g}^\top\boldsymbol{f}$,. The flow on an edge is allowed to be both positive or negative.

Mixed ℓ_2-ℓ_p-norm minimizing voltages. Consider a graph $G = (V, E)$ along with non-negative diagonal matrices $\boldsymbol{W} \in \mathbb{R}^{E\times E}$ and $\boldsymbol{U} \in \mathbb{R}^{E\times E}$, and demands $\boldsymbol{d} \in \mathbb{R}^V$. We refer to the diagonal entries of $\boldsymbol{W}$ and $\boldsymbol{U}$ as ℓ_2-conductances and ℓ_p-conductances respectively.

In this case, we want to minimize the objective $\mathcal{E}(\boldsymbol{v}) = \boldsymbol{d}^\top \boldsymbol{v} + \|\boldsymbol{W}^{1/2}\boldsymbol{B}\boldsymbol{v}\|_2^2 + \|\boldsymbol{U}\boldsymbol{v}\|_p^p$ in minimization problem

$$\min_{\boldsymbol{v}} \mathcal{E}(\boldsymbol{v}) \tag{2}$$

In the voltage setting, we only require $\boldsymbol{d} \perp \mathbf{1}$ so the problem has bounded minimum value.

Obtaining good solutions. For both these problems, we study high accuracy approximation algorithms that provide feasible solutions $\boldsymbol{x}$ (a flow or a voltage respectively), that approximately minimize the objective function from some starting point $\boldsymbol{x}^{(0)}$, i.e., for some small $\varepsilon > 0$, we have

$$\mathcal{E}(\boldsymbol{x}) - \mathcal{E}(\boldsymbol{x}^\star) \leq \varepsilon(\mathcal{E}(\boldsymbol{x}^{(0)}) - \mathcal{E}(\boldsymbol{x}^\star))$$

wher $\boldsymbol{x}^\star$ denotes an optimal feasible solution. Our algorithms apply to problems with quasipolynomially bounded parameters, including quasipolynomial bounds on non-zero singular values of matrices we work with. Below we state our main algorithmic results.

▶ **Theorem 1** (Flow Algorithmic Result)**.** *Consider a graph G with n vertices and m edges, equipped with non-negative ℓ_2 and ℓ_p-weights, as well as a gradient and demands, all with quasi-polynomially bounded entries. For $p = \omega(1)$, in $p(m^{1+o(1)} + n^{4/3+o(1)}) \log^2 1/\varepsilon$ time we can compute an ε-approximately optimal flow solution to Problem* (1) *with high probability.*

This improves upon [1, 2, 3] which culminated in a $pm^{4/3+o(1)} \log^2 1/\varepsilon$ time algorithm.

▶ **Theorem 2** (Voltage Algorithmic Result)**.** *Consider a graph G with n vertices and m edges, equipped with non-negative ℓ_2 and ℓ_p-conductances, as well as demands, all with quasi-polynomially bounded entries. For $p = \omega(1)$, in $p(m^{1+o(1)} + n^{4/3+o(1)}) \log^2 1/\varepsilon$ time we can compute an ε-approximately optimal voltage solution to Problem* (2) *with high probability.*

Background: Iterative Refinement for Mixed ℓ_2-ℓ_p-norm Flow Objectives. Adil et al. [1] developed a notion of iterative refinement for mixed $(\ell_2^2 + \ell_p^p)$-objectives which in the flow setting, i.e. Problem (1), corresponds to approximating $\mathcal{E}'(\boldsymbol{\delta}) = \mathcal{E}(\boldsymbol{f} + \boldsymbol{\delta})$ using another $(\ell_2^2 + \ell_p^p)$-objective which roughly speaking corresponds to the 2nd degree Taylor series approximation of $\mathcal{E}'(\boldsymbol{\delta})$ combined with an ℓ_p-norm term $\|\boldsymbol{S}\boldsymbol{\delta}\|_p^p$, while ensuring feasibility of $\boldsymbol{f} + \boldsymbol{\delta}$ through a constraint $\boldsymbol{B}\boldsymbol{\delta} = \mathbf{0}$. We call the resulting problem a *residual* problem. Adil et al. [1] showed that obtaining a constant-factor approximate solution to the residual problem in δ is sufficient to ensure that $\mathcal{E}(\boldsymbol{f} + \boldsymbol{\delta})$ is closer to the optimal solution by a multiplicative factor depending only on p. In [2], this result was sharpened to show that such an approximate solution for the residual problem can be used to make $(1 - \Omega(1/p))$ multiplicative progress to the optimum, so that $O(p \log(m/\varepsilon))$ iterations suffice to produce an ε-accurate solution.

In order to solve the residual problem to a constant approximation, Adil et al. [1] developed an accelerated multiplicative weights method for $(\ell_2^2 + \ell_p^p)$-flow objectives, or more generally, for mixed $(\ell_2^2 + \ell_p^p)$-regression in an underconstrained setting.

Sparsification results. Our central technical results in this paper concern sparsification of residual flow and voltage problems, in the sense outlined in the previous paragraph. Concretely, in nearly-linear time, we can take a residual problem on a dense graph and produce a residual problem on a sparse graph with $\widetilde{O}(n)$ edges, with the property that constant factor solutions to the sparse residual problem still make $(1 - \Omega(m^{-\frac{2}{p-1}} p))$ multiplicative

progress on the original problem. This leads to an iterative refinement that converges in $O(pm^{\frac{2}{p-1}}\log(m/\varepsilon))$ steps. However, the accelerated multiplicative weights algorithm that we use for each residual problem now only requires $\widetilde{O}(n^{4/3})$ time to compute a crude solution.

Flow residual problem sparsification. In the flow setting, we show the following:

▶ **Theorem 3** (Informal Flow Sparsification Result). *Consider a graph G with n vertices and m edges, equipped with non-negative ℓ_2 and ℓ_p-weights, as well as a gradient. In $\widetilde{O}(m)$ time, we can compute a graph H with n vertices and $\widetilde{O}(n)$ edges, equipped with non-negative ℓ_2 and ℓ_p-weights, as well as a gradient, such that a constant factor approximation to the flow residual problem on H, when scaled by $m^{\frac{-1}{p-1}}$ results in an $\widetilde{O}(m^{\frac{2}{p-1}})$ approximate solution to the flow residual problem on G. The algorithm works for all $p \geq 2$ and succeeds with high probability.*

Our sparsification techniques build on [25], require a new bucketing scheme to deal with non-uniform ℓ_p-weights, as well as a prepreprocessing step to handle cycles with zero ℓ_2-weight and ℓ_p-weight. This preprocessing scheme in turn necessitates a more careful analysis of additive errors introduced by gradient rounding, and we provide a more powerful framework for this than [25].

Voltage residual problem sparsification. In the voltage setting, we show the following.

▶ **Theorem 4** (Voltage Sparsification Result (Informal)). *Consider a graph G with n vertices and m edges, equipped with non-negative ℓ_2 and ℓ_p-conductances. In $\widetilde{O}(m)$ time, we can compute a graph H with n vertices and $\widetilde{O}(n)$ edges, equipped with non-negative ℓ_2 and ℓ_p-conductances, such that constant factor approximation to the voltage residual problem on H, when scaled by $m^{\frac{-1}{p-1}}$ results in an $\widetilde{O}(m^{\frac{1}{p-1}})$ approximate solution to the voltage residual problem on G. The algorithm works for all $p \geq 2$ and succeeds with high probability.*

Note that our voltage sparsification is slightly stronger than our flow sparsification, as the former loses only a factor $\widetilde{O}(m^{\frac{1}{p-1}})$ in the approximation while the latter loses a factor $\widetilde{O}(m^{\frac{2}{p-1}})$. Our voltage sparsification uses a few key observations: In voltage space, surprisingly, we can treat treat the ℓ_2 and ℓ_p costs separately. This behavior is very different than the flow case, and arises becase in voltage space, every edge provides an "obstacle", i.e. adding an edge increases cost, whereas in flow space, every edge provides an "opportunity", i.e. adding an edge decreases cost. This means that in voltage space, we can separately account for the energy costs created by our ℓ_2 and ℓ_p terms, whereas in flow space, the ℓ_2 and ℓ_p weights must be highly correlated in a sparsifier. Armed with this decoupling observation, we preserve ℓ_2 cost using standard tools for spectral graph sparsification, and we preserve ℓ_p cost approximately by a reduction to graph distance preservation, which we in turn achieve using weighted undirected graph spanners.

Voltage space accelerated multiplicative weights solver. The algorithm from [1] for constant approximate solutions to the residual problem works in the flow setting. Using iterative refinement, the algorithm could be used to compute high-accuracy solutions. Because we can use high-accuracy flow solutions to extract high-accuracy solutions to the dual voltage problem, [1] were also able to produce solutions to ℓ_q-norm minimizing voltage problems (where ℓ_q for $q = p/(p-1)$ is the dual norm to ℓ_p). Hence, by solving ℓ_p-flow problems for all $p \in (2, \infty)$, [1] were able to solve ℓ_q-norm minimizing voltage problems for all $q \in (1, 2)$.

Our sparsification of flow and voltage problems works only for $p \geq 2$. Thus, in order to solve for q-norm minimizing voltages for $q > 2$, we require a solver that works directly in voltage space for mixed $(\ell_2^2 + \ell_p^p)$-voltage objectives.

We develop an accelerated multiplicative weights algorithm along the lines of [11, 10, 1] that works directly in voltage space for mixed $(\ell_2^2 + \ell_p^p)$-objectives, or more generally for overconstrained mixed $(\ell_2^2 + \ell_p^p)$-objective regression. Concretely, this directly gives an algorithm for computing crude solutions to the residual problems that arise from applying [1] iterative refinement to Problem (2). Our solver produces an improved $O(1)$-approximation to the residual problem rather than a $p^{O(p)}$-approximation from [1]. This gives an $\widetilde{O}(m^{4/3})$ high-accuracy algorithm for mixed $(\ell_2^2 + \ell_p^p)$-objective voltage problems for $p > 2$, unlike [1], which could only solve pure $p > 2$ voltage problems. We then speed this up to a $p(m^{1+o(1)} + n^{4/3+o(1)})$ time algorithm for $p = \omega(1)$ by developing a sparsification procedure that applies directly to mixed $(\ell_2^2 + \ell_p^p)$-voltage problems for $p > 2$.

Mixed ℓ_2-ℓ_p-norm regression. Our framework can also be applied outside of a graph setting, where our new accelerated multiplicative weights algorithm for overconstrained mixed $(\ell_2^2 + \ell_p^p)$-regression gives new state-of-the-art results in some regimes when combined with new sparsification results. In this setting we develop sparsification techniques based on the Lewis weights sampling from the work of Cohen and Peng [17]. We focus on the case $2 < p < 4$, where [17] provided fast algorithms for Lewis weight sampling.

▶ **Theorem 5** (General Matrices Sparsification Result). *Let $p \in [2, 4)$, let $\boldsymbol{M} \in \mathbb{R}^{m_1 \times n}, \boldsymbol{N} \in \mathbb{R}^{m_2 \times n}$ be matrices, $m_1, m_2 \geq n$, and let $LSS(\boldsymbol{B})$ denote the time to solve a linear system in $\boldsymbol{B}^\top \boldsymbol{B}$. Then, we may compute $\widetilde{\boldsymbol{M}}, \widetilde{\boldsymbol{N}} \in \mathbb{R}^{O(n^{p/2}\log(n)) \times n}$ such that with probability at least $1 - \frac{1}{n^{\Omega(1)}}$, for all $\Delta \in \mathbb{R}^n$,*

$$\|\widetilde{\boldsymbol{M}}\Delta\|_2^2 + \|\widetilde{\boldsymbol{N}}\Delta\|_p^p \approx_{O(1)} \|\boldsymbol{M}\Delta\|_2^2 + \|\boldsymbol{N}\Delta\|_p^p,$$

in time $\tilde{O}\Big(\mathrm{nnz}(\boldsymbol{M}) + \mathrm{nnz}(\boldsymbol{N}) + LSS(\widehat{\boldsymbol{M}}) + LSS(\widehat{\boldsymbol{N}})\Big)$, for some $\widehat{\boldsymbol{M}}$ and $\widehat{\boldsymbol{N}}$ each containing $O(n\log(n))$ rescaled rows of $\boldsymbol{M}$ and $\boldsymbol{N}$, respectively.

▶ **Theorem 6** (General Matrices Algorithmic Result). *For $p \in [2, 4)$, with high probability we can find an ε-approximate solution to (3) in time*

$$\widetilde{O}\Big(\Big(\mathrm{nnz}(\boldsymbol{M}) + \mathrm{nnz}(\boldsymbol{N}) + \Big(LSS(\widetilde{\boldsymbol{M}}) + LSS(\widetilde{\boldsymbol{N}})\Big)n^{\frac{p(p-2)}{6p-4}}\Big)\log^2(1/\varepsilon)\Big),$$

for some $\widetilde{\boldsymbol{M}}$ and $\widetilde{\boldsymbol{N}}$ each containing $O(n^{p/2}\log(n))$ rescaled rows of $\boldsymbol{M}$ and $\boldsymbol{N}$, respectively, where $LSS(\boldsymbol{A})$ is the time required to solve a linear equation in $\boldsymbol{A}^\top \boldsymbol{A}$ to quasipolynomial accuracy.

Note that for all $p \in (2, 4)$, we have that the exponent $\frac{p(p-2)}{6p-4} \leq 0.4$.

▶ Remark 7. By [14], a linear equation in $\boldsymbol{A}^\top \boldsymbol{A}$, where $\boldsymbol{A} \in \mathbb{R}^{m \times n}$ can be solved to quasipolynomial accuracy in time $\widetilde{O}(\mathrm{nnz}(\boldsymbol{A}) + n^\omega)$.

Using the above result for solving the required linear systems, we get a running time of $\widetilde{O}(\mathrm{nnz}(\boldsymbol{M}) + \mathrm{nnz}(\boldsymbol{N}) + (n^{p/2} + n^\omega)n^{\frac{p(p-2)}{6p-4}})$, matching an earlier input sparsity result by Bubeck et al. [9] that achieves $\widetilde{O}((\mathrm{nnz}(\boldsymbol{M}) + \mathrm{nnz}(\boldsymbol{N}))(1 + n^{\frac{1}{2}}m^{-\frac{1}{p}}) + m^{\frac{1}{2}-\frac{1}{p}}n^2 + n^\omega)$, where $\boldsymbol{M} \in \mathbb{R}^{m_1 \times n}, \boldsymbol{N} \in \mathbb{R}^{m_2 \times n}$ and $m = \max\{m_1, m_2\}$.

3 Main Algorithm

In this section, we prove Theorems 1, 2, and 6. We first design an algorithm to solve the following general problem:

▶ **Definition 8.** *For matrices* $\boldsymbol{M} \in \mathbb{R}^{m_1 \times n}$, $\boldsymbol{N} \in \mathbb{R}^{m_2 \times n}$ *and* $\boldsymbol{A} \in \mathbb{R}^{d \times n}$, $m_1, m_2 \geq n, d \leq n$, *and vectors* $\boldsymbol{b} \perp \{\ker(\boldsymbol{M}) \cap \ker(\boldsymbol{N}) \cap \ker(\boldsymbol{A})\}$ *and* $\boldsymbol{c} \in im(\boldsymbol{A})$, *we want to solve*

$$\begin{aligned} \min_{\boldsymbol{x}} \quad & \boldsymbol{b}^\top \boldsymbol{x} + \|\boldsymbol{M}\boldsymbol{x}\|_2^2 + \|\boldsymbol{N}\boldsymbol{x}\|_p^p \\ s.t. \quad & \boldsymbol{A}\boldsymbol{x} = \boldsymbol{c}. \end{aligned} \tag{3}$$

In order to solve the above problem, we use the iterative refinement framework from [2] to obtain a residual problem which is defined as follows.

▶ **Definition 9.** *For any* $p \geq 2$, *we define the residual problem* $res(\Delta)$, *for* (3) *at a feasible* $\boldsymbol{x}$ *as,*

$$\max_{\boldsymbol{A}\Delta=0} \quad res(\Delta) \stackrel{\text{def}}{=} \boldsymbol{g}^\top \Delta - \Delta^\top \boldsymbol{R} \Delta - \|\boldsymbol{N}\Delta\|_p^p, \text{ where,}$$

$$\boldsymbol{g} = \frac{1}{p}\boldsymbol{b} + \frac{2}{p}\boldsymbol{M}^\top \boldsymbol{M}\boldsymbol{x} + |\boldsymbol{N}\boldsymbol{x}|^{p-2}\boldsymbol{N}\boldsymbol{x} \quad \text{and} \quad \boldsymbol{R} = \frac{2}{p^2}\boldsymbol{M}^\top \boldsymbol{M} + 2\boldsymbol{N}^\top Diag(|\boldsymbol{N}\boldsymbol{x}|^{p-2})\boldsymbol{N}.$$

This residual problem can further be reduced by moving the term linear in $\boldsymbol{x}$ to the constraints via a binary search. This leaves us with a problem of the form,

$$\begin{aligned} \min_{\Delta} \quad & \Delta^\top \boldsymbol{R}\Delta + \|\boldsymbol{N}\Delta\|_p^p \\ \text{s.t.} \quad & \boldsymbol{g}^\top \Delta = a, \boldsymbol{A}\Delta = 0, \end{aligned}$$

for some constant a.

In order to solve the above problem with $\ell_2^2 + \ell_p^p$ objective, we reduce the instance size via a sparsification routine, and then solve the smaller problem by a multiplicative weights algorithm. We adapt the multiplicative-weights algorithm from [1] to work in the voltage space while improving the p dependence of the runtime from $p^{O(p)}$ to p, and the approximation quality from $p^{O(p)}$ to $O(1)$. The precise sparsification routines are described in later sections.

For large p, i.e., $p > \log m$, in order to get a linear dependence on the running time on p, we need to reduce the residual problem in ℓ_p-norm to a residual problem in $\log m$-norm by using the framework from [3].

The entire meta-algorithm is described formally in Algorithm 1, and its guarantees are described by the next theorem. Most proof details are deferred to the full version.

▶ **Theorem 10.** *For an instance of Problem* (3), *suppose we are given a starting solution* $\boldsymbol{x}^{(0)}$ *that satisfies* $\boldsymbol{A}\boldsymbol{x}^{(0)} = \boldsymbol{c}$ *and is a* κ *approximate solution to the optimum. Consider an iteration of the while loop, line 8 of Algorithm 1 for the* ℓ_p*-norm residual problem at* $\boldsymbol{x}^{(t)}$. *We can define* μ_1 *and* κ_1 *such that if* $\bar{\Delta}$ *is a* β *approximate solution to a corresponding* p'*-norm residual problem, then* $\mu_1\bar{\Delta}$ *is a* κ_1*-approximate solution to the* p*-residual problem. Further, suppose we have the following procedures,*

1. SPARSIFY*: Runs in time* K, *takes as input any matrices* $\boldsymbol{R}, \boldsymbol{N}$ *and vector* $\boldsymbol{g}$ *and returns* $\widetilde{\boldsymbol{R}}, \widetilde{\boldsymbol{N}}, \widetilde{\boldsymbol{g}}$ *having sizes at most* $\tilde{n} \times n$ *for the matrices , such that if* $\widetilde{\Delta}$ *is a* β *approximate solution to,*

$$\max_{\boldsymbol{A}\Delta=0} \quad \widetilde{\boldsymbol{g}}^\top \Delta - \|\widetilde{\boldsymbol{R}}\Delta\|_2^2 - \|\widetilde{\boldsymbol{N}}\Delta\|_{p'}^{p'},$$

for any $p' \geq 2$, *then* $\mu_2\widetilde{\Delta}$, *for a computable* μ_2 *is a* $\kappa_2\beta$*-approximate solution for,*

$$\max_{\boldsymbol{A}\Delta=0} \quad res(\Delta) \stackrel{\text{def}}{=} \boldsymbol{g}^\top \Delta - \|\boldsymbol{R}^{1/2}\Delta\|_2^2 - \|\boldsymbol{N}\Delta\|_{p'}^{p'}.$$

2. SOLVER*: Approximately solves* (4) *to return* $\bar{\Delta}$ *such that* $\|\widetilde{\boldsymbol{R}}\bar{\Delta}\|_2^2 \le \kappa_3\nu$ *and* $\|\widetilde{\boldsymbol{N}}\Delta\|_p^p \le \kappa_4\nu$ *in time* $\tilde{K}(\tilde{n})$ *for instances of size at most* $\tilde{n}$*.*

Algorithm 1 finds an ε*-approximate solution for Problem* (3) *in time*

$$\widetilde{O}\left(p\kappa_4^{1/(p-1)}\kappa_3\kappa_2\kappa_1(K+\tilde{K}(\tilde{n}))\log\left(\frac{\kappa p}{\varepsilon}\right)^2\right).$$

Algorithm 1 Meta-Algorithm for ℓ_p Flows and Voltages.

procedure SPARSIFIED-P-PROBLEMS($\boldsymbol{A}, \boldsymbol{M}, \boldsymbol{N}, \boldsymbol{c}, \boldsymbol{b}, p$)
  $\boldsymbol{x} \leftarrow \boldsymbol{x}^{(0)}$, such that $\boldsymbol{f}(\boldsymbol{x}^{(0)}) \le \kappa\text{OPT}$
  $T \leftarrow \widetilde{O}\big(p\kappa_1\kappa_2\kappa_3 \log\big(\frac{\kappa}{\varepsilon}\big)\big)$
  for $t = 0$ to T **do**
    At $\boldsymbol{x}^{(t)}$ define $\boldsymbol{g}, \boldsymbol{R}, \boldsymbol{N}$ and $res(\Delta)$, the residual problem (Definition 9)
    $a \leftarrow \frac{1}{2}, b \leftarrow 1, \mu_1 \leftarrow 1, \kappa_1 \leftarrow 1$
    $\nu \leftarrow \boldsymbol{f}(\boldsymbol{x}^{(0)})$
    while $\nu \ge \varepsilon\frac{\boldsymbol{f}(\boldsymbol{x}^{(0)})}{\kappa p}$ **do**
      if $p > \log m$ **then** ▷ Convert ℓ_p-norm residual to $\log m$-norm residual
        $p' \leftarrow \log m$
        $\boldsymbol{N}' \leftarrow \frac{1}{2^{1/p'}}\big(\frac{\nu}{m}\big)^{\frac{1}{p'}-\frac{1}{p}}\boldsymbol{N}$
        $a \leftarrow \frac{1}{33}, b \leftarrow O(1)m^{o(1)}$
        $\mu_1 \leftarrow m^{-o(1)}, \kappa_1 \leftarrow m^{o(1)}$ ▷ Lose κ_1 in approx. when scaled by μ_1
        $(\widetilde{\boldsymbol{g}}, \widetilde{\boldsymbol{R}}, \widetilde{\boldsymbol{N}}) \leftarrow$ SPARSIFY$(\boldsymbol{g}, \boldsymbol{R}, \boldsymbol{N}')$ ▷ Lose κ_2 in approx. when scaled by μ_2
      else
        $(\widetilde{\boldsymbol{g}}, \widetilde{\boldsymbol{R}}, \widetilde{\boldsymbol{N}}) \leftarrow$ SPARSIFY$(\boldsymbol{g}, \boldsymbol{R}, \boldsymbol{N})$ ▷ Lose κ_2 in approx. when scaled by μ_2
        $p' \leftarrow p$
      Use SOLVER to compute κ_3, κ_4 approximate solution to

$$\widetilde{\Delta}^{(\nu)} \leftarrow \arg\min_{\Delta} \quad \|\widetilde{\boldsymbol{R}}^{1/2}\Delta\|_2^2 + \|\widetilde{\boldsymbol{N}}\Delta\|_{p'}^{p'} \quad \text{s.t.} \quad \widetilde{\boldsymbol{g}}^\top\Delta = a\nu, \quad \boldsymbol{A}\Delta = 0. \tag{4}$$

      $\bar{\Delta}^{(\nu)} \leftarrow \frac{a}{2b\kappa_3\kappa_4^{1/(p'-1)}}\mu_2\mu_1\widetilde{\Delta}^{(\nu)}$
      $\nu \leftarrow \nu/2$
    $\Delta \leftarrow \arg\min_{\bar{\Delta}^{(\nu)}} \boldsymbol{f}\Big(\boldsymbol{x} - \frac{\bar{\Delta}^{(\nu)}}{p}\Big)$
    $\boldsymbol{x} \leftarrow \boldsymbol{x} - \frac{\Delta}{p}$
  return $\boldsymbol{x}$

3.1 Algorithms for ℓ_p-norm Problems

The problems discussed in Section 2 are special cases of Problem (3), which means we can use Algorithm 1. To prove our results, we will utilize Theorem 10, with the respective sparsification procedures and the following multiplicative-weights based algorithm for solving problems of the form,

$$\min_{\Delta} \quad \Delta^\top\boldsymbol{M}^\top\boldsymbol{M}\Delta + \|\boldsymbol{N}\Delta\|_p^p \quad \text{s.t.} \quad \boldsymbol{A}\Delta = \boldsymbol{c}. \tag{5}$$

We describe our solver formally and prove the following theorem about its guarantees in the full version.

▶ **Theorem 11.** *Let $p \geq 2$. Consider an instance of Problem (5) described by matrices $\boldsymbol{A} \in \mathbb{R}^{d\times n}, \boldsymbol{N} \in \mathbb{R}^{m_1\times n}, \boldsymbol{M} \in \mathbb{R}^{m_2\times n}$, $d \leq n \leq m_1, m_2$, and vector $\boldsymbol{c} \in \mathbb{R}^d$. If the optimum of this problem is at most ν, Procedure* Residual-Solver *returns an $\boldsymbol{x}$ such that $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{c}$, and $\boldsymbol{x}^\top \boldsymbol{M}^\top \boldsymbol{M}\boldsymbol{x} \leq O(1)\nu$ and $\|\boldsymbol{N}\boldsymbol{x}\|_p^p \leq O(3^p)\nu$. The algorithm makes $O\left(pm_1^{\frac{p-2}{(3p-2)}}\right)$ calls to a linear system solver.*

We utilize Procedure Residual-Solver as the Procedure Solver in Algorithm Sparsified-p-Problems. The algorithm uses the procedure only for solving problems instances with $p \leq \log m$. Thus, its running time is $\tilde{K}(\tilde{n}) = \widetilde{O}\left(\tilde{n}^{\frac{p-2}{3p-2}} \cdot \mathrm{LSS}(\tilde{n})\right) \leq \widetilde{O}\left(\tilde{n}^{1/3} \cdot \mathrm{LSS}(\tilde{n})\right)$, where $\mathrm{LSS}(\tilde{n})$ denotes the time required to solve a linear system in matrices of size $\tilde{n}$. We also have, $\kappa_3 = O(1), \kappa_4^{1/(p-1)} = O(1)$.

We next estimate the values of κ_1 and μ_1. If $p \leq \log m$, we have $\mu_1 = 1$ and $\kappa_1 = 1$. Otherwise, $\mu_1 = \widetilde{O}(1)$ and $\kappa_1 = O(m^{o(1)})$ (Refer to the full version).

In order to obtain an initial solution, we usually solve an ℓ_2-norm problem. This gives an $m^{p/2}$ approximate initial solution which results in a factor of p^2 in the running time. To avoid this, we can do a homotopy on p similar to [3], i.e., start with an ℓ_2 solution and solve the ℓ_{2^2} problem to a constant approximation, followed by $\ell_{2^3}, ..\ell_p$. We note that a constant approximate solution to the $\ell_{p/2}$-norm problem gives an $O(m)$ approximation to the ℓ_p problem and thus, we can solve $\log p$ problems where we can assume $\kappa = O(m)$.

We now complete the proof of our various algorithmic results by utilizing sparsification procedures specific to each problem.

ℓ_p Flows

We will prove Theorem 1 (Flow Algorithmic Result), with explicit p dependencies.

Proof. From Theorem 3, we obtain a sparse graph in $K = \widetilde{O}(m)$ time with $\tilde{n} = \widetilde{O}(n)$ edges. A constant factor approximation to the flow residual problem on this sparse graph when scaled by $\mu_2 = m^{-\frac{1}{p-1}}$ gives a $\kappa_2 = \widetilde{O}\left(m^{\frac{2}{p-1}}\right)$-approximate solution to the flow residual problem on the original graph. We can solve linear systems on the sparse graph in $\widetilde{O}(\tilde{n}) = \widetilde{O}(n)$ time using fast Laplacian solvers. Using all these values in Theorem 10, we get the final runtime to be $pm^{\frac{2}{p-1}+o(1)}\left(m + n^{1+\frac{p-2}{3p-2}}\right)\log^2\left(\frac{pm}{\varepsilon}\right)$ as claimed. We prove Theorem 3 in the full version. ◀

ℓ_p Voltages

We will prove Theorem 2 (Voltage Algorithmic Result), with explicit p dependencies.

Proof. From Theorem 4, we obtain a sparse graph in $K = \widetilde{O}(m)$ time with $\tilde{n} = \widetilde{O}(n)$ edges. A constant factor approximation to the voltage residual problem on this sparse graph when scaled by $\mu_2 = m^{-\frac{1}{p-1}}$ gives a $\kappa_2 = \widetilde{O}\left(m^{\frac{1}{p-1}}\right)$-approximate solution to the voltage residual problem on the original graph. We can solve linear systems on the sparse graph in $\widetilde{O}(\tilde{n}) = \widetilde{O}(n)$ time using fast Laplacian solvers. Using these values in Theorem 10, we get the final runtime to be $pm^{\frac{1}{p-1}+o(1)}\left(m + n^{1+\frac{p-2}{3p-2}}\right)\log^2\left(\frac{pm}{\varepsilon}\right)$ as claimed. We prove Theorem 4 in Section 4. ◀

General Matrices

We will now prove Theorem 6.

Proof. We assume Theorem 5, which we prove in Appendix (refer to full version). From the theorem, we have $\kappa_2 = O(1)$ and $\mu_2 = O(1)$. Note that $K = \mathrm{LSS}(\widehat{\boldsymbol{M}}) + \mathrm{LSS}(\widehat{\boldsymbol{N}})$ for some $\widehat{\boldsymbol{M}}, \widehat{\boldsymbol{N}} \in \mathbb{R}^{O(n\log(n))\times n}$, which is the time required to solve linear systems in $\widehat{\boldsymbol{M}}^\top \widehat{\boldsymbol{M}}$ and $\widehat{\boldsymbol{N}}^\top \widehat{\boldsymbol{N}}$, respectively. Since, by Theorem 5, the size of $\widetilde{\boldsymbol{M}}$ and $\widetilde{\boldsymbol{N}}$ is $\tilde{n} = O(n^{p/2}\log(n))$, the cost from the solver in Theorem 11 is $\tilde{O}_p\Big(\Big(\mathrm{LSS}(\widetilde{\boldsymbol{M}}) + \mathrm{LSS}(\widetilde{\boldsymbol{N}})\Big)n^{\frac{p(p-2)}{6p-4}}\Big)$. ◀

4 Construction of Sparsifiers for $\ell_2^2 + \ell_p^p$ Voltages

In this section, we prove a formal version of the voltage sparsification result (Theorem 4):

▶ **Theorem 12.** *Consider a graph $G = (V, E)$ with non-negative 2-weights $\boldsymbol{w} \in \mathbb{R}^E$ and non-negative p-weights $\boldsymbol{s} \in \mathbb{R}^E$, with m and n vertices. We can produce a graph $H = (V, F)$ with edges $F \subseteq E$, ℓ_2-weights $\boldsymbol{u} \in \mathbb{R}^F$, and ℓ_p-weights $\boldsymbol{t} \in \mathbb{R}^F$, such that with probability at least $1 - \delta$ the graph H has $O(n\log(n/\varepsilon))$ edges and*

$$\frac{1}{1.5}\|\boldsymbol{W}\boldsymbol{B}_G\boldsymbol{x}\|_2 \le \|\boldsymbol{U}\boldsymbol{B}_H\boldsymbol{x}\|_2 \le 1.5\|\boldsymbol{W}\boldsymbol{B}_G\boldsymbol{x}\|_2 \tag{6}$$

and for any $p \in [1, \infty]$

$$\frac{1}{m^{1/p}\log(n)}\|\boldsymbol{S}\boldsymbol{B}_G\boldsymbol{x}\|_p \le \|\boldsymbol{T}\boldsymbol{B}_H\boldsymbol{x}\|_p \le \|\boldsymbol{S}\boldsymbol{B}_G\boldsymbol{x}\|_p \tag{7}$$

where $\boldsymbol{W} = \textsc{Diag}(\boldsymbol{w})$, $\boldsymbol{U} = \textsc{Diag}(\boldsymbol{u})$, $\boldsymbol{S} = \textsc{Diag}(\boldsymbol{s})$, $\boldsymbol{T} = \textsc{Diag}(\boldsymbol{t})$. We denote the routine computing H and $\boldsymbol{u}, \boldsymbol{t}$ by SpannerSparsify, so that $(H, \boldsymbol{u}, \boldsymbol{t}) = \textsc{SpannerSparsify}(G, \boldsymbol{w}, \boldsymbol{s})$. This algorithm runs in $\widetilde{O}(m\log(1/\delta))$ time.

We will first define some terms required for our result. Given a undirected graph $G = (V, E)$, with edge lengths $\boldsymbol{l} \in \mathbb{R}^E$, and $u, v \in V$, we let $d_G(u, v)$ denote the shortest path distance in G w.r.t $\boldsymbol{l}$, so that if P is the shortest path w.r.t $\boldsymbol{l}$ then

$$d_{G,\boldsymbol{l}}(u, v) = \sum_{e\in P} \boldsymbol{l}(e)$$

▶ **Definition 13.** *Given a undirected graph $G = (V, E)$ with edge lengths $\boldsymbol{l} \in \mathbb{R}^E$, a K-spanner is a subgraph H of G with the same edge lengths s.t. $d_H(u, v) \le K d_G(u, v)$.*

Baswana and Sen showed the following result on spanners [7].

▶ **Theorem 14.** *Given an undirected graph $G = (V, E, \boldsymbol{l})$ with m edges and n vertices, and an integer $k > 1$, we can compute a $(2k-1)$-spanner H of G with $O(n^{1+1/k})$ edges in expected time $O(km)$.*

▶ **Lemma 15.** *Given an undirected graph $G = (V, E)$ with positive edge lengths $\boldsymbol{l} \in \mathbb{R}^E$, and a K-spanner $H = (V, F)$ of G, for all $\boldsymbol{x} \in \mathbb{R}^V$ we have*

$$\max_{(u,v)\in F} \frac{1}{\boldsymbol{l}(u,v)}|\boldsymbol{x}(u) - \boldsymbol{x}(v)| \le \max_{(u,v)\in E} \frac{1}{\boldsymbol{l}(u,v)}|\boldsymbol{x}(u) - \boldsymbol{x}(v)| \le K \max_{(u,v)\in F} \frac{1}{\boldsymbol{l}(u,v)}|\boldsymbol{x}(u) - \boldsymbol{x}(v)|$$

Proof. The inequality $\max_{(u,v)\in F} \frac{1}{\boldsymbol{l}(u,v)}|\boldsymbol{x}(u)-\boldsymbol{x}(v)| \leq \max_{(u,v)\in E} \frac{1}{\boldsymbol{l}(u,v)}|\boldsymbol{x}(u)-\boldsymbol{x}(v)|$ is immediate from $F \subseteq E$.

To prove the second inequality, we note that if $(u,v) \in E$ has shortest path P in H then

$$\frac{1}{\boldsymbol{l}(u,v)}|\boldsymbol{x}(u)-\boldsymbol{x}(v)| \leq \frac{K}{\sum_{(z,y)\in P}\boldsymbol{l}(z,y)}\left|\sum_{(z,y)\in P}\boldsymbol{x}(z)-\boldsymbol{x}(y)\right| \leq \max_{(z,y)\in P}\frac{K}{\boldsymbol{l}(z,y)}|\boldsymbol{x}(z)-\boldsymbol{x}(y)|.$$

◀

▶ **Definition 16.** *Given a undirected graph $G=(V,E)$ with m edges and n vertices with positive edge ℓ_2-weights $\boldsymbol{w} \in \mathbb{R}^E$, a spectral ε-approximation of G is a graph $H=(V,F)$ with $F \subseteq E$ with positive edge ℓ_2-weights $\boldsymbol{u} \in \mathbb{R}^F$ s.t.*

$$\frac{1}{1+\varepsilon}\|\boldsymbol{W}\boldsymbol{B}_G\boldsymbol{x}\|_2 \leq \|\boldsymbol{U}\boldsymbol{B}_H\boldsymbol{x}\|_2 \leq (1+\varepsilon)\|\boldsymbol{W}\boldsymbol{B}_G\boldsymbol{x}\|_2$$

where $\boldsymbol{W} = \textsc{Diag}(\boldsymbol{w})$ and $\boldsymbol{U} = \textsc{Diag}(\boldsymbol{u})$.

The following result on spectral sparsifiers was shown by Spielman and Srivastava [40] (see also [43]).

▶ **Theorem 17.** *Given a graph $G=(V,E)$ with positive ℓ_2-weights $\boldsymbol{w} \in \mathbb{R}^E$ with m edges and n vertices, for any $\varepsilon \in (0,1/2]$, we can produce a graph $H=(V,F)$ with edges $F \subseteq E$ and ℓ_2-weights $\boldsymbol{u} \in \mathbb{R}^F$ such that H has $O(n\varepsilon^{-2}\log(n/\delta))$ edges and with probability at least $1-\delta$ we have that $(H,\boldsymbol{u})$ is a spectral ε-approximation of $(G,\boldsymbol{w})$. We denote the routine computing H and $\boldsymbol{u}$ by SpectralSparsify, so that $(H,\boldsymbol{u}) = \textsc{SpectralSparsify}(G,\boldsymbol{s},\varepsilon,\delta)$. This algorithm runs in $\widetilde{O}(m)$ time. Furthermore, if the weights $\boldsymbol{w}$ are quasipolynomially bounded, then so are the weights of $\boldsymbol{u}$.*

We can now prove our main result.

Proof of Theorem 12. We consider a graph $G=(V,E)$ with m edges and n vertices, and with non-negative ℓ_p-weights $\boldsymbol{r} \in \mathbb{R}^E$, non-negative ℓ_2-weights $\boldsymbol{s} \in \mathbb{R}^E$. We define $\hat{E} \subseteq E$ to be the edges s.t. $\boldsymbol{s}(e) > 0$, and then let $\boldsymbol{l} \in \mathbb{R}^{\hat{E}}$ by $\boldsymbol{l}(e) = 1/\boldsymbol{s}(e)$, and $\hat{G} = (V,\hat{E})$. We then apply Theorem 14 to $\hat{G}$ with $\boldsymbol{l}$ as edge lengths, and with $k = \log(n)$. We turn the algorithm of Theorem 14 into running time $\widetilde{O}(m\log(1/\delta))$, instead of expected time $\widetilde{O}(m)$, by applying the standard Las Vegas to Monte-Carlo reduction. With probability $1-\delta/2$, this gives us a $\log n$-spanner H_1 of $\hat{G}$, and we define $\boldsymbol{t}$ by restricting $\boldsymbol{s}$ to the edges of H_1. By Lemma 15, we then have

$$\|\boldsymbol{T}\boldsymbol{B}_{H_1}\boldsymbol{x}\|_\infty \leq \|\boldsymbol{S}\boldsymbol{B}_G\boldsymbol{x}\|_\infty \leq \log(n)\|\boldsymbol{T}\boldsymbol{B}_{H_1}\boldsymbol{x}\|_\infty$$

Because $\boldsymbol{T}\boldsymbol{B}_{H_1}\boldsymbol{x}$ is a restriction of $\boldsymbol{S}\boldsymbol{B}_G\boldsymbol{x}$ to a subset of the coordinates, we always have for any $p \geq 1$ that $\|\boldsymbol{T}\boldsymbol{B}_{H_1}\boldsymbol{x}\|_p \leq \|\boldsymbol{S}\boldsymbol{B}_G\boldsymbol{x}\|_p$.

At the same time, we also have

$$\|\boldsymbol{S}\boldsymbol{B}_G\boldsymbol{x}\|_p \leq m^{1/p}\|\boldsymbol{S}\boldsymbol{B}_G\boldsymbol{x}\|_\infty \leq m^{1/p}\log(n)\|\boldsymbol{T}\boldsymbol{B}_{H_1}\boldsymbol{x}\|_\infty \leq m^{1/p}\log(n)\|\boldsymbol{T}\boldsymbol{B}_{H_1}\boldsymbol{x}\|_p$$

We define $\tilde{E} \subseteq E$ to be the edges s.t. $\boldsymbol{r}(e) > 0$, and the let $\tilde{G} = (V,\tilde{E})$. Now, appealing to Theorem 17, we let $(H_2,\boldsymbol{u}) = \textsc{SpectralSparsify}(\tilde{G},\boldsymbol{r},1/2,\varepsilon/2)$.

Finally, we form H by taking the union of the edge sets of H_1 and H_2 and extending $\boldsymbol{u}$ and $\boldsymbol{t}$ to the new edge set by adding zero entries as needed. By a union bound, the approximation guarantees of Equations (6) and (7) simultaneously hold with probability at least $1-\delta$.

The edge set remains bounded in size by $O(n\log n)$. ◀

To see Theorem 4, note that from Theorem 12, we get,

$$m^{-\frac{1}{p-1}}\left(m^{-\frac{1}{p-1}}\|\boldsymbol{W}\boldsymbol{B}_G\boldsymbol{x}\|_2^2+m^{-1}\|\boldsymbol{S}\boldsymbol{B}_G\boldsymbol{x}\|_p^p\right)\leq m^{-\frac{1}{p-1}}\left(\|\boldsymbol{U}\boldsymbol{B}_H\boldsymbol{x}\|_2^2+\|\boldsymbol{T}\boldsymbol{B}_H\boldsymbol{x}\|_p^p\right)$$

The other direction is easy to see.

5 Extensions of Our Results and Open Problems

Solving dual problems: q-norm minimizing flows and voltages for $q < 2$

When the mixed $(\ell_2^2+\ell_p^p)$-objective flow problem (Problem (1)) is restricted to the case $\boldsymbol{g}=\boldsymbol{0}$ and $\boldsymbol{R}=\boldsymbol{0}$, it becomes a pure ℓ_p-norm minimizing flow problem, and its dual problem can be slightly rearranged to give

$$\min_{\boldsymbol{v}}\boldsymbol{d}^\top\boldsymbol{v}+\left\|\boldsymbol{S}^{-1}\boldsymbol{B}\boldsymbol{v}\right\|_q^q \tag{8}$$

where $q=p/(p-1)=1+1/(p-1)$. We refer to the diagonal entries of $\boldsymbol{S}^{-1}$ as ℓ_q-conductances. Because we can solve Problem (1) to high-accuracy in near-linear time for $p=\omega(1)$, this allows us to solve Problem (8), the dual voltage ℓ_q-norm minimization, in time $p(m^{1+o(1)}+n^{4/3+o(1)})\log^2 1/\varepsilon$ (see [1, Section 7] for the reduction). We summarize this in the theorem below.

▶ **Theorem 18** (Voltage Algorithmic Result, $q<2$ (Informal)). *Consider a graph G with n vertices and m edges, equipped with positive ℓ_q-conductances, as well as a demand vector. For $1<q<2$, when $q=1+o(1)$, in $\mathrm{poly}\left(\frac{1}{q-1}\right)(m^{1+o(1)}+n^{4/3+o(1)})\log^2 1/\varepsilon$ time, we can compute an ε-approximately optimal voltage solution to Problem (8) with high probability.*

Similarly, we can solve ℓ_q-norm minimizing flows for $q<2$ as dual to the ℓ_p-voltage problem, a special case of the mixed $(\ell_2^2+\ell_p^p)$-voltage problem. Picking $\boldsymbol{W}=\boldsymbol{0}$ in Problem (2), we obtain a pure ℓ_p-norm minimizing voltage problem, and its dual problem can be slightly rearranged to give

$$\min_{\boldsymbol{B}^\top\boldsymbol{f}=\boldsymbol{d}}\left\|\boldsymbol{U}^{-1}\boldsymbol{f}\right\|_q^q \tag{9}$$

where $q=p/(p-1)=1+1/(p-1)$. We refer to the diagonal entries of $\boldsymbol{U}^{-1}$ as q-weights. Again, because we can solve Problem (2) to high-accuracy in near-linear time for $p=\omega(1)$, this allows us to solve Problem (9), the dual flow ℓ_q-norm minimization, in time $p(m^{1+o(1)}+n^{4/3+o(1)})\log^2 1/\varepsilon$.

▶ **Theorem 19** (Flow Algorithmic Result, $q<2$ (Informal)). *Consider a graph G with n vertices and m edges, equipped with positive q-weights, as well as a demand vector. For $1<q<2$, when $q=1+o(1)$, in $\mathrm{poly}\left(\frac{1}{q-1}\right)(m^{1+o(1)}+n^{4/3+o(1)})\log^2 1/\varepsilon$ time, we can compute an ε-approximately optimal flow solution to Problem (9) with high probability.*

Open Questions

Mixed ℓ_2,ℓ_q problems for small $q<2$. In this work, we provided new state-of-the-art algorithms for weighted mixed ℓ_2,ℓ_p-norm minimizing flow and voltage problems for $p>>2$, and for pure ℓ_q-norm minimizing flow and voltage problems for q near 1.

A reasonable definition of mixed ℓ_2,ℓ_q-norm problems for $q<2$ is based on gamma-functions as introduced in [9] and used in [1]. We believe that with minor adjustments to our multiplicative weights solver, these objectives could be handled too, by solving their dual ℓ_2,ℓ_p-gamma function problem for $p>2$.

Directly sparsifying mixed ℓ_2, ℓ_q problems for $q < 2$. A second approach to developing a fast ℓ_2, ℓ_q-gamma function solver for $q < 2$ would be to directly develop sparsification in this setting. We believe this might be possible, and in the general matrix setting might provide better algorithms than alternative approaches.

Removing the $m^{\frac{O(1)}{p-1}}$ loss in sparsification. Our current approaches to graph mixed ℓ_2, ℓ_p-sparsification lose a factor $m^{\frac{O(1)}{p-1}}$ in their quality of approximation, which leads to a $m^{\frac{O(1)}{p-1}}$ factor slowdown in running time, and makes our algorithms less useful for small p. We believe a more sophisticated graph sparsification routine could remove this loss and result in significantly faster algorithms for p close to 2.

Using mixed ℓ_2, ℓ_p-objectives as oracles for ℓ_∞ regression. The current state-of-the-art algorithm for computing maximum flow in unit capacity graphs runs in $\widetilde{O}(m^{4/3})$ time [21], and uses the almost-linear-time algorithm from [25] for solving *unweighted* $\ell_2^2 + \ell_p^p$ instances as a key ingredient.

References

1. Deeksha Adil, Rasmus Kyng, Richard Peng, and Sushant Sachdeva. Iterative refinement for ℓ_p-norm regression. In *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1405–1424. SIAM, 2019.
2. Deeksha Adil, Richard Peng, and Sushant Sachdeva. Fast, provably convergent irls algorithm for p-norm linear regression. In *Advances in Neural Information Processing Systems*, pages 14189–14200, 2019.
3. Deeksha Adil and Sushant Sachdeva. Faster p-norm minimizing flows, via smoothed q-norm problems. In *Proceedings of the Fourteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 892–910. SIAM, 2020.
4. Ravindra K. Ahuja, Thomas L. Magnanti, and James B. Orlin. *Network flows - theory, algorithms and applications*. Prentice Hall, 1993.
5. Zeyuan Allen-Zhu, Yuanzhi Li, Rafael Oliveira, and Avi Wigderson. Much faster algorithms for matrix scaling. In *2017 IEEE 58th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 890–901. IEEE, 2017.
6. Ingo Althöfer, Gautam Das, David Dobkin, Deborah Joseph, and José Soares. On sparse spanners of weighted graphs. *Discrete & Computational Geometry*, 9(1):81–100, 1993. doi:10.1007/BF02189308.
7. Surender Baswana and Sandeep Sen. A simple and linear time randomized algorithm for computing sparse spanners in weighted graphs. *Random Struct. Algorithms*, 30(4):532–563, 2007.
8. András A. Benczúr and David R. Karger. Approximating s-t minimum cuts in $\tilde{O}(n^2)$ time. In *Proceedings of the twenty-eighth annual ACM symposium on Theory of computing*, STOC '96, pages 47–55, New York, NY, USA, 1996. ACM. doi:10.1145/237814.237827.
9. Sébastien Bubeck, Michael B. Cohen, Yin Tat Lee, and Yuanzhi Li. An homotopy method for lp regression provably beyond self-concordance and in input-sparsity time. In *Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing*, STOC 2018, pages 1130–1137, New York, NY, USA, 2018. ACM. doi:10.1145/3188745.3188776.
10. Hui Han Chin, Aleksander Madry, Gary L. Miller, and Richard Peng. Runtime guarantees for regression problems. In *Proceedings of the 4^{th} conference on Innovations in Theoretical Computer Science*, ITCS '13, pages 269–282, New York, NY, USA, 2013. ACM.
11. Paul Christiano, Jonathan A. Kelner, Aleksander Madry, Daniel A. Spielman, and Shang-Hua Teng. Electrical flows, laplacian systems, and faster approximation of maximum flow in undirected graphs. In *Proceedings of the 43rd annual ACM symposium on Theory of computing*, STOC '11, pages 273–282, New York, NY, USA, 2011. ACM. doi:10.1145/1993636.1993674.

12 T. Chu, Y. Gao, R. Peng, S. Sachdeva, S. Sawlani, and J. Wang. Graph sparsification, spectral sketches, and faster resistance computation, via short cycle decompositions. In *2018 IEEE 59th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 361–372, October 2018. doi:10.1109/FOCS.2018.00042.

13 M. B. Cohen, A. Madry, D. Tsipras, and A. Vladu. Matrix scaling and balancing via box constrained newton's method and interior point methods. In *2017 IEEE 58th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 902–913, October 2017. doi:10.1109/FOCS.2017.88.

14 Michael B. Cohen, Yin Tat Lee, Cameron Musco, Christopher Musco, Richard Peng, and Aaron Sidford. Uniform sampling for matrix approximation. In *Proceedings of the 2015 Conference on Innovations in Theoretical Computer Science*, ITCS '15, page 181–190, New York, NY, USA, 2015. Association for Computing Machinery. doi:10.1145/2688073.2688113.

15 Michael B. Cohen, Yin Tat Lee, and Zhao Song. Solving linear programs in the current matrix multiplication time. In *Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing*, STOC 2019, page 938–942, New York, NY, USA, 2019. Association for Computing Machinery. doi:10.1145/3313276.3316303.

16 Michael B. Cohen, Aleksander Madry, Piotr Sankowski, and Adrian Vladu. Negative-weight shortest paths and unit capacity minimum cost flow in $\tilde{O}(m^{10/7} \log w)$ time (extended abstract). In *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2017, Barcelona, Spain, Hotel Porta Fira, January 16-19*, pages 752–771, 2017.

17 Michael B. Cohen and Richard Peng. ℓ_p row sampling by lewis weights. In *Proceedings of the Forty-Seventh Annual ACM Symposium on Theory of Computing*, STOC '15, page 183–192, New York, NY, USA, 2015. Association for Computing Machinery. doi:10.1145/2746539.2746567.

18 Samuel I. Daitch and Daniel A. Spielman. Faster approximate lossy generalized flow via interior point algorithms. In *Proceedings of the 40th annual ACM symposium on Theory of computing*, STOC '08, pages 451–460, New York, NY, USA, 2008. ACM. Available at arXiv:0803.0988. doi:10.1145/1374376.1374441.

19 David Durfee, John Peebles, Richard Peng, and Anup B. Rao. Determinant-preserving sparsification of SDDM matrices with applications to counting and sampling spanning trees. In *FOCS*, pages 926–937. IEEE Computer Society, 2017.

20 Andrew V. Goldberg and Robert Endre Tarjan. Efficient maximum flow algorithms. *Commun. ACM*, 57(8):82–89, 2014.

21 Tarun Kathuria, Yang P. Liu, and Aaron Sidford. Unit capacity maxflow in almost $m^{4/3}$ time. In *2020 IEEE 61st Annual Symposium on Foundations of Computer Science (FOCS)*, pages 119–130, 2020. doi:10.1109/FOCS46700.2020.00020.

22 Jonathan A. Kelner, Yin Tat Lee, Lorenzo Orecchia, and Aaron Sidford. An almost-linear-time algorithm for approximate max flow in undirected graphs, and its multicommodity generalizations. In *Proceedings of the Twenty-Fifth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2014, Portland, Oregon, USA, January 5-7, 2014*, pages 217–226, 2014.

23 Ioannis Koutis, Gary L. Miller, and Richard Peng. A nearly-$m \log n$ time solver for SDD linear systems. In *Proceedings of the 2011 IEEE 52nd Annual Symposium on Foundations of Computer Science*, FOCS '11, pages 590–598, Washington, DC, USA, 2011. IEEE Computer Society. doi:10.1109/FOCS.2011.85.

24 Rasmus Kyng, Yin Tat Lee, Richard Peng, Sushant Sachdeva, and Daniel A Spielman. Sparsified cholesky and multigrid solvers for connection laplacians. In *Proceedings of the 48th Annual ACM SIGACT Symposium on Theory of Computing*, pages 842–850. ACM, 2016.

25 Rasmus Kyng, Richard Peng, Sushant Sachdeva, and Di Wang. Flows in almost linear time via adaptive preconditioning. In *Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing*, STOC 2019, pages 902–913, New York, NY, USA, 2019. ACM. doi:10.1145/3313276.3316410.

26 Rasmus Kyng, Anup Rao, Sushant Sachdeva, and Daniel A. Spielman. Algorithms for lipschitz learning on graphs. In Peter Grünwald, Elad Hazan, and Satyen Kale, editors, *Proceedings of The 28th Conference on Learning Theory*, volume 40 of *Proceedings of Machine Learning Research*, pages 1190–1223, Paris, France, July 2015. PMLR.

27 Y. T. Lee and A. Sidford. Path finding methods for linear programming: Solving linear programs in $\tilde{O}(\text{vrank})$ iterations and faster algorithms for maximum flow. In *FOCS*, 2014.

28 Yang P. Liu and Aaron Sidford. Faster energy maximization for faster maximum flow. In *Proccedings of the 52nd Annual ACM SIGACT Symposium on Theory of Computing, STOC 2020, Chicago, IL, USA, June 22-26, 2020*, pages 803–814. ACM, 2020. doi:10.1145/3357713.3384247.

29 A. Madry. Navigating central path with electrical flows: From flows to matchings, and back. In *FOCS*, 2013.

30 Aleksander Madry. Fast approximation algorithms for cut-based problems in undirected graphs. In *Foundations of Computer Science (FOCS), 2010 51st Annual IEEE Symposium on*, pages 245–254. IEEE, 2010.

31 Y. Nesterov and A. Nemirovskii. *Interior-Point Polynomial Algorithms in Convex Programming*. Society for Industrial and Applied Mathematics, 1994. doi:10.1137/1.9781611970791.

32 Lorenzo Orecchia, Sushant Sachdeva, and Nisheeth K. Vishnoi. Approximating the exponential, the lanczos method and an $\tilde{O}(m)$-time spectral algorithm for balanced separator. In *Proceedings of the Forty-fourth Annual ACM Symposium on Theory of Computing*, STOC '12, pages 1141–1160, New York, NY, USA, 2012. ACM. doi:10.1145/2213977.2214080.

33 David Peleg and Alejandro A. Schäffer. Graph spanners. *Journal of Graph Theory*, 13(1):99–116, 1989. doi:10.1002/jgt.3190130114.

34 Richard Peng and Daniel A. Spielman. An efficient parallel solver for SDD linear systems. In *Proceedings of the 46th Annual ACM Symposium on Theory of Computing*, STOC '14, pages 333–342, New York, NY, USA, 2014. ACM.

35 Harald Racke. Optimal hierarchical decompositions for congestion minimization in networks. In *Proceedings of the 40th annual ACM symposium on Theory of computing*, STOC '08, pages 255–264, New York, NY, USA, 2008. ACM.

36 Harald Racke, Chintan Shah, and Hanjo Taubig. Computing cut-based hierarchical decompositions in almost linear time. In *Proceedings of the 25th Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '14, pages 227–238, 2014.

37 Alexander Schrijver. On the history of the transportation and maximum flow problems. *Math. Program.*, 91(3):437–445, 2002.

38 Jonah Sherman. Nearly maximum flows in nearly linear time. In *54th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2013, 26-29 October, 2013, Berkeley, CA, USA*, pages 263–269, 2013.

39 Jonah Sherman. Generalized preconditioning and undirected minimum-cost flow. In *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '17, pages 772–780, 2017.

40 D. Spielman and N. Srivastava. Graph sparsification by effective resistances. *SIAM Journal on Computing*, 40(6):1913–1926, 2011. doi:10.1137/080734029.

41 D. Spielman and S. Teng. Spectral sparsification of graphs. *SIAM Journal on Computing*, 40(4):981–1025, 2011. doi:10.1137/08074489X.

42 D.A. Spielman and S. Teng. Nearly-linear time algorithms for graph partitioning, graph sparsification, and solving linear systems. In *STOC*, 2004.

43 Daniel A. Spielman. Spectral graph theory lectures: Sparsification by effective resistance sampling, 2015.

44 Jan van den Brand, Yin-Tat Lee, Danupon Nanongkai, Richard Peng, Thatchaphol Saranurak, Aaron Sidford, Zhao Song, and Di Wang. Bipartite matching in nearly-linear time on moderately dense graphs. In *2020 IEEE 61st Annual Symposium on Foundations of Computer Science (FOCS)*, pages 919–930. IEEE, 2020.

An Output-Sensitive Algorithm for Computing the Union of Cubes and Fat Boxes in 3D

Pankaj K. Agarwal
Department of Computer Science, Duke University, Durham, NC, USA

Alex Steiger
Department of Computer Science, Duke University, Durham, NC, USA

Abstract

Let $\mathcal{C}$ be a set of n axis-aligned cubes of arbitrary sizes in $\mathbb{R}^3$. Let $\mathcal{U}$ be their union, and let κ be the number of vertices on $\partial\mathcal{U}$; κ can vary between $O(1)$ and $O(n^2)$. We show that $\mathcal{U}$ can be computed in $O(n\log^3 n+\kappa)$ time if $\mathcal{C}$ is in general position. The algorithm also computes the union of a set of fat boxes (i.e., boxes with bounded aspect ratio) within the same time bound. If the cubes in $\mathcal{C}$ are congruent or have bounded depth, the running time improves to $O(n\log^2 n)$, and if both conditions hold, the running time improves to $O(n\log n)$.

2012 ACM Subject Classification Theory of computation → Computational geometry

Keywords and phrases union of cubes, fat boxes, plane-sweep

Digital Object Identifier 10.4230/LIPIcs.ICALP.2021.10

Category Track A: Algorithms, Complexity and Games

Funding Partially supported by NSF grants IIS-18-14493 and CCF-20-07556.

1 Introduction

Let $\mathcal{C}$ be a set of n axis-aligned cubes of arbitrary sizes in $\mathbb{R}^3$. Let $\mathcal{U}:=\mathcal{U}(\mathcal{C})$ be their union, and let κ be the number of vertices on $\partial\mathcal{U}$. It is known that $\kappa=\Theta(n^2)$ in the worst case, though it is linear or near-linear in many special cases. For example, $\kappa=O(n)$ if the cubes in $\mathcal{C}$ have roughly the same size [6] or if they have bounded depth (i.e., any point in $\mathbb{R}^3$ lies in $O(1)$ cubes) [14]. If their sizes are drawn independently from an arbitrary distribution, the expected complexity of their union is $O(n\log^2 n)$ [3]. A natural problem is to develop an algorithm for computing $\partial\mathcal{U}$, by which we mean compute its vertices, edges, and faces. Although $\mathcal{U}$ can be computed in $O(n^2\log n)$ time by computing $\partial\mathcal{U}$ on each face of every cube of $\mathcal{C}$, an output-sensitive algorithm with $O(n\log n+\kappa)$ running time has remained elusive. In this paper we present an algorithm that almost matches this running time.

Related work. Motivated by VLSI design and other applications, the problem of computing the union of a set of axis-aligned rectangles in $\mathbb{R}^2$ and its variants have been studied since the 1970's; see [17]. An optimal $O(n\log n+\kappa)$-time algorithm was presented by Güting [12].

A closely related problem, which also has been studied extensively, is the so-called *Klee's measure problem*, which asks to compute the *volume* of the union of axis-aligned boxes in $\mathbb{R}^d$. For $d=2$, Bentley presented an $O(n\log n)$-time algorithm for this problem, which extends to higher dimensions and computes the volume in $O(n^{d-1}\log n)$ time. For $d\geq 3$, Overmars and Yap [16] gave an $O(n^{d/2}\log n)$-time algorithm. The running time was improved to $O(n^{d/2})$ by Chan [8]. It was an open question whether a faster algorithm exists if the input boxes are hypercubes or fat. Agarwal *et al.* [2] described an $O(n^{4/3}\log n)$-time algorithm for cubes in 3D, which was subsequently improved to $O(n\log^4 n)$ in [1]. Bringmann [7] presented an $O(n^{(d+2)/3})$-time algorithm for fat boxes in $\mathbb{R}^d$, which was later improved to $O(n^{(d+1)/3}\,\mathrm{polylog}(n))$ in [8]; see also [19].

Despite extensive work on Klee's measure problem, relatively less is known about computing the union boundary. For example, no $O(n \log n)$-time algorithm is known even for computing the union of unit cubes in $\mathbb{R}^3$. For $d = 3$, the Overmars-Yap algorithm can be adapted to compute the boundary of the union of boxes in $O((n^{3/2} + \kappa) \log n)$ time, where κ is the output size. But it is not obvious whether the algorithm in [1] can be adapted to compute the union of axis-aligned cubes in $O((n + \kappa)\,\mathrm{polylog}(n))$ time. Agarwal *et al.* [4] have presented a randomized algorithm that computes the union of congruent cubes in $\mathbb{R}^3$ in $O(n^{1+\varepsilon})$ expected time, for any constant $\varepsilon > 0$.

A related line of work is to partition the union (or its complement) into few axis-aligned boxes. Chew *et al.* [9] describe an algorithm to compute a partition of the union of n congruent cubes into $O(n)$ axis-aligned boxes in $O(n \log n)$ time, assuming that the union has already been computed[1]. For a special case of orthants in $\mathbb{R}^d$, Kaplan *et al.* [15] describe an algorithm to partition the union of n such orthants into $O(\kappa)$ axis-aligned (semiunbounded) boxes in $O((n + \kappa) \log^{d-1} n)$ time, for any $d \geq 1$, where κ is the union complexity.

Our results. The main result of the paper is an output-sensitive algorithm to compute $\mathcal{U}$. That is, our algorithm computes the vertices, edges, and 2D faces of $\partial\mathcal{U}$. For each 2D face f, it computes the components of ∂f. We say that $\mathcal{C}$ is in *general position* if any plane contains the boundary face of at most one cube in $\mathcal{C}$. Our algorithm assumes $\mathcal{C}$ to be in general position. Although it can be extended to degenerate configurations using symbolic perturbation techniques (e.g., [10]), the running time depends on the union complexity of the perturbed configuration.

▶ **Theorem 1.** *Given a set $\mathcal{C}$ of n axis-aligned cubes in $\mathbb{R}^3$, $\mathcal{U}(\mathcal{C})$ can be computed in $O(n \log^3 n + \kappa)$ time if $\mathcal{C}$ is in general position. If $\mathcal{C}$ is not in general position, the running time is $O(n \log^3 n + \widehat{k})$, where $\widehat{k}$ is the maximum complexity of $\mathcal{U}(\mathcal{C})$ under an infinitesmal perturbation.*

A 3D box is *fat* if its aspect ratio (i.e., the ratio of its longest side length and its shortest side length) is bounded by a constant. A fat box can be decomposed into $O(1)$ (possibly intersecting) cubes. Replacing fat boxes in general position with $O(1)$ such cubes then perturbing them into general position increases the union complexity at most by a constant factor. From Theorem 1, we have the following:

▶ **Corollary 2.** *Given a set $\mathcal{B}$ of n axis-aligned fat boxes in $\mathbb{R}^3$, where the aspect ratio of each box is bounded by a constant, $\mathcal{U}(\mathcal{B})$ can be computed in $O(n \log^3 n + \kappa)$ time, assuming $\mathcal{B}$ is in general position. If $\mathcal{B}$ is not in general position, the running time is $O(n \log^3 n + \widehat{k})$, where $\widehat{k}$ is the maximum complexity of $\mathcal{U}(\mathcal{B})$ under an infinitesmal perturbation.*

For some special cases, simpler algorithms compute the union slightly more efficiently: when all cubes in $\mathcal{C}$ are congruent or when they have bounded *depth* (i.e., any point in $\mathbb{R}^3$ is contained in at most c cubes of $\mathcal{C}$, for a constant $c > 0$). In both cases, $\kappa = O(n)$. Since the output size is always linear after perturbing the cubes to be in general position, we do not need the general-position assumption here.

▶ **Theorem 3.** *Let $\mathcal{C}$ be a set of n axis-aligned cubes in $\mathbb{R}^3$. If all cubes in $\mathcal{C}$ are congruent or they have bounded depth, then $\mathcal{U}(\mathcal{C})$ can be computed in $O(n \log^2 n)$ time. If the cubes in $\mathcal{C}$ are congruent and have bounded depth, then $\mathcal{U}(\mathcal{C})$ can be computed in $O(n \log n)$ time.*

---

[1] Theorem 1 in [9] suggests that the union $\mathcal{U}(\mathcal{C})$ of n unit axis-aligned cubes $\mathcal{C}$ in $\mathbb{R}^3$ can be computed in $O(n \log n)$, but no such algorithm is presented in the paper. It shows that given $\mathcal{U}(\mathcal{C})$, it can be decomposed into $O(n)$ boxes in $O(n \log n)$ time [11].

Analogous to Corollary 2, we obtain the following:

▶ **Corollary 4.** *Given a set $\mathcal{B}$ of n axis-aligned fat boxes in $\mathbb{R}^3$. If the ratio of the largest to the smallest size box is bounded by a constant or they have bounded depth, then $\mathcal{U}(\mathcal{B})$ can be computed in $O(n \log^2 n)$ time. If both conditions hold, then the running time is $O(n \log n)$.*

At a high level, the general approach of our algorithm is similar to Agarwal's [1] to compute the volume of the union of 3D cubes, but ours is considerably simpler and more efficient. We reduce the problem to maintaining the union of a set of squares in the plane under insertions and deletions, which are the xy-projections of the faces of cubes in $\mathcal{C}$. At the core of the data structure in [1] is a hierarchical decomposition of the plane using a variant of a kd-tree. In constrast, our data structure is a variant of a quadtree whose regions are squares. Using this property – having square regions – we crucially circumvent much of the intracacies in [1] and attain a simpler algorithm. We use a sweep-line algorithm to compute the changes in the union-boundary of squares and rely on auxiliary data structures, which are also somewhat simpler than in [1] to perform this sweep efficiently. Finally, we are also able to simplify and improve the algorithm because we can charge time to the $O(\kappa)$ vertices reported.

Roadmap of the paper. As a warm-up, we first present in Section 2 an algorithm for a special case where the spread of the xy-projections of the vertices of the cubes in $\mathcal{C}$ (regarded as a 2D point set) is (polynomially) bounded, i.e., the ratio of the distance between the farthest and closest pairs of such points in $\mathbb{R}^2$ is bounded by n^c, for some constant $c > 0$. In Section 3 we describe how to remove this assumption to obtain our main result (Theorem 1). Finally, we describe the more efficient algorithms for congruent cubes and cubes with bounded depth (Theorem 3) in Section 4.

2 Algorithm for the Bounded Spread Case

Let $\mathcal{C} := \{C_1, \ldots, C_n\}$ be a set of n axis-aligned cubes in $\mathbb{R}^3$ in general position. We assume that the spread of the xy-projections of the vertices of $\mathcal{C}$ is polynomially bounded. For any set A of objects (e.g. segments in $\mathbb{R}$, squares in $\mathbb{R}^2$, or cubes in $\mathbb{R}^3$), let $\mathcal{U}(A)$ denote the union of the objects in A, and let $V(A)$ be the vertices of $\mathcal{U}(A)$. Set $\mathcal{U} := \mathcal{U}(\mathcal{C})$. For any 3D object a, let $a^{\downarrow}$ be the xy-projection of a.

In this section, we describe an algorithm to compute the boundary of $\mathcal{U}$, denoted by $\partial\mathcal{U}$, namely its vertices, edges, and faces. Once the vertices of $\partial\mathcal{U}$ have been computed, the edges and faces can be computed using standard techniques, so we first focus only on computing the vertices and remark at the end of the section how to extend it to compute edges and faces of $\partial\mathcal{U}$.

2.1 Overview of the algorithm

We first introduce some notation. For a cube $C_i \in \mathcal{C}$, let $S_i := C_i^{\downarrow}$ be the xy-projection of C_i, which is an axis-aligned square in $\mathbb{R}^2$. Let $z_1 < \ldots < z_{2n}$ be the z-coordinates of the vertices of cubes in $\mathcal{C}$, sorted in decreasing order. For all i with $1 \le i \le 2n$, let $\mathcal{C}_i \subseteq \mathcal{C}$ be the set of cubes whose z-spans cover the interval (z_i, z_{i+1}), and let $\mathcal{S}_i = \{S_j \mid C_j \in \mathcal{C}_i\}$. Let V_i denote the set of vertices of $\mathcal{U}(\mathcal{S}_i)$.

Note that every vertex of $\mathcal{U}$ is the intersection of three orthogonal faces of cubes in $\mathcal{C}$ – in particular, every vertex lies on a xy-face of some cube, i.e., the z-coordinate of every vertex is one of the z_i's, and is incident to a z-edge of $\mathcal{U}$. Therefore, our high-level approach is to

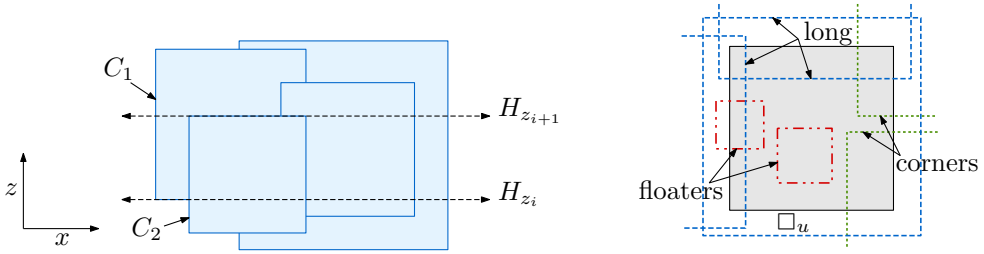


Figure 1 (left) A 2D view of $\mathcal{C}$. The bottom (resp. top) face of cube C_1 (resp. C_2) lies on the xy-plane $H_{z_i} : z = z_i$ (resp. $H_{z_{i+1}} : z = z_{i+1}$). (right) Various long and short squares at node u.

sweep a horizontal plane H in the $(+z)$-direction, from z_1 to z_{2n}, and maintain $H \cap \mathcal{U}$ in the process. We stop the sweep at each z_i and report the vertices of $\mathcal{U}$ that are incident on the face of the cube of $\mathcal{C}$ lying in the plane H_{z_i}.

For any value $a \in (z_i, z_{i+1})$, let $\mathcal{U}_a := \mathcal{U} \cap H_a$. Then we have

$$\mathcal{U}_a = \mathcal{U}\big(\{H_a \cap C \mid C \in \mathcal{C}_i\}\big) = \mathcal{U}(\{S_i \times \{a\} \mid S_i \in \mathcal{S}_i\}) = \mathcal{U}(\mathcal{S}_i) \times \{a\}$$

so the combinatorial structure of $\mathcal{U}$, and $\mathcal{U}(\mathcal{S}_i) = \mathcal{U}_a^{\downarrow}$, does not change in $[z_i, z_{i+1})$. See Figure 1. Furthermore, each vertex (x_0, y_0, a) of $\mathcal{U}_a$ is the intersection point of a z-edge e of $\mathcal{U}$ with H_a and $(x_0, y_0) \in V_i$. Let p^- (resp. p^+) be the lower (resp. upper) endpoint of e. Let $z_i := z(p^-)$ and $z_j := z(p^+)$, where $i < j$. Then $(p^-)^{\downarrow} \in V_i \setminus V_{i-1}$, and $(p^+)^{\downarrow} \in V_{j-1} \setminus V_j$. Conversely, for any i with $1 \le i \le 2n$, every vertex of $\Delta V_i := V_i \oplus V_{i-1}$ is the projection of an endpoint of a z-edge of $\mathcal{U}$ and thus a vertex of $\mathcal{U}$, where $\oplus$ is symmetric difference. (So that ΔV_1 is well-defined, set $\Delta V_0 := \varnothing$.) Thus, to compute the vertices of $\mathcal{U}$, we report ΔV_i when the plane H stops at z_i, for all i, $1 \le i \le 2n$.

We report the vertices of ΔV_i using a data structure $\mathbb{T}$ that stores $\mathcal{S}_i$ and implicitly maintains $\mathcal{U}(\mathcal{S}_i)$ as we sweep H. A square S_j is inserted into $\mathcal{S}_i$ when H reaches the bottom face of C_j, and it is deleted from $\mathcal{S}_i$ when H reaches the top face of C_j. Let $|A|$ denote the length of the longest side length of A, where A is a box or rectangle. Since C_j's are cubes, the sequence of updates satisfies the following property:

(P1): Let $C_u, C_v \in \mathcal{C}$ be two cubes such that $|C_u| < |C_v|$ and S_u is inserted before S_v. Then S_u is also deleted before S_v.

When the sweep stops at z_i, the insertion or deletion of a square to $\mathcal{S}_{i-1}$ (to obtain $\mathcal{S}_i$) into $\mathbb{T}$ is performed in $O(\log^3 n)$ amortized time, and ΔV_i is reported in $O(\log^3 n + |\Delta V_i|)$ amortized time. Thus, the sweep takes $O(\sum_{i=1}^{2n} \log^3 n + \sum \Delta V_i) = O(n \log^3 n + \kappa)$ time. With $O(n \log n)$ additional preprocessing to initialize $\mathbb{T}$ before the sweep, and $O(n \log n + \kappa)$ postprocessing to compute the edges and faces of $\mathcal{U}$ using the vertices reported during the sweep, the algorithm takes $O(n \log^3 n + \kappa)$ overall time, proving Theorem 1 for the bounded-spread case.

2.2 Reporting ΔV_i

Next, we describe our dynamic data structure $\mathbb{T}$ that stores a set $\mathcal{S}$ of squares in $\mathbb{R}^2$ and implicitly maintains $\mathcal{U}(\mathcal{S})$. After each update – an insertion or deletion of a square – it reports $\Delta V := V(\mathcal{S}^{\text{new}}) \oplus V(\mathcal{S}^{\text{old}})$, where $\mathcal{S}^{\text{old}}$ (resp. $\mathcal{S}^{\text{new}}$) is $\mathcal{S}$ before (resp. after) the update operation. Let $\mathsf{X} \subset \mathbb{R}^2$ be the set of points corresponding to the xy-projections of vertices of cubes in $\mathcal{C}$. Recall that the spread of X is polynomially bounded. Let $\square_0$ be a square

containing X. $\mathbb{T}$ is a quadtree built on X with $\square_0$ being the square associated with the root node of $\mathbb{T}$. Without loss of generality, we assume that no point of X lies on the boundary of a square $\square_u$ for any node $u \in \mathbb{T}$.

Each node $u \in \mathbb{T}$ is associated with a square $\square_u$. For the root node r, $\square_r = \square_0 \supset \mathsf{X}$. If $|\mathsf{X} \cap \square_u| \leq 1$, u is a leaf, otherwise $\square_u$ is partitioned into four congruent squares, each associated with a child of u. Set $\mathsf{X}_u \coloneqq \mathsf{X} \cap \square_u$. The height of $\mathbb{T}$ is $O(\log n)$. A square S that intersects the region $\square_u$, for a node $u \in \mathbb{T}$, is *long* at u if no vertex of S lies in $\square_u$, and S is *short* at u otherwise. A short square S at node u is called a *floater square* if at least two vertices of S lie in $\operatorname{int}(\square_u)$, and a *corner square* otherwise. A corner square S contains exactly one vertex of $\square_u$. See Figure 1 (right). For each node $u \in \mathbb{T}$, let $\mathcal{L}_u \subseteq \mathcal{S}$ (resp. $\mathcal{S}_u \subseteq \mathcal{S}$) be the set of long (resp. short) squares at u, and let $\mathcal{L}^*_u \coloneqq \mathcal{L}_u \setminus \mathcal{L}_{p(u)} = \mathcal{L}_u \cap \mathcal{S}_{p(u)}$, where $p(u)$ is the parent node of u in $\mathbb{T}$; $\mathcal{L}_{\text{root}} \coloneqq \varnothing$. Let $\mathcal{F}_u \subseteq \mathcal{S}_u$ (resp. $\mathcal{R}_u \subseteq \mathcal{S}_u$) denote the set of floater (resp. corner) squares at u. Note that any square $S \in \mathcal{S}$ is short at no more than four nodes in any level of $\mathbb{T}$. Hence S is short at $O(\log n)$ nodes of $\mathbb{T}$.

At each node $u \in \mathbb{T}$, we maintain $\mathcal{L}^*_u$ and $\mathcal{S}_u$. A long square $S \in \mathcal{L}^*_u$ contains at least one edge of $\square_u$; it may contain all four edges of $\square_u$ if $\square_u \subseteq S$. We partition $\mathcal{L}^*_u$ into four sets: for each edge $e \in \square_u$, let $\mathcal{L}^*_{u,e} \subseteq \mathcal{L}^*_u$ be the set of squares that contain e. If $\square_u \subseteq S$, then S is assigned only to the set associated with the top edge of $\square_u$. Suppose e is the top edge of $\square_u$. Then the bottom edges of squares in $\mathcal{L}^*_{u,e}$ intersect $\square_u$ or lie below $\square_u$. We store $\mathcal{L}^*_{u,e}$ in a red-black tree, sorted in increasing order of the distances of their bottom edges from e (i.e., in decreasing order of their y-coordinates). We similarly store $\mathcal{L}^*_{u,e}$ for the other three edges e of $\square_u$. Let $\mathcal{E}^x_u$ (resp. $\mathcal{E}^y_u$) be the sequences of x-edges (resp. y-edges) of $\mathcal{F}_u$ in increasing order of their y-coordinates (resp. x-coordinates). We maintain $\mathcal{E}^x_u, \mathcal{E}^y_u$ using red-black trees.

For each edge e of $\square_u$, we store a value ℓ_e that maintains the position of a sweep-line L_e associated with the edge e. The roles of ℓ_e and L_e will become clear in the insertion and deletion procedures. The value of ℓ_e (and the sweep-line) changes dynamically. Initially, for each x-edge (resp. y-edge) e of $\square_u$, ℓ_e is the y-coordinate (resp. x-coordinate) of e, and it always lies in the y-span (resp. x-span) of $\square_u$. If e is an x-edge (resp. y-edge), then let $\mathcal{I}_e$ be the set of x-projections (resp. y-projections) of the floater squares in $\mathcal{F}_u$ whose y-spans (resp. x-spans) contain ℓ_e. $\mathcal{I}_e$ changes dynamically with ℓ_e.

We also maintain two secondary structures for each edge e of $\square_u$:

- *Wall data structure* $\mathbb{W}_e(\mathcal{I}_e)$: It supports the following two operations:
  - Insert/Delete(I): Insert or delete an interval I to $\mathcal{I}_e$.
  - Report-Holes(ζ): For a query interval ζ, return the endpoints of $\zeta \setminus \operatorname{int}(\mathcal{U}(\mathcal{I}_e))$. If an endpoint x of ζ does not lie in $\operatorname{int}(\mathcal{U}(\mathcal{I}_e))$, x is also returned.

  The Insert, Delete, and Membership operations take $O(\log n)$ time and Report-Holes takes $O(\log n + \kappa)$ time, where κ is the number of points reported.

- *Corner data structure* $\mathbb{C}_e(\mathcal{R}_u, \mathcal{I}_e)$: It supports the following four operations:
  - Insert/Delete(R): Insert or delete a corner square R to $\mathcal{R}_u$.
  - Insert/Delete(I): Insert or delete an interval I to $\mathcal{I}_e$.
  - Report-Hole(σ): Given a query axis-aligned segment $\sigma \subset \square_u$, return the (at most one) interval of $\sigma \setminus \operatorname{int}(\mathcal{U}(\mathcal{R}_u))$.
  - Report-Vertices(ρ): Given a query rectangle $\rho \subseteq \square_u$, return $V(\mathcal{R}_u \cup \mathcal{W}_e) \cap \rho$, where $\mathcal{W}_e \coloneqq \{I \times \mathbb{R} \mid I \in \mathcal{I}_e\}$ if e is an x-edge and $\mathcal{W}_e \coloneqq \{\mathbb{R} \times I \mid I \in \mathcal{I}_e\}$ if e is a y-edge. Namely, the vertices of $\mathcal{U}(\mathcal{R}_u)$ that do not lie in $\mathcal{U}(\mathcal{W}_e)$ or the intersection points of the edges of $\mathcal{U}(\mathcal{R}_u)$ and $\mathcal{U}(\mathcal{W}_e)$ that lie in ρ are reported.

  An Insert/Delete takes $O(\log^2 n)$ time, Report-Hole takes $O(\log n)$ time, and Report-Vertices takes $O(\log n + \kappa)$ time, where κ is the number of vertices reported.

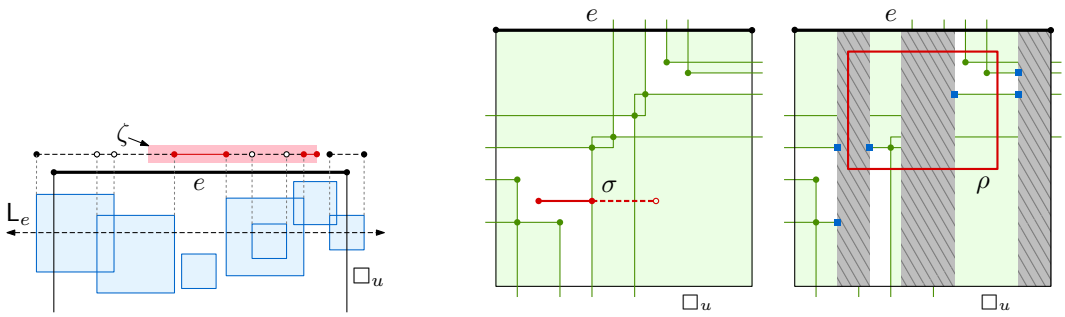


Figure 2 (left) An illustration of call Report-Hole(ζ) to $\mathbb{W}_e$. The x-projections of floater squares intersecting L_e that compose $\mathcal{I}_e$. The intervals of $\mathcal{U}(\mathcal{I}_e)$ are dashed. The solid red intervals in ζ are the answer to the query. (right) Illustrations of calls Report-Hole(σ) and Report-Vertices(ρ) to $\mathbb{C}_e$. The vertices of $\mathcal{U}(\mathcal{R}_u)$ are shown as green circles, and the vertices of $\mathcal{U}(\mathcal{R}_u \cup \mathcal{W}_e)$ that are the intersection points of a "strip" in $\mathcal{W}_e$ (gray hatched) and an edge of $\mathcal{U}(\mathcal{R}_u)$ are shown as blue squares. The solid portion of σ is the answer to Report-Hole(σ), and the two blue vertices and three green vertices (inside ρ) are the answer to Report-Vertices(ρ).

See Figure 2. We describe these data structures in Section 2.4. For now, we assume that, for all four edges of $\square_u$ and for all nodes $u \in \mathbb{T}$, $\mathbb{W}_e$, and $\mathbb{C}_e$ are at our disposal.

For each square $S \in \mathcal{S}$, let $\Lambda(S) := \{u \in \mathbb{T} \mid S \in \mathcal{L}^*_u\}$ and $\Sigma(S) := \{u \in \mathbb{T} \mid S \in \mathcal{S}_u\}$. Any square $S \in \mathcal{S}$ is short at no more than four nodes of any fixed level of $\mathbb{T}$. The nodes in $\Sigma(S)$ lie along at most four root-to-leaf paths to the leaves whose squares contain the vertices of S, and the nodes in $\Lambda(S)$ are those children u of nodes in $\Sigma(S)$ for which $S \cap \square_u \neq \varnothing$ and S is not short at u. Since the height of $\mathbb{T}$ is $O(\log n)$, $|\Lambda(S)|, |\Sigma(S)| = O(\log n)$. Finally, let $\Xi(S)$ be $\Lambda(S)$ and the four leaves of $\Sigma(S)$, and set $\Pi(S) := \{S \cap \square_u \mid \square_u \in \Xi(S)\}$. See Figure 3 (left). The following lemma is straightforward.

▶ **Lemma 5.** *For any square S, $\Pi(S)$ is a partition of S into $O(\log n)$ rectangles.*

When we insert or delete a square S, every vertex of ΔV lies in S. Thus, to report ΔV, we report $\Delta V_u := \Delta V \cap \square_u$ for each node $u \in \Xi(S)$. We now describe how to update $\mathbb{T}$ and compute ΔV_u at each node $u \in \Xi(S)$.

Insertion of S. There are four main steps:

(1) At each node $u \in \Lambda(S) \cup \Sigma(S)$, we compute $\rho_u := \mathrm{cl}(\square_u \setminus \mathcal{U}(\mathcal{L}_u))$, as described below.

(2) At each node $u \in \Xi(S)$, we report ΔV_u, which is also described below.

(3) For each node $u \in \Lambda(S)$, we insert S to $\mathcal{L}^*_u$. In particular, if S contains the edge e of $\square_u$, S is inserted into $\mathcal{L}^*_{u,e}$; recall that if $\square_u \subseteq S$, then S is associated with the top edge of $\square_u$.

(4) For each $u \in \Sigma(S)$, we update $\mathcal{S}_u$ and its secondary structures, as follows. If S is a corner square at u, we insert S into all four corner data structures $\mathbb{C}_e$ stored at u for each edge of $\square_u$. If S is a floater, we first insert the x-edges (resp. y-edges) of S which intersect $\square_u$ into $\mathcal{E}^*_u$ (resp. $\mathcal{E}^y_u$). Then, for each x-edge (resp. y-edge) e of $\square_u$ such that ℓ_e lies in the y-span (resp. x-span) of S, we insert the x-projection (resp. y-projection) of S to $\mathbb{W}_e$ and $\mathbb{C}_e$.

We now describe the first two steps in more detail. We compute ρ_u at each node $u \in \Lambda(S) \cup \Sigma(S)$, as follows. Observe that $\rho_u \subseteq \square_u$ is a (possibly empty) rectangle. Similarly, $\mathrm{cl}(\square_u \setminus \mathcal{U}(\mathcal{L}^*_u))$ is a rectangle $\rho^*_u \subseteq \square_u$. By definition, $\rho_u = \mathrm{cl}(\square_u \setminus (\mathcal{U}(\mathcal{L}_{p(u)}) \cup \mathcal{U}(\mathcal{L}^*_u))) =$

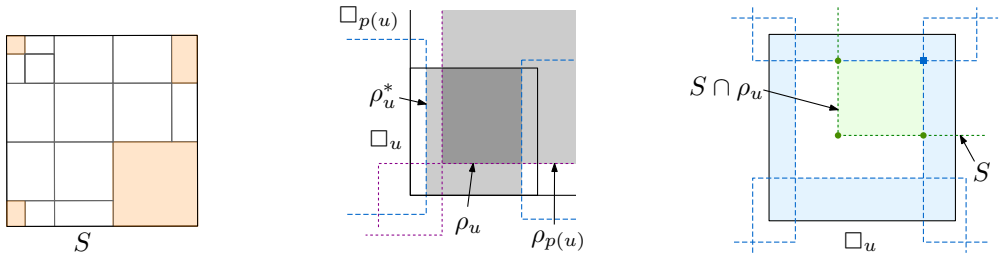


Figure 3 (left) The rectangles of $\Pi(S)$. The shaded (resp. empty) rectangles lie in squares $\square_u$ of nodes u in $\Sigma(S)$ (resp. $\Lambda(S)$). (middle) Computing ρ_u using ρ_u^* and $\rho_{p(u)}$. The squares at the ends of the sequences $\mathcal{L}^*_{u,e}$ for each edge e of $\square_u$ are dashed and the squares of $\mathcal{L}_{p(u)}$ that contribute to $\mathcal{U}(\mathcal{L}_{p(u)})$ are dotted. (right) An example of a square S being inserted at $u \in \Sigma(S)$. The four vertices of ΔV_u are shown, where the blue vertex is old and the rest are new.

$\rho_{p(u)} \cap \rho_u^*$. See Figure 3 (middle). Furthermore, for each edge e_i, $1 \le i \le 4$, let S_i be the last square in the sequence $\mathcal{L}^*_{u,e_i}$, and let h_i be the halfplane bounded by the line supporting the edge of S_i that intersects $\square_u$ and does not contain S_i (if $S_i \supseteq \square_u$, then choose the halfplane bounded by the edge of $\square_u$ opposite e_i and not containing $\square_u$). Then $\rho_u^* = \bigcap_{i=1}^4 h_i \cap \square_u$. Hence, ρ_u^* can be computed in $O(1)$ time. Also, with $\rho_{p(u)}$ at our disposal $\rho_u = \rho_{p(u)} \cap \rho_u^*$ can be computed in $O(1)$ time. Thus, using a top-down traversal of $\mathbb{T}$, starting at the root node r (with $\mathcal{L}_r = \varnothing$) and ending at the nodes of $\Xi(S)$, we compute ρ_u at each visited node; $O(\log n)$ nodes are visited in this process, so the step takes $O(\log n)$ time overall. This completes step (1).

Next, let u be a node of $\Xi(S)$. We report ΔV_u as follows. If $u \in \Sigma(S)$, then $\mathcal{S}_u = \varnothing$ (before the insertion of S) and hence ΔV_u can be reported in $O(1)$ time; see Figure 3 (right). We now focus on reporting ΔV_u when $u \in \Lambda(S)$. For sake of concreteness, suppose S contains the top edge of $\square_u$. Let S_u^{exp} be the rectangle $S \cap \rho_u = \mathrm{cl}((S \cap \square_u) \setminus \mathcal{U}(\mathcal{L}_u)))$, i.e., the portion of $S \cap \square_u$ that is left "exposed" by the squares of $\mathcal{L}_u$. All vertices of ΔV_u lie in S_u^{exp}. Note that S_u^{exp} may be empty, e.g., if $\square_u \subseteq \mathcal{U}(\mathcal{L}_u)$ or $S \cap \rho_u = \varnothing$, in which case $\Delta V_u = \varnothing$ and there is nothing to do; see Figure 4 (middle). If $S_u^{\mathsf{exp}} \neq \varnothing$, we sweep a line L_e in the $(-y)$-direction from the top edge of S_u^{exp} (defined by $\mathcal{U}(\mathcal{L}_u)$ or $\square_u$) to its bottom edge (defined by S, $\square_u$, or $\mathcal{U}(\mathcal{L}_u)$), and report all vertices of ΔV_u in the process by using the secondary data structures as described in Section 2.3. Recall that ℓ_e keeps track of the position of L_e.

Deletion of S. Intuitively, the deletion is "undoing" the insertion of S.

(1) At each node $u \in \Lambda(S) \cup \Sigma(S)$, we compute $\rho_u := \mathrm{cl}(\square_u \setminus \mathcal{U}(\mathcal{L}_u \setminus \{S\}))$ as in the insertion case.

(2) At each node $u \in \Xi(S)$, we report ΔV_u, as described below.

(3) For each node $u \in \Lambda(S)$, we delete S from $\mathcal{L}_u^*$.

(4) For each node $u \in \Sigma(S)$, we update $\mathcal{S}_u$ and its secondary structures, as follows. If S is a corner square at u, we delete S from all four corner data structures $\mathbb{C}_e$ stored at u for each edge e of $\square_u$. If S is a floater, we first delete the x-edges (resp. y-edges) of S which intersect $\square_u$ from $\mathcal{E}_u^*$ (resp. $\mathcal{E}_u^y$). Then, for each x-edge (resp. y-edge) e of $\square_u$, we delete the x-projection (resp. y-projection) of S from $\mathbb{W}_e$ and $\mathbb{C}_e$ if ℓ_e lies in the y-span (resp. x-span) of S.

Let S_u^{exp} be the rectangle $S \cap \rho_u = \mathrm{cl}(S \cap \square_u) \setminus \mathcal{U}(\mathcal{L}_u \setminus \{S\}))$ i.e., the portion of $S \cap \square_u$ that is left "exposed" by the other squares of $\mathcal{L}_u$. Again, all vertices of ΔV_u lie in S_u^{exp}. We note that S_u^{exp} may be empty because ρ_u is empty or the bottom edge of S lies above the

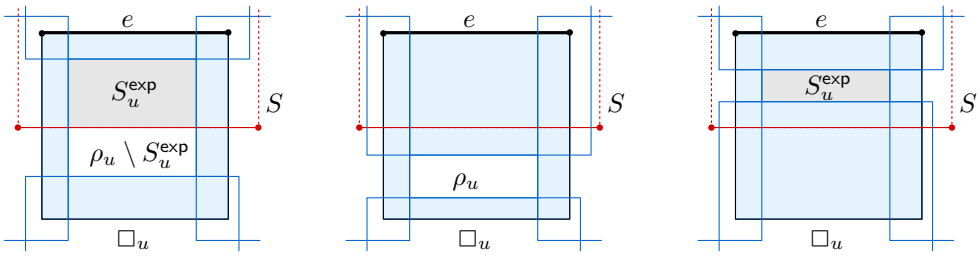


Figure 4 Various cases for S_u^{exp} when $\rho_u \neq \varnothing$: (left) $S_u^{\mathsf{exp}} \subset \rho_u$. (middle) $S_u^{\mathsf{exp}} = \varnothing$. (right) $S_u^{\mathsf{exp}} = \rho_u$.

top edge of S_u^{exp}. If $S_u^{\mathsf{exp}} = \varnothing$, $\Delta V_u = \varnothing$ and there is nothing to do. So assume $S_u^{\mathsf{exp}} \neq \varnothing$. We report ΔV_u by sweeping a line L_e in the $(+y)$-direction from the bottom edge of S_u^{exp} (defined by S, $\square_u$, or $\mathcal{U}(\mathcal{L}_u)$), to its top edge (which is the top edge of ρ_u).

Runtime analysis. We now analyze the amortized running time of inserting or deleting a square. Let c be a sufficiently large constant. For each node $u \in \mathbb{T}$, we assign $4c \log^2 n$ credits to each of the four edges of every floater in $\mathcal{F}_u$. These credits will be used to pay part of the cost of reporting ΔV_u (see Lemma 8). Note that S is a floater at $O(\log n)$ nodes. Therefore $O(\log^3 n)$ credits are assigned to a square S. This cost is charged to the insertion of S.

Suppose a square S is being inserted. The total time spent in computing ρ_u at all nodes $u \in \Lambda(S) \cup \Sigma(S)$ is $O(\log n)$. By Lemma 8, which is given in Section 2.3, the amortized cost of reporting ΔV_u and updating the secondary structures at all nodes in $\Xi(S)$ is $O(\log^2 n + |\Delta V_u|)$. Summing at all nodes of $\Xi(S)$, the total amortized time spent in reporting ΔV is $O(\log^3 n + |\Delta V|)$. Finally, we spend $O(\log n)$ time at each node $u \in \Lambda(S)$ to insert S into $\mathcal{L}_u^*$ and $O(\log^2 n)$ time to insert S into the secondary structures at each node in $\Sigma(S)$. Summing this cost over all nodes in $\Lambda(S) \cup \Sigma(S)$ and adding the cost of credits assigned to S, the total amortized time spent in inserting S is $O(\log^3 n + |\Delta V|)$ time. A similar analysis shows that the amortized time spent in deleting a square is $O(\log^3 n + |\Delta V|)$. Hence, we obtain the following:

▶ **Lemma 6.** *The amortized cost of inserting or deleting a square in $\mathbb{T}$ and reporting ΔV is $O(\log^3 n + |\Delta V|)$.*

2.3 Reporting ΔV_u via Sweep-Line

We describe the sweep-line procedure for the insertion of a square S; the deletion case is symmetric. Without loss of generality, assume that S contains the top edge of $\square_u$; the procedures for the other cases are similar. Let $\rho_u := \mathrm{cl}(\square_u \setminus \mathcal{U}(\mathcal{L}_u))$ and $S_u^{\mathsf{exp}} := S \cap \rho_u$ as defined above. If $S_u^{\mathsf{exp}} = \varnothing$, there is nothing to do. So assume $S_u^{\mathsf{exp}} \neq \varnothing$. Suppose S_u^{exp} is of the form $\gamma \times [a^-, a^+]$, where γ is the x-span of S_u^{exp} and $a^- < a^+$ are the y-coordinates of the bottom and top edges of S_u^{exp}; the bottom edge of S_u^{exp} is the bottom edge of S, ρ_u, or $\square_u$.

Let $V_u^{\mathrm{new}} \subseteq \Delta V_u$ be the set of vertices that are created by the insertion of S (which appear on ∂S) and let $V_u^{\mathrm{old}} \subseteq \Delta V_u$ be the set of vertices that no longer appear on $\mathcal{U}$ after the insertion of S (which lie in $\mathrm{int}(S)$). The vertices of V_u^{new} lie on $S_u^{\mathsf{exp}} \cap \partial S$, i.e., on the bottom edge of S_u^{exp} if it is contained in the bottom edge of S, and $V_u^{\mathrm{new}} = \varnothing$ if the bottom edge of $\square_u$ is not contained in that of S. Next, a vertex v of V_u^{old} can be classified into the following categories depending on the types of the two (not necessarily distinct) squares S_1, S_2 whose edges contain v: v is a *long-long* (LL) vertex if both S_1 and S_2 are long, in which case v

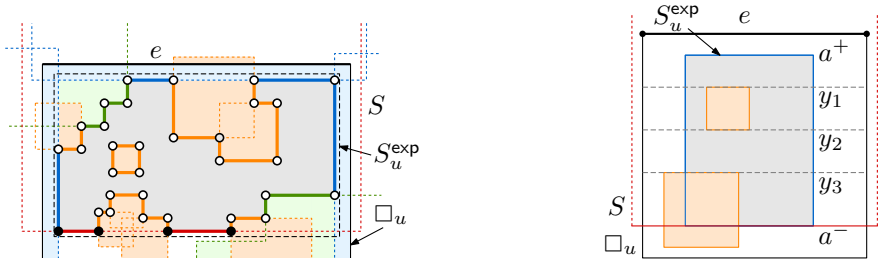


Figure 5 (left) A zoomed-in example of $S_u^{\exp}$ (slightly enlarged for visibility) and the vertices of ΔV where the bottom edge of $S_u^{\exp}$ lies on the bottom edge of the inserted square S. $\mathcal{U}(\mathcal{L}_u)$ is blue, $\mathcal{U}(\mathcal{F}_u) \setminus \mathcal{U}(\mathcal{L}_u)$ is orange, $\mathcal{U}(\mathcal{R}_u) \setminus \mathcal{U}(\mathcal{L}_u \cup \mathcal{F}_u)$ is green, and the edges of S are red. The edges of $\mathcal{U} \cap S_u^{\exp}$ are thick, the old vertices covered by S are hollow, and the new vertices on ∂S are solid. (right) Dashed lines supporting floater edges (orange) with y-coordinates in (a^-, a^+) partition $S_u^{\exp}$ (blue) into rectangles.

is a vertex of $S_u^{\exp}$ not contained in $\mathcal{U}(\mathcal{S}_u)$, a *long-short* (LS) vertex if S_1 is long and S_2 is short, in which case v is an intersection point of an edge of $\mathcal{U}(\mathcal{S}_u)$ with an edge of $S_u^{\exp}$, and a *short-short* (SS) vertex if both S_1 and S_2 are short, in which case v is a vertex of $\mathcal{U}(\mathcal{S}_u)$ that lies in $\operatorname{int}(S_u^{\exp})$. SS vertices are further classified into three categories, CC, CF, or FF, depending on whether S_1 and S_2 are corners (C) or floaters (F). See Figure 5 (left).

With this characterization of ΔV_u at hand, we report ΔV_u by sweeping downward ($(-y)$-direction) with a horizontal line L_e from $y = a^+$ to $y = a^-$. Let $\mathsf{Y} = \langle y_0 = a^+, y_1, y_2, \ldots, y_t = a^- \rangle$ where $y_1, \ldots, y_{t-1}$ are the y-coordinates of edges of floaters $F \in \mathcal{F}_u$ in the interval (a^-, a^+), sorted in decreasing order. Y can be constructed from $\mathcal{E}_u^y$. The lines $y = y_i$, $0 < i < t$, partition $S_u^{\exp}$ into rectangles. See Figure 5 (right).

The sweep line starts at $y = y_0$ and stops at every y_i, for $0 \le i \le t$. At each y_i, we perform two steps:

(i) Report all vertices of ΔV_u that lie on the line $\mathsf{L}_i : y = y_i$.

(ii) For $i \ge 1$, we report the vertices of ΔV_u that lie in the semi-open rectangle $\sigma_i := \gamma \times (y_i, y_{i-1})$, i.e., all vertices that lie in σ_i but not on its x-edges, using the secondary structures.

We now describe the details of the sweep-line algorithm. As we sweep L_e from a^+ to a^-, we vary ℓ_e, the value associated with the top edge e, to the current position of the sweep-line, so we update the set $\mathcal{I}_e$ (and the secondary structures $\mathbb{W}_e$ and $\mathbb{C}_e$ that store it) as ℓ_e changes. We note that $\mathcal{I}_e$ does not change in the interval (y_i, y_{i-1}). However, when we initialize L_e to y_0, we need to reset ℓ_e to y_0 and update $\mathcal{I}_e$, $\mathbb{W}_e$, and $\mathbb{C}_e$. We will describe this initialization step later and for now assume that $\mathcal{I}_e, \mathbb{W}_e$, and $\mathbb{C}_e$ are consistent with $\ell_e = y_0$. At each y_i, we perform steps (i) and (ii) follows.

Performing step (i). For $i = 0, t$, we set $\delta := \gamma \times \{y_i\}$ to be the top (or bottom) edge of $S_u^{\exp}$. By definition $\delta \cap \operatorname{int}(\mathcal{U}(\mathcal{L}_u)) = \varnothing$. Next, by calling Report-Hole(δ), we compute $\delta_0 := \delta \setminus \operatorname{int}(\mathcal{U}(\mathcal{R}_u)) = \delta \setminus \operatorname{int}(\mathcal{U}(\mathcal{L}_u \cup \mathcal{R}_u))$. Finally, set $\delta_0^{\downarrow}$ to be the x-projection of δ_0. By querying the wall data structure $\mathbb{W}_e$ for edge e with Report-Holes($\delta_0^{\downarrow}$), we compute the intervals of $\delta_0^{\downarrow} \setminus \mathcal{U}(\mathcal{I}_e)$. Let $x_0, \ldots, x_s$ be the endpoints of these intervals. Then, for $0 < j < s$, (x_j, y_i) is a LS vertex. If (x_0, y_i),(x_s, y_i) are endpoints of δ then they are LL vertices, otherwise they are LS vertices.

For $0 < i < t$, L_i contains an x-edge η of a floater $F \in \mathcal{F}_u$. Let $\delta := \eta \cap S_u^{\exp}$. As above, by calling Report-Hole(δ) on the corner data structure $\mathbb{C}_e$ for edge e, we compute $\delta_0 := \delta \setminus \operatorname{int}(\mathcal{U}(\mathcal{L}_u \cup \mathcal{R}_u))$. If η is the bottom edge of F then we first delete the interval $\eta^{\downarrow}$,

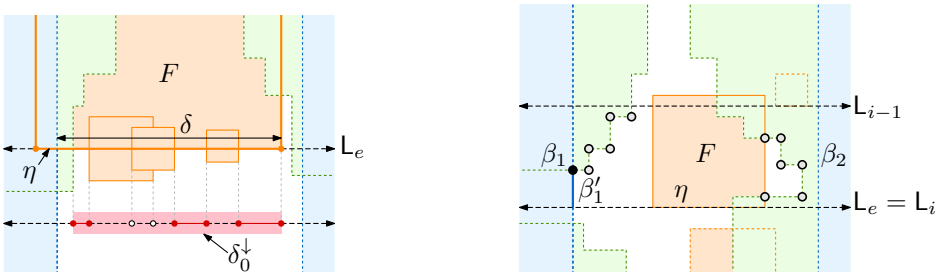


Figure 6 Zoomed-in illustrations of steps (i) and (ii) where the sweep-line L_e reaches the bottom edge η of floater F. Floaters are shown in orange, $\mathcal{U}(\mathcal{L}_u)$ is shown in blue, and $\mathcal{U}(\mathcal{R}_u) \setminus \mathcal{U}(\mathcal{L}_u)$ is shown in green. (left) Step i: The endpoints of the solid intervals in $\delta_0^{\downarrow}$ are the x-projections of the vertices of $\Delta V_u \cap \mathsf{L}_e$. (right) Step ii: The SS vertices in σ_i are hollow, and the lone LS vertex is solid. β_2, the right edge of σ_i, is contained in $\mathcal{U}(\mathcal{R}_u)$ so $\beta_2' = \varnothing$ and it contains no LS vertices.

the x-projection of η, from $\mathbb{W}_e$. Next, we query $\mathbb{W}_e$ with REPORT-HOLES($\delta_0^{\downarrow}$) to report the endpoints of the intervals of $\delta_0^{\downarrow} \setminus \text{int}(\mathcal{U}(\mathcal{I}_e))$. If $x_0, \ldots, x_s$ are these endpoints then (x_j, y_i), for $0 \le j \le s$, are vertices of ΔV_u lying on L_i: If x_0 or x_s lies on $\partial S_u^{\mathsf{exp}}$ then it is an LS vertex, and if it lies on an edge of a square in $\mathcal{R}_u$, (i.e., an endpoint of δ_0 lying inside S_u^{exp}), it is a CF vertex. All other vertices are FF vertices. See Figure 6 (left).

Performing step (ii). Our goal is to report all vertices of $V_u \cap \int(\sigma_i)$. A vertex of ΔV_u not lying on L_{i-1} or L_i lies on an x-edge of a corner square. Since no x-edge of any floater square lies in the interval (y_i, y_{i-1}), the set $\mathcal{I}_e$, and thus $\mathcal{W}_e$, remains the same for all y-values in this interval. Furthermore, $V(\mathcal{R}_u \cup \mathcal{F}_u) \cap \sigma_i = V(\mathcal{R}_u \cap \mathcal{W}_e) \cap \sigma_i$. We can thus report all CF and CC vertices lying in σ_i by querying $\mathcal{R}_e$ with REPORT-VERTICES(σ_i). The $O(1)$ LS vertices in σ_i are defined by a long square, namely a long square that defines a y-edge of σ_i, and a corner square. For each y-edge β_j of σ_i, we call $\mathbb{W}_e$ with REPORT-HOLE($\beta_j^{\downarrow}$). If $\beta_j^{\downarrow}$ is returned, then β_j is not contained in $\mathcal{U}(\mathcal{W}_e) \cap \sigma_i = \mathcal{U}(\mathcal{F}_u) \cap \sigma_i$ and we call $\mathbb{C}_e$ with REPORT-HOLE(β_j); let $\beta_j' \subseteq \beta_j$ be the returned (possibly empty) interval. The endpoints of β_j^* that lie in int(β_j), if any, are the LS vertices on β_j. See Figure 6 (right). Finally, for $i < t$, if η is the top edge of F, we insert $\eta^{\downarrow}$ into $\mathbb{W}_e$ and $\mathbb{C}_e$, otherwise η is the bottom edge of F and we delete $\eta^{\downarrow}$ from $\mathbb{C}_e$. (In the latter case, $\eta^{\downarrow}$ was already deleted from $\mathbb{W}_e$ in the previous step.) Note that if η is an edge of a floater F, then $\eta^{\downarrow}$ is accordingly inserted or deleted to $\mathbb{W}_e$ and $\mathbb{C}_e$ by the end of these two steps, and hence they are made consistent with ℓ_e.

When the sweep procedure ends, we set $\ell_e := a^-$. Note that now $\mathcal{I}_e$ consists of the x-projections of floaters that intersect the line $y = a^-$, as desired. Therefore the secondary structures $\mathbb{W}_e$ and $\mathbb{C}_e$ are consistent with the new value of ℓ_e.

Initializing the sweep line. At the beginning of the procedure, the value of ℓ_e is the value at which the sweep procedure stopped after inserting or deleting a long square at u that contained the top edge of $\square_u$. To initialize the sweep line at $y = a^+$, the top edge of S_u^{exp}, and to initialize $\mathbb{W}_e$ and $\mathbb{C}_e$ correctly for $y = a^+$, we again perform a line sweep from the current value of ℓ_e to a^+ by a horizontal line L_e as above, except that no reporting of vertices occurs. That is, as we sweep, we stop at each encountered x-edge η of a floater square, and insert or delete the interval $\eta^{\downarrow}$ in $\mathbb{W}_e$ and $\mathbb{C}_e$ without (querying for and) reporting any vertices.

Runtime analysis. Let κ_u be the number of vertices in ΔV_u, and let a_{pr} be the value of ℓ_e before the sweep line is initialized. For the insertion of a square S, let φ_u be the number of y-coordinates of floater edges in the intervals $[a^-, a^+]$ and $[\ell_e, a^+]$ (resp. $[a^+, a_{pr}]$) if $a_{pr} \leq a^+$ (resp. $a_{pr} > a^+$). For the deletion of a square S, let φ_u be the number of y-coordinates of floater edges in the intervals $[a^-, a^+]$ and $[\ell_e, a^-]$ (resp. $[a^-, a_{pr}]$) if $a_{pr} \leq a^-$ (resp. $a_{pr} > a^-$). (Some y-coordinates may be counted twice by φ_u.) Thus, the total time spent for the insertion or deletion of S at each node $u \in \Xi(S)$ is $(1 + \varphi_u) \cdot O(\log^2 n) + O(\kappa_u)$.

We charge $O(\log^2 n)$ units of time to each of the φ_u floater edges that are parallel to the sweep line and that were crossed by the two sweep-line procedures – one in the initialization step and one for reporting the vertices. This charging pays for the $\varphi_u \log^2 n$ term in the running time. The amortized cost of reporting ΔV_u at u is $O(\log^2 n + \kappa_u)$, provided that each floater at u had enough credits to pay for the costs charged to it. The following lemma proves that the floater is not charged too many times.

▶ **Lemma 7.** *Let f be an edge of a floater F at a node $u \in \mathbb{T}$. Then f is charged by the sweeps at $\square_u$ at most six times during the entire algorithm.*

Proof. Without loss of generality assume that f is an x-edge. Then f is charged by the *top* and *bottom* sweeps, i.e., sweeps performed when a square $S \in \mathcal{L}^*_u$ containing the top or the bottom edge of $\square_u$ is inserted or deleted. We claim that f is charged at most three times by the top sweep – a similar argument holds for the bottom sweep. Let e denote the top edge of $\square_u$. Recall that f is charged whenever the horizontal sweep-line L_e crosses f – either in the initialization step or in the reporting step. Furthermore L_e will cross f from opposite directions, i.e., when L_e is sweeping *downward* ($(-y)$-direction) or *upward* ($(+y)$-direction), in any two consecutive crossings. We claim that L_e crosses f in at most three top sweeps.

Recall that during the insertion or deletion of a square S, either $S_u^{\mathsf{exp}} = \varnothing$ and no sweeps occur, otherwise $S_u^{\mathsf{exp}} \neq \varnothing$ and two sweeps occur (one in the initialization step and one in the reporting step). We classify downward (resp. upward) sweeps that occur during an insertion of a square as DI (resp. UI) sweeps, and classify downward (resp. upward) sweeps that occur during a deletion of a square as DD (resp. UD) sweeps. Consider the sequence of sweeps that cross f while $F \in \mathcal{F}_u$, i.e., after the insertion of F and before the deletion of F at node u. We claim the sequence satisfies the following two constraints:

(i) No sweeps can occur after a DI sweep.

(ii) No sweeps can occur before a DD sweep.

Clearly the longest valid sequences of sweeps are of the form "DD, UD, DI" or "DD, UI, DI," and hence f is crossed by L_e in at most three top sweeps. (It can be shown that the latter sequence is not possible, but this fact is not needed in order to prove the lemma.) It remains to prove the constraints above.

Proof of claim (i). Suppose L_e crosses f in a DI sweep, and let $S \in \mathcal{L}^*_{u,e}$ be the square being inserted. We argue that if the bottom edge of g of S lies above f then f could not have been crossed in this sweep, either during the initialization step or during the reporting step, as follows. $S_u^{\mathsf{exp}} \neq \varnothing$, so the bottom edge of $S_u^{\mathsf{exp}} = S \cap \rho_u$ is either contained in g or lies above it. A downward sweep in the initialization step sweeps to the top edge of S_u^{exp}, and a downward sweep in the reporting step sweeps from the top edge of S_u^{exp} to the bottom edge of S_u^{exp}; in either case, the sweeps stop above g and hence above f, so f is not crossed. So we assume that g lies below f; see Figure 7. Note that an upward sweep, either in the initialization step of the insertion of a square or in the reporting step of the deletion of a square, always stops at the top edge of ρ_u. After S has been inserted into $\mathcal{L}^*_{u,e}$ and until

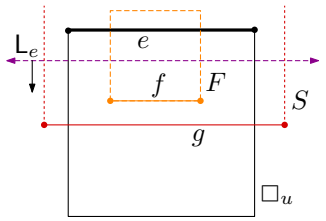


Figure 7 An illustration of the proof of Lemma 7: Sweep-line L_e is swept downward toward edge f of floater F during the insertion or deletion of S at node u.

it is deleted from $\mathcal{L}^*_{u,e}$, the top edge of ρ_u is contained in g or lies below it, which implies that no upward sweep will cross g until after S is deleted. Let $|B|$ denote the length of the longest side of any rectangle B. Since F is a floater at u and S contains an edge of $\square_u$ (as S is long at u), $|F| < |\square_u| < |S|$. Then, since S is inserted after F, S will be deleted after F has been deleted by property (P1), which implies that after L_e crosses f during the insertion of S, f will not be crossed by L_e in any subsequent sweeps. This proves claim (i).

Proof of claim (ii). Suppose L_e crosses f in a DD sweep, where $S \in \mathcal{L}^*_{u,e}$ is the square being deleted. This sweep occurs in the initialization step of the deletion of S as L_e moves from some position above f to the bottom edge of S^{exp}_u, which must be below f. The latter edge lies above the bottom edge g of S. See Figure 7 again. As noted above, in an upward sweep, either in the initialization step of the insertion of a square or in the reporting step of the deletion of a square, always stops at the top edge of ρ_u. While S is present, the top edge of ρ_u is contained in g or lies below it, which implies that no upward sweep can cross g until after S is deleted. Again, we have that $|F| < |\square_u| < |S|$, and hence property (P1) implies S is inserted before F since f is present during the deletion of S. Since g is below f, no upward sweeps preceding the deletion of S can cross f, proving claim (ii). ◀

Lemma 7 implies that each floater edge in $\mathcal{F}_u$ has sufficient credits to pay for the cost charged to it. Hence, we obtain the following:

▶ **Lemma 8.** *The amortized cost of inserting/deleting a square S at a node $u \in \Xi(S)$ is $O(\log^2 n + |\Delta V_u|)$ and at a node $u \in \Sigma(S) \setminus \Xi(S)$ is $O(\log^2 n)$.*

▶ Remark 9. The sweep-line algorithm and secondary data structures can be extended so that not only they compute ΔV_u, but also compute (the xy-projection of) $\partial\mathcal{U}$ within $S \cap \square_u$ in the same time bound. Hence, we can compute $\partial\mathcal{U}$ within each rectangle of $\Pi(S)$. By merging these pieces together over all rectangles of $\Pi(S)$, we can compute (the xy-projection of) $\partial\mathcal{U}$ within S. We thus compute $\partial\mathcal{U}$ on each horizontal face of the cubes in $\mathcal{C}$ in time $O(n \log^3 n + \kappa)$. By performing the plane-sweep in the x-direction and y-direction, we can compute $\partial\mathcal{U}$ on the other faces of cubes in $\mathcal{C}$ as well. These sweeps will be simpler because we already have computed the vertices of $\mathcal{U}$. We omit the details from this version.

2.4 Secondary structures

In this section, we describe the details of the wall and corner data structures at every node $u \in \mathbb{T}$ and edge e of $\square_u$.

Wall data structure. For $\mathbb{W}_e$, we use the data structure described by Wood [18], which is a standard segment tree augmented with additional information stored at each of its nodes. It supports our desired operations, Insert, Delete, Membership, and Report-Holes

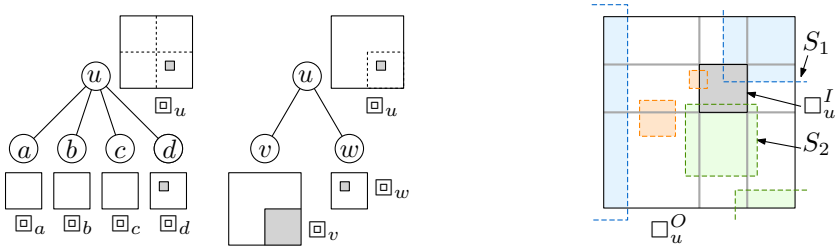


Figure 8 (left) Two examples of internal nodes in $\mathbb{T}$ with identical regions $\boxdot_u$. On the left, the outer square of $\boxdot_u$ is partitioned into four congruent squares, and on the right, $\boxdot_u$ is partitioned by a quadrant of its outer box. (right) Examples of long (blue), floater (orange), and corner (green) squares at rectangles of ∇_u. S_1 (resp. S_2) is a long (resp. corner) square for each rectangle of ∇_u that it intersects (even though S_1 is not a long square for $\square_u^O$).

in the time mentioned earlier. Its size is $O(|\mathsf{X}_u|)$ and is constructed in $O(|\mathsf{X}_u|)$ time at the preprocessing stage of the algorithm. Every point $p \in \mathsf{X}$ is contained in region $\square_u$ of exactly one node u in each level of $\mathbb{T}$. $\mathbb{T}$ has $O(\log n)$ levels, so constructing all wall data structures takes $O(\sum_u |\mathsf{X}_u|) = O(n \log n)$ time.

Corner data structure. For each edge e of $\square_u$, we construct the data structure of Agarwal [1, Lemma 5]. Although it was originally described to support *area* queries (report the area of $\rho \cap \mathcal{U}(\mathcal{R}_u) \setminus \mathcal{U}(\mathcal{W}_e)$ for a given query rectangle ρ), it is straightforward to extend the data structure to support our required operations, Report-Hole and Report-Vertices. In particular, to answer the area queries, it maintains the x-edges (resp. y-edges) of $\mathcal{U}(\mathcal{R}_u \cup \mathcal{W}_e)$ that lie on x-edges (resp. y-edges) of corner squares sorted by their y-coordinates (resp. x-coordinates), which is sufficient to answer our desired operations by searching over these lists of edges. Like $\mathbb{W}_e$, it is constructed in $O(|\mathsf{X}_u|)$ time at the preprocessing stage of the algorithm, so constructing all corner data structures takes $O(\sum_u |\mathsf{X}_u|) = O(n \log n)$ time.

3 Algorithm for the Unbounded-Spread Case

In this section, we extend the algorithm of Section 2 to the case when the spread of X, the xy-projections of the vertices of $\mathcal{C}$, is arbitrary. The algorithm is largely the same. The main challenge is that we cannot use a standard quadtree for our dynamic data structure $\mathbb{T}$, as it may have $\Omega(n)$ depth even if we use a compressed quadtree. Instead, $\mathbb{T}$ is a compressed quadtree with fingers[2] [13]. Its height is $O(\log n)$ regardless of the spread of X, its size is $O(n)$, and it can be constructed in $O(n \log n)$ time.

The properties of $\mathbb{T}$ are as follows. Every node $u \in \mathbb{T}$ is associated with a region $\boxdot_u$ that is a square or the difference of two nested squares $\square_u^O \setminus \square_u^I$, where $\square_u^O \supset \square_u^I$. For nodes u at which $\boxdot_u$ is a square, we say $\square_u^O = \boxdot_u$ and $\square_u^I = \varnothing$ so that $\square_u^O, \square_u^I$ are well-defined for all nodes of $\mathbb{T}$. For the root node r, $\boxdot_r \supset \mathsf{X}$. If $|\mathsf{X} \cap \boxdot_u| \leq 1$, u is a leaf; otherwise u is an internal node and $\boxdot_u$ is partitioned either into four regions with congruent outer squares, or it is partitioned into $\boxdot_u \setminus \square_u^S$ and $\square_u^S$ by a square $\square_u^S$ such that $\square_u^O \supset \square_u^S \supset \square_u^I$. In either

[2] The regions associated with the nodes of a compressed quadtree with fingers as described in [13] may have multiple holes, whereas the *BBD trees* proposed in [5] have at most one hole per node but rectangular regions (which may not be squares). By combining some of the ideas from the construction of BBD trees in [5] to ensure each region has at most one hole, our tree $\mathbb{T}$ can be constructed with all stated properties.

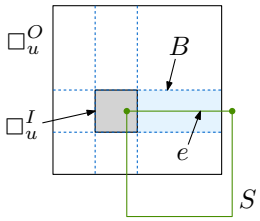


Figure 9 An illustration of the proof of Lemma 10: An edge e of a long square (green) intersecting rectangle $B \in \nabla_u$.

case, the regions in the partition of $\boxdot_u$ are associated with a child of u, and hence u has either two or four children. See Figure 8 (left). Recall that $|\rho|$ denotes the length of the longest side length of any rectangle ρ. Any nodes where $\square_u^I \neq \varnothing$, $\mathbb{T}$ has the property that $\square_u^I$ is *sticky*; that is, the distance between the top (resp. right, bottom, left) edge of $\square_u^O$ and the top (resp. right, bottom, left) edge of $\square_u^I$ is either 0, in which case the former contains the latter, or at least $|\square_u^I|$. For example, if $[o_1, o_2]$ (resp. $[i_1, i_2]$) is the x-projection of $\square_u^O$ (resp. $\square_i^I$), then for $j = 1, 2$ we have that $|o_j - i_j|$ is 0 or at least $|i_1 - i_2|$.

For a node u, we define long and short squares and the sets $\mathcal{S}_u$, $\mathcal{L}_u$, and $\mathcal{L}_u^*$ as earlier. Note that a vertex of a square $S \in \mathcal{L}_u$ may lie in the inner square $\square_u^I$ of $\boxdot_u$ (in fact, at most one vertex of S); see Figure 8 (right). For a square S, let $\Lambda(S)$, $\Sigma(S)$, $\Xi(S)$, and $\Pi(S)$ be the same as before. We report $\Delta V_u \coloneqq \Delta V \cap \square_u$ for each node $u \in \Xi(S)$. However, because $\boxdot_u$ may have a hole, reporting ΔV_u at u is more involved than a single sweep through $\boxdot_u$. We describe what we store at each node u, then how we update $\mathbb{T}$ and ultimately report ΔV_u at each node $u \in \Xi(S)$ due to the insertion or deletion of a square S at u.

The information at node u. We avoid all issues involving the inner square $\square_u^I$ by further partitioning $\boxdot_u$ into rectangles, then using essentially the same algorithm. In particular, the lines supporting the inner square $\square_u^I$ (if it exists) induce a partition of $\square_u$ into a set ∇_u of $m \leq 8$ non-empty rectangles $\{B_u^1, \ldots, B_u^m\}$; if $\square_u^I = \varnothing$, set $\nabla_u \coloneqq \{\boxdot_u\}$. See Figure 8 (right). For a rectangle $B \in \nabla_u$, let $\mathcal{L}_B$ (resp. $\mathcal{S}_B$) be the set of long (resp. short) squares at u that intersect B, and let $\mathcal{L}_B^* \coloneqq \mathcal{L}_B \cap \mathcal{L}_u^*$. Let $\mathcal{F}_B$ be the subset of short squares with at least two vertices in $\text{int}(B)$, and let $\mathcal{R}_B \coloneqq \mathcal{S}_B \setminus \mathcal{F}_B$ be the set of short squares with at most one vertex in $\text{int}(B)$. The following lemma will be crucial for our runtime analysis.

▶ **Lemma 10.** *For any node u of $\mathbb{T}$, rectangle $B \in \nabla$, and square $S \in \mathcal{L}_B$, $|S| > |B|$ and hence S contains an edge of B.*

Proof. Let S be a square in $\mathcal{L}_B$. S intersects $\text{int}(\boxdot_u)$ but no vertex of S lies in $\text{int}(\boxdot_u)$, so its vertices either lie in $\text{int}(\square_u^I)$ or outside $\square_u^O$. There are two cases. First, suppose all vertices of S lie outside $\square_u^O$. Then $|S| > |\square_u^O| > |B|$, as desired.

Next, suppose a vertex v_1 of S lies in $\text{int}(\square_u^I)$. If $S \supset B$, we are done, so suppose otherwise. No vertices of S lie in $\text{int}(B)$, so an edge $e \coloneqq v_1v_2$ of S spans B where v_2 is a vertex of S that lies outside $\square_u^O$. (If all vertices of S lie inside $\text{int}(\square_u^I)$ then S does not intersect B.) See Figure 9. By construction of ∇_u, the edges of B perpendicular to e have length $|\square_u^I|$ (in particular, one is an edge of $\square_u^I$ and the other is a portion of an edge of $\square_u^O$). By the sticky property of $\square_u^I$, the edges of B parallel to e has length at least $|\square_u^I|$, and thus at least as long as the former ones. Since e is longer than the edges of B parallel to it, $|S| > |B|$. ◀

To be consistent with the previous algorithm, we refer to the squares in $\mathcal{F}_B$ as *floater squares* and the squares in $\mathcal{R}_B$ as *corner squares*, even though a square $S \in \mathcal{R}_B$ may not have any vertices in $\text{int}(B)$ and may even contain B. For such a corner square $S \in \mathcal{R}_B$, we associate S with the top-left vertex v_1 of B if $v_1 \in S$, otherwise we associate S with the bottom-right vertex v_2 of B (in which case $v_2 \in S$).

We maintain $\mathcal{L}^*_B$ and $\mathcal{S}_B$ for each $B \in \nabla_u$ in the same fashion as $\mathcal{L}^*_u, \mathcal{S}_u$ were stored at nodes u of the quadtree in the previous algorithm. Note that by Lemma 10, we have the property that any square $S \in \mathcal{L}_B$ contains an edge of B as before. For each edge e of B, we store the long squares that contain e, $\mathcal{L}^*_{B,e}$, in red-black trees as before. We similarly store the x-edges (resp. y-edges) of floater squares in $\mathcal{F}_B$ sorted by their y-coordinates (resp. x-coordinates), which we call $\mathcal{E}^x_B$ (resp. $\mathcal{E}^y_B$).

For each edge e of a rectangle $B \in \nabla_u$, we maintain a value ℓ_e that is the position of the sweep-line L_e associated with edge e, a wall data structure $\mathbb{W}_e(\mathcal{I}_e)$ and corner data structure $\mathbb{C}_e(\mathcal{R}_B, \mathcal{I}_e)$, where $\mathcal{I}_e$ are the x-projections (resp. y-projections) of squares in $\mathcal{F}_B$ intersected by L_e if e is an x-edge (resp. y-edge). Recall their descriptions from Section 2.2.

Reporting ΔV_u. To report ΔV_u when we insert (or delete) a square S, we report $\Delta V_B := B \cap \Delta V_u$ for each rectangle $B \in \nabla_u$ intersected by S (there are no vertices of ΔV_B if $S \cap B = \varnothing$). The insertion and deletion procedures are largely the same as before. In particular, we employ the sweep-line approach from the previous algorithm and essentially treat B as if it was $\square_u$ to report ΔV_B, as follows.

3.1 Reporting ΔV

We describe the procedure to report ΔV and update the data structures for the insertion of a square S; the procedure for the deletion of S is a similar extension of the deletion procedure described in Section 2.2 for bounded spread. There are four main steps:

(1) At each node $u \in \Lambda(S) \cup \Sigma(S)$, we compute $\rho_B := \text{cl}(B \setminus \mathcal{U}(\mathcal{L}_B))$ for all rectangles $B \in \nabla_u$ using a top-down traversal of $\mathbb{T}$ and the sequences of $\mathcal{L}^*_B$, as described below.

(2) At each node $u \in \Xi(S)$, we report ΔV_B for all rectangles $B \in \nabla_u$ by sweeping a line from an edge of rectangle $S^{\text{exp}}_B := S \cap \rho_B$, i.e., the portion of $S \cap B$ that is left "exposed" by the squares of $\mathcal{L}_B$.

(3) For each node $u \in \Lambda(S)$, we insert S to $\mathcal{L}^*_B$ for each rectangle $B \in \nabla_u$ that S intersects. In particular, if S contains the edge e of B, S is inserted into $\mathcal{L}^*_{B,e}$; recall that if $B \subseteq S$, then S is associated with the top edge of B.

(4) For each rectangle $B \in \nabla_u$ at node $u \in \Sigma(S)$, we update $\mathcal{S}_B$ and its secondary structures, as follows. If $S \cap B = \varnothing$, there is nothing to do, so suppose otherwise. If S is a corner square for B, we insert S into all four corner data structures $\mathbb{C}_e$ stored at B for each of its edges. Otherwise, S is a floater square, and we first insert the x-edges (resp. y-edges) of S which intersect B into $\mathcal{E}^x_B$ (resp. $\mathcal{E}^y_B$). Then, for each x-edge (resp. y-edge) e of B such that ℓ_e lies in the y-span (resp. x-span) of S, we insert the x-projection (resp. y-projection) of S to $\mathbb{W}_e$ and $\mathbb{C}_e$.

We now describe the first two steps in more detail. Consider a node $u \in \Lambda(S) \cup \Sigma(S)$, and assume that the rectangle ρ_D has been computed for all $D \in \nabla_{p(u)}$. Let $\rho^*_B := B \setminus \mathcal{U}(\mathcal{L}^*_B)$, which can be computed in $O(1)$ time using the sequences of $\mathcal{L}^*_B$. Then ρ_B can be computed in $O(1)$ time using ρ^*_B and the rectangles ρ_D for each $D \in \nabla_{p(u)}$, using the fact that

$$\begin{aligned}\rho_B &= B \setminus \mathcal{U}(\mathcal{L}_B) = B \setminus \mathcal{U}(\mathcal{L}^*_B \cup \mathcal{L}_{p(u)}) = (B \setminus \mathcal{U}(\mathcal{L}^*_B)) \cap (B \setminus \mathcal{U}(\mathcal{L}_{p(u)})\\ &= \rho^*_B \cap (\boxdot_{p(u)} \setminus \mathcal{U}(\mathcal{L}_{p(u)})) = \rho^*_B \cap \Big(\bigcup_{D \in \nabla_{p(u)}} B_D \setminus \mathcal{U}(\mathcal{L}_D)\Big)\\ &= \rho^*_B \cap \Big(\bigcup_{D \in \nabla_{p(u)}} \rho_D\Big) = \bigcup_{D \in \nabla_{p(u)}} (\rho^*_B \cap \rho_D).\end{aligned}$$

That is, ρ_B is the union of at most eight interior-disjoint rectangles, each of which is of the intersection of rectangles ρ^*_B and ρ_D for a rectangle $D \in \nabla_{p(u)}$, which we assume have been precomputed. Hence, ρ_B can be computed in $O(1)$ time. This completes step (1).

Let B be a rectangle of ∇_u for a node $u \in \Xi(S)$. Note that S^{exp}_B and ρ_B are indeed rectangles, since any square in $\mathcal{L}_u$ contains an edge of B by Lemma 10 and B is a rectangle. The main idea is that even though the sweep-line procedure to report ΔV_u in Section 2.3 was described for a square, the correctness is invariant of whether it is a square or a rectangle as in this scenario. We report ΔV_B in the same way as before: If $u \in \Sigma(S)$, then ΔV_B is the set of $O(1)$ vertices of $S \cap \rho_B$ that lie in $\mathrm{int}(S \cap B)$ and hence can be computed in $O(1)$ time. Otherwise, $u \in \Lambda(S)$. Suppose S contains the top edge e of B. If $S^{\mathsf{exp}}_B = \varnothing$, $\Delta V_B = \varnothing$, and there is nothing to. So assume $S^{\mathsf{exp}}_B \neq \varnothing$. We first initialize the sweep line L_e to be at the top edge of rectangle S^{exp}_B, then we sweep it to its bottom edge and report all vertices of ΔV_B using $\mathbb{W}_e$ and $\mathbb{C}_e$ in the process. The details of the sweep-line procedure are the same as in the bounded spread case; see Section 2.3.

Runtime analysis. Most of the runtime analysis of the previous algorithm easily extends to this algorithm. Let S be a square being inserted or deleted. Since $|\nabla_u| \leq 8$ for each node $u \in \mathbb{T}$, inserting S to the secondary structures at each rectangle $B \in \nabla_u$ of a node u in $\Lambda(S)$ (resp. $\Sigma(S)$) still takes $O(\log n)$ (resp. $O(\log^2 n)$) time. To extend the amortized time to report ΔV_B for a rectangle $B \in \nabla_u$ of a node $u \in \Xi(S)$, we again charge $O(\log^2 n)$ time to the edges of floaters encountered during the sweep-line procedure at B. Then, as in Lemma 7, it can be shown that each such edge is charged at most six times by the sweeps at B during the entire algorithm. However, there is a subtle issue when extending the proof of the lemma to this setting: the proof used the inequalities $|S| > |B| > |F|$ for any floater $F \in \mathcal{F}_B$ in order to conclude $|S| > |F|$ and apply property (P1), but our justification for those inequalities relied on B being a square in that setting. We instead use Lemma 10 to conclude $|S| > |F|$, which crucially relies on the sticky property of $\mathbb{T}$.

The remainder of the analysis follows as before: it follows that the amortized runtime to perform the at most eight sweeps (at most one per rectangle of ∇_u) at node $u \in \Xi(S)$ is $O(\log^2 n + |\Delta V_u|)$, and hence inserting or deleting S to $\mathbb{T}$ and reporting ΔV takes $O(\log^3 n + |\Delta V|)$ time. Thus, the entire algorithm takes $O(n \log^3 n + \kappa)$ time, where κ is the number of vertices of $\mathcal{U}$, proving Theorem 1.

4 Algorithms for Special Cases

In this section, we simplify the algorithm for two special cases in which $\mathcal{C} := \{C_1, \ldots, C_n\}$ is a set of n axis-aligned cubes in $\mathbb{R}^3$ in general position that:

1. have bounded *depth*, i.e., the maximum number of cubes in $\mathcal{C}$ that contain any point $p \in \mathbb{R}^3$ is bounded by a constant $c > 0$, or
2. are all congruent.

4.1 Cubes with Bounded Depth

We assume that the depth of the cubes in $\mathcal{C}$ is bounded by a constant. For simplicity, we also assume that the cubes in $\mathcal{C}$ have bounded spread; it is straightforward but slightly more tedious to extend the following techniques to the unbounded spread case. In this case, we employ the same algorithm as described in Section 2, except that we no longer need the corner data structures and can perform their operations in $O(1)$ time, as explained below. As a result, the amortized runtime to report ΔV due to the insertion or deletion of a square in $\mathcal{S}$ improves to $O(\log^2 n + |\Delta V|)$, as follows.

Fix a rectangle $B \in \nabla_u$ for a node $u \in \mathbb{T}$, and let $v \coloneqq (x_i, y_i)$ be one of the four vertices of B. For any $a \in \mathbb{R}$, $O(1)$ cubes of $\mathcal{C}$ contain (x_i, y_i, a), and hence $O(1)$ squares in $\mathcal{S}$ contain v at any point during the plane-sweep. In particular, $O(1)$ squares contain any of the four vertices of B. Any square in $\mathcal{R}_B$ contains a vertex of B, and hence $|\mathcal{R}_B| = O(1)$. Then $\mathcal{U}(\mathcal{R}_B)$, and thus $\mathcal{U}(\mathcal{R}_B) \cap \square_u$, has constant complexity, so the latter can be maintained explicitly in $O(1)$ time per insertion or deletion to $\mathcal{R}_u$ (at worst, it takes $O(1)$ time to recompute it from scratch any time that it is needed). Furthermore, given any query segment $\sigma \supset B$, $\sigma \setminus \operatorname{int}(\mathcal{U}(\mathcal{R}_u))$ can be computed in $O(1)$ time, which replaces the need for the Report-Hole(σ) operation of the corner data structures.

Lastly, we need to implement the Report-Vertices(ρ) operation of the corner data structures for each edge e of B using $\mathcal{U}(\mathcal{R}_u) \cap \square_u$ and $\mathbb{W}_e$, where $\rho \subset \square_u$ is a query rectangle. Suppose e is an x-edge of B. We first compute $\mathcal{U}_\rho \coloneqq \mathcal{U}(\mathcal{R}_u) \cap \rho$ in $O(1)$ time, and then, for each edge σ of $\mathcal{U}_\rho$, we compute the endpoints of the intervals of $\sigma^\downarrow \setminus \operatorname{int}(\mathcal{U}(\mathcal{I}_e))$ in $O(\log n + \kappa_\sigma)$ time, where $\sigma^\downarrow$ is the x-projection of σ and κ_σ is the number of endpoints reported. These 1D vertices correspond to the 2D vertices of $\mathcal{U}(\mathcal{S})$ on σ, and any vertex is reported at most twice. Thus, the overall runtime for this operation is $O(\log n + \kappa)$, where κ is the total number of vertices reported.

Plugging these improved bounds for the corresponding operations of the corner data structures in the analysis of the sweep-line procedure, we have that the amortized time to report ΔV for the insertion or deletion of a square S to $\mathcal{S}$ is $O(\log^2 n + |\Delta V|)$. Given that κ, the total number of vertices of $\mathcal{U}$, is $O(n)$ in this case, the overall runtime is $O(n \log^2 n)$, as claimed in Theorem 3.

4.2 Congruent Cubes

Without loss of generality, assume that all cubes in $\mathcal{C}$ are unit cubes. In this case, $\mathcal{S}$ is now a set of unit squares in $\mathbb{R}^2$. Whenever the sweeping plane reaches the top or bottom face of a cube in $\mathcal{C}$, we neither need a tree data structure nor do we need a 2D sweep-line procedure to report ΔV. Instead we only need a 2D grid and a simpler version of the corner data structure, as described below.

The data structure. Let $\mathcal{G}$ be the 2-dimensional integer grid. Without loss of generality, no point of X lies in on the grid lines of $\mathbb{R}^2$. For $i, j \in \mathbb{Z}$, let $\square_{i,j}$ denote the grid cell $[i, i+1] \times [j, j+1]$. For all $i, j \in \mathbb{Z}$, let $\mathsf{X}_{i,j} \coloneqq \mathsf{X} \cap \square_{i,j}$, and let $\mathcal{G}^*$ be the non-empty grid cells of $\mathcal{G}$, i.e., $\mathcal{G}^* = \{\square_{i,j} \in \mathcal{G} \mid |\mathsf{X}_{i,j}| > 0\}$.

For any square $S \in \mathcal{S}$, any grid cell $\square \in \mathcal{G}^*$ intersected by S contains exactly one vertex of S; that is, S is a *corner square* for $\square$. For any grid cell $\square \in \mathcal{G}^*$, let $\mathcal{S}_\square$ be the set of squares in $\mathcal{S}$ that intersect $\square$. Since there are no long or floater squares at any grid cell $\square \in \mathcal{G}^*$, there are no (projections of) floaters to maintain, nor any vertices of ΔV defined by such squares to report, which accounts for much of the intricacies of the previous algorithms.

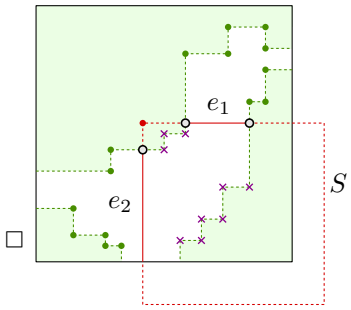


Figure 10 An illustration of a square S (red) being inserted at a grid cell $\square \in \Sigma(S)$ (black). The solid portions of edges e_1,e_2 of S are the portions of the edges returned by the calls to Report-Hole. The vertices of $V^{\text{new}} \cap \square$ are marked as hollow and lie on e_1 and e_2, and the vertices of $V^{\text{old}} \cap \square$ are marked as crosses and lie in $\text{int}(S) \cap \square$.

For each $\square \in \mathcal{G}^*$, we build one *corner data structure* $\mathbb{C}_\square$, that maintains $\mathcal{S}_\square$ and supports the following operations:

- Insert/Delete(S): Insert or delete a corner square S to $\mathcal{R}_\square$.
- Report-Hole(σ): Given a query axis-aligned segment $\sigma \subset \square$, return the (at most one) interval of $\sigma \setminus \text{int}(\mathcal{U}(\mathcal{S}_\square))$.
- Report-Vertices(ρ): Let $\rho \subseteq \square$ be a corner rectangle, i.e., one of its vertices is a vertex of $\square$. Return $V(\mathcal{S}_\square) \cap \rho$.

As in Section 2.1, we implement $\mathbb{C}_\square$ using the data structure of Agarwal [1, Lemma 5], which supports the operations above without modification. An Insert/Delete takes $O(\log^2 n)$ time, Report-Hole takes $O(\log n)$ time, and Report-Vertices takes $O(\log n + \kappa)$ time, where κ is the number of vertices reported. Note that, in contrast with the corner data structures used in the previous two algorithms, we build only one for each cell $\square$ instead of one per edge of $\square$, and this data structure does not maintain a set of intervals in addition to the set of corner squares $\mathcal{S}_\square$. Using the fact that there are no intervals and the query rectangle is a corner rectangle, the corner data structure in [1] can be simplified, but we skip the details here.

Reporting ΔV. We describe the procedure to report ΔV and update the data structures for the insertion of a square S; the procedure for the deletion of S is a similar extension of the deletion procedure described in Section 2.2 for bounded spread. Let $\Sigma(S)$ be the four grid cells that S intersects. The grid cells in $\Sigma(S)$ partition S. We report $\Delta V_\square := \Delta V \cap \square$ for each grid cell $\square \in \Sigma(S)$.

Let $V^{\text{new}} \subseteq \Delta V$ be the set of vertices that are created by the insertion of S (which appear on ∂S), and let V^{old} be the set of vertices that no longer appear on $\mathcal{U}$ after the insertion of S (which lie in $\text{int}(S)$).

Fix a grid cell $\square \in \Sigma(S)$. Let e_1, e_2 be the edges incident to the vertex of S in $\text{int}(\square)$. The vertices of $V^{\text{new}} \cap \square$ that lie on e_1 (resp. e_2) are the endpoints of $e_1 \setminus \text{int}(\mathcal{U}(\mathcal{S}_\square))$ (resp. $e_2 \setminus \text{int}(\mathcal{U}(\mathcal{S}_\square))$ that lie in $\text{int}(\square)$. See Figure 10. There are at most two such vertices lying on each edge e_i, and they are computed by calling $\mathbb{C}_\square$ with Report-Hole($e_i \cap \square$)). Then we report $V^{\text{old}} \cap \square$, i.e., the vertices of $\mathcal{U}(\mathcal{S}_\square) \cap \square$, by calling $\mathbb{C}_\square$ with Report-Vertices($S \cap \square$). The vertices reported account for all vertices of ΔV. Finally, we insert S to $\mathcal{S}_\square$ by calling $\mathbb{C}_\square$ with Insert(S). Repeating this step for all four cells in $\Sigma(S)$, we report ΔV.

The eight Report-Hole calls take $O(\log n)$ time overall, the four Report-Vertices calls take $O(\log n + |\Delta V|)$ time, and the four Insert calls take $O(\log^2 n)$ time. Hence, the total time spent for the insertion of S is $O(\log^2 n + |\Delta V|)$. Similarly, the deletion of a square takes $O(\log^2 n + |\Delta V|)$ time.

Runtime analysis. $\mathcal{G}^*$ and $\Sigma(S)$ for all squares S can be computed in $O(n \log n)$ time. At the beginning of the algorithm, no cubes intersect the sweeping plane and hence $\mathcal{S}_\square = \varnothing$, so building $\mathbb{C}_\square$ for $\square \in \mathcal{G}^*$ takes $O(n)$ time overall. Given that κ, the total number of vertices of $\mathcal{U}$, is $O(n)$ in this case, the overall runtime is $O(n \log^2 n)$, as claimed in Theorem 3.

Unit cubes with bounded depth. Suppose the cubes of $\mathcal{C}$ are unit cubes with bounded depth. Let $\mathcal{G}$ be the 3D integer grid, which partitions $\mathbb{R}^3$ into unit cubes, and let $\mathcal{G}^* \subset \mathcal{G}$ be the grid cells intersected by at least one cube in $\mathcal{C}$; $|\mathcal{G}^*| \leq 8n$. Let $\mathcal{C}_G \subseteq \mathcal{C} := \{C \cap G \neq \varnothing \mid C \in \mathcal{C}\}$ for each grid cell $G \in \mathcal{G}^*$; $\sum_{G \in \mathcal{G}^*} |\mathcal{C}_G| \leq 8n$. For any grid cell $G \in \mathcal{G}^*$, any cube in $\mathcal{C}_G$ contains a vertex of G, which implies that $|\mathcal{C}_G|$ is bounded by a constant since the cubes of $\mathcal{C}$ have bounded depth. Therefore $\mathcal{U}(\mathcal{C}_G) \cap G = \mathcal{U} \cap G$ can be computed in $O(1)$ time. Then $\mathcal{U}$ can be computed by merging the portions within each grid cell of $\mathcal{G}^*$ in $O(n + \kappa) = O(n)$ time, where κ is the number of vertices on $\mathcal{U}$, which is bounded by $O(n)$ in this case. Computing $\mathcal{C}_G$ for all grid cells $G \in \mathcal{G}^*$ takes $O(n \log n)$ time, so the running time of the entire algorithm is $O(n \log n)$. Note that if the maximum distance between any two centers of cubes in $\mathcal{C}$ is polynomially bounded (i.e., is at most n^c for a constant $c > 0$) and $\lfloor x \rfloor$ can be computed in constant time for any $x \in \mathbb{R}$, computing the $\mathcal{C}_G$'s can be done in $O(n)$ time, which improves the overall running time to $O(n)$.

5 Conclusion

We have described algorithms to compute (the boundary of) the union of axis-aligned 3D cubes (or fat boxes) in general position in an output-sensitive manner. In particular, if the cubes have different sizes the union can be computed in $O(n \log^3 n + \kappa)$ time, where κ is the number of union vertices. If all cubes have the same size or they have bounded depth, then the union can be computed in $O(n \log^2 n)$ time, and if both conditions hold then the running time improves to $O(n \log n)$. We conclude by mentioning two open problems:

- **(i)** Can the running time be improved to $O(n \log n + \kappa)$?
- **(ii)** Can the union of a set of n cubes in $\mathbb{R}^d$ be computed in $O(n^{\lfloor d/2 \rfloor} + \kappa)$ time? In particular, can the union of n hypercubes in $\mathbb{R}^4$ be computed in $O(n \operatorname{polylog}(n) + \kappa)$ time? Kaplan *et al.* [15] have shown that for a special case of orthants in $\mathbb{R}^d$, the union of n such orthants can be computed in $O(n + \kappa) \log^{d-1} n)$ time, but their algorithm does not extend to hypercubes.

References

1. Pankaj K. Agarwal. An improved algorithm for computing the volume of the union of cubes. In *Proc. 26th Annu. Sympos. Comp. Geom.*, pages 230–239. ACM, 2010.
2. Pankaj K. Agarwal, Haim Kaplan, and Micha Sharir. Computing the volume of the union of cubes. In *Proc. 23rd Annu. Sympos. Comp. Geom.*, pages 294–301. ACM, 2007.
3. Pankaj K. Agarwal, Haim Kaplan, and Micha Sharir. Union of hypercubes and 3d minkowski sums with random sizes. *Discret. Comput. Geom.*, 65(4):1136–1165, 2021.
4. Pankaj K. Agarwal, Micha Sharir, and Alex Steiger. Decomposing the complement of the union of cubes in three dimensions. In *Proc. 2021 ACM-SIAM Sympos. Disc. Alg.*, pages 1425–1444. SIAM, 2021.

5 Sunil Arya, David M. Mount, Nathan S. Netanyahu, Ruth Silverman, and Angela Y. Wu. An optimal algorithm for approximate nearest neighbor searching fixed dimensions. *J. ACM*, 45(6):891–923, 1998.

6 Jean-Daniel Boissonnat, Micha Sharir, Boaz Tagansky, and Mariette Yvinec. Voronoi diagrams in higher dimensions under certain polyhedral distance functions. *Discret. Comput. Geom.*, 19(4):485–519, 1998.

7 Karl Bringmann. An improved algorithm for Klee's measure problem on fat boxes. *Comput. Geom.*, 45(5-6):225–233, 2012.

8 Timothy M. Chan. Klee's measure problem made easy. In *Proc. 54th Annu. IEEE Sympos. Found. Comp. Sci.*, pages 410–419. IEEE Computer Society, 2013.

9 L. Paul Chew, Dorit Dor, Alon Efrat, and Klara Kedem. Geometric pattern matching in d-dimensional space. *Discret. Comput. Geom.*, 21(2):257–274, 1999.

10 Herbert Edelsbrunner and Ernst P. Mücke. Simulation of simplicity: a technique to cope with degenerate cases in geometric algorithms. *ACM Trans. Graph.*, 9(1):66–104, 1990.

11 Alon Efrat. Private communication.

12 Ralf Hartmut Güting. An optimal contour algorithm for iso-oriented rectangles. *J. Algorithms*, 5(3):303–326, 1984.

13 Sariel Har-Peled. *Geometric Approximation Algorithms*. American Mathematical Society, 2011.

14 Sariel Har-Peled and Kent Quanrud. Approximation algorithms for polynomial-expansion and low-density graphs. *SIAM J. Comput.*, 46(6):1712–1744, 2017.

15 Haim Kaplan, Natan Rubin, Micha Sharir, and Elad Verbin. Efficient colored orthogonal range counting. *SIAM J. Comput.*, 38(3):982–1011, 2008.

16 Mark H. Overmars and Chee-Keng Yap. New upper bounds in Klee's measure problem. *SIAM J. Comput.*, 20(6):1034–1045, 1991.

17 Franco P. Preparata and Michael Ian Shamos. *Computational Geometry - An Introduction*. Springer, 1985.

18 Derick Wood. The contour problem for rectilinear polygons. *Inf. Process. Lett.*, 19(5):229–236, 1984.

19 Hakan Yildiz and Subhash Suri. Computing Klee's measure of grounded boxes. *Algorithmica*, 71(2):307–329, 2015.

Dynamic Enumeration of Similarity Joins

Pankaj K. Agarwal
Duke University, Durham, NC, USA

Xiao Hu
Duke University, Durham, NC, USA

Stavros Sintos
University of Chicago, IL, USA

Jun Yang
Duke University, Durham, NC, USA

Abstract

This paper considers enumerating answers to *similarity-join* queries under dynamic updates: Given two sets of n points A, B in $\mathbb{R}^d$, a metric $\phi(\cdot)$, and a distance threshold $r > 0$, report all pairs of points $(a, b) \in A \times B$ with $\phi(a, b) \leq r$. Our goal is to store A, B into a dynamic data structure that, whenever asked, can enumerate all result pairs with worst-case *delay* guarantee, i.e., the time between enumerating two consecutive pairs is bounded. Furthermore, the data structure can be efficiently updated when a point is inserted into or deleted from A or B.

We propose several efficient data structures for answering similarity-join queries in low dimension. For exact enumeration of similarity join, we present near-linear-size data structures for ℓ_1, ℓ_∞ metrics with $\log^{O(1)} n$ update time and delay. We show that such a data structure is not feasible for the ℓ_2 metric for $d \geq 4$. For *approximate* enumeration of similarity join, where the distance threshold is a soft constraint, we obtain a unified linear-size data structure for ℓ_p metric, with $\log^{O(1)} n$ delay and update time. In high dimensions, we present an efficient data structure with worst-case delay-guarantee using *locality sensitive hashing* (LSH).

2012 ACM Subject Classification Theory of computation → Data structures and algorithms for data management

Keywords and phrases dynamic enumeration, similarity joins, worst-case delay guarantee

Digital Object Identifier 10.4230/LIPIcs.ICALP.2021.11

Category Track A: Algorithms, Complexity and Games

Related Version *Full Version*: https://arxiv.org/abs/2105.01818

Funding This work has partially supported by NSF grants IIS-1814493 and CCF-2007556.

1 Introduction

There has been extensive work in many areas including theoretical computer science, computational geometry, and database systems on designing efficient dynamic data structures to store a set $\mathscr{O}$ of objects so that certain queries on $\mathscr{O}$ can be answered quickly and objects can be inserted into or deleted from $\mathscr{O}$ dynamically. A query $\mathscr{Q}$ is specified by a set of constraints and the goal is to report the subset $\mathscr{Q}(\mathscr{O}) \subseteq \mathscr{O}$ of objects that satisfy the constraints, the so-called *reporting* or *enumeration* queries. More generally, $\mathscr{Q}$ may be specified on k-tuples of objects in $\mathscr{O}$, and we return the subset of $\mathscr{O}^k$ that satisfy $\mathscr{Q}$. One may also ask to return certain statistics on $\mathscr{Q}(\mathscr{O})$ instead of $\mathscr{Q}(\mathscr{O})$ itself, but here we focus on enumeration queries. As an example, $\mathscr{O}$ is set of points in $\mathbb{R}^d$ and a query $\mathscr{Q}$ specifies a simple geometric region Δ (e.g., box, ball, simplex) and asks to return $\mathscr{O} \cap \Delta$, the so-called *range-reporting* problem. As another example, $\mathscr{O}$ is again a set of points in $\mathbb{R}^d$, and $\mathscr{Q}$ now specifies a value $r \geq 0$ and asks to return all pairs $(p, q) \in \mathscr{O} \times \mathscr{O}$ with $\|p - q\| \leq r$. Traditionally, the performance of a

data structure has been measured by its size, the time needed to update the data structure when an object is inserted or deleted, and the *total time* spent in reporting $\mathscr{Q}(\mathscr{O})$. In some applications, especially in exploratory or interactive data analysis of large datasets, it is desirable to report $\mathscr{Q}(\mathscr{O})$ incrementally one result at a time so that users can start exploiting the first answers while waiting for the remaining ones. To offer guarantees on the regularity during the enumeration process, we consider an important complexity measure of such data structures as the maximum *delay* between the enumeration of two consecutive objects [11]. Formally speaking, δ-*delay enumeration* requires that the time between the start of the enumeration process to the first result, the time between any consecutive pair of results, and the time between the last result and the termination of the enumeration process should all be at most δ.

In this paper, we are interested in dynamic data structures for *(binary) similarity join* queries, which have numerous applications in data cleaning, data integration, collaborative filtering, etc. Given two sets of points A and B in $\mathbb{R}^d$, a metric $\phi(\cdot)$, and a distance threshold $r > 0$, the similarity join asks to report all pairs of $(a, b) \in A \times B$ with $\phi(a, b) \leq r$. Similarity joins have been studied extensively in the database and data mining literature [19, 33, 40, 43, 45], but it is still unclear how to enumerate similarity join results efficiently when the underlying data is updated. Our goal is to design a dynamic data structure that can be efficiently updated when an input point is inserted or deleted; and whenever an enumeration query is issued, all join results can be enumerated from it with *worst-case delay* guarantee.

1.1 Previous results

We briefly review the previous work on similarity join and related problems. See surveys [8, 10, 44] for more results.

Enumeration for Conjunctive Queries. Conjunctive queries are built upon natural join ($\bowtie$), which is a special case of similarity join with $r = 0$, i.e., two tuples can be joined if and only if they have the same value on the join attributes. Enumeration for conjunctive queries has been extensively studied in the static settings [11, 42, 16] for a long time. In 2017, two papers [14, 31] started to study dynamic enumeration for conjunctive query. Both obtained a dichotomy. First, a linear-size data structure that can be updated in $O(1)$ time while supporting $O(1)$-delay enumeration exists for a conjunctive query if and only if it is *q-hierarchical* (e.g., the degenerated natural join over two tables is q-hierarchical). However, for non-q-hierarchical queries with input size n, they showed a lower bound $\Omega(n^{\frac{1}{2}-\varepsilon})$ on the update time for any small constant $\varepsilon > 0$, if aiming at $O(1)$ delay. This result is very negative since q-hierarchical queries are a very restricted class; for example, the matrix multiplication query $\pi_{X,Z} R_1(X, Y) \bowtie R_2(Y, Z)$, where $\pi_{X,Y}$ denotes the projection on attributes X, Y, and the triangle join $R_1(X, Y) \bowtie R_2(Y, Z) \bowtie R_3(Z, X)$ are already non-q-hierarchical. Later, Kara et al. [34] designed optimal data structures supporting $O(\sqrt{n})$-time maintenance for some selected non-q-hierarchical queries such as the triangle queries. However, it is still unclear if a data structure of $O(\sqrt{n})$-time maintenance exists for a large class of queries. Some additional trade-off results have been obtained in [35, 46].

Range search. A widely studied problem related to similarity join is *range searching* [2, 3, 13, 47]: Preprocess a set A of points in $\mathbb{R}^d$ with a data structure so that for a query range γ (e.g., rectangle, ball, simplex), all points of $A \cap \gamma$ can be reported quickly. A particular instance of range searching, the so-called *fixed-radius-neighbor* searching, in which the range is a ball of fixed radius centered at query point is particularly relevant for similarity joins.

Table 1 Summary of Results: n is the input size; r is the distance threshold; d is the dimension of input points; $\rho \leq \frac{1}{(1+\varepsilon)^2} + o(1)$ is the quality of LSH family for the ℓ_2 metric. For ℓ_1, Hamming $\rho \leq \frac{1}{1+\varepsilon}$. $\widetilde{O}$ notation hides a $\log^{O(1)} n$-factor; for the results where d is constant the $O(1)$ exponent is at most linear on d, while for the high dimensional case the exponent is at most 3.

Enumeration	Metric	Properties	Data Structures		
			Space	Update	Delay
Exact	ℓ_1/ℓ_∞	r is fixed	$\widetilde{O}(n)$	$\widetilde{O}(1)$	$\widetilde{O}(1)$
	ℓ_2	r is fixed	$\widetilde{O}(n)$	$\widetilde{O}(n^{1-\frac{1}{d+1}})$	$\widetilde{O}(n^{1-\frac{1}{d+1}})$
ϵ-Approximate	ℓ_p	r is fixed	$O(n)$	$\widetilde{O}(\epsilon^{-d})$	$\widetilde{O}(\epsilon^{-d})$
		r is variable spread is poly(n)	$O(\varepsilon^{-d}n)$	$\widetilde{O}(\varepsilon^{-d})$	$O(1)$
	ℓ_1, ℓ_2, hamming	r is fixed high dimension	$\widetilde{O}(dn + n^{1+\rho})$	$\widetilde{O}(dn^{2\rho})$	$\widetilde{O}(dn^{2\rho})$

For a given metric ϕ, let $\mathscr{B}_\phi(x, r)$ be the ball of radius r centered at x. A similarity join between two sets A, B can be answered by querying A with ranges $\mathscr{B}_\phi(b, r)$ for all $b \in B$. Notwithstanding this close relationship between range searching and similarity join, the data structures for the former cannot be used for the latter: It is too expensive to query A with $\mathscr{B}_\phi(b, r)$ for every $b \in B$ whenever an enumeration query is issued, especially since many such range queries may return empty set, and it is not clear how to maintain the query results as the input set A changes dynamically.

Reporting neighbors. The problem of reporting neighbors is identical to our problem in the offline setting. In particular, given a set P of n points in $\mathbb{R}^d$ and a parameter r, the goal is to report all pairs of P within distance r. The algorithm proposed in [36] can be modified to solve the problem of reporting neighbors under the ℓ_∞ metric in $O(n + k)$ time, where k is the output size. Aiger et al. [7] proposed randomized algorithms for reporting neighbors using the ℓ_2 metric in $O((n + k) \log n)$ time, for constant d.

Scalable continuous query processing. There has been some work on scalable *continuous query* processing, especially in the context of data streams [21, 18, 49] and publish/subscribe [25], where the queries are standing queries and whenever a new data item arrives, the goal is to report all queries that are affected by the new item [6, 5]. In the context of similarity join, one can view A as the data stream and $\mathscr{B}_\phi(b, r)$ as standing queries, and we update the results of queries as new points in A arrive. There are, however, significant differences with similarity joins – arbitrary deletions are not handled; continuous queries do not need to return previously produced results; basing enumeration queries on a solution for continuous queries would require accessing previous results, which can be prohibitive if stored explicitly.

1.2 Our results

We present several dynamic data structures for enumerating similarity joins under different metrics. Table 1 summarizes our main results. It turns out that dynamic similarity join is hard for some metrics, e.g., ℓ_2. Therefore we also consider *approximate similarity join* where the distance threshold r is a soft constraint. Formally, given parameter $r, \varepsilon > 0$, the *ε-approximate similarity join* relaxes the distance threshold: (1) all pairs of $(a, b) \in A \times B$ with $\phi(a, b) \leq r$ should be returned; (2) no pair of $(a, b) \in A \times B$ with $\phi(a, b) > (1 + \varepsilon)r$ is returned; (3) some pairs of $(a, b) \in A \times B$ with $r < \phi(a, b) \leq (1 + \varepsilon)r$ may be returned. We classify our results in four broad categories:

Exact similarity join. Here we assume that d is constant and the distance threshold is fixed. Our first result (Section 2.1) is an $\widetilde{O}(1)$-size data structure for similarity join under the ℓ_1/ℓ_∞ metrics that can be updated in $\widetilde{O}(1)$ time whenever A or B is updated, and ensures $\widetilde{O}(1)$ delay during enumeration. Based on range trees [12, 23], the data structure stores the similarity join pairs *implicitly* so that they can be enumerated without probing every input point. We extend these ideas to construct a data structure for similarity join under the ℓ_2 metric (in Section 2.3) with $\widetilde{O}(n^{1-1/d})$ amortized update time while supporting $\widetilde{O}(n^{1-1/d})$-delay enumeration. Lower bounds on ball range searching [1, 20] rule out the possibility of a linear-size data structure with $\widetilde{O}(1)$ delay.

Approximate similarity join in low dimensions. Due to the negative result for ℓ_2 metric, we shift our attention to ε-approximate similarity join. We now allow the distance threshold to be part of the query, but the value of ε, the error parameter, is fixed. We present a simple linear-size data structure based on quad trees and the notion of well-separated pair decomposition, with $O(\epsilon^{-d})$ update time and $O(1)$ delay. If we fix the distance threshold, then the data structure can be further simplified and somewhat improved by replacing the quad tree with a simple uniform grid.

Approximate similarity join in high dimensions. So far we assumed d to be constant and the big O notation in some of the previous bounds hides a constant that is exponential in d. Our final result is an LSH-based [27] data structure for similarity joins in high dimensions. Two technical issues arise when enumerating join results from LSH: one is to ensure bounded delay because we do not want to enumerate false positive results identified by the hash functions, and the other is to remove duplicated results as one join result could be identified by multiple hash functions. For the ℓ_2 metric (the results can also be extended to ℓ_1 and Hamming metrics) we propose a data structure of $\widetilde{O}(nd + n^{1+\rho})$ size and $\widetilde{O}(dn^{2\rho})$ amortized update time that supports $(1 + 2\varepsilon)$-approximate enumeration with $\widetilde{O}(dn^{2\rho})$ delay with high probability, where $\rho \leq \frac{1}{(1+\varepsilon)^2} + o(1)$ is the quality of the LSH family. Alternatively, we present a data structure with $\widetilde{O}(dn^{\rho})$ amortized update time and $\widetilde{O}(dn^{3\rho})$ delay. Our data structure can be extended to the case when the distance threshold r is variable. If we allow worse approximation error we can improve the results for the Hamming distance. Finally, we show a lower bound by relating similarity join to the *approximate nearest neighbor* query.

We also consider similarity join beyond binary joins.

Triangle similarity join in low dimensions. Given three sets of points A, B, S in $\mathbb{R}^d$, a metric $\phi(\cdot)$, and a distance threshold $r > 0$, the *triangle similarity join* asks to report the set of all triples of $(a, b, s) \in A \times B \times S$ with $\phi(a, b) \leq r, \phi(a, s) \leq r, \phi(b, s) \leq r$. The *$\varepsilon$-approximate triangle similarity join* can be defined similarly by taking the distance threshold r as a soft constraint. In the full version [4], we extend our data structures to approximate *triangle similarity join* by paying an extra factor of $\log^{O(1)} n$ in the performance.

High-level framework. All our data structures rely on the following common framework. We model the (binary) similarity join as a bipartite graph $G' = (A \cup B, E)$, where an edge $(a, b) \in E$ if and only if $\phi(a, b) \leq r$. A naive solution by maintaining all edges of G' explicitly leads to a data structure of $\Theta(n^2)$ size that can be updated in $\Theta(n)$ time while supporting $O(1)$-delay enumeration. To obtain a data structure with poly-logarithmic update time and delay enumeration, we find a compact representation of G' with a set $\mathscr{F} = \{(A_1, B_1), (A_2, B_2), \ldots, (A_u, B_u)\}$ of edge-disjoint bi-cliques such that (i) $A_i \subseteq A$,

$B_i \subseteq B$ for any i, (ii) $E = \bigcup_{i=1}^{u} A_i \times B_i$, and (iii) $(A_i \times B_i) \cap (A_j \times B_j) = \emptyset$ for any $i \neq j$. We represent $\mathscr{F}$ using a tripartite graph $\mathscr{G} = (A \cup B \cup C, E_1 \cup E_2)$ where $C = \{c_1, \ldots, c_u\}$ has a node for each bi-clique in $\mathscr{F}$ and for every $i \leq u$, we have the edges $(a_j, c_i) \in E_1$ for all $a_j \in A_i$ and $(b_k, c_i) \in E_1$ for all $b_k \in B_i$. We *cannot* afford to maintain E_1 and E_2 explicitly. Instead, we store some auxiliary information for each c_i and use geometric data structures to recover the edges incident to a vertex $c_i \in C$. We also use data structures to maintain the set C and the auxiliary information dynamically as A and B are being updated. We will not refer to this framework explicitly but it provides the intuition behind all our data structures. Section 2 describes the data structures to support this framework for exact similarity join, and Section 3 presents simpler, faster data structures for approximate similarity join. Both Sections 2 and 3 assume d to be constant. Section 4 describes the data structure for approximate similarity join when d is not constant.

2 Exact Similarity Join

In this section, we describe the data structure for exact similarity joins under the $\ell_\infty, \ell_1, \ell_2$ metrics, assuming d is constant. We first describe the data structure for the ℓ_∞ metric. We show that similarity join under the ℓ_1 metric in $\mathbb{R}^d$ can be reduced to that under the ℓ_∞ metric in $\mathbb{R}^{d+1}$. Finally, we describe the data structure for the ℓ_2 metric. Throughout this section, the threshold r is fixed, which is assumed to be 1 without loss of generality.

2.1 Similarity join under ℓ_∞ metric

Let A and B be two point sets in $\mathbb{R}^d$ with $|A| + |B| = n$. For a point $p \in \mathbb{R}^d$, let $\mathscr{B}(p) = \{x \in \mathbb{R}^d \mid \|p - x\|_\infty \leq 1\}$ be the hypercube of side length 2. We wish to enumerate pairs $(a, b) \in A \times B$ such that $a \in \mathscr{B}(b)$.

Data structure. We build a d-dimensional dynamic range tree $\mathscr{T}$ on the points in A. For $d = 1$, the range tree on A is a balanced binary search tree $\mathscr{T}$ of $O(\log n)$ height. The points of A are stored at the leaves of $\mathscr{T}$ in increasing order, while each internal node v stores the smallest and the largest values, α_v^- and α_v^+, respectively, contained in its subtree. The node v is associated with an interval $I_v = [\alpha_v^-, \alpha_v^+]$ and the subset $A_v = I_v \cap A$. For $d > 1$, $\mathscr{T}$ is constructed recursively: We build a 1D range tree $\mathscr{T}_d$ on the x_d-coordinates of points in A. Next, for each node $v \in \mathscr{T}_d$, we recursively construct a $(d-1)$-dimensional range tree $\mathscr{T}_v$ on A_v^*, which is defined as the projection of A_v onto the hyperplane $x_d = 0$, and attach $\mathscr{T}_v$ to v as its secondary tree. The size of $\mathscr{T}$ in $\mathbb{R}^d$ is $O(n \log^{d-1} n)$ and it can be constructed in $O(n \log^d n)$ time. See [23] for details.

For a node v at a level-i tree, let $p(v)$ denote its parents in that tree. If v is the root of that tree, $p(v)$ is undefined. For each node u of the d-th level of $\mathscr{T}$, we associate a d-tuple $\pi(u) = \langle u_1, u_2, \ldots, u_d = u \rangle$, where u_i is the node at the i-th level tree of $\mathscr{T}$ to which the level-$(i+1)$ tree containing u_{i+1} is connected. We associate the rectangle $\square_u = \prod_{j=1}^{d} I_{u_j}$ with the node u. For a rectangle $\rho = \prod_{i=1}^{d} \delta_i$, a d-level node u is called a *canonical node* if for every $i \in [1, d]$, $I_{u_i} \subseteq \delta_i$ and $I_{p(u_i)} \not\subseteq \delta_i$. For any rectangle ρ, there are $O(\log^d n)$ canonical nodes in $\mathscr{T}$, denoted by $\mathscr{N}(\rho)$, and they can be computed in $O(\log^d n)$ time [23]. $\mathscr{T}$ can be maintained dynamically, as points are inserted into A or deleted from A using the standard partial-reconstruction method, which periodically reconstructs various bottom subtrees. The amortized time is $O(\log^d n)$; see [39] for details.

We query $\mathscr{T}$ with $\mathscr{B}(b)$ for all $b \in B$ and compute $\mathscr{N}(b) := \mathscr{N}(\mathscr{B}(b))$ the sets of its canonical nodes. For each level-d tree node u of $\mathscr{T}$, let $B_u = \{b \in B \mid u \in \mathscr{N}(b)\}$. We have $\sum_u |B_u| = O(n \log^d n)$. By construction, for all pairs $(a, b) \in A_u \times B_u$, $\|a - b\|_\infty \leq 1$, so

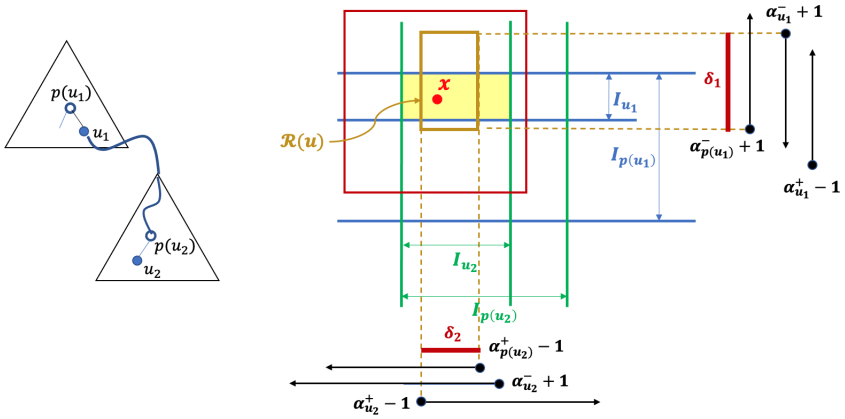


Figure 1 Left: Two levels of the range tree. Right: Definition of $\mathscr{R}(u)$.

(A_u, B_u) is a bi-clique of join results. We call u *active* if both $A_u, B_u \neq \emptyset$. A naive approach for reporting join results is to maintain A_u, B_u for every d-level node u of $\mathscr{T}$ as well as the set $\mathscr{C}$ of all active nodes. Whenever an enumerate query is issued, we traverse $\mathscr{C}$ and return $A_u \times B_u$ for all $u \in \mathscr{C}$ (referring to the tripartite-graph framework mentioned in Introduction, C is the set of all level-d nodes of $\mathscr{T}$). The difficulty with this approach is that when A changes and $\mathscr{T}$ is updated, some d-level nodes change and we have to construct B_u for each new level-d node $u \in \mathscr{T}$. It is too expensive to scan the entire B at each update. Furthermore, although the average size of B_u is small, it can be very large for a particular u and this node may appear and disappear several times. So we need a different approach. The following lemma is the key observation.

▶ **Lemma 1.** *Let u be a level-d node, and let $\pi(u) = \langle u_1, \ldots, u_d = u\rangle$. Then there is a d-dimensional rectangle $\mathscr{R}(u) = \prod_{i=1}^{d} \delta_i$, where the endpoints of δ_i, for $i \in [1, d]$, are defined by the endpoints of I_{u_i} and $I_{p(u_i)}$, such that for any $x \in \mathbb{R}^d$, $u \in \mathscr{N}(x)$ if and only if $x \in \mathscr{R}(u)$. Given u_i's and $p(u_i)$'s, $\mathscr{R}(u)$ can be constructed in $O(1)$ time.*

Proof. Notice that $\mathscr{B}(x)$ is the hypercube of side length 2 and center x. Let $I_{u_i} = [\alpha^-_{u_i}, \alpha^+_{u_i}]$ for any u_i and $i \in [1, d]$. Recall that $u \in \mathscr{N}(x)$ if and only if for each $i \in [1, d]$,

$$I_{u_i} \subseteq [x_i - 1, x_i + 1] \text{ and } I_{p(u_i)} \not\subseteq [x_i - 1, x_i + 1], \quad (*)$$

Fix a value of i. From the construction of a range tree either $\alpha^-_{u_i} = \alpha^-_{p(u_i)}$ or $\alpha^+_{u_i} = \alpha^+_{p(u_i)}$. Without loss of generality, assume $\alpha^-_{u_i} = \alpha^-_{p(u_i)}$; the other case is symmetric. Then $(*)$ can be written as: $x_i \leq \alpha^-_{u_i} + 1$ and $\alpha^+_{u_i} - 1 \leq x_i < \alpha^+_{p(u_i)} - 1$. Therefore x_i has to satisfy three 1D linear constraints. The feasible region of these constraints is an interval δ_i and $x_i \in \delta_i$ (see also Figure 1). Hence, u is a canonical node of $\mathscr{B}(x)$ if and only if for all $i \in [1, d]$, $x_i \in \delta_i$. In other words, $x = (x_1, \ldots, x_d) \in \prod_{i=1}^{d} \delta_i := \mathscr{R}(u)$. The endpoints of δ_i are the endpoints of I_{u_i} or $I_{p(u_i)}$. In order to construct $\mathscr{R}(u)$, we only need the intervals I_{u_i} and $I_{p(u_i)}$ for each $i \in [1, d]$, so it can be constructed in $O(d) = O(1)$ time. ◀

In view of Lemma 1, we proceed as follows. We build a dynamic range tree $\mathscr{Z}$ on B. Furthermore, we augment the range tree $\mathscr{T}$ on A as follows. For each level-d node $u \in \mathscr{T}$, we compute and store $\mathscr{R}(u)$ and $\beta_u = |B_u|$. By construction, $|A_u| \geq 1$ for all u. We also store a pointer at u to the leftmost leaf of the subtree of $\mathscr{T}$ rooted at u, and we thread all the leaves

of a d-level tree so that for a node u, A_u can be reported in $O(|A_u|)$ time. Updating these pointers as $\mathscr{T}$ is updated is straightforward. Whenever a new node u of $\mathscr{T}$ is constructed, we query $\mathscr{Z}$ with $\mathscr{R}(u)$ to compute β_u. Finally, we store $\mathscr{C}$, the set of all active nodes of $\mathscr{T}$, in a red-black tree so that a node can be inserted or deleted in $O(\log n)$ time. The total size of the data structure is $O(n \log^{d-1} n)$, and it can be constructed in $O(n \log^d n)$ time.

Update and Enumerate. Updating A is straightforward. We update $\mathscr{T}$, query $\mathscr{Z}$ with $\mathscr{R}(u)$, for all newly created d-level nodes u in $\mathscr{T}$ to compute β_u, and update $\mathscr{C}$ to delete all active nodes that are no longer in $\mathscr{T}$ and to insert new active nodes. Since the amortized time to update $\mathscr{T}$ as a point is inserted or deleted is $O(\log^d n)$, the amortized update time of a point in A is $O(\log^{2d} n)$ – we spend $O(\log^d n)$ time to compute β_u for each of the $O(\log^d n)$ newly created nodes. If a point b is inserted (resp. deleted) in B, we update $\mathscr{Z}$ and query $\mathscr{T}$ with $\mathscr{B}(b)$. For all canonical nodes u in $\mathscr{N}(b)$, we increment (resp. decrement) b_u. If u becomes active (resp. inactive), we insert (resp. delete) u in $\mathscr{C}$ in $O(\log n)$ time. The amortized update time for b is $O(\log^{d+1} n)$.

Finally, to enumerate the pairs in join results, we traverse the active nodes $\mathscr{C}$ and for each $u \in \mathscr{C}$, we first query $\mathscr{Z}$ with $\mathscr{R}(u)$ to recover B_u. Recall that B_u is reported as a set of $O(\log^d n)$ canonical nodes of $\mathscr{Z}$ whose leaves contain the points of B_u. We simultaneously traverse the leaves of the subtree of $\mathscr{T}$ rooted at u to compute A_u and report $A_u \times B_u$. The traversals can be performed in $O(\log^d n)$ maximum delay. Putting everything together, we obtain:

▶ **Theorem 2.** *Let A, B be two sets of points in $\mathbb{R}^d$, where $d \geq 1$ is a constant, with $|A| + |B| = n$. A data structure of $\widetilde{O}(n)$ size can be built in $\widetilde{O}(n)$ time and updated in $\widetilde{O}(1)$ amortized time, while supporting $\widetilde{O}(1)$-delay enumeration of similarity join under ℓ_∞ metric.*

2.2 Similarity join under ℓ_1 metric

For $d \leq 2$ it is straightforward to reduce similarity join under ℓ_1 metric to ℓ_∞ metric. For $d = 1$, ℓ_1 metric is obviously equivalent to the ℓ_∞ metric. For $d = 2$, notice that the ℓ_1 ball is a diamond, while the ℓ_∞ ball is a square. Hence, given an instance of the similarity join under the ℓ_1 metric we can rotate $A \cup B$ by 45 degrees to create an equivalent instance of the similarity join problem under the ℓ_∞ metric.

Next, we focus on $d \geq 3$. The data structure we proposed in Section 2.1 for the ℓ_∞ norm can be straightforwardly extended to the *rectangle-containment* problem in which for each $b \in B$, $\mathscr{B}(b)$ is an arbitrary axis-aligned hyper-rectangle with center b, and the goal is to report all $(a, b) \in A \times B$ such that $a \in \mathscr{B}(b)$. Lemma 1 can be extended so that $\mathscr{R}(u)$ is a $2d$-dimensional rectangle. Overall, Theorem 2 remains the same assuming $\mathscr{B}(b)$ are hyper-rectangles (and not hypercubes).

Given an instance of similarity join under ℓ_1 metric in $\mathbb{R}^d$, we next show how to reduce it to 2^d $(d+1)$-dimensional rectangle-containment problems. As above, assume $r = 1$, so our goal is to report all pairs $a = (a_1, \ldots, a_d) \in A$, $b = (b_1, \ldots, b_d) \in B$ such that $\sum_{i=1}^d |a_i - b_i| \leq 1$.

Let $E = \{-1, +1\}^d$ be the set of all 2^d vectors in $\mathbb{R}^d$ with coordinates either 1 or -1. For each vector $e \in E$, we construct an instance of the rectangle-containment problem. For each $e = (e_1, \ldots, e_d) \in E$, we map each point $a = (a_1, \ldots, a_d) \in A$ to a point $\bar{a}_e = (a_1, \ldots, a_d, \sum_{i=1}^d e_i a_i) \in \mathbb{R}^{d+1}$. Let $\bar{A}_e = \{\bar{a}_e \mid a \in A\}$. For each point $b = (b_1, \ldots, b_d) \in B$, we construct the axis-align rectangle $\bar{b}_e = \prod_{i=1}^{d+1} b_e^{(i)}$ in $\mathbb{R}^{d+1}$, where $b_e^{(i)}$ is the interval $[b_i, \infty)$ if $e_i = 1$ and $(-\infty, b_i]$ if $e_i = -1$ for each $i = 1, \ldots, d$, and $b_e^{(d+1)} = (-\infty, 1 + \sum_{i=1}^d e_i b_i]$. Let $\bar{B}_e = \{\bar{b}_e \mid b \in B\}$. See Figure 2.

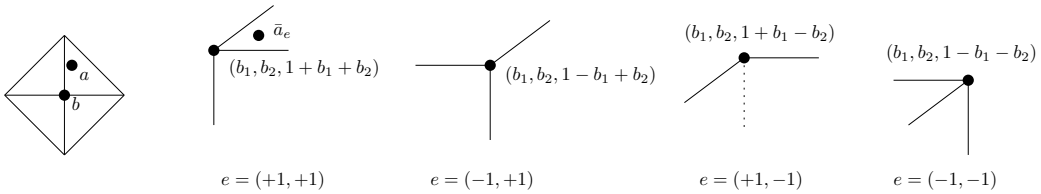


Figure 2 An illustration of mapping each b to rectangles.

For each $e \in E$, we construct the dynamic data structure for $\bar{A}_e, \bar{B}_e$. Whenever A or B is updated, we update all 2^d rectangle-containment data structures. A similarity join enumeration query on A, B is answered by enumerating containment pairs $(\bar{A}_e, \bar{B}_e)$ for $e \in E$. If a pair $(\bar{a}_e, \bar{b}_e)$ is reported, we report (a, b). The update time and delay are $\widetilde{O}(1)$. The correctness of the algorithm follows from the following lemma. Let $\mathrm{sgn}(x) = +1$ if $x \geq 0$ and -1 otherwise.

▶ **Lemma 3.** *Let $a = (a_1, \ldots, a_d) \in A$, $b = (b_1, \ldots, b_d) \in B$ be an arbitrary pair of points. Let $e^* = (e_1^*, \ldots, e_d^*)$ where $e_i^* = \mathrm{sgn}(a_i - b_i)$ for $1 \leq i \leq d$. Then $\bar{a}_e \notin \bar{b}_e$ for all $e \in E \setminus \{e^*\}$. Furthermore, $\bar{a}_{e^*} \in \bar{b}_{e^*}$ if and only if $\|a - b\|_1 \leq 1$.*

Proof. First, we note that for any $e \in E \setminus \{e^*\}$, there must exist some i such that $e_i \neq e_i^*$. Without loss of generality, assume $e_j = 1$ when $a_j < b_j$. By the definition of $\bar{a}_e, \bar{b}_e$, $a_j \notin [b_j, \infty)$, thus $\bar{a}_e \notin \bar{b}_e$. Next, we show that $\bar{a}_{e^*} \in \bar{b}_{e^*}$ if and only if $\|a - b\|_1 \leq 1$. On one hand, we assume $\bar{a}_{e^*} \in \bar{b}_{e^*}$. By definition, $\sum_{i=1}^d e_i^* a_i$ lies in the interval associated with $b_{e^*}^{d+1}$, i.e., $\sum_{i=1}^d e_i^* a_i \leq 1 + \sum_{i=1}^d e_i^* b_i$, or $\sum_{i=1}^d e_i^* (a_i - b_i) \leq 1$. Implied by the fact that $\|a - b\|_1 = \sum_{i=1}^d e_i^* (a_i - b_i)$, we have $\|a - b\|_1 \leq 1$. On the other hand, assume $\|a - b\|_1 \leq 1$. Similarly, we have $\|a - b\|_1 = \sum_{i=1}^d e_i^* (a_i - b_i) \leq 1 \Leftrightarrow \sum_{i=1}^d e_i^* a_i \leq 1 + \sum_{i=1}^d e_i^* b_i$, or $\sum_{i=1}^d e_i^* a_i \in (-\infty, 1 + \sum_{i=1}^d e_i^* b_i]$. Moreover, for any $i \in \{1, \ldots, d\}$, we have: (1) if $e_i^* = 1$, $a_i \geq b_i$, i.e., $a_i \in [b_i, \infty)$; (2) if $e_i^* = -1$, $a_i \leq b_i$, i.e., $a_i \in (-\infty, b_i]$. Hence, $\bar{a}_{e^*} \in \bar{b}_{e^*}$. ◀

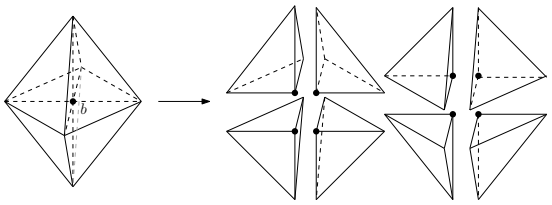


Figure 3 An illustration of ℓ_1 ball in $\mathbb{R}^3$. It is decomposed to $2^d = 8$ types of simplices.

▶ Remark. Roughly speaking, we partition the ℓ_1-ball centered at 0 into 2^d simplices $\Delta_1, \ldots, \Delta_{2^d}$ (see Figure 3) and build a separate data structure for each simplex Δ_i. Namely, let $\mathscr{B}_i = \{b + \Delta_i \mid b \in B\}$ and we report all pairs $(a, b) \in A \times B$ such that $a \in b + \Delta_i$. If $\|a - b\|_1 \leq 1$ then a lies in exactly one simplex $b \in \Delta_i$. We map each simplex to a rectangle in $\mathbb{R}^{d+1}$ and use the previous data structure.

Using Theorem 2, we obtain:

▶ **Theorem 4.** *Let A, B be two sets of points in $\mathbb{R}^d$, where $d \geq 1$ is a constant, with $|A| + |B| = n$. A data structure of $\widetilde{O}(n)$ size can be built in $\widetilde{O}(n)$ time and updated in $\widetilde{O}(1)$ amortized time, while supporting $\widetilde{O}(1)$-delay enumeration of similarity join under ℓ_1 metric.*

2.3 Similarity join under ℓ_2 metric

In this section, we consider the similarity join between two point sets A and B in $\mathbb{R}^d$ under the ℓ_2 metric.

Reduction to halfspace containment. We use the *lifting transformation* [23] to convert an instance of the similarity join problem under ℓ_2 metric to the halfspace-containment problem in $\mathbb{R}^{d+1}$. For any two points $a = (a_1, \ldots, a_d) \in A$ and $b = (b_1, \ldots, b_d) \in B$, $\|a - b\|_2 \leq 1$ if and only if $(a_1 - b_1)^2 + \ldots + (a_d - b_d)^2 \leq 1$, or a lies in the unit sphere centered at b. The above condition can be rewritten as

$$a_1^2 + b_1^2 + \cdots + a_d^2 + b_d^2 - 2a_1b_1 - \cdots - 2a_db_d - 1 \geq 0.$$

We map the point a to a point $a' = (a_1, \ldots, a_d, a_1^2 + \cdots + a_d^2)$ in $\mathbb{R}^{d+1}$ and the point b to a halfspace b' in $\mathbb{R}^{d+1}$ defined as

$$b' : -2b_1z_1 - \cdots - 2b_dz_d + z_{d+1} + b_1^2 + \cdots + b_d^2 - 1 \geq 0.$$

Note that $\|a - b\|_2 \leq 1$ if and only if $a' \in b'$. Set $A' = \{a' \mid a \in A\}$ and $B' = \{b' \mid b \in B\}$. Thus, in the following, we study the halfspace-containment problem, where given a set of points A' and a set of halfspaces B' we construct a dynamic data structure that reports all pairs $(a \in A', b \in B')$, such that a belongs in the halfspace b, with delay guarantee.

Partition tree. A partition tree on a set P of points in $\mathbb{R}^d$ [17, 37, 48] is a tree data structure formed by recursively partitioning a set into subsets. Each point is stored in exactly one leaf and each leaf usually contains a constant number of points. Each node u of the tree is associated with a simplex Δ_u and the subset $P_u = P \cap \Delta_u$; the subtree rooted at u is a partition tree of P_u. We assume that the simplices associated with the children of a node u are pairwise disjoint and lie inside Δ_u, as in [17]. In general, the degree of a node is allowed to be non-constant. Given a query simplex Δ, a partition tree finds a set of $O(n^{1-1/d})$ *canonical* nodes whose cells contain the points of $P \cap \Delta$. Roughly speaking, a node u is a canonical node for Δ if $\Delta_u \subset \Delta$ and $\Delta_{p(u)} \not\subseteq \Delta$. A simplex counting (resp. reporting) query can be answered in $O(n^{1-1/d})$ (resp. $O(n^{1-1/d} + k)$) time using a partition tree. Chan [17] proposed a randomized algorithm for constructing a linear size partition tree with constant degree, that runs in $O(n \log n)$ time and it has $O(n^{1-1/d})$ query time with high probability.

Data structure. For simplicity, with slight abuse of notation, let A be a set of points in $\mathbb{R}^d$ and B a set of halfspaces in $\mathbb{R}^d$ each lying below the hyperplane bounding it, and our goal is to build a dynamic data structure for halfspace-containment join on A, B. The overall structure of the data structure is the same as for rectangle containment described in Section 2.1, so we simply highlight the difference.

Instead of constructing a range tree, we construct a dynamic partition tree $\mathscr{T}_A$ for A so that the points of A lying in a halfspace can be represented as the union of $O(n^{1-1/d})$ canonical subsets. For a halfplane bounding a halfspace $b \in B$, let $\bar{b}$ denote its dual point in $\mathbb{R}^d$ (see [23] for the definition of duality transform). Note that a point a lies in b if and only if the dual point $\bar{b}$ lies in the halfspace lying below the hyperplane dual to a. Set $\bar{B} = \{\bar{b} \mid b \in B\}$. We construct a multi-level dynamic partition tree on $\bar{B}$, so that for a pair of simplices Δ_1 and Δ_2, it returns the number of halfspaces of B that satisfy the following two conditions: (i) $\Delta_1 \subseteq b$ and (ii) $\Delta_2 \cap \partial b \neq \emptyset$, where ∂b is the hyperplane boundary defined by the halfspace b. This data structure uses $O(n)$ space, can be constructed in $\widetilde{O}(n)$ time, and answers a query in $\widetilde{O}(n^{1-1/d})$ time.

For each node $u \in \mathscr{T}_A$, we issue a counting query to $\mathscr{T}_B$ and get the number of halfspaces in B that have u as a canonical node. Hence, $\mathscr{T}_A$ can be built in $\widetilde{O}(n^{2-1/d})$ time. For a node u, $\mu_A(u)$ can be computed in $O(1)$ time by storing A_u at each node $u \in \mathscr{T}_A$. Recall that $\mu_B(u)$ is the number of halfspaces b of B for which u is a canonical node, i.e., $\Delta_u \subseteq b$ and $\Delta_{p(u)} \cap \partial b \neq \emptyset$, where $p(u)$ is the parent of u. Using $\mathscr{T}_B$, $\mu_B(u)$ can be computed in $\widetilde{O}(n^{1-1/d})$ time.

Update and enumeration. The update procedure is the same that in Section 2.1, however the query time now on $\mathscr{T}_A$ or $\mathscr{T}_B$ is $\widetilde{O}(n^{1-\frac{1}{d}})$ so the amortized update time is $\widetilde{O}(n^{1-\frac{1}{d}})$. The enumeration query is also the same as in Section 2.1 but a reporting query in $\mathscr{T}_B$ takes $\widetilde{O}(n^{1-\frac{1}{d}}+k)$ time (and it has delay at most $\widetilde{O}(n^{1-\frac{1}{d}})$), so the overall delay is $\widetilde{O}(n^{1-\frac{1}{d}})$.

▶ **Theorem 5.** *Let A be a set of points and B be a set of half-spaces in $\mathbb{R}^d$ with $|A|+|B| = n$. A data structure of $\widetilde{O}(n)$ size can be built in $\widetilde{O}(n^{2-\frac{1}{d}})$ time and updated in $\widetilde{O}(n^{1-\frac{1}{d}})$ amortized time while supporting $\widetilde{O}(n^{1-\frac{1}{d}})$-delay enumeration of halfspace-containment query.*

Using Theorem 5 and the lifting transformation described at the beginning of this section we conclude with Corollary 6.

▶ **Corollary 6.** *Let A, B be two sets of points in $\mathbb{R}^d$, where $d \geq 1$ is a constant, with $|A| + |B| = n$. A data structure of $\widetilde{O}(n)$ size can be constructed in $\widetilde{O}(n^{2-\frac{1}{d+1}})$ time and updated in $\widetilde{O}(n^{1-\frac{1}{d+1}})$ amortized time, while supporting $\widetilde{O}(n^{1-\frac{1}{d+1}})$-delay enumeration of similarity join under the ℓ_2 metric.*

Lower bound. We show a lower bound for the similarity join in the pointer-machine model under the ℓ_2 metric based on the hardness of unit sphere reporting problem. Let P be a set of n points in $\mathbb{R}^d$ for $d > 3$. The unit-sphere reporting problem asks for a data structure on the points in P, such that given any unit-sphere b report all points of $P \cap b$. If the space is $\widetilde{O}(n)$, it is not possible to get a data structure for answering unit-sphere reporting queries in $\widetilde{O}(k+1)$ time in the pointer-machine model, where k is the output size for $d \geq 4$ [1].

For any instance of sphere reporting problem, we construct an instance of similarity join over two sets, with $A = \emptyset$, $B = P$, and $r = 1$. Given a query unit-sphere of center q, we insert point q in A, issue an enumeration query, and then remove q from A. All results enumerated (if any) are the results of the sphere reporting problem. If there exists a data structure for enumerating similarity join under ℓ_2 metric using $\widetilde{O}(n)$ space, with $\widetilde{O}(1)$ update time and $\widetilde{O}(1)$ delay, we would break the barrier.

▶ **Theorem 7.** *Let A, B be two sets of points in $\mathbb{R}^d$ for $d > 3$, with $|A| + |B| = n$. If using $\widetilde{O}(n)$ space, there is no data structure under the pointer-machine model that can be updated in $\widetilde{O}(1)$ time, while supporting $\widetilde{O}(1)$-delay enumeration of similarity join under the ℓ_2 metric.*

3 Approximate Enumeration

In this section we propose a dynamic data structure for answering approximate similarity-join queries under any ℓ_p metric. For simplicity, we use the ℓ_2 norm to illustrate the main idea and assume $\phi(a,b) = ||a-b||_2$. Recall that all pairs of $(a,b) \in A \times B$ with $\phi(a,b) \leq r$ must be reported, along with (potentially) some pairs of (a',b') with $\phi(a',b') \leq (1+\varepsilon)r$, but no pair (a,b) with $\phi(a,b) > (1+\varepsilon)r$ is reported.

We will start with the setting where the distance threshold r is not fixed and specified as part of a query, and then move to a simpler scenario where r is fixed.

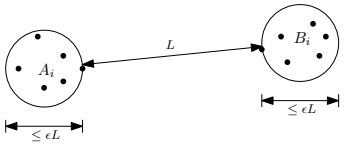


Figure 4 An example pair of ε-WSPD.

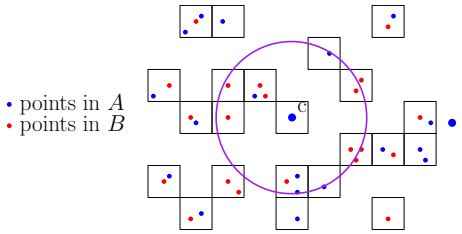


Figure 5 An example of active cell c in the grid.

3.1 Variable Similarity Threshold

We describe the data structure when r is part of the query. In this subsection we assume that the spread of $A \cup B$ is polynomially bounded, i.e., $sp(A \cup B) = \frac{\max_{p,q \in A \cup B} \phi(p,q)}{\min_{p \neq q \in A \cup B} \phi(p,q)} = n^{O(1)}$. We use a quad tree and well-separated pair decomposition (WSPD) for our data structure. We describe them briefly here and refer the reader to [28, 41] for details.

Quad tree and WSPD. A d-dimensional quad tree over a point set P is a tree data structure $\mathscr{T}$ in which each node u is associated with a hypercube $\square_u$ in $\mathbb{R}^d$, called a *cell*, and each internal node has 2^d children. The root is associated with a hypercube containing P. For a node u, let $P_u = P \cap \square_u$. A node u is a leaf if $|P_u| \leq 1$. The tree recursively subdivides the space into 2^d congruent hypercubes until a box contains at most one point from P. If $sp(P) = n^{O(1)}$, the height of $\mathscr{T}$ is $O(\log n)$.

Given two point sets $A, B \subset \mathbb{R}^d$, with $|A| + |B| = n$, and a parameter $0 < \varepsilon < \frac{1}{2}$, a family of pairs $\mathscr{W} = \{(A_1, B_1), (A_2, B_2), \cdots, (A_s, B_s)\}$ is an ε-WSPD if the following conditions hold: (1) for any $i \leq s$, $A_i \subseteq A$, $B_i \subseteq B$ (2) for each pair of points $(a, b) \in A \times B$, there exists a unique pair $(A_j, B_j) \in \mathscr{W}$ such that $a \in A_j$ and $b \in B_j$ (3) for any $i \leq s$, $\max\{\text{diam}(A_i), \text{diam}(B_i)\} \leq \varepsilon \cdot \phi(A_i, B_i)$, where $\text{diam}(X) = \max_{x,y \in X} \phi(x, y)$ and $\phi(X, Y) = \min_{x \in X, y \in Y} \phi(x, y)$ (see Figure 4). As shown in [28, 30] if $sp(A \cup B) = n^{O(1)}$, a quad tree T on $A \cup B$ can be used to construct, in time $O(n \log n + \varepsilon^{-d} n)$, a WSPD $\mathscr{W}$ of size $O(\varepsilon^{-d} n)$ such that each pair $(A_i, B_i) \in \mathscr{W}$ is associated with pair of cells $(\square_i, \boxplus_i)$ in $\mathscr{T}$ where $A_i = A \cap \square_i$ and $B_i = B \cap \boxplus_i$. It is also known that for each pair $(A_i, B_i) \in \mathscr{W}$ (i) $\square_i \cap \boxplus_i = \emptyset$, (ii) $\max\{\text{diam}(\square_i), \text{diam}(\boxplus_i)\} \leq \varepsilon \phi(\square_i, \boxplus_i)$, and each cell appears in $O(\varepsilon^{-d} \log n)$ cells (see Figure 4). We will use $\mathscr{W} = \{(\square_1, \boxplus_i), \ldots, (\square_s, \boxplus_s)\}$ to denote the WSPD, with A_i, B_i being implicitly defined from the cells. Using the techniques in [15, 26], the quad tree $\mathscr{T}$ and the WSPD $\mathscr{W}$ can be maintained under insertions and deletions of points in $\widetilde{O}(\varepsilon^{-d})$ time.

Data structure. We construct a quad tree $\mathscr{T}$ on $A \cup B$. For each node $u \in \mathscr{T}$, we store a pointer A_u (and B_u) to the leftmost leaf of subtree $\mathscr{T}_u$ that contains a point from A (and B). Furthermore, we store sorted lists L_A and L_B of the leaves that contain points from A and B, respectively. We use these pointers and lists to report points in $\square_u$ with $O(1)$ delay. Using $\mathscr{T}$, we can construct a WSPD $\mathscr{W} = \{(\square_1, \boxplus_1), \ldots, (\square_s, \boxplus_s)\}$, $s = O(\varepsilon^{-d})$. For each i, let $\Delta_i = \min_{p \in \square_i, q \in \boxplus_i} \phi(p, q)$. We store all pairs $(\square_i, \boxplus_i)$ in a red-black tree $\mathscr{Z}$ using Δ_i as the key. The data structure has $O(\varepsilon^{-d} n)$ size and $O(\varepsilon^{-d} n \log n)$ construction time.

Update. After inserting or deleting an input point, the quad tree $\mathscr{T}$ and W can be updated in $\widetilde{O}(\varepsilon^{-d})$ time, following the standard techniques in [15, 26]. As there are at most $\widetilde{O}(\varepsilon^{-d})$ pairs changed, we can update the tree $\mathscr{Z}$ in $\widetilde{O}(\varepsilon^{-d})$ time. Furthermore, we note that there are only $O(1)$ changes in the structure of quad tree $\mathscr{T}$ and the height of $\mathscr{T}$ is $O(\log n)$, so we can update all necessary pointers A_u, B_u and sorted lists L_A, L_B in $O(\log n)$ time.

Enumeration. Let r be the threshold parameter specified as part of a query. We traverse the tree $\mathscr{Z}$ in order and report pairs of cells until we reach a pair $(\square_j, \boxplus_j)$ with $\Delta_j > r$. For each pair $(\square_i, \boxplus_i)$ reported, we traverse we enumerate $(a, b) \in (A \cap \square_i) \times (B \cap \boxplus_i)$ using the stored pointers and the sorted lists L_A, L_B. The delay guarantee is $O(1)$.

Let $(a, b) \in A \times B$ be a pair with $\phi(a, b) \leq r$. Implied by the definition, there exists a unique pair $(A_i, B_i) \in \mathscr{W}$ such that $a \in A_i$ and $b \in B_i$. Notice that $\phi(\square_i, \boxplus_i) \leq \phi(a, b) \leq r$. Thus, all results of $A_i \times B_i$ will be reported, including (a, b). Next, let $(\square_i, \boxplus_i)$ be a pair that is reported by the enumeration procedure in $\mathscr{Z}$, with $\phi(\square_i, \boxplus_i) \leq r$. For any pair of points $x \in \square_i, y \in \boxplus_i$, we have $\phi(x, y) \leq \phi(\square_i, \boxplus_i) + \operatorname{diam}(\square_i) + \operatorname{diam}(\boxplus_i) \leq (1 + 2 \cdot \frac{\varepsilon}{2}) \cdot \phi(\square_i, \boxplus_i) \leq (1 + \varepsilon)r$, thus $\phi(a, b) \leq (1 + \varepsilon)r$ for any pair $(a, b) \in A_i \times B_i$.

▶ **Theorem 8.** *Let A, B be two sets of points in $\mathbb{R}^d$ for constant d, with $O(n^{O(1)})$ spread and $|A| + |B| = n$. A data structure of $O(\varepsilon^{-d} n)$ space can be built in $\widetilde{O}(\varepsilon^{-d} n)$ time and updated in $\widetilde{O}(\varepsilon^{-d})$ time, while supporting ε-approximate enumeration for similarity join under any ℓ_p metric with $O(1)$ delay, for any query similarity threshold r.*

3.2 Fixed distance threshold

Without loss of generality we assume that $r = 1$. We use a grid-based data structure for enumerating similarity join with fixed distance threshold r.

Data structure. Let $\mathscr{G}$ be an infinite uniform grid[1] in $\mathbb{R}^d$, where the size of each grid cell is $\frac{\varepsilon}{2\sqrt{d}}$ and the diameter is $\frac{\varepsilon}{2}$. For a pair of cells $c, c' \in$, define $\phi(c, c') = \min_{p \in c, q \in c'} \phi(p, q)$. Each grid cell $c \in \mathscr{G}$ is associated with (1) $A_c = A \cap c$; (2) $B_c = B \cap c$; (3) $m_c = \sum_{c' : \phi(c, c') \leq 1} |B_{c'}|$ as the number of points in B that lie in a cell c' within distance 1 from cell c. Let $\mathscr{C}_{NE} \subseteq \mathscr{G}$ be the set of all non-empty cells, $\mathscr{C}_{NE} = \{c \in \mathscr{G} \mid A_c \cup B_c \neq \emptyset\}$. A grid cell $c \in \mathscr{C}_{NE}$ is *active* if and only if $A_c \neq \emptyset$ and $m_c > 0$ (see Figure 5 for an example). Let $\mathscr{C} \subseteq \mathscr{C}_{NE}$ be the set of active grid cells (Figure 5). Notice that a grid cell is stored when there is at least one point from A or B lying inside it, so $|\mathscr{C}_{NE}| \leq n$. Finally, we build a balanced search tree on $\mathscr{C}$ so that whether a cell c is stored in $\mathscr{C}$ can be answered in $O(\log n)$ time. Similarly, we build another balanced search tree to store the set of non-empty cells $\mathscr{C}_{NE}$.

Update. Assume point $a \in A$ is inserted into cell $c \in \mathscr{G}$. If c is already in $\mathscr{C}_{NE}$, simply add a to A_c. Otherwise, we add c to $\mathscr{C}_{NE}$ with $A_c = \{a\}$ and update m_c as follows. We visit each cell $c' \in \mathscr{C}_{NE}$ with $\phi(c, c') \leq 1$, and add $|B_{c'}|$ to m_c. A point of A is deleted in a similar manner. Assume point $b \in B$ is inserted into cell $c \in \mathscr{G}$. If $c \notin \mathscr{C}_{NE}$, we add it to $\mathscr{C}_{NE}$. In any case, we first insert b into B_c and for every cell $c' \in \mathscr{C}_{NE}$ with $\phi(c, c') \leq 1$, we increase $m_{c'}$ by 1 and add c' to $\mathscr{C}$ if c' turns from inactive to active. A point from B is deleted in a similar manner. As there are $O(\varepsilon^{-d})$ cells within distance 1 from c, this procedure takes $\widetilde{O}(\varepsilon^{-d})$ time.

Enumeration. For each active cell $c \in \mathscr{C}$, we visit each cell $c' \in \mathscr{C}_{NE}$ within distance 1. If $B_{c'} \neq \emptyset$, we report all pairs of points in $A_c \times B_{c'}$. It is obvious that each pair of points is enumerated at most once. For an active cell c, there must exists a pair $(a \in A_c, b \in B_{c'})$ for some cell $c' \in \mathscr{C}_{NE}$ such that $\phi(a, b) \leq \phi(c, c') + \operatorname{diam}(c) + \operatorname{diam}(c') \leq 1 + \varepsilon$. So it takes at most $O(\varepsilon^{-d} \log n)$ time before finding at least one result for c; thus, the delay is $O(\varepsilon^{-d} \log n)$.

[1] When extending it to any ℓ_p norm, the size of each grid cell is $\varepsilon/(2d^{1/p})$ and the diameter is $\frac{\epsilon}{2}$.

Furthermore, consider every pair of points a, b with $\phi(a, b) \leq 1$. Assume $a \in c$ and $b \in c'$. By definition, c must be an active grid cell. Thus, (a, b) will definitely be enumerated in this procedure, thus guaranteeing the correctness of ε-enumeration.

▶ **Theorem 9.** *Let A, B be two sets of points in $\mathbb{R}^d$ for some constant d, with $|A| + |B| = n$. A data structure of $O(n)$ size can be constructed in $O(n\varepsilon^{-d} \log n)$ time and updated in $O(\varepsilon^{-d} \log n)$ time, while supporting ε-approximate enumeration of similarity join under any ℓ_p metric with $O(\varepsilon^{-d} \log n)$ delay.*

Note that if for each active cell $c \in \mathscr{C}$, we store the cells within distance 1 that contain at least a point from B, i.e., $\{c' \in C \mid \phi(c, c') \leq 1, B_c \neq \emptyset\}$, then the delay can be further reduced to $O(1)$ but the space becomes $O(\varepsilon^{-d} n)$.

4 Similarity Join in High Dimensions

So far, we have treated the dimension d as a constant. In this section we describe a data structure for approximate similarity join using the *locality sensitive hashing* (LSH) technique, so that the dependency on d is a small polynomial. For simplicity, we assume that r is fixed, however our results can be extended to the case in which r is part of the enumeration query.

For $\varepsilon > 0$, $0 < p_2 < p_1 \leq 1$, a family $\mathscr{H}$ of hash functions is $(r, (1+\varepsilon)r, p_1, p_2)$-sensitive, if for any uniformly chosen hash function $h \in H$ and any two points x, y:

- $\Pr[h(x) = h(y)] \geq p_1$ if $\phi(x, y) \leq r$;
- $\Pr[h(x) = h(y)] \leq p_2$ if $\phi(x, y) \geq (1+\varepsilon)r$.

The quality of $\mathscr{H}$ is measured by $\rho = \frac{\ln p_1}{\ln p_2} < 1$, which is upper bounded by a number that depends only on ε; and $\rho = \frac{1}{1+\varepsilon}$ for many common distance functions [27, 22, 29]. For ℓ_2 the best result is $\rho \leq \frac{1}{(1+\varepsilon)^2} + o(1)$ [9].

The essence of LSH is to hash "similar" points into the same buckets with high probability. A simple approach based on LSH is to (i) hash points into buckets; (ii) probe each bucket and check for each pair of points $(a, b) \in A \times B$ inside the same bucket whether $\phi(a, b) \leq r$; and (iii) report (a, b) if the inequalities holds. However, two challenges arise for enumeration. First, without any knowledge of false positive results inside each bucket, checking every pair of points could lead to a huge delay. Our key insight is that after checking a small number (to be determined later) of pairs of points in one bucket, we can safely skip the bucket since any pair of result missed in this bucket will be found in another one with high probability. Second, one pair of points may collide under multiple hash functions, so an additional step is necessary in the enumeration to remove duplicates. If we wish to keep the size of data structure to be near-linear and if we are not allowed to store the reported pairs (so that the size remains near linear), detecting duplicates requires care.

Since we do not define new hash functions, our results hold for any metric for which LSH works, in particular for Hamming, ℓ_2, ℓ_1 metrics.

Data structure. We fix an LSH family $\mathscr{H}$. Let ρ be its quality parameter. To ensure high-probability guarantee, we maintain $O(\log n)$ copies of the whole data structure below.

We randomly choose $\tau = O(n^\rho)$ hash functions. Each possible value in the range of hash functions defines a bucket. We maintain some extra statistics for all buckets. We choose a parameter $M = O(n^\rho)$. For a bucket $\square$, let $A_\square = A \cap \square$ and $B_\square = B \cap \square$. We choose two arbitrary subsets $\bar{A}_\square, \bar{B}_\square$ of $A_\square, B_\square$, respectively, of M points each. For each point $a \in \bar{A}_\square$, we maintain a counter $\beta_a = |\{b \in \bar{B}_\square \mid \phi(a, b) \leq 2(1+\varepsilon)r\}|$, i.e., the number of points in $\bar{B}_\square$ with distance at most $2(1+\varepsilon)r$ from a. We store $\bar{A}_\square$ in an increasing order of their

β values. If there exists some positive counter $\beta_a > 0$, we denote bucket $\square$ as *active* and store an arbitrary pair $(a, b) \in \bar{A}_\square \times \bar{B}_\square$ with $\phi(a, b) \leq 2(1+\varepsilon)r$ as its *representative* pair, denoted as $(a_\square, b_\square)$. Let $\mathscr{C}$ denote the set of active buckets.

Before diving into the details of update and enumeration, we give some intuition about active buckets. Given a set P of points and a distance threshold r, let $\overline{\mathscr{B}}(q, P, r) = \{p \in P \mid \phi(p, q) > r\}$. For any pair of points $(a, b) \in A \times B$ and a hashing bucket $\square$, we refer to $\square$ as the *proxy bucket* for (a, b) if (i) $a \in A_\square, b \in B_\square$; (ii) $|\bar{\mathscr{B}}(a, A_\square \cup B_\square, (1+\varepsilon)r)| \leq M$. A crucial property of proxy bucket is captured by Lemma 10. Moreover, it can be shown that with high probability each close pair of points has a proxy bucket in Lemma 11 (more details are given in the full version [4]). In this way, it is safe to skip a bucket after we have seen up to M^2 faraway pairs of points inside, since the close pairs of points in this bucket will be captured by other buckets. In this way, we only need to report pairs from active buckets.

▶ **Lemma 10.** *For any bucket $\square$, if there exist M points from $A_\square$ and $B_\square$ each, such that none of the M^2 pairs has its distance within $2(1+\varepsilon)r$, $\square$ is not a proxy bucket for any pair $(a, b) \in A_\square \times B_\square$ with $\phi(a, b) \leq r$.*

Proof of Lemma 10. Let A', B' be two sets of M points from $A_\square, B_\square$ respectively. We assume that all pairs of points in $A' \times B'$ have their distances larger than $2(1+\varepsilon)r$. Observe that $\square$ is not a proxy bucket for any pair $(a \in A', b \in B')$. It remains to show that $\square$ is not a proxy bucket for any pair $(a \in A_\square \setminus A', b \in B_\square)$. Assume $b \in B_\square \setminus B'$ (the case is similar if $b \in B'$). If $A' \subseteq \bar{\mathscr{B}}(a, A, (1+\varepsilon)r)$ or $B' \subseteq \bar{\mathscr{B}}(a, B, (1+\varepsilon)r)$, $\square$ is not a proxy bucket for (a, b). Otherwise, there must exist at least one point $a' \in A'$ as well as $b' \in B'$ such that $\phi(a, a') \leq (1+\varepsilon)r$ and $\phi(a, b') \leq (1+\varepsilon)r$, so $\phi(a', b') \leq \phi(a, a') + \phi(a, b') \leq 2(1+\varepsilon)r$. Thus, $(a', b') \in A' \times B'$ is a pair within distance $2(1+\varepsilon)r$, coming to a contradiction. ◀

▶ **Lemma 11** ([27, 28, 32])**.** *For $M = O(n^\rho)$, with probability $1 - 1/n$, every pair of points (a, b) with $\phi(a, b) \leq r$ has a proxy bucket.*

Update. Assume a point $a \in A$ is being inserted. We visit every bucket $\square$ into which a is hashed and insert a to $A_\square$. If $|\bar{A}_\square| \geq M$, we do nothing. Otherwise, we insert a to $\bar{A}_\square$ and compute its counter β_a. If $\beta_a > 0$ and $\square \notin \mathscr{C}$, we add $\square$ to $\mathscr{C}$ and store an arbitrary pair (a, b) for some $b \in \bar{B}_\square$ with $\phi(a, b) \leq 2(1+\varepsilon)r$, as the representative pair of $\square$. Notice that there always exists such a point b since $\beta_a > 0$.

Assume a point $a \in A$ is being deleted. We visit every bucket $\square$ into which a is hashed and delete a from $A_\square$. If $a \in \bar{A}_\square$, we delete it from $\bar{A}_\square$ and insert an arbitrary point (if any) from $A_\square \setminus \bar{A}_\square$ into $\bar{A}_\square$. If $a = a_\square$, i.e., a participates in the representative pair of $\square$, we find a new representative pair by considering an arbitrary point $a' \in \bar{A}_\square$ with $\beta_{a'} > 0$. If no such point exists, we remove $\square$ from $\mathscr{C}$.

The case when point $b \in B$ is inserted or deleted is similar but with slight differences. Assume a point $b \in B$ is being inserted. We visit every bucket $\square$ into which b is hashed and inserted b to $B_\square$. If $|\bar{B}_\square| \geq M$, we do nothing. Otherwise, we insert b to $\bar{B}_\square$ and increment counter β_a for every point $a \in \bar{A}_\square$ with $\phi(a, b) \leq 2(1+\epsilon)r$. Moreover, if $\square \notin \mathscr{C}$ and there exists some point $a \in \bar{A}_\square$ with $\beta_a > 0$ after update, say a', we store (a', b) as the representative pair of $\square$ and add $\square$ to $\mathscr{C}$.

Assume a point $b \in B$ is being deleted. We visit every bucket $\square$ into which b is hashed and delete b from $B_\square$. If $b \in \bar{B}_\square$, we delete it from $\bar{B}_\square$ and insert an arbitrary point (if any) from $B_\square \setminus \bar{B}_\square$ into $\bar{B}_\square$. Moreover, we need to update counter β_a for every point $a \in \bar{A}_\square$. If

$b = b_\square$, i.e., b participates in the representative pair of $\square$, we find a new representative pair by considering an arbitrary point $a \in \bar{A}_\square$ with $\beta_a > 0$. If no such pair exists, we remove $\square$ from $\mathscr{C}$.

After performing $n/2$ updates, we reconstruct the entire data structure from scratch.

Enumeration. Let $\mathscr{R}$ be the set of representative pairs. As mentioned, the high-level idea is to enumerate representative pairs from active buckets. More specifically, we start with an arbitrary representative pair $(a, b) \in \mathscr{R}$, and enumerate all pairs involving point a from $\mathscr{C}(a)$, where $\mathscr{C}(a) \subseteq \mathscr{C}$ is the set of active buckets containing a. Then, we remove a from each bucket $\square \in \mathscr{C}(a)$. If $a \in \bar{A}_\square$, we remove a from $\bar{A}_\square$; moreover, we add a point $a' \in \bar{A} - \bar{A}_\square$ to $\bar{A}_\square$ to ensure $|\bar{A}_\square| = M$ if $|A_\square| \geq M$ and compute $\beta_{a'}$. If $a = a_\square$, we compute a new representative pair $(a_\square, b_\square)$ of $\square$. If no such new representative pair exists, we just remove $\square$ from the set of active buckets; otherwise, we add the new pair $(a_\square, b_\square)$ to $\mathscr{R}$. We repeat this process until $\mathscr{R}$ becomes empty. The whole procedure is described in Algorithm 1.

Algorithm 1 ENUMERATE.

```
1  R ← {(a□, b□) : □ ∈ C};
2  while R ≠ ∅ do
3      (a, b) ← R;
4      C(a) ← {□ ∈ C : a ∈ A□};
5      REPORT(a, C(a));
6      foreach □ ∈ C(a) do
7          A□ ← A□ − {a};
8          if a ∈ Ā□ then
9              Ā□ ← Ā□ − {a};
10             if |A□| ≥ M then
11                 Pick arbitrary point a' ∈ A□ − Ā□;
12                 Ā□ ← Ā□ ∪ {a'};
13         if a = a□ then
14             R ← R − {(a□, b□)};
15             Recompute (a□, b□) for □;
16             if (a□, b□) = ∅ then
17                 C ← C − {□};
18             else
19                 R ← R ∪ {(a□, b□)};
```

In line 5 of Algorithm 1, we report all pairs including point a is described in Algorithm 2. For a bucket $\square \in \mathscr{C}(a)$, whenever we report a pair (a, b), we *mark* b with a. If b was already marked by some other point a' because (a', b) was reported, we overwrite a' with a. Let $X(\square, a) \subseteq B_\square$ be the set of points in $B_\square$ marked with a. We visit every bucket $\square \in \mathscr{C}(a)$ and check the distances between a and points in $B_\square \setminus X(\square, a)$. Each time a pair (a, b) with $\phi(a, b) \leq 2(1 + \varepsilon)r$ is found, we report it and mark b with a to ensure that we will not report (a, b) again. More specifically, we go over each active bucket $\square \in \mathscr{C}(a)$ into which b is also hashed, and put a marker on b with respect to a. Implied by line 3 of Algorithm 2, we only

consider points not marked by $X(\square, a)$, thus avoiding repeated enumeration.[2] Whenever more than M points from $B_\square$ have been checked without finding a pair with distance less than $2(1+\varepsilon)r$ (or if all points in $B_\square$ have been considered), we just skip this bucket.

Algorithm 2 REPORT$(a, \mathscr{C}(a))$.

```
1  foreach □ ∈ C(a) do
2  |  i ← 0;
3  |  foreach b ∈ B_□ − X(□, a) do
4  |  |  if φ(a, b) ≤ 2(1 + ε)r then
5  |  |  |  Report (a, b);
6  |  |  |  foreach □′ ∈ C(a) with b ∈ B_□′ do
7  |  |  |  |  X(□′, a) ← X(□′, a) ∪ {b};
8  |  |  else
9  |  |  |  i ← i + 1;
10 |  |  |  if i > M then break;
```

Correctness analysis. The report procedure guarantees that each pair of points is enumerated at most once. It remains to show that $(1+2\varepsilon)$-approximate enumeration is supported.

We show that $(1+2\varepsilon)$-approximate enumeration is supported with probability $1-1/n$. It can be easily checked that any pair of points farther than $2(1+\varepsilon)r$ will not be enumerated. Hence, it suffices to show that all pairs within distance r are enumerated with high probability. From Lemma 11, with high probability every pair within distance 1 has a proxy bucket. Let $\square$ be a proxy bucket for pair (a, b). Implied by Lemma 10, there exist no M points from $A_\square$ (for example $\bar{A}_\square$) and M points from $B_\square$ (for example $\bar{B}_\square$) such that all M^2 pairs have their distance larger than $2(1+\varepsilon)r$, so $\square$ is active. Moreover, from the definition of M and the proof of Lemma 10 (check [4] the complete proof) there exist no M points from $B_\square$ such that all of them have distance more than $2(1+\varepsilon)r$ from a, so Algorithm 2 will report (a, b).

Complexity analysis. Recall that $\tau, M = O(n^\rho)$. The data structure uses $O(dn + n\tau \log n)$ space since we only use linear space with respect to the points in each bucket. The update time is $\widetilde{O}(dM \cdot \tau)$ as there are $\widetilde{O}(\tau)$ buckets to be investigated and it takes $\widetilde{O}(dM)$ time to update the representative pair. After $n/2$ updates we re-build the data structure so the update time is amortized. The delay is $\widetilde{O}(dM \cdot \tau)$. In order to replace a with an arbitrary point $a' \in A_\square - \bar{A}_\square$ in line 8 of Algorithm 1 we need $O(dM)$ time and there are $\widetilde{O}(\tau)$ buckets that we need to visit. In total, this step takes $\widetilde{O}(dM\tau)$ time. In Algorithm 2, we spend $\widetilde{O}(dM\tau)$ time to report a pair of results and $\widetilde{O}(\tau)$ time to mark point b over all buckets.

We conclude with the following result:

▶ **Theorem 12.** *Let A and B be two sets of points in $\mathbb{R}^d$, where $|A| + |B| = n$ and let ε, r be positive parameters. For $\rho = \frac{1}{(1+\varepsilon)^2} + o(1)$, a data structure of $\widetilde{O}(dn + n^{1+\rho})$ size can be constructed in $\widetilde{O}(dn^{1+2\rho})$ time, and updated in $\widetilde{O}(dn^{2\rho})$ amortized time, while supporting $(1+2\varepsilon)$-approximate enumeration for similarity join under the ℓ_2 metric with $\widetilde{O}(dn^{2\rho})$ delay.*

[2] To avoid conflicts with the markers made by different enumeration queries, we can generate them randomly and delete old values by lazy updates [24, 38, 39] after finding new pairs to report.

▶ Remark. Alternatively, we can insert or delete points from $A \cup B$ without maintaining the sets $\bar{A}_\square, \bar{B}_\square$ for every bucket $\square$. In the enumeration phase, given a bucket $\square$, we can visit M arbitrary points from $A_\square$ and M arbitrary points from $B_\square$ and compute their pairwise distances. If there is no pair $(a \in A_\square, b \in B_\square)$ with $\phi(a,b) \leq 2(1+\varepsilon)r$, we just skip this bucket. Otherwise, we report the pair (a,b) and invoke Algorithm 2 for point a. In this case, the update time decreases to $\widetilde{O}(dn^\rho)$ but the delay will increases to $\widetilde{O}(dn^{3\rho})$.

The same result holds for Hamming and ℓ_1 metrics with $\rho = \frac{1}{1+\varepsilon}$. Using [32], for the Hamming metric and $\varepsilon > 1$ we can get $M = O(1)$. Skipping the details, we have:

▶ **Theorem 13.** *Let A and B be two sets of points in $\mathbb{H}^d$, where $|A| + B| = n$ and let ε, r be positive parameters. For $\rho = \frac{1}{1+\varepsilon}$, a data structure of $\widetilde{O}(dn + n^{1+\rho})$ size can be built in $\widetilde{O}(dn^{1+\rho})$ time, and updated in $\widetilde{O}(dn^\rho)$ amortized time, while supporting $(3+2\varepsilon)$-approximate enumeration for similarity join under the Hamming metric with $\widetilde{O}(dn^\rho)$ delay.*

In the full version [4], we show that our results can be extended to the case where r is part of the enumeration procedure, and we also prove a lower bound relating similarity join to the approximate nearest neighbor query.

5 Conclusion

In this paper, we presented dynamic data structures for enumerating similarity join queries with delay guarantees. We present several efficient data structures for dynamic enumeration of similarity joins in constant or higher dimensions over various metrics.

Note that our data structures provide worst-case delay guarantee for arbitrary input data and arbitrary updates. In practice, most real-world update sequences are "nice", nowhere near these worst-case scenarios; and input points from two sets might be dependent, or follow certain parameterized distributions. A more fine-grained analysis on the intrinsic difficulty of update sequences in dynamic enumeration of similarity joins is quite interesting but still open. Similar instance-dependent analysis has been considered in [46].

Another interesting direction is to investigate other variants of similarity join queries under more general metrics, such as doubling metric space, and more complicated distance functions, such as the cosine distance.

References

1 Peyman Afshani. Improved pointer machine and I/O lower bounds for simplex range reporting and related problems. In *Proc. 28th Annual Symp. Comput. Geom.*, pages 339–346, 2012.

2 P. K. Agarwal. Simplex range searching and its variants: A review. In M. Loebl, J. Nešetřil, and R. Thomas, editors, *A Journey Through Discrete Mathematics*, pages 1–30. Springer, 2017.

3 P. K. Agarwal and J. Erickson. Geometric range searching and its relatives. In B. Chazelle, J. E. Goodman, and R. Pollack, editors, *Contemporary Mathematics*, volume 223, pages 1–56. American Math. Society, 1999.

4 Pankaj K. Agarwal, Xiao Hu, Stavros Sintos, and Jun Yang. Dynamic enumeration of similarity joins. `arXiv:2105.01818`.

5 Pankaj K Agarwal, Junyi Xie, Jun Yang, and Hai Yu. Monitoring continuous band-join queries over dynamic data. In *Int. Symp. Algorithms Comput.*, pages 349–359. Springer, 2005.

6 Pankaj K Agarwal, Junyi Xie, Jun Yang, and Hai Yu. Scalable continuous query processing by tracking hotspots. In *Proc. 32nd Int. Conf. Very Large Databases*, pages 31–42, 2006.

7 Dror Aiger, Haim Kaplan, and Micha Sharir. Reporting neighbors in high-dimensional euclidean space. *SIAM J. Comput.*, 43(4):1363–1395, 2014.

8 A. Al-Badarneh, A. Al-Abdi, M. Sana'a, and H. Najadat. Survey of similarity join algorithms based on mapreduce. *MATTER: International Journal of Science and Technology*, 2016.

9 Alexandr Andoni and Piotr Indyk. Near-optimal hashing algorithms for approximate nearest neighbor in high dimensions. In *Proc. 47th Annual IEEE Symp. Foundations Comput. Sci.*, pages 459–468, 2006.

10 N. Augsten and M. Böhlen. Similarity joins in relational database systems. *Synthesis Lectures on Data Management*, 5(5):1–124, 2013.

11 Guillaume Bagan, Arnaud Durand, and Etienne Grandjean. On acyclic conjunctive queries and constant delay enumeration. In *Proc. 21th Int. Workshop Computer Science Logic*, pages 208–222, 2007.

12 J. Bentley. Decomposable searching problems. Technical report, CMU, 1978.

13 J. Bentley and J. Friedman. Data structures for range searching. *ACM Comput. Surv.*, 11(4):397–409, 1979.

14 Christoph Berkholz, Jens Keppeler, and Nicole Schweikardt. Answering conjunctive queries under updates. In *Proc. 36th ACM SIGMOD-SIGACT-SIGAI Symp. Principles Database Systems*, pages 303–318, 2017.

15 P. Callahan. *Dealing with Higher dimensions: The Well-Separated Pair Decomposition and Its Applications.* PhD thesis, Johns Hopkins University, 1995.

16 N. Carmeli and M. Kröll. Enumeration complexity of conjunctive queries with functional dependencies. *Theory of Computing Systems*, pages 1–33, 2019.

17 T. Chan. Optimal partition trees. *Discrete & Comput. Geom.*, 47(4):661–690, 2012.

18 Sirish Chandrasekaran and Michael J Franklin. Streaming queries over streaming data. In *Proc. 28th Int. Conf. Very Large Databases*, pages 203–214, 2002.

19 S. Chaudhuri, V. Ganti, and R. Kaushik. A primitive operator for similarity joins in data cleaning. In *Proc. 22nd Int. Conf. Data Eng.*, pages 5–5, 2006.

20 B. Chazelle and B. Rosenberg. Simplex range reporting on a pointer machine. *Comput. Geom.: Theory Apps.*, 5(5):237–247, 1996.

21 Jianjun Chen, David J DeWitt, Feng Tian, and Yuan Wang. Niagaracq: A scalable continuous query system for internet databases. In *Proc. 19th ACM SIGMOD Int. Conf. Management Data*, pages 379–390, 2000.

22 Mayur Datar, Nicole Immorlica, Piotr Indyk, and Vahab S Mirrokni. Locality-sensitive hashing scheme based on p-stable distributions. In *Proc. 20th Annual Symp. Comput. Geom.*, pages 253–262, 2004.

23 M. De Berg, M. Van Kreveld, M. Overmars, and O. C. Schwarzkopf. *Computational Geometry: Algorithms and Applications.* Springer, 3rd edition, 2008.

24 J. Erickson. Static-to-dynamic transformations. `http://jeffe.cs.illinois.edu/teaching/datastructures/notes/01-statictodynamic.pdf`.

25 Françoise Fabret, H Arno Jacobsen, François Llirbat, João Pereira, Kenneth A Ross, and Dennis Shasha. Filtering algorithms and implementation for very fast publish/subscribe systems. In *Proc. 20th ACM SIGMOD Int. Conf. Management Data*, pages 115–126, 2001.

26 John Fischer and Sariel Har-Peled. Dynamic well-separated pair decomposition made easy. In *Proc. 17th Canadian Conf. Comput. Geom.*, pages 235–238, 2005.

27 Aristides Gionis, Piotr Indyk, Rajeev Motwani, et al. Similarity search in high dimensions via hashing. In *Proc. 25nd Int. Conf. Very Large Databases*, volume 99 (6), pages 518–529, 1999.

28 S. Har-Peled. *Geometric Approximation Algorithms.* Number 173 in Mathematical Surveys and Monographs. American Math. Society, 2011.

29 Sariel Har-Peled, Piotr Indyk, and Rajeev Motwani. Approximate nearest neighbor: Towards removing the curse of dimensionality. *Theory of Computing*, 8(1):321–350, 2012.

30 Sariel Har-Peled and Manor Mendel. Fast construction of nets in low-dimensional metrics and their applications. *SIAM J. Comput.*, 35(5):1148–1184, 2006.

31 Muhammad Idris, Martín Ugarte, and Stijn Vansummeren. The dynamic yannakakis algorithm: Compact and efficient query processing under updates. In *Proc. 36th ACM SIGMOD Int. Conf. Management Data*, pages 1259–1274, 2017.

32 Piotr Indyk and Rajeev Motwani. Approximate nearest neighbors: towards removing the curse of dimensionality. In *Proc. 30th Annual ACM Symp. Theory of Comput.*, pages 604–613, 1998.

33 Edwin H Jacox and Hanan Samet. Metric space similarity joins. *ACM Trans. Database Systems*, 33(2):1–38, 2008.

34 Ahmet Kara, Hung Q Ngo, Milos Nikolic, Dan Olteanu, and Haozhe Zhang. Counting triangles under updates in worst-case optimal time. In *Proc. 22nd Int. Conf. Database Theory*, pages 4:1–4:18, 2019.

35 Ahmet Kara, Milos Nikolic, Dan Olteanu, and Haozhe Zhang. Trade-offs in static and dynamic evaluation of hierarchical queries. In *Proc. 39th ACM SIGMOD-SIGACT-SIGAI Symp. Principles Database Systems*, pages 375–392, 2020.

36 Hans-Peter Lenhof and Michiel Smid. Sequential and parallel algorithms for the k closest pairs problem. *Int. J. Comput. Geom. & Appl.*, 5(03):273–288, 1995.

37 J. Matoušek. Efficient partition trees. *Discrete Comput. Geom.*, 8(3):315–334, 1992.

38 M. Overmars and J. van Leeuwen. Worst-case optimal insertion and deletion methods for decomposable searching problems. *Inf. Process. Lett.*, 12(4):168–173, 1981.

39 Mark H Overmars. *The design of dynamic data structures*, volume 156. Springer Science & Business Media, 1987.

40 Rodrigo Paredes and Nora Reyes. Solving similarity joins and range queries in metric spaces with the list of twin clusters. *J. Discrete Algorithms*, 7(1):18–35, 2009.

41 H. Samet. Spatial data structures: Quadtree, octrees and other hierarchical methods, 1989.

42 Luc Segoufin. Enumerating with constant delay the answers to a query. In *Proc. 16th Int. Conf. Database Theory*, pages 10–20, 2013.

43 Y. Silva, W. Aref, and M. Ali. The similarity join database operator. In *Proc. 26th Int. Conf. Data Engi.*, pages 892–903, 2010.

44 Yasin N Silva, Jason Reed, Kyle Brown, Adelbert Wadsworth, and Chuitian Rong. An experimental survey of mapreduce-based similarity joins. In *Int. Conf. Similarity Search Appl.*, pages 181–195. Springer, 2016.

45 J. Wang, G. Li, and J. Feng. Can we beat the prefix filtering? an adaptive framework for similarity join and search. In *Proc. 31th ACM SIGMOD Int. Conf. Management Data*, pages 85–96, 2012.

46 Q. Wang and K. Yi. Maintaining acyclic foreign-key joins under updates. In *Proc. 39th ACM SIGMOD Int. Conf. Management Data*, pages 1225–1239, 2020.

47 D. E. Willard. Applications of range query theory to relational data base join and selection operations. *J. Comput. Syst. Sci.*, 52(1):157–169, 1996.

48 Dan E Willard. Polygon retrieval. *SIAM J. Comput.*, 11(1):149–165, 1982.

49 Kun-Lung Wu, Shyh-Kwei Chen, and Philip S Yu. Interval query indexing for efficient stream processing. In *Proc. 30th ACM Int. Conf. Inform. Knowledge Management*, pages 88–97, 2004.

Faster Algorithms for Bounded Tree Edit Distance

Shyan Akmal
MIT, EECS and CSAIL, Cambridge, MA, USA

Ce Jin
MIT, EECS and CSAIL, Cambridge, MA, USA

Abstract

Tree edit distance is a well-studied measure of dissimilarity between rooted trees with node labels. It can be computed in $O(n^3)$ time [Demaine, Mozes, Rossman, and Weimann, ICALP 2007], and fine-grained hardness results suggest that the weighted version of this problem cannot be solved in truly subcubic time unless the APSP conjecture is false [Bringmann, Gawrychowski, Mozes, and Weimann, SODA 2018].

We consider the *unweighted* version of tree edit distance, where every insertion, deletion, or relabeling operation has unit cost. Given a parameter k as an upper bound on the distance, the previous fastest algorithm for this problem runs in $O(nk^3)$ time [Touzet, CPM 2005], which improves upon the cubic-time algorithm for $k \ll n^{2/3}$. In this paper, we give a faster algorithm taking $O(nk^2 \log n)$ time, improving both of the previous results for almost the full range of $\log n \ll k \ll n/\sqrt{\log n}$.

2012 ACM Subject Classification Theory of computation → Pattern matching

Keywords and phrases tree edit distance, edit distance, dynamic programming

Digital Object Identifier 10.4230/LIPIcs.ICALP.2021.12

Category Track A: Algorithms, Complexity and Games

Funding *Shyan Akmal*: Supported by NSF Grant CCF-1909429.
Ce Jin: Supported by an MIT Akamai Presidential Fellowship.

Acknowledgements We thank Virginia Vassilevska Williams for several helpful discussions.

1 Introduction

Many tasks involve measuring the similarity between two sets of data. When the data is naturally represented as a string of characters, one of the most popular and well-studied ways of measuring similarity is via the (string) edit distance, defined to be the minimum number of characters that must be deleted, inserted, and substituted to turn one string into the other. Although edit distance is a fundamental problem in computer science and has been employed to great effect in many other areas, it can be less useful for applications where we are interested in comparing data that is not just linearly ordered, but has some hierarchical organization. When the data admits a tree structure, a natural measure of similarity is the *tree edit distance*, first introduced by Tai [34] as a generalization of the string edit distance problem [38]. Computing this metric has a wide variety of applications in a diverse array of fields including computational biology [22, 32, 23, 39], structured data analysis [14, 16, 21], and image processing [7, 26, 25, 31].

Given two *rooted ordered* trees with node labels, the tree edit distance is the minimum number of node deletions, insertions, and relabelings needed to turn one tree into the other. When we delete a node, its children become children of the parent of the deleted node. Beyond this widely studied definition, there are many other variants of the tree edit distance problem, including those defined for unrooted trees or unordered trees, or parameterized by the depth or the number of leaves, which we do not consider in this paper. We refer interested readers to the survey by Bille [8] for a comprehensive review.

We now recount the development of exact algorithms for tree edit distance. In 1979, Tai [34] gave the first algorithm that computes the tree edit distance between two node-labeled rooted trees on n nodes in $O(n^6)$ time. The time complexity was improved to $O(n^4)$ by Zhang and Shasha [40] using a dynamic programming approach. Later, Klein [24] applied the heavy-light decomposition technique to obtain an $O(n^3 \log n)$ time algorithm. Finally, Demaine, Mozes, Rossman, and Weimann [17] improved the running time by a log-factor to $O(n^3)$, and further showed that this running time is optimal among a certain class of dynamic programming algorithms termed *decomposition strategy algorithms* by Dulucq and Touzet [19, 20]. When the two input trees have different sizes $m \leq n$, their algorithm runs in $O\left(nm^2(1 + \log \frac{n}{m})\right)$ time.

All algorithms mentioned above actually compute tree edit distance in the general *weighted* setting where the cost of deleting, inserting, or relabeling is a function of the labels (so that deleting nodes with certain labels might be cheaper than deleting other nodes with different labels). In this setting, Bringmann, Gawrychowski, Mozes, and Weimann [13] showed conditional hardness results for the tree edit distance problem: a truly subcubic time algorithm for this problem would imply a truly subcubic time algorithm for the All-Pairs Shortest Paths (APSP) problem (assuming alphabet of size $\Theta(n)$), and an $O(n^{k(1-\varepsilon)})$ time algorithm for the Max-weight k-clique problem (assuming a sufficiently large constant-size alphabet). However, the instances produced by their fine-grained reduction have non-unit edit costs, and it is not clear yet how to prove a conditional hardness result for the *unweighted* tree edit distance problem with unit edit costs. In contrast, the quadratic-time fine-grained lower bound for the *string* edit distance problem (based on the Strong Exponential Time Hypothesis) holds for unit-cost operations [6, 1].

Therefore, it is natural to consider the unweighted unit-cost setting, where every elementary operation has cost 1, independent of the labels. In this case, the distance between two trees of sizes n and m cannot be larger than $n + m$, and is arguably even smaller in practical scenarios. In 2005, Touzet [35, 36] gave an algorithm in this context that computes the unweighted tree edit distance in $O(nk^3)$ time, assuming the distance is at most k. When $k = \Theta(n)$, Touzet's algorithm has the same performance as the $O(n^4)$ time algorithm by Zhang and Shasha [40]. However, the running time significantly improves if the upper bound k is much smaller than n. We remark that similar progress was shown earlier for the string edit distance problem: although the best known running time for the general case is $O(n^2/\log^2 n)$ [29, 9], when the distance is at most k, Ukkonen [37] gave an $O(nk)$ time algorithm, which was later improved to $\widetilde{O}(n + k^2)$ time[1] by Myers [30], Landau and Vishkin [28] using suffix trees.

Although we focus on exact algorithms in this work, approximation algorithms for the tree edit distance problem have also been studied [2, 11]. Boroujeni, Ghodsi, Hajiaghayi, and Seddighin [11] showed an algorithm that computes a $(1 + \varepsilon)$-approximation of the tree edit distance in $\widetilde{O}(\varepsilon^{-3}n^2)$ time. If an upper bound k on the distance is known, the running time can be improved to $\widetilde{O}(\varepsilon^{-3}nk)$. For the easier problem of approximating string edit distance, there is a longer line of research [5, 3, 10, 15, 12, 27] culminating in a near-linear time constant-factor approximation algorithm [4].

[1] In this paper, $\widetilde{O}(f)$ stands for $f \cdot (\log f)^{O(1)}$.

1.1 Our contribution

We present a faster algorithm for exactly computing the unweighted tree edit distance (where every elementary operation has unit cost), with a parameter $k \leq O(n)$ given as an upper bound on the distance.

▶ **Theorem 1.** *Given two node-labeled rooted trees T_1, T_2 each of size at most n, we can compute the unweighted tree edit distance between T_1 and T_2 exactly in $O(nk^2 \log n)$ time, assuming the distance is at most k.*

When the distance parameter k is constant our algorithm runs in quasilinear time, and as k reaches its upper bound $O(n)$ we recover the $O(n^3 \log n)$ time algorithm by Klein [24]. Our algorithm outperforms the $O(n^3)$ time algorithms of Demaine et al. [17] when $k = o(n/\sqrt{\log n})$. As mentioned earlier, the previous best algorithm for bounded tree edit distance by Touzet [36] takes $O(nk^3)$ time. The time complexity of our algorithm improves upon this prior work whenever $k = \omega(\log n)$.

1.2 High-level Overview

Touzet's $O(nk^3)$-time algorithm is based on Zhang and Shasha's $O(n^4)$-time dynamic programming algorithm [40]. The improvement was achieved by pruning unuseful DP states, and only considering $O(nk^3)$ many states instead of $O(n^4)$. This pruning technique was inspired by an idea used in the previous $O(nk)$-time algorithm for string edit distance [37]: for input strings whose edit distance is at most k, when building the dynamic programming table for computing the edit distance, it suffices to only compute entries of the table corresponding to prefixes whose lengths differ by at most k. Touzet's improvement for tree edit distance employs a similar technique and relies on measuring the "distance" between two DP states with respect to the preorder tree traversal, which is compatible with the DP transitions of Zhang and Shasha.

We modify Klein's $O(n^3 \log n)$ time algorithm by further reducing the number of useful states, similar in spirit to the algorithm by Touzet [36]. The main difficulty in adapting this idea is that unlike the algorithm of Zhang and Sasha, Klein's DP algorithm does not follow the same preorder traversal of the nodes. Hence we need completely new arguments to bound the number of useful DP states. Beyond considering the sizes of the subproblems generated, our proofs examine how various subforests are generated by different transition rules and employ some combinatorial arguments about how the subgraphs of deleted nodes can be structured when the edit distance is known to be bounded.

1.3 Paper Organization

In Section 2 we formally define the tree edit distance problem and introduce the notation used throughout the rest of the paper. Next, in Section 3, we review Klein's algorithm [24] which our algorithm builds off of. Then, in Section 4, we present our improved algorithm. Finally, we conclude by mentioning several open questions relevant to our work in Section 5.

2 Preliminaries

In this paper, we consider rooted trees that are *ordered*, meaning that the order between siblings is significant. We also consider *forests* consisting of disjoint rooted trees, where the order between these trees is also significant. It is convenient to treat the tree roots of a forest as the children of a virtual root node. Let $\mathsf{par}(v)$ denote the parent node of v, or the virtual root node if v is a tree root in the forest.

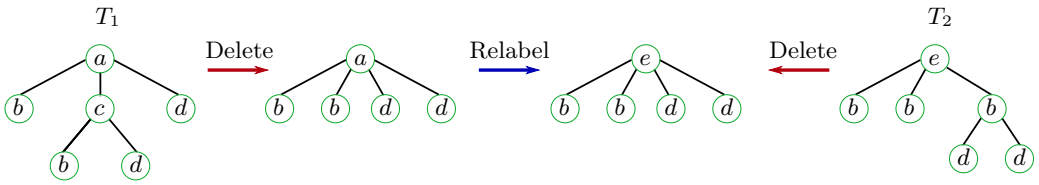


Figure 1 To turn T_1 and T_2 into the same tree with a minimum number of operations, we can delete a node from each and relabel a node in T_1. So in this example $\mathsf{ed}(T_1, T_2) = 3$.

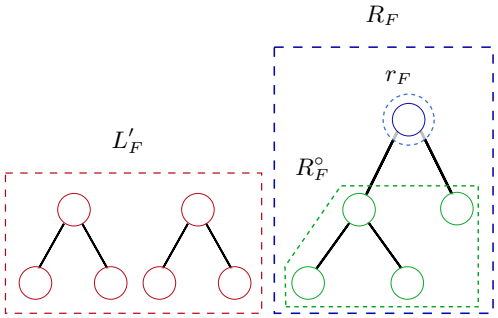


Figure 2 The example forest F above is partitioned into L'_F, r_F, and R°_F.

We define the *node removal operation* in the following natural way: after removing a node v from the forest F, the children of v become children of $\mathsf{par}(v)$, preserving the same relative order. We use $F - v$ to denote the forest obtained by removing v from F.

We now formally define the *tree edit distance* as a metric on ordered rooted trees with node labels.

▶ **Definition 2** ((Unweighted) Tree Edit Distance). *Let T_1 and T_2 be two ordered rooted trees whose nodes are labeled with symbols from some alphabet Σ. There are two types of allowed operations:*

- *Relabeling: change the label of a node from one symbol in Σ to another.*
- *Deletion: remove a node.*

Then the tree edit distance between T_1 and T_2, denoted by $\mathsf{ed}(T_1, T_2)$, is the minimum number of operations that must be performed on T_1 and T_2 to obtain two identical forests.

Figure 1 provides an example of these operations in action.

▶ Remark 3. An alternative definition of tree edit distance is the minimum number of insertions, deletions, and relabeling needed to turn one tree into the other. It is easy to see that these two definitions are equivalent.

Since the operations of relabeling and deletion also apply to labeled forests, the above definition naturally extends to measure the edit distance between two *forests* F_1 and F_2, and for the rest of the paper we write $\mathsf{ed}(F_1, F_2)$ to denote this edit distance as well.

Given a forest F, we write L_F (or R_F) to denote the leftmost (or rightmost) tree in F, and write ℓ_F (or r_F) to denote the root of L_F (or R_F). For convenience, let L'_F denote $F - R_F$, and let R°_F denote $R_F - r_F$ (similarly, $R'_F = F - L_F$ and $L^\circ_F = L_F - \ell_F$). Hence, the nodes of a nonempty forest F can be partitioned into three parts: L'_F, r_F, and R°_F (an example is given in Figure 2). Finally, $\mathsf{size}(F)$ or $|F|$ denote the number of nodes in F (where F can also be any subset of nodes).

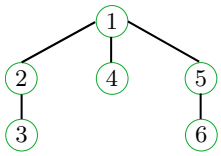


Figure 3 The subforests of this rooted tree are: $\{1,2,3,4,5,6\}, \{2,3,4,5,6\}, \{3,4,5,6\}, \{4,5,6\}, \{5,6\}, \{6\}, \emptyset, \{2,3,4,6\}, \{3,4,6\}, \{4,6\}, \{2,3,4\}, \{3,4\}, \{4\}, \{2,3\}, \{3\}$. For example, the subforest $\{3,4,6\}$ can be obtained by first removing the leftmost root 1, then removing the rightmost root 5, and finally removing the leftmost root 2.

▶ **Definition 4** (Subforest). *Given a rooted tree T, we say F is a* subforest *of T if we can obtain F from T by repeatedly deleting the leftmost or rightmost root.*

An example illustrating the definition of subforests is given in Figure 3.

▶ **Proposition 5.** *A rooted tree T of n nodes has at most $O(n^2)$ subforests.*

Proof. Although a subforest may result from interleaving operations of removing the leftmost root and removing the rightmost root, it is not hard to see that every such subforest F can also be obtained from T by first removing the leftmost root a times, and then removing the rightmost root b times, for some nonnegative integer a, b with $a+b \le n$. Specifically, let u be the node in F with the smallest index $\mathsf{pre}(u)$ in the *preorder traversal* of T ($1 \le \mathsf{pre}(u) \le n$), and we can set $a = \mathsf{pre}(u) - 1$ and $b = n - a - \mathsf{size}(F)$. The claim then follows from the number of choices of (a, b). ◀

For a subforest F of T, define $\mathsf{LCA}_T(F)$ as the lowest common ancestor in T of all nodes in F. When the identity T is clear from context, we may write $\mathsf{LCA}(F)$ and leave the underlying tree implicit. Observe that $\mathsf{LCA}(F)$ is in F precisely when F is a subtree of T.

Throughout, we use T_1, T_2 to denote the input trees (or T if we do not specify which one of the two) we want to compute the edit distance between.

3 Review of Klein's Algorithm

We briefly review Klein's algorithm [24] in the context of computing the unweighted tree edit distance $\mathsf{ed}(T_1, T_2)$ (see [17, 8, 18] for other overviews of this algorithm).

The algorithm uses dynamic programming (DP) over pairs (F_1, F_2), where F_1, F_2 are subforests of T_1, T_2, respectively. Let the node relabeling cost $\delta(x, y) = 1$ if nodes x, y have different labels, and $\delta(x, y) = 0$ otherwise. Then $\mathsf{ed}(F_1, F_2)$ can be computed recursively as follows [40]:

- The base case is where either of F_1, F_2 is empty (denoted as $\emptyset$), and we have

$$\mathsf{ed}(F_1, \emptyset) = \mathsf{size}(F_1), \mathsf{ed}(\emptyset, F_2) = \mathsf{size}(F_2). \tag{1}$$

- When both F_1, F_2 are nonempty, if $\mathsf{size}(L_{F_1}) > \mathsf{size}(R_{F_1})$, then we recurse with

$$\mathsf{ed}(F_1, F_2) = \min \begin{cases} \mathsf{ed}(F_1 - r_{F_1}, F_2) + 1 \\ \mathsf{ed}(F_1, F_2 - r_{F_2}) + 1 \\ \mathsf{ed}(R^\circ_{F_1}, R^\circ_{F_2}) + \mathsf{ed}(L'_{F_1}, L'_{F_2}) + \delta(r_{F_1}, r_{F_2}). \end{cases} \tag{2}$$

- Otherwise, $\mathsf{size}(L_{F_1}) \leq \mathsf{size}(R_{F_1})$, and we recurse with

$$\mathsf{ed}(F_1, F_2) = \min \begin{cases} \mathsf{ed}(F_1 - \ell_{F_1}, F_2) + 1 \\ \mathsf{ed}(F_1, F_2 - \ell_{F_2}) + 1 \\ \mathsf{ed}(L^\circ_{F_1}, L^\circ_{F_2}) + \mathsf{ed}(R'_{F_1}, R'_{F_2}) + \delta(\ell_{F_1}, \ell_{F_2}). \end{cases} \tag{3}$$

Taking Equation (2) as an example, the recursion considers three options concerning the rightmost roots of F_1, F_2: (1) r_{F_1} is removed. (2) r_{F_2} is removed. (3) The two roots are matched to each other, generating two subproblems of matching their subtrees $R^\circ_{F_1}, R^\circ_{F_2}$, and matching the remaining parts L'_{F_1}, L'_{F_2}. The other recursion rule in Equation (3) is symmetric and considers the leftmost roots.

We can easily verify that, if we compute $\mathsf{ed}(T_1, T_2)$ using this recursion, the DP states visited by the recursion are indeed pairs of subforests of T_1 and T_2. We call a subforest F_1 or F_2 which appears in the above dynamic programming procedure a *relevant* subforest. Klein showed the following bound on the number of relevant subforests F_1 of T_1 generated by the DP procedure.

▶ **Lemma 6** (Lemma 3 of [24]). *If we use top-down dynamic programming to compute* $\mathsf{ed}(T_1, T_2)$ *with respect to the recursion defined in Equations* (1)–(3)*, we only ever need to compute* $\mathsf{ed}(F_1, F_2)$ *for* $O(|T_1| \log |T_1|)$ *distinct subforests* F_1 *of* T_1*.*

The proof of this lemma uses a heavy-light decomposition argument, which crucially relies on choosing the "direction" of recursion (Equations (2) and (3)) based on the sizes of the leftmost and rightmost trees in F_1. This improves upon the previous DP algorithm by Zhang and Shasha [40], which always recurses on the rightmost roots and could only give an $O(|T_1|^2)$ bound instead of $O(|T_1| \log |T_1|)$.

Since there are only $O(|T_2|^2)$ possible subforests F_2 of T_2 (Proposition 5), Lemma 6 shows that we can compute $\mathsf{ed}(T_1, T_2)$ in $O(|T_1||T_2|^2 \log |T_1|)$ time. In the next section, we show how to use the assumption that $\mathsf{ed}(T_1, T_2) \leq k$ to bound the number of relevant F_2 as well, and through this get a faster algorithm.

4 Improved Algorithm

4.1 DP state transition graph

Our algorithm builds on Klein's DP algorithm described in Section 3. For the sake of analysis, it is helpful to consider the DP state transition graph, which is a directed acyclic graph with vertices representing the DP states (F_1, F_2) and edges representing DP transitions. Each edge is associated with a proxy cost that lower bounds the true incurred cost when using this transition in the actual DP. These will be based off the trivial lower bound

$$\mathsf{ed}(F_1, F_2) \geq |\mathsf{size}(F_1) - \mathsf{size}(F_2)|, \tag{4}$$

which holds because each operation changes the size of a tree by at most 1, and at the end of applying $\mathsf{ed}(F_1, F_2)$ operations the trees must have the same size.

To define the DP state transition graph, we distinguish three types of DP transition that can occur from following the recursion of Klein's algorithm described in Equations (2) and (3). The first type corresponds to the first two cases of Equations (2) and (3) where we delete the rightmost or leftmost root of the forest. The second and third types of transition capture the two subproblems generated from the third case of Equations (2) and (3) where we match nodes in the trees. Hence, the edges in the DP state transition graph and their proxy costs are defined as follows:

Type 1 (Node Removal) We delete the rightmost (or leftmost) root of F_1 (or F_2).
For example, we can transition $(F_1, F_2) \to (F_1 - r_{F_1}, F_2)$. This transition has cost 1.

Type 2 (Subtree Removal) We remove the rightmost (or leftmost) subtrees of F_1 and F_2.
For example, we can transition $(F_1, F_2) \to (L'_{F_1}, L'_{F_2})$. This transition costs at least $|\mathsf{size}(R_{F_1}) - \mathsf{size}(R_{F_2})|$ by Equation (4) and the last case of Equation (2).

Type 3 (Subtree Selection) We focus on the subtrees below the rightmost (or leftmost) roots of F_1 and F_2.
For example, we can transition $(F_1, F_2) \to (R^\circ_{F_1}, R^\circ_{F_2})$. This transition costs at least $|\mathsf{size}(L'_{F_1}) - \mathsf{size}(L'_{F_2})|$ by Equation (4) and the last case of Equation (2).

4.2 Pruning DP states

Each pair of subforests (F_1, F_2) is a potential state in the DP table. We say a state (F_1, F_2) is "not useful" or *useless* if we do not need to evaluate $\mathsf{ed}(F_1, F_2)$ to compute the overall tree edit distance $\mathsf{ed}(T_1, T_2)$. Having defined the DP state transition graph, we use the following simple observation to label some states as useless.

▶ **Proposition 7** (DP State Pruning Rule 1). *Suppose input trees T_1, T_2 satisfy $\mathsf{ed}(T_1, T_2) \le k$. Then if a state cannot be reached from (T_1, T_2) by traversing a sequence of edges with total cost at most k in the DP state transition graph, that state is useless.*

We will also make use of the following pruning rule, which is a direct application of Equation (4).

▶ **Proposition 8** (DP State Pruning Rule 2). *Suppose input trees T_1, T_2 satisfy $\mathsf{ed}(T_1, T_2) \le k$. If $|\mathsf{size}(F_1) - \mathsf{size}(F_2)| > k$, then the DP state (F_1, F_2) is useless.*

The two pruning rules will enable us to prove the following core result, which shows that when the tree edit distance is bounded, each relevant subforest cannot occur in too many useful states.

▶ **Lemma 9** (Number of useful DP states). *Suppose input trees T_1, T_2 satisfy $\mathsf{ed}(T_1, T_2) \le k$. For each relevant subforest F_1 of T_1, there are at most $O(k^2)$ subforests F_2 of T_2 such that (F_1, F_2) is a useful DP state.*

Lemma 9 together with Lemma 6 immediately shows an $O(nk^2 \log n)$ bound on the number of useful DP states, which will suffice to prove Theorem 1, so in the remainder of this section, we setup the proof of this lemma.

▶ **Definition 10** (Upper parts). *Given a subforest F of T, we partition the nodes of $T \setminus F$ into three disjoint upper parts MU_F, LU_F, and RU_F as follows.*

- *The middle upper part MU_F contains the nodes on the path from the root of T to $\mathsf{LCA}(F)$ (excluding $\mathsf{LCA}(F)$ if $\mathsf{LCA}(F) \in F$).*
- *The left upper part is defined as $LU_F := \{u \in T \setminus MU_F \mid \mathsf{pre}(u) < \mathsf{pre}(v) \text{ for all } v \in F\}$, where $\mathsf{pre}(u)$ denote the index of u in the preorder traversal of T ($1 \le \mathsf{pre}(u) \le |T|$). The right upper part RU_F is defined symmetrically using the postorder traversal of T. Intuitively, LU_F consists of the nodes to the left of the path MU_F, and RU_F consists of the nodes to the right of this path.*

See Figure 4 for some examples.

If a DP state (G_1, G_2) can be reached from (T_1, T_2) in the DP state transition graph, it means that we obtain G_1 and G_2 by removing some nodes in T_1 and T_2 respectively, following the DP transition rules. We classify the removed nodes in $T_1 \setminus G_1$ according to which of

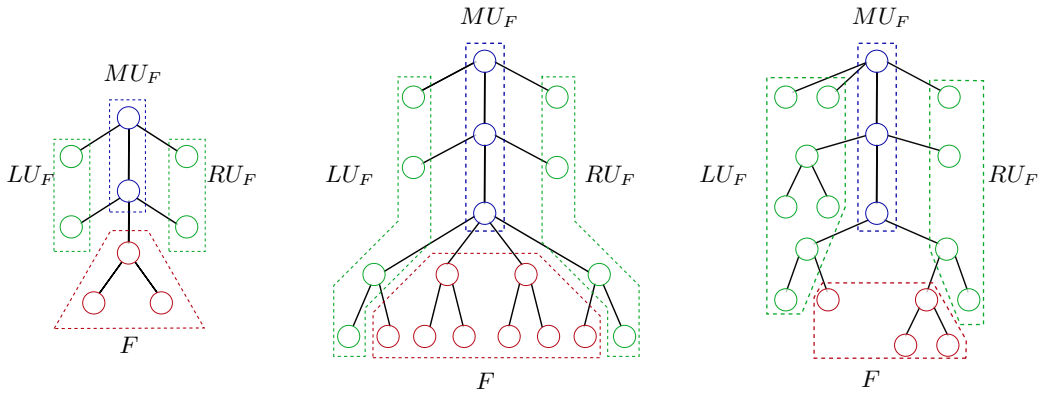


Figure 4 Three examples of subforests F in different underlying trees, with upper parts labeled.

the three upper parts they belong to. For node $v \in T_1 \setminus G_1$, if $v \in LU_{G_1}$ (or $v \in RU_{G_1}$, $v \in MU_{G_1}$), then we say v is *left-removed* (or *right-removed*, *middle-removed*) with respect to subforest G_1. If during a DP transition $(F_1, F_2) \to (G_1, G_2)$, a node $v \in F_1 \setminus G_1$ is left-removed (or right-removed, middle-removed) with respect to not only G_1, but also all subforests $G_1' \subseteq G_1$ (which may be reached by later DP transitions), then we simply say v is left-removed (or right-removed, middle-removed) during this DP transition, without specifying the subforest G_1. The above discussion also similarly applies to the second input tree T_2 and its subforests.

By inspecting the DP transition rules described in Section 4.1, we immediately have the following simple but useful observation.

▶ **Lemma 11.** *Let (F_1, F_2) be a DP state. The following hold:*

- *A type 2 transition from this state either right-removes $\mathsf{size}(R_{F_1})$ nodes, or left-removes $\mathsf{size}(L_{F_1})$ nodes from F_1, depending on whether the right or left subtree were removed.*
- *A type 3 transition from this state either left-removes $\mathsf{size}(F_1 - R_{F_1})$ nodes and middle-removes one node, or right-removes $\mathsf{size}(F_1 - L_{F_1})$ nodes and middle-removes one node from F_1, depending on whether the transition zoomed in on the right or left subtree.*

Similar statements hold for removals in F_2.

Note that in the case of type 1 transitions, we cannot tell whether the node being removed was a left, middle, or right-removal. However, we observe that a type 1 transition always has cost 1. Combining this observation with Lemma 11 and the pruning rule in Proposition 7, we obtain the following property of useful DP states (G_1, G_2):

▶ **Lemma 12.** *If DP state (G_1, G_2) survives the pruning rule in Proposition 7, then*

$$|\mathsf{size}(LU_{G_1}) - \mathsf{size}(LU_{G_2})| \leq k,$$

and

$$|\mathsf{size}(RU_{G_1}) - \mathsf{size}(RU_{G_2})| \leq k.$$

Proof. Consider the sets LU_{G_1} and LU_{G_2} of left-removed nodes in G_1 and G_2. Suppose k_1 nodes of LU_{G_1} and k_2 nodes of LU_{G_2} were removed by type 1 transitions, incurring a total cost of $k_1 + k_2$. The remaining $\mathsf{size}(LU_{G_1}) - k_1$ nodes in LU_{G_1} and $\mathsf{size}(LU_{G_2}) - k_2$ nodes in LU_{G_2} must be the result of type 2 and 3 transitions.

From Lemma 11 and the discussion in Section 4.1, we know that when a type 2 or 3 transition t left-removes $c_1^{(t)}$ nodes from T_1 and $c_2^{(t)}$ nodes from T_2, the incurred cost is at least $|c_1^{(t)} - c_2^{(t)}|$. Then by triangle inequality, the total cost from all type 2 and 3 transitions is at least

$$\sum_t \left|c_1^{(t)} - c_2^{(t)}\right| \geq \left|\sum_t c_1^{(t)} - \sum_t c_2^{(t)}\right| = \left|(\mathsf{size}(LU_{G_1}) - k_1) - (\mathsf{size}(LU_{G_2}) - k_2)\right|,$$

where the sum is over all type 2 and 3 transitions t leading from state (T_1, T_2) to state (G_1, G_2). Then, by applying triangle inequality once more, the total cost from all transitions is at least

$$k_1 + k_2 + \left|(\mathsf{size}(LU_{G_1}) - k_1) - (\mathsf{size}(LU_{G_2}) - k_2)\right| \geq \left|\mathsf{size}(LU_{G_1}) - \mathsf{size}(LU_{G_2})\right|.$$

This proves the first inequality. The second inequality follows from identical reasoning, applied to the right-removed instead of the left-removed nodes of G_1 and G_2. ◀

We have just derived the useful Lemma 12 from the first pruning rule in Proposition 7. To prove Lemma 9, we still need to apply the second pruning rule in Proposition 8 as well. We will use the following lemma.

▶ **Lemma 13.** *Given three integers a, b, c, the number of subforests F of a tree T which simultaneously satisfy $|\mathsf{size}(LU_F) - a| \leq k$, $|\mathsf{size}(RU_F) - b| \leq k$, and $|\mathsf{size}(F) - c| \leq k$ is at most $O(k^2)$.*

Before proving Lemma 13, we show that it implies the desired upper bound on the number of useful DP states that survive both pruning rules in Proposition 7 and Proposition 8.

Proof of Lemma 9 given Lemma 13. We are given a relevant subforest F_1 of T_1, and want to bound the number of subforests F_2 of T_2 such that (F_1, F_2) is a useful state. By Lemma 12, the state (F_1, F_2) is useful only if

$$\left|\mathsf{size}(LU_{F_2}) - a\right|, \left|\mathsf{size}(RU_{F_2}) - b\right| \leq k$$

for $a = \mathsf{size}(LU_{F_1})$ and $b = \mathsf{size}(RU_{F_1})$. Moreover, by Proposition 8 if the state is useful then

$$\left|\mathsf{size}(F_2) - c\right| \leq k$$

for $c = \mathsf{size}(F_1)$. Hence, applying Lemma 13 with $T = T_2$ immediately implies that there are $O(k^2)$ possibilities for F_2, which proves the desired result. ◀

4.3 Proof of Lemma 13

Suppose $\mathsf{size}(LU_F) = \ell$ and $\mathsf{size}(RU_F) = r$ for some integers ℓ and r within k of a and b respectively. Then we claim the following algorithm outputs all possible subforests F satisfying the hypotheses of the lemma:

1. Initialize $F = T$ as the given tree.
2. While $\ell \neq 0$ or $r \neq 0$:
   a. If F has only root remaining, delete this root (middle-removal) from F
   b. Otherwise, F has more than one root remaining:
      i. If $\mathsf{size}(L_F) \leq \ell$: remove the leftmost tree and update $\ell \leftarrow \ell - \mathsf{size}(L_F)$
      ii. Else if $\mathsf{size}(R_F) \leq r$: remove the rightmost tree and update $r \leftarrow r - \mathsf{size}(R_F)$
      iii. Otherwise remove the leftmost root ℓ times, remove the rightmost root r times, and return F *(unique solution case)*

3. If F has one root remaining: repeatedly remove the only root (middle-removal) until we no longer have a single root. Return all the forests encountered during this procedure as possible solutions of F *(multiple solutions case)*
4. Otherwise, F has more than one root remaining: return F *(unique solution case)*

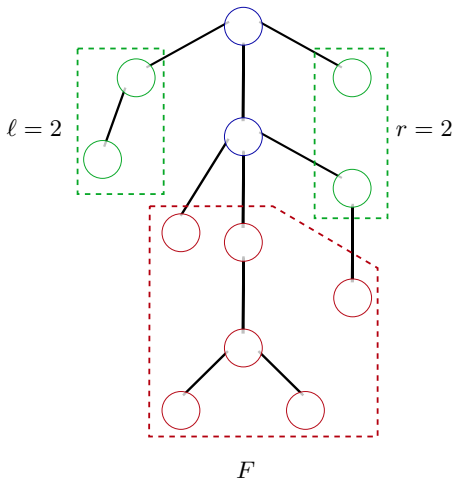


Figure 5 An example of the unique solution case, where $\ell = 2$ and $r = 2$.

At each step of the algorithm, we are either at a state with multiple roots or at a state with one root. In the former case, we have to left-remove or right-remove which we do (unless we have already left-removed ℓ times and right-removed r times, in which case we halt). In the latter case, if we still have not left-removed or right-removed the full number of times, we must keep middle-removing until we can make left or right removals. Terminating in one of these states corresponds to the *unique solution* cases of the algorithm (at step 2(b)iii or step 4). An example is given in Figure 5.

In these situations, the algorithm halts on the unique subforest F of T with $\mathsf{size}(LU_F) = \ell$ and $\mathsf{size}(RU_F) = r$. Since there are $O(k)$ possible values for ℓ and r individually, we get that there are at most $O(k^2)$ distinct subforests F which can be outputted as a "unique solution" in the above procedure.

The only other possibility is that we find ourselves in step 3 of the algorithm at a point where we have already left-removed ℓ times and right-removed r times, and there is only one root u remaining. In this case F might not be uniquely determined: we can continue to middle-remove the remaining root for some number of times and then return a possible solution of F. Formally, let w be the deepest descendant of the remaining root u, such that for every node v on the path from w to u, v has no siblings. Then, for every such node v, the subtree rooted at v (denoted T_v) and $T_v - v$ can be a valid solution for F. This describes the *multiple solutions case* annotated in step 3 of the above procedure. An example of the multiple solutions case is given in Figure 6.

By the above discussion, a subforest F from the multiple solutions case can be determined uniquely by the identity of the lowest common ancestor $v = \mathsf{LCA}(F)$, and the choice of whether v is in F or not. We now prove that, over all choices of valid ℓ and r, there are only $O(k)$ many possibilities for the node v. Combined with the unique solution case, this will immediately finish the proof of the lemma.

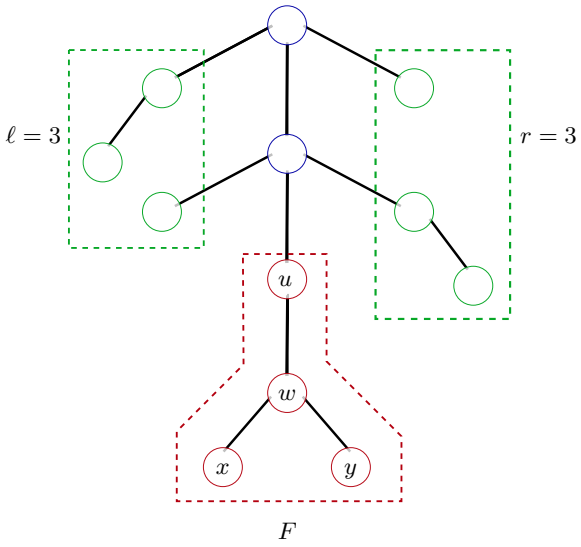


Figure 6 An example of the multiple solutions case, where $l = 3, r = 3$. Before executing step 3 of the algorithm, the remaining F consists of $\{u, w, x, y\}$. Then the algorithm returns three possible solutions: $\{u, w, x, y\}, \{w, x, y\}$, and $\{x, y\}$.

We first consider the case where, among all possible node choices for v, there are two such that neither is an ancestor of the other. Then pick the leftmost (with respect to post-order traversal) and the rightmost (with respect to preorder traversal) of such possibilities for v, denoted v_1 and v_2 respectively. Let G_1 and G_2 be the subtrees in T rooted at v_1 and v_2, respectively. Then by the assumptions on F we necessarily have

$$|\mathsf{size}(RU_{G_1}) - b|, |\mathsf{size}(RU_{G_2}) - b| \leq k.$$

Note that $RU_{G_2} \subseteq RU_{G_1}$. Write $D = RU_{G_1} \setminus RU_{G_2}$ for the difference of the right upper part of G_1 and the right upper part G_2. Thus, by triangle inequality, we get that

$$\mathsf{size}(D) = \mathsf{size}(RU_{G_1}) - \mathsf{size}(RU_{G_2}) \leq 2k.$$

By our choice of v_1 and v_2, we know that any possible choice for v is either a node in D or an ancestor of v_1. For the former case, we have already shown that there are at most $O(k)$ nodes in D. In the latter case, each distinct v which is an ancestor of v_1 determines a subforest F of a different size. Then because we are assuming that $|\mathsf{size}(F) - c| \leq k$, there are only $O(k)$ possibilities for the choices of v which are ancestors of v_1.

The previous argument applies whenever there are two choices for v, neither of which is an ancestor of the other. If there do not exist such options for v, then all possible choices of v lie on the a single root-to-leaf path of T. By the same reasoning as before, the number of possible cases for v here is again at most $O(k)$, because each v would determine a different-sized subforest and $\mathsf{size}(F)$ is allowed to take on $O(k)$ distinct values.

This completes the proof of Lemma 13. As noted earlier, this implies Lemma 9. We conclude by tying these results back to our main theorem.

Proof of Theorem 1. Set up a table which can be indexed by pairs of subforests (F_1, F_2) of T_1 and T_2. Begin using Klein's dynamic programming approach outlined in Section 3 and Lemma 6 but avoid generating subproblems according to the pruning rules described

in Proposition 7 and Proposition 8, and store solutions $\mathsf{ed}(F_1, F_2)$ produced. In particular, when Klein's algorithm would normally generate a subproblem, we first check if the produced subproblem would be a useful state according to our previous definitions. Proposition 8 and the proof of Lemma 13 make it clear that we can quickly check if a state is useful provided we know the sizes of $F_1, F_2, LU_{F_1}, RU_{F_1}, LU_{F_2}$, and RU_{F_2}, and this information can be kept track of easily simply by updating the sizes according to the type of transition we follow in the table.

So, we can compute $\mathsf{ed}(T_1, T_2)$ while only computing $\mathsf{ed}(F_1, F_2)$ for useful states. By Lemma 6 there are $O(n \log n)$ possibilities for F_1 and by Lemma 9 there are $O(k^2)$ choices for F_2 for each F_1. So overall we only fill in at most $O(nk^2 \log n)$ entries of the DP table. Since we do a constant amount of work to get the value at each entry of the table, our algorithm has the desired running time. ◀

5 Open problems

For trees of bounded edit distance $k = O(1)$ our algorithm runs in linear time. However, for larger tree edit distances $k = \Theta(n)$ our algorithm requires $O(n^3 \log n)$ time, which is slower than the fastest known algorithm [17] for general tree edit distance by a logarithmic factor. This motivates the question: can we solve the bounded tree edit distance problem in $O(nk^2)$ time instead of $O(nk^2 \log n)$?

The easier problem of *string* edit distance can be solved in $\widetilde{O}(n + k^2)$ time [30, 28], which is quasilinear even for super constant distance parameter $k = O(\sqrt{n})$. This motivates the question of whether it is possible to get similar speedups for tree edit distance. It would be especially interesting to see if the bounded tree edit distance problem can be solved in $\widetilde{O}(n + k^3)$ time. Perhaps the suffix tree techniques used in [33] (and discussed in [11, Appendix]) could prove useful in showing such a result.

Regarding variants of tree edit distance, it remains an open question to get faster algorithms for the harder problem of *unrooted* tree edit distance [24, 18] (where the elementary operations are edge contraction, insertion, and relabeling) when the distance is bounded by k. The best known algorithm for unrooted tree edit distance was recently given by Dudek and Gawrychowski [18] and runs in $O(n^3)$ time. The previous $O(n^3 \log n)$ time algorithm by Klein [24] also applies to the unrooted setting. Although we extended Klein's algorithm to tackle the rooted tree edit distance problem in $O(nk^2 \log n)$ time, it is not obvious how to extend their approach to the unrooted bounded distance setting. This is because Klein solves the unrooted version of the problem by dynamic programming over the subproblems generated by all possible rootings of T_2. This is fine for computing general edit distance because the number of subforests over all possible rootings is $O(n^2)$ just like the number of subforests for a fixed rooted tree on n nodes. However, when the tree edit distance is bounded, the number of possible relevant subproblems over all possible rootings can be $\Omega(n)$ even when k is small. Although our algorithm can be used to recover a near quadratic time algorithm for unrooted tree edit distance when $k = O(1)$ is constant, it remains open whether we can obtain a quasilinear time algorithm in this setting.

Finally, although general tree edit distance with *arbitrary weights* cannot be solved in truly subcubic time unless certain popular conjectures are false [13], analogous fine-grained hardness results rule out truly subquadratic time algorithms for string edit distance even when deletions and insertions have unit cost [6]. Can we show conditional hardness for tree edit distance with unit costs, or can we find a subcubic time algorithm for this problem?

References

1. Amir Abboud, Arturs Backurs, and Virginia Vassilevska Williams. Tight hardness results for LCS and other sequence similarity measures. In *Proceedings of the 56th IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 59–78, 2015. doi:10.1109/FOCS.2015.14.
2. Tatsuya Akutsu, Daiji Fukagawa, and Atsuhiro Takasu. Approximating tree edit distance through string edit distance. *Algorithmica*, 57(2):325–348, 2010. doi:10.1007/s00453-008-9213-z.
3. Alexandr Andoni, Robert Krauthgamer, and Krzysztof Onak. Polylogarithmic approximation for edit distance and the asymmetric query complexity. In *Proceedings of the 51st Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 377–386, 2010. doi:10.1109/FOCS.2010.43.
4. Alexandr Andoni and Negev Shekel Nosatzki. Edit distance in near-linear time: it's a constant factor. In *Proceedings of the 61st IEEE Annual Symposium on Foundations of Computer Science (FOCS)*, 2020. arXiv:2005.07678.
5. Alexandr Andoni and Krzysztof Onak. Approximating edit distance in near-linear time. *SIAM J. Comput.*, 41(6):1635–1648, 2012. doi:10.1137/090767182.
6. Arturs Backurs and Piotr Indyk. Edit distance cannot be computed in strongly subquadratic time (unless SETH is false). *SIAM J. Comput.*, 47(3):1087–1097, 2018. doi:10.1137/15M1053128.
7. John Bellando and Ravi Kothari. Region-based modeling and tree edit distance as a basis for gesture recognition. In *Proceedings of the 10th International Conference on Image Analysis and Processing (ICIAP)*, pages 698–703. IEEE Computer Society, 1999. doi:10.1109/ICIAP.1999.797676.
8. Philip Bille. A survey on tree edit distance and related problems. *Theor. Comput. Sci.*, 337(1-3):217–239, 2005. doi:10.1016/j.tcs.2004.12.030.
9. Philip Bille and Martin Farach-Colton. Fast and compact regular expression matching. *Theor. Comput. Sci.*, 409(3):486–496, 2008. doi:10.1016/j.tcs.2008.08.042.
10. Mahdi Boroujeni, Soheil Ehsani, Mohammad Ghodsi, Mohammad Taghi Hajiaghayi, and Saeed Seddighin. Approximating edit distance in truly subquadratic time: Quantum and MapReduce. In *Proceedings of the 29th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1170–1189, 2018. doi:10.1137/1.9781611975031.76.
11. Mahdi Boroujeni, Mohammad Ghodsi, MohammadTaghi Hajiaghayi, and Saeed Seddighin. $1+\varepsilon$ approximation of tree edit distance in quadratic time. In *Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing (STOC)*, pages 709–720, 2019. doi:10.1145/3313276.3316388.
12. Joshua Brakensiek and Aviad Rubinstein. Constant-factor approximation of near-linear edit distance in near-linear time. In *Proccedings of the 52nd Annual ACM SIGACT Symposium on Theory of Computing (STOC)*, pages 685–698, 2020. doi:10.1145/3357713.3384282.
13. Karl Bringmann, Paweł Gawrychowski, Shay Mozes, and Oren Weimann. Tree edit distance cannot be computed in strongly subcubic time (unless APSP can). *ACM Trans. Algorithms*, 16(4):48:1–48:22, 2020. doi:10.1145/3381878.
14. Peter Buneman, Martin Grohe, and Christoph Koch. Path queries on compressed XML. In *Proceedings of the 29th International Conference on Very Large Data Bases (VLDB)*, pages 141–152, 2003. doi:10.1016/B978-012722442-8/50021-5.
15. Diptarka Chakraborty, Debarati Das, Elazar Goldenberg, Michal Koucký, and Michael E. Saks. Approximating edit distance within constant factor in truly sub-quadratic time. In *Proceedings of the 59th IEEE Annual Symposium on Foundations of Computer Science (FOCS)*, pages 979–990. IEEE Computer Society, 2018. doi:10.1109/FOCS.2018.00096.
16. Sudarshan S. Chawathe. Comparing hierarchical data in external memory. In *Proceedings of the 25th International Conference on Very Large Data Bases (VLDB)*, pages 90–101, 1999. URL: http://www.vldb.org/conf/1999/P8.pdf.

17 Erik D. Demaine, Shay Mozes, Benjamin Rossman, and Oren Weimann. An optimal decomposition algorithm for tree edit distance. *ACM Trans. Algorithms*, 6(1):2:1–2:19, 2009. doi:10.1145/1644015.1644017.

18 Bartłomiej Dudek and Paweł Gawrychowski. Edit distance between unrooted trees in cubic time. In *Proceedings of the 45th International Colloquium on Automata, Languages, and Programming (ICALP)*, pages 45:1–45:14, 2018. doi:10.4230/LIPIcs.ICALP.2018.45.

19 Serge Dulucq and Hélène Touzet. Analysis of tree edit distance algorithms. In *Proceedings of the 14th Annual Symposium on Combinatorial Pattern Matching (CPM)*, volume 2676 of *Lecture Notes in Computer Science*, pages 83–95. Springer, 2003. doi:10.1007/3-540-44888-8_7.

20 Serge Dulucq and Hélène Touzet. Decomposition algorithms for the tree edit distance problem. *J. Discrete Algorithms*, 3(2-4):448–471, 2005. doi:10.1016/j.jda.2004.08.018.

21 Paolo Ferragina, Fabrizio Luccio, Giovanni Manzini, and S. Muthukrishnan. Compressing and indexing labeled trees, with applications. *J. ACM*, 57(1):4:1–4:33, 2009. doi:10.1145/1613676.1613680.

22 Dan Gusfield. *Algorithms on Strings, Trees, and Sequences: Computer Science and Computational Biology*. Cambridge University Press, 1997. doi:10.1017/CBO9780511574931.

23 Matthias Höchsmann, Thomas Töller, Robert Giegerich, and Stefan Kurtz. Local similarity in RNA secondary structures. In *Proceedings of 2nd IEEE Computer Society Bioinformatics Conference, CSB*, pages 159–168. IEEE Computer Society, 2003. doi:10.1109/CSB.2003.1227315.

24 Philip N. Klein. Computing the edit-distance between unrooted ordered trees. In *Proceedings of the 6th Annual European Symposium on Algorithms (ESA)*, volume 1461 of *Lecture Notes in Computer Science*, pages 91–102. Springer, 1998. doi:10.1007/3-540-68530-8_8.

25 Philip N. Klein, Thomas B. Sebastian, and Benjamin B. Kimia. Shape matching using edit-distance: an implementation. In *Proceedings of the 12th Annual Symposium on Discrete Algorithms (SODA)*, pages 781–790, 2001. URL: http://dl.acm.org/citation.cfm?id=365411.365779.

26 Philip N. Klein, Srikanta Tirthapura, Daniel Sharvit, and Benjamin B. Kimia. A tree-edit-distance algorithm for comparing simple, closed shapes. In David B. Shmoys, editor, *Proceedings of the 11th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 696–704, 2000. URL: http://dl.acm.org/citation.cfm?id=338219.338628.

27 Michal Koucký and Michael E. Saks. Constant factor approximations to edit distance on far input pairs in nearly linear time. In *Proccedings of the 52nd Annual ACM SIGACT Symposium on Theory of Computing (STOC)*, pages 699–712. ACM, 2020. doi:10.1145/3357713.3384307.

28 Gad M. Landau and Uzi Vishkin. Fast string matching with k differences. *J. Comput. Syst. Sci.*, 37(1):63–78, 1988. doi:10.1016/0022-0000(88)90045-1.

29 William J. Masek and Mike Paterson. A faster algorithm computing string edit distances. *J. Comput. Syst. Sci.*, 20(1):18–31, 1980. doi:10.1016/0022-0000(80)90002-1.

30 Eugene W. Myers. An O(ND) difference algorithm and its variations. *Algorithmica*, 1(2):251–266, 1986. doi:10.1007/BF01840446.

31 Thomas B. Sebastian, Philip N. Klein, and Benjamin B. Kimia. Recognition of shapes by editing their shock graphs. *IEEE Trans. Pattern Anal. Mach. Intell.*, 26(5):550–571, 2004. doi:10.1109/TPAMI.2004.1273924.

32 Bruce A. Shapiro and Kaizhong Zhang. Comparing multiple RNA secondary structures using tree comparisons. *Bioinformatics*, 6(4):309–318, October 1990. doi:10.1093/bioinformatics/6.4.309.

33 Dennis E. Shasha and Kaizhong Zhang. Fast algorithms for the unit cost editing distance between trees. *J. Algorithms*, 11(4):581–621, 1990. doi:10.1016/0196-6774(90)90011-3.

34 Kuo-Chung Tai. The tree-to-tree correction problem. *J. ACM*, 26(3):422–433, 1979. doi:10.1145/322139.322143.

35 Hélène Touzet. A linear tree edit distance algorithm for similar ordered trees. In *Proceedings of the 16th Annual Symposium on Combinatorial Pattern Matching (CPM)*, volume 3537 of *Lecture Notes in Computer Science*, pages 334–345. Springer, 2005. doi:10.1007/11496656_29.

36 Hélène Touzet. Comparing similar ordered trees in linear-time. *J. Discrete Algorithms*, 5(4):696–705, 2007. doi:10.1016/j.jda.2006.07.002.

37 Esko Ukkonen. Algorithms for approximate string matching. *Inf. Control.*, 64(1-3):100–118, 1985. doi:10.1016/S0019-9958(85)80046-2.

38 Robert A. Wagner and Michael J. Fischer. The string-to-string correction problem. *J. ACM*, 21(1):168–173, 1974. doi:10.1145/321796.321811.

39 Michael S. Waterman. *Introduction to computational biology: maps, sequences and genomes*. CRC Press, 1995.

40 Kaizhong Zhang and Dennis E. Shasha. Simple fast algorithms for the editing distance between trees and related problems. *SIAM J. Comput.*, 18(6):1245–1262, 1989. doi:10.1137/0218082.

Improved Approximation for Longest Common Subsequence over Small Alphabets

Shyan Akmal
MIT, EECS and CSAIL, Cambridge, MA, USA

Virginia Vassilevska Williams
MIT, EECS and CSAIL, Cambridge, MA, USA

Abstract

This paper investigates the approximability of the Longest Common Subsequence (LCS) problem. The fastest algorithm for solving the LCS problem exactly runs in essentially quadratic time in the length of the input, and it is known that under the Strong Exponential Time Hypothesis the quadratic running time cannot be beaten. There are no such limitations for the approximate computation of the LCS however, except in some limited scenarios. There is also a scarcity of approximation algorithms. When the two given strings are over an alphabet of size k, returning the subsequence formed by the most frequent symbol occurring in both strings achieves a $1/k$ approximation for the LCS. It is an open problem whether a better than $1/k$ approximation can be achieved in truly subquadratic time ($O(n^{2-\delta})$ time for constant $\delta > 0$).

A recent result [Rubinstein and Song SODA'2020] showed that a $1/2 + \epsilon$ approximation for the LCS over a binary alphabet is possible in truly subquadratic time, provided the input strings have the same length. In this paper we show that if a $1/2 + \epsilon$ approximation (for $\epsilon > 0$) is achievable for binary LCS in truly subquadratic time when the input strings can be unequal, then for every constant k, there is a truly subquadratic time algorithm that achieves a $1/k + \delta$ approximation for k-ary alphabet LCS for some $\delta > 0$. Thus the binary case is the hardest. We also show that for every constant k, if one is given two strings of *equal* length over a k-ary alphabet, one can obtain a $1/k + \epsilon$ approximation for some constant $\epsilon > 0$ in truly subquadratic time, thus extending the Rubinstein and Song result to all alphabets of constant size.

2012 ACM Subject Classification Theory of computation → Design and analysis of algorithms

Keywords and phrases approximation algorithms, longest common subsequence, subquadratic

Digital Object Identifier 10.4230/LIPIcs.ICALP.2021.13

Category Track A: Algorithms, Complexity and Games

Funding *Shyan Akmal*: Supported by NSF Grant CCF-1909429.
Virginia Vassilevska Williams: Supported by an NSF CAREER Award, NSF Grant CCF-1909429, a BSF Grant BSF:2012338, a Google Research Fellowship and a Sloan Research Fellowship.

Acknowledgements We thank Jenny Kaufmann for suggesting several helpful revisions.

1 Introduction

In a large variety of applications, from spell-checkers to DNA sequence alignment, one seeks to compute how similar two given sequences of letters are. Arguably the most popular measures of sequence similarity are the *longest common subsequence (LCS)* and the *edit distance*. The LCS of two given sequences A and B (as the name suggests) measures the maximum length of a sequence whose symbols appear in both A and B in the same order. The edit distance, on the other hand, measures how far apart two strings are by counting the minimum number of insertions, deletions and substitutions of characters that must be performed on one string to transform it into the other. These two measures are related: the complement of the LCS is the version of edit distance in which one minimizes only the

number of insertions and deletions (in fact, both versions of edit distance are the same up to a factor of 2, so with respect to constant factor approximation algorithms they are equivalent). Both the LCS and the edit distance of two length n strings can be computed in $O(n^2)$ time using a classic dynamic programming approach. The fastest algorithm for both problems is the $O(n^2/(\log n)^2)$ time algorithm of Masek and Paterson [20].

Hardness results from fine-grained complexity have shown that truly subquadratic time algorithms (those running in time $O(n^{2-\delta})$ for some constant $\delta > 0$) for LCS and edit distance cannot exist under the Strong Exponential Time Hypothesis [2, 9, 14] and other even more believable hypotheses [3, 16]. Consequently, much of the recent interest around LCS and edit distance has concerned approximation algorithms for the problems. A long chain of progress on edit distance approximation (e.g. [19, 11, 19, 10, 8, 6, 12]) has culminated in the breakthrough constant-factor approximation algorithm of [15] running in truly subquadratic time. Several improvements followed this breakthrough including [5, 18, 13], with the most recent result being a constant factor approximation algorithm running in near linear time [7].

In contrast, much less is known about how well LCS can be approximated in truly subquadratic time. For inputs over non-constant size alphabets, some fine-grained hardness results [1, 4, 16] and nontrivial super-constant approximations [17] are known. When the input strings come from a fixed alphabet of constant size k, there is a trivial $1/k$-approximation algorithm that returns in linear time the longest common *unary* subsequence of two inputs. Despite the simplicity of this algorithm, until recently no better constant-factor approximation algorithm was known for *any* constant size alphabet. There are also no existing hardness results that rule out better approximation algorithms.

Recently, for the case of *binary* strings of *equal* length, Rubinstein and Song [21] were able to improve upon the simple 1/2-approximation algorithm described above, obtaining for some constants $\epsilon, \delta > 0$ an $O(n^{2-\delta})$ time algorithm that returns a $(1/2 + \epsilon)$-approximation. It is still open, however, whether one can obtain such an algorithm for binary strings of unequal length, and whether one can extend the result to achieve a truly subquadratic time algorithm achieving a better than k-approximation for strings over an alphabet of size k, for every constant k.

Our results

In this paper we present two results. The first result shows that if one can obtain a truly subquadratic time, better than 2-approximation algorithm for the LCS of binary strings of possibly unequal lengths, then one can use this algorithm to obtain a truly subquadratic time, better than k-approximation algorithm for the LCS of strings over a k-ary alphabet, for any constant k.

▶ **Theorem 1.** *For any fixed integer $k \geq 2$ there is an $O(n)$ time algorithm that given an instance of LCS for strings of length at most n over an alphabet of size k, reduces it to $O(k^2)$ instances of LCS over* binary *strings of length at most n, so that $(1/2 + \epsilon)$-approximate solutions (for $\epsilon > 0$) for these LCS instances can be translated in $O(n)$ time into a $(1/k + \epsilon_k)$ approximation of the k-ary alphabet LCS instance, where $\epsilon_k > 0$ is a constant only depending on ϵ and k.*

In other words, in order to beat the longstanding $1/k$-approximation algorithm for LCS, one merely needs to obtain a better than 1/2 approximation for the binary case, i.e. to extend the Rubinstein and Song result to strings of possibly unequal length.

Our second result generalizes the Rubinstein-Song result by proving that one can beat the simple $1/k$-approximation algorithm for every constant k, as long as the input strings have equal length.

▶ **Theorem 2.** *Given two strings A and B of length n over an arbitrary alphabet of size k, there exist positive constants ϵ and δ such that we can compute a $1/k + \epsilon$ approximation for the longest common subsequence of A and B in $O(n^{2-\delta})$ time.*

In fact, our algorithm can actually $(1/k + \epsilon)$-approximate the LCS in near-linear time. Note that our result applies only to strings with equal input lengths. The relevance of this restriction is discussed in Section 2, which also contains the proof of Theorem 1. We present the proof of Theorem 2 in Section 3.

Preliminaries

We write approximation ratios as constants less than 1, so that for example a $1/2 + \epsilon$ approximation algorithm for the LCS of A and B is an algorithm that returns a common subsequence of A and B with length at least $(1/2 + \epsilon) \cdot \text{LCS}(A, B)$.

When we discuss edit distance in the rest of the paper we mean the version of edit distance that does not allow symbol substitutions but only measures the number of insertions and deletions. For constant factor approximation algorithms, this version of the problem is equivalent to the original.

2 Reduction to Binary Alphabets & Input Length Conditions

We begin by showing how to reduce nontrivial constant factor approximations of LCS over large alphabets to better than 1/2 approximations of LCS over binary alphabets. Although we do not directly apply this reduction in our proof of Theorem 2, the reduction is elegant and motivates the approach we end up using. Moreover, the reduction works even for strings of non-equal lengths, thus showing that one merely needs to extend the Rubinstein-Song result to non-equal length strings in order to truly improve upon the trivial alphabet-size approximation algorithm. We will need the following definition.

▶ **Definition 3** (Restrictions). *Given an alphabet Σ, we call a subset $\Sigma' \subseteq \Sigma$ a* subalphabet. *Given a string A from alphabet Σ the* restriction *of A to a subalphabet Σ' is the maximum subsequence of A whose characters are all in Σ'.*

▶ **Theorem 4.** *Fix integers s and ℓ with $s > \ell \geq 2$. Suppose that there is a $T(n, \ell)$ time algorithm that achieves an $1/(\ell - \epsilon)$-approximation of the LCS of two strings of length at most n from an alphabet of size ℓ. Then, there is also a $O((n + T(n, \ell))\binom{s}{\ell})$ time algorithm that achieves an $1/(s(1 - \epsilon/\ell))$-approximation of the LCS of two strings of length at most n from an alphabet of size s.*

Proof. We will show how to reduce the LCS for two strings of length at most n over a s-ary alphabet, to the LCS for two strings of length at most n over an ℓ-ary alphabet for any $\ell < s$. The reduction runs in $O(n\binom{s}{\ell})$ time and produces $\binom{s}{\ell}$ instances of ℓ-ary alphabet LCS.

Let A and B two strings of length at most n over an alphabet Σ of size s. Let C be the longest common subsequence of A and B (we do not know C). For the sake of argument, sort the alphabet symbols according to their number of occurrences in C.

Let x be the collection of the ℓ most frequent alphabet symbols in C. Let C_x be the subsequence of C obtained by restricting C to the subalphabet of Σ that contains the symbols of x. Since x has the ℓ most frequent symbols in C, C_x contains at least an ℓ/s fraction of C.

Now, let us describe our algorithm. Given A and B, we consider all subsets of the alphabet consisting of precisely ℓ symbols (one of these subsets will be x.) For each such collection y, consider the sub-instance of the LCS instance restricted to the symbols of y. Let $\text{OPT}(y)$ be the optimal LCS for this instance.

We know that $|\operatorname{OPT}(x)| \geq |C_x| \geq (\ell/s)|C|$. So when we consider $y = x$, if we can efficiently obtain an $1/(\ell - \epsilon)$ approximation for $\operatorname{OPT}(x)$, we will get a common subsequence of A and B of length at least

$$\frac{|\operatorname{OPT}(x)|}{\ell - \epsilon} \geq \frac{|C|}{s(1 - \epsilon/\ell)},$$

and thus yields the desired approximation for the LCS of A and B. The running time is multiplied by $\binom{s}{\ell}$ which is a constant, as long as s is. ◀

By setting $\epsilon = \ell\delta s/(1+\delta s)$ in the above Theorem statement, we obtain a linear time reduction from obtaining a $1/s+\delta$-approximation for s-ary strings to a $1/\ell+(\delta s)/\ell$-approximation for ℓ-ary strings. We obtain Theorem 1 as a corollary:

▶ **Corollary 5.** *Fix an integer $s \geq 3$ and a constant $\delta > 0$. The problem of obtaining a $1/s + \delta$ approximation for the LCS of two strings from an alphabet of size s can be reduced in linear time to the problem of obtaining a $1/2 + (\delta s)/2$ approximation for the LCS of two strings binary strings (i.e. strings from an alphabet of size two).*

Proof. This follows from Theorem 4 by taking $\ell = 2$. The reduction has $O(s^2)$ overhead. ◀

The reason we cannot prove Theorem 2 by combining Corollary 5 with the result of [21] is that the latter gives a better than $1/2$ approximation for strings from an alphabet of size 2 only when the input strings have *equal length*. Note that in the reduction from the proof of Theorem 4, the subsequences obtained from restrictions to subalphabets may be of different lengths even if the original strings have equal lengths.

Although at first it may seem that extending the Rubinstein-Song result to strings of differing length should not be too hard, generalizing the result does not appear straightforward. In the following we will discuss why this not simple. The main hurdles come from the lemma and algorithm used in [21] stated below.

▶ **Lemma 6** (LCS and Edit Distance Connection). *For any strings X and Y of length n and m respectively, we have*

$$2 \cdot \operatorname{LCS}(X, Y) + \operatorname{ED}(X, Y) = n + m.$$

For the sake of completeness, we include a proof of the above lemma.

Proof. Consider an optimal alignment between X and Y, which matches the maximum possible number of characters of X with identical characters of Y while respecting the order in which the characters appear in each string. The characters that are matched in X and Y correspond to a longest common subsequence. This is because if there were a longer common subsequence, we could get a larger alignment by matching the characters of the subsequence in X and Y, but this would contradict the optimality of X and Y.

Similarly, the unmatched characters correspond to a minimum set of symbols that need to be deleted from X and inserted from Y to turn X into Y. If there were a smaller edit distance computation, then all the characters which were not deleted or inserted could be paired up to form a larger alignment, again contradicting optimality.

Thus there are exactly $2 \cdot \operatorname{LCS}(X, Y)$ characters paired up in the alignment and $\operatorname{ED}(X, Y)$ unmatched characters. These encompass all the characters in X and Y, and thus account for $n + m$ symbols. ◀

▶ **Definition 7** (Approximating LCS through Edit Distance). *Given strings A and B of length n, the algorithm* ApproxED(A, B) *approximates the edit-distance between A and B and then returns the lower bound on the LCS implied by this. More precisely, the algorithm computes an approximate edit distance* $\widetilde{\text{ED}}(A, B)$ *and then returns*

$$n - \frac{1}{2} \cdot \widetilde{\text{ED}}(A, B). \tag{1}$$

As stated ApproxED can use any edit distance approximation $\widetilde{\text{ED}}$ as a black box. For concreteness, we will take $\widetilde{\text{ED}}$ to be the edit distance algorithm from [5] which can achieve an approximation ratio of c for any constant $c > 3$ and runs in truly subquadratic time.

In [21], the authors use the ApproxED algorithm to handle the case of binary strings with large LCS. In this case, they notice that if the LCS is large then the edit distance must be small. Then constant factor approximations to edit distance will give good lower bounds on the LCS because in the calculation from Equation (1) we are subtracting off a small quantity from the maximum possible LCS value of n.

However, if we tried extending this algorithm to the case where the inputs X and Y have lengths n and m with $n = 100m$ (for example) by using the identity from Lemma 6, then even when the LCS is large (say of length $(1 - \epsilon)m$ for some small positive ϵ) the edit distance will still be very large compared to the length of the smaller string (at least $(99 + \epsilon)m$). So even a 3-approximation to edit-distance would incur massive error when trying to approximate LCS by computing

$$\frac{1}{2} \cdot \left(n + m - \widetilde{\text{ED}}(X, Y)\right)$$

and the result would not give any nontrivial lower bound for the LCS. In other words, when the input strings have very different lengths it is not clear how to use approximate edit distance in general to obtain good approximations for LCS. This is essentially why the algorithm from [21] and Theorem 2 both require equal length inputs.

3 Extending Alphabet Size

This section proves Theorem 8 which restates Theorem 2 from the introduction slightly.

▶ **Theorem 8.** *Given two strings A and B of length n over an arbitrary alphabet Σ of size s, there exist a positive constant ϵ such that we can compute a $1/s + \epsilon$ approximation for the longest common subsequence* LCS(A, B) *of A and B in truly subquadratic time.*

Throughout the rest of this section, we assume A and B refer to strings satisfying the conditions of Theorem 8 and that $s \geq 3$. We begin by establishing lemmas corresponding to easy instances of the problem. The following definition is useful for identifying these easy cases.

▶ **Definition 9** (Balanced Strings). *Given a string A of length n from an alphabet of size s and a parameter $\rho > 0$, we say a string is* ρ-balanced *if all its character frequencies are within ρn of n/s.*

▶ **Lemma 10** (Balanced Inputs, adapted from Lemma 3.2 of [21]). *For all sufficiently small $\rho > 0$, if either A or B is ρ-balanced, we can $(1/s + \gamma)$-approximate* LCS(A, B) *in truly subquadratic time, where γ is some positive constant depending on ρ.*

Proof. We reduce the problem to approximating the edit distance between A and B.

Without loss of generality assume A is ρ-balanced. This means that all of its character frequencies are at least $(1/s - \rho)n$. So there exists a unary common subsequence of A and B of at least this length. If this is a $(1/s + \gamma)$ approximation we are done. Otherwise the LCS must be quite large:

$$\mathrm{LCS}(A, B) > s(1/s - \rho)n - s\gamma n = (1 - s(\rho + \gamma))n.$$

Recall from Lemma 6 that

$$\mathrm{ED}(A, B) + 2 \cdot \mathrm{LCS}(A, B) = 2n,$$

where $\mathrm{ED}(A, B)$ denotes the (no substitutions) edit distance of A and B. Using the ED approximation of [5] with some approximation ratio $c > 3$ we recover an LCS approximation of length at least

$$n - c\,(n - \mathrm{LCS}(A, B)) > n\,(1 - cs(\rho + \gamma))$$

so as long as we have $1 - cs(\rho + \gamma) \geq 1/s + \gamma$, this approximation is strong enough. This inequality holds when

$$\gamma \leq \frac{s - 1 - cs^2\rho}{s(1 + cs)}.$$

We can ensure it does by picking ρ small enough that the numerator of the right hand side above is positive and then taking γ smaller than the right hand side. Note that smaller values of ρ correspond to larger values of γ. ◀

▶ **Lemma 11** (Balanced LCS). *If the LCS of A and B is not ρ-balanced, then in linear time we can $(1/s + \rho/(s-1))$-approximate* $\mathrm{LCS}(A, B)$.

Proof. Returning the longest common unary subsequence gives the desired approximation. To see this, let $\sigma_{\max}$ and $\sigma_{\min}$ be the most frequent and least frequent characters in the LCS respectively. Since the LCS is not balanced, either $\sigma_{\max}$ makes up more than a $(1/s + \rho)$ fraction of all symbols in the LCS, or $\sigma_{\min}$ makes up fewer than a $(1/s - \rho)$ fraction of the symbols in the LCS.

In the latter case, the $s - 1$ members of the alphabet besides $\sigma_{\min}$ must account for at least an $((s-1)/s + \rho)$ fraction of characters in the LCS. Among these, $\sigma_{\max}$ appears the most often, which means by averaging that $\sigma_{\max}$ accounts for at least a

$$\frac{1}{s} + \frac{\rho}{s - 1}$$

fraction of all symbols in the LCS.

In either case, $\sigma_{\max}$ makes up at least a $(1/s + \rho/(s-1))$ fraction of the symbols in the LCS. Then the string consisting of $\sigma_{\max}$ repeated $\min(\sigma_{\max}(A), \sigma_{\max}(B))$ times is a common subsequence of A and B which has at least as many instances of $\sigma_{\max}$ as the LCS does, and thus yields the desired approximation. ◀

After handling these easy cases, our approach is to restrict A and B to binary strings and invoke the frequency arguments from previous work. As we noted before, we cannot directly use the reduction in Corollary 5 because the better than 1/2 approximation of [21] only applies when the inputs have the same length. It turns out however, that the arguments

of [21] *do* hold for strings of differing length as long as the inputs satisfy a nice frequency condition. The following lemma demonstrates how by careful choice of subalphabets we can find restrictions meeting this condition. To describe this condition, we introduce some new notation: given a string A from an alphabet Σ, for any symbol $\sigma \in \Sigma$ we let $\sigma(A)$ denote the number of times σ appears in A.

▶ **Lemma 12** (Binary Restriction). *Suppose neither A nor B are ρ-balanced. Then there exists $\Sigma' \subseteq \Sigma$ with $|\Sigma'| = 2$ such that the restrictions of A and B to Σ' are not ρ/s-balanced.*

Proof. Let $\alpha_1 \leq \cdots \leq \alpha_s$ be the number of times the distinct symbols $\sigma_1, \ldots, \sigma_s$ of Σ appear respectively in A, so that σ_s is the most frequent symbol of A and σ_1 is the least common character in A. By averaging we know that $\alpha_s \geq n/s \geq \alpha_1$. Since A is not ρ-balanced we deduce that

$$\alpha_s - \alpha_1 > \rho n.$$

We can decompose the left hand side of the above equation to

$$\alpha_s - \alpha_1 = \sum_{i=2}^{s} (\alpha_i - \alpha_{i-1}).$$

Since all the summands on the right hand side are positive, there exists some index j with

$$\alpha_j - \alpha_{j-1} > (\rho/s)n. \tag{2}$$

Note that we can find such a j in linear time by scanning through A and B and keeping counts of all the characters that appear. Now, consider the $(s-1)$ two-element sets

$$\{\sigma_s, \sigma_{j-1}\}, \{\sigma_{s-1}, \sigma_{j-1}\}, \ldots, \{\sigma_j, \sigma_{j-1}\}, \{\sigma_j, \sigma_{j-2}\}, \ldots, \{\sigma_j, \sigma_1\}. \tag{3}$$

We claim that one of these sets satisfies the properties of Σ' from the lemma statement. First, note that Equation (2) ensures that the restriction of A to any of the above sets is not ρ/s-balanced, so it suffices to verify that the restriction of B will not be balanced for one of these sets. Suppose to the contrary that none of the sets from Equation (3) satisfies the desired conditions. We will use a triangle-inequality argument on the character frequencies of B to derive a contradiction. Let M and m be indices such that σ_M is the most frequent character of B and σ_m is the least frequent. Since B is not ρ-balanced, we know that $M \neq m$. If $M \geq j > m$ we may write

$$|\sigma_M(B) - \sigma_m(B)| \leq |\sigma_M(B) - \sigma_{j-1}(B)| + |\sigma_{j-1}(B) - \sigma_j(B)| + |\sigma_j(B) - \sigma_m(B)|.$$

By assumption, the restrictions of B to the sets $\{\sigma_M, \sigma_{j-1}\}$, $\{\sigma_j, \sigma_{j-1}\}$, and $\{\sigma_j, \sigma_m\}$ are (ρ/s)-balanced. Thus, each addend on the right hand side of the above inequality is bounded above by $(\rho/s)n$. It follows that

$$|\sigma_M(B) - \sigma_m(B)| \leq (3\rho/s)n.$$

By similar reasoning, if $m = j$ and $M \geq j$ then we have

$$|\sigma_M(B) - \sigma_m(B)| = |\sigma_M(B) - \sigma_j(B)| \leq |\sigma_M(B) - \sigma_{j-1}(B)| + |\sigma_{j-1}(B) - \sigma_j(B)| \leq (2\rho/s)n.$$

If instead $M, m > j$ we know that

$$|\sigma_M(B) - \sigma_m(B)| \leq |\sigma_M(B) - \sigma_{j-1}(B)| + |\sigma_{j-1}(B) - \sigma_m(B)| \leq (2\rho/s)n$$

since now the subalphabets $\{\sigma_M, \sigma_{j-1}\}$ and $\{\sigma_m, \sigma_{j-1}\}$ both occur in Equation (3). Similar reasoning on the remaining cases of the values of M and m relative to j to establishes the inequality

$$\sigma_M(B) - \sigma_m(B) \leq (3\rho/s)n.$$

We have dropped absolute value signs because the left hand side of the above equation is positive by definition of M and m. Since $s \geq 3$ this contradicts the assumption that B is not ρ-balanced, and the desired result follows. Note that we can find which of subalphabets from Equation (3) satisfies the conditions of the lemma in $O(n + s)$ time by scanning through B to get the counts of each of its characters and trying out all the restrictions. ◀

In our proof of Theorem 2, we will require a stronger form of the result from Section 4 of [21]. This variant of their theorem is useful because it applies to strings of different length.

▶ **Lemma 13.** *Let X and Y be binary strings of length n and m respectively, where $m \leq n$. Suppose the frequencies $0(Y)$ and $1(X)$ are both at most $(1/2-\rho)m$ for some positive constant ρ. Then there exist positive constants δ and ϵ such that if $0(Y)$ and $1(X)$ are within δm of each other, we can compute a $(1/2+\epsilon)$ approximation of $\mathrm{LCS}(X, Y)$ in subquadratic time.*

Although the realization that this type of result holds for strings of differing length is novel, the proof of Lemma 13 itself is conceptually identical to the frequency analysis used in [21], requiring only minor changes to make the argument go through. For completeness we include the detailed casework proof of this result in Section 4.

Finally we apply a lemma that provides some frequency information about the strings.

▶ **Lemma 14** (Lemma 3.1 from [21])**.** *For any $\delta > 0$ if*

$$\min(0(X), 0(Y)) > (1+\delta)\min(1(X), 1(Y))$$

or

$$\min(1(X), 1(Y)) > (1+\delta)\min(0(X), 0(Y))$$

then there is a unary $(1+\delta)/(2+\delta)$ approximation for $\mathrm{LCS}(X, Y)$.

Proof. Observe that for any binary strings X and Y we have

$$\mathrm{LCS}(X, Y) \leq \min(0(X), 0(Y)) + \min(1(X), 1(Y)). \tag{4}$$

This equation holds because the LCS is a subsequence of X and Y, and thus cannot contain more zeros or ones than either of the strings X or Y individually have.

Suppose by symmetry that $\min(0(X), 0(Y))$ is the larger of the two addends on the right. Then the we can return the all zeros string of this length. By the first inequality we get

$$\min(0(X), 0(Y)) > (1+\delta)(\mathrm{LCS}(X, Y) - \min(0(X), 0(Y)))$$

and then rearranging proves the claim. ◀

We now combine these results to improve LCS approximation on all alphabets.

Proof of Theorem 8. Let ρ and ρ' be positive parameters whose values will be specified later. If either of the strings A or B are ρs-balanced, we are done by Lemma 10 (by taking ρ to be small enough so that the lemma applies). If the LCS of A and B is not ρ'-balanced we are done by Lemma 11. So, we may assume that neither A nor B are ρs-balanced and that their LCS is ρ'-balanced.

By Lemma 12 we can find binary alphabet restrictions X and Y of A and B respectively with the property that neither X nor Y are ρ-balanced. Informally, since the LCS of A and B is balanced, a better than $1/2$ approximation for the LCS of X and Y acts as a better than $1/s$ approximation for $\text{LCS}(A, B)$. More precisely, since $\text{LCS}(A, B)$ is ρ'-balanced, each of its characters occurs at least $(1/s - \rho')\,\text{LCS}(A, B)$ times in the LCS. By construction, we also know that $\text{LCS}(X, Y)$ is at least as large as a binary alphabet restriction of $\text{LCS}(A, B)$. Thus, it follows that given a positive constant ϵ', a $1/2 + \epsilon'$ approximation of $\text{LCS}(X, Y)$ has length at least

$$(1/2 + \epsilon') \cdot 2\,(1/s - \rho')\,\text{LCS}(A, B) = (1/s + 2\epsilon'/s - \rho' - 2\epsilon'\rho')\,\text{LCS}(A, B).$$

By setting $\epsilon = \epsilon'/s$ (for example) and ρ' sufficiently small in terms of ϵ', the above calculation shows that a $1/2 + \epsilon'$ approximation for the LCS of X and Y acts as a $1/s + \epsilon$ approximation for $\text{LCS}(A, B)$. Hence to complete the proof, it suffices to get a better than $1/2$ approximation for $\text{LCS}(X, Y)$. We may assume that

$$\min(0(X), 0(Y)) \leq (1 + \delta)\min(1(X), 1(Y)) \tag{5}$$

and

$$\min(1(X), 1(Y)) \leq (1 + \delta)\min(0(X), 0(Y)) \tag{6}$$

for some constant δ since otherwise Lemma 14 yields a better than $1/2$ approximation.

As mentioned previously, [21] gives a better than $1/2$ approximation for the LCS when X and Y have equal length. If X and Y have different lengths, without loss of generality we assume that $|X| > |Y|$ and $0(Y) \leq 1(Y)$. We do casework on frequencies in X, relative to the frequencies in Y.

Since X is longer than Y, we cannot have both $1(X) \leq 1(Y)$ and $0(X) \leq 0(Y)$.

If $1(X) > 1(Y)$ and $0(X) > 0(Y)$ simultaneously, then Equation (6) implies that

$$1(Y) \leq (1 + \delta) \cdot 0(Y)$$

which contradicts the fact that Y is not ρ-balanced as long as we take $\delta \leq 2\rho$.

If instead $1(X) \leq 1(Y)$ and $0(X) \leq 0(Y)$, by Equation (6) we similarly have

$$0(X) \leq 0(Y) \leq 1(Y) \leq (1 + \delta) \cdot 0(X)$$

which again contradicts the fact that Y is not ρ-balanced for the same choice of δ.

Thus, the only possibility is that $1(X) \leq 1(Y)$ and $0(X) > 0(Y)$. Let $m = |Y|$ be the length of string Y. Then Equation (5) implies that

$$0(Y) \leq (1 + \delta) \cdot 1(X) \leq 1(X) + \delta m$$

while Equation (6) implies that

$$1(X) \leq (1 + \delta) \cdot 0(Y) \leq 0(Y) + \delta m.$$

Consequently, $0(Y)$ and $1(X)$ are within δm of each other. Then by Lemma 13, as long as we take δ small enough in terms of ρ we get a subquadratic $1/2 + \epsilon'$ approximation for $\text{LCS}(X, Y)$. As noted earlier, this then yields the desired $1/s + \epsilon$ approximation for $\text{LCS}(A, B)$ in truly subquadratic time. In fact, because we only ever use subroutines that run in linear time or constant factor approximations to edit distance which take near-linear time, the overall algorithm takes near-linear time. ◀

4 Proof of Lemma 13

This section is devoted to proving Lemma 13. We do this working through the individual arguments in the case analysis of [21] and verifying that the arguments still hold in the case where the strings have different lengths, as long as they satisfy the frequency requirements included in the hypotheses of the lemma. Throughout this section we fix the binary alphabet $\Sigma = \{0, 1\}$ and assume that all strings come from this alphabet.

We carry over the following subroutines from [21].

▶ **Definition 15.** *Given strings A and B and a symbol σ, the algorithm* $\mathrm{Match}(A, B, \sigma)$ *returns the largest subsequence of A and B consisting entirely of copies of σ.*

▶ **Definition 16.** *Given strings A and B, the algorithm* $\mathrm{BestMatch}(A, B)$ *returns the longer of the strings*

$$\mathrm{Match}(A, B, 0) \quad \textit{and} \quad \mathrm{Match}(A, B, 1).$$

In other words, the algorithm returns the largest common unary subsequence.

Note that BestMatch is a $1/2$ approximation algorithm for LCS.

▶ **Definition 17.** *Given strings A_1, A_2, and B, the algorithm* $\mathrm{Greedy}(A_1, A_2, B)$ *returns*

$$\max_{B=B_1 \sqcup B_2} \mathrm{BestMatch}(A_1, B_1) + \mathrm{BestMatch}(A_2, B_2)$$

taken over all possible splits of B into two contiguous left and right substrings B_1 and B_2.

These algorithms can all be implemented to run in linear time by scanning through the counting how many 0s and 1s appear in each string. The final procedure we will invoke utilizes an approximation algorithm for computing the edit distance of two strings as a black-box. Although any fast enough constant factor approximation for edit distance will work, for concreteness we assume that it leverages the algorithm $\widetilde{\mathrm{ED}}$ from [5], which runs in subquadratic time and can approximate the edit distance to a $c = 3 + \epsilon'$ factor for any fixed positive constant ϵ'. Note that this algorithm can also return a common subsequence achieving the given length.

Proof of Lemma 13. Let $1(X) = \alpha m$ for some $\alpha \in [0, 1]$. By assumption, we know that $\alpha < 1/2$ is bounded away from $1/2$ by some constant amount. We can also assume that $0(Y)$ is within δm of $1(X)$ for some positive parameter $\delta < 0$ to be picked later on. We will end up setting δ to be some sufficiently small constant depending on α. We will use the notation $\approx$ to denote quantities that are within δm of each other. For example $1(X) \approx 0(Y)$. This lets us avoid having to stick in $\pm\delta m$ symbols in all the inequalities and helps make the arguments cleaner without affecting the correctness (since we just care about getting a $1/2 + \epsilon$ approximation for *some* constant $\epsilon > 0$).

Note that by Equation (4) the LCS of the input strings

$$\mathrm{LCS}(X, Y) \leq (2\alpha + \delta)m \tag{7}$$

cannot be too large, because the LCS contain at most αm 1s from X and $\approx \alpha m$ 0s from Y.

Let R_X and R_Y denote the substrings of X and Y consisting of their rightmost αm characters respectively. Similarly define L_X and L_Y as the substrings of X and Y consisting of the leftmost αm characters. Set $M_X = X \setminus (L_X \cup R_X)$ and $M_Y = Y \setminus (L_Y \cup R_Y)$ to be the middle substrings of X and Y that remain when these left and right ends are chopped off.

We now follow the casework of [21], explaining at each step why the arguments still hold in our more general setting. These cases are based off the frequencies of 0s and 1s in the left and right ends of the inputs, and consider separately the situation where these substrings are pseudorandom (balanced) or structured (imbalanced). In the former case we can appeal to edit distance as in the proof of Lemma 10, and in the latter situation we can exploit the imbalance in the strings to use the simpler unary algorithms described in definitions 15, 16, and 17. Intuitively, we succeed in using edit distance approximation arguments (and overcome the barrier described the end of Section 2) in this particular case because even though X and Y may have different size, we identify right and left substrings which all have equal length and only employ edit distance approximation around these areas.

Recall that we assumed that $\alpha < 1/2 - \rho$ for some constant ρ. Take a parameter $\beta < \rho/20$ to represent deviation from the balanced case. By choosing δ sufficiently small in terms of β we may additionally assume that

$$1(X), 0(Y) < (1/2 - 10\beta)\, m \tag{8}$$

since in the hypothesis of the lemma we supposed that $1(X) < (1/2 - \rho)m$ and that $0(Y) \approx 1(X)$. Finally, as one last piece of notation, we write $|A|$ to denote length of an arbitrary string A. We now begin the casework, maintaining consistency with [21].

Case 1(a): $1(R_Y), 0(R_X) \in [(\alpha/2 - 4\beta)\, m, (\alpha/2 + 4\beta)\, m]$

In this case both right ends of the strings are balanced. We will give a good approximation for the LCS by splitting these strings at the right ends and using the aforementioned algorithms.

Consider an optimal alignment between X and Y (i.e. a maximum partial matching of identical characters in X and Y, corresponding to the LCS of X and Y). Let $\widehat{R_Y}$ be the minimal suffix (from the right end) of the string Y with the property that every character from R_X which is matched in the alignment is paired up with some character in $\widehat{R_Y}$.

Without loss of generality, we may assume that $\widehat{R_Y}$ is a substring of R_Y. This is because if $\widehat{R_Y}$ was not contained in R_Y, we could define an analogous substring of $\widehat{R_X}$ of X satisfying $\widehat{R_X} \subseteq R_X$ and then use a symmetric argument to get the desired approximation. Let $\widehat{L_Y} = Y \setminus \widehat{R_Y}$ be the left substring of Y that remains after chopping of $\widehat{R_Y}$.

Optimality of the alignment implies that

$$\mathrm{LCS}(X, Y) = \mathrm{LCS}(X \setminus R_X, \widehat{L_Y}) + \mathrm{LCS}(R_X, \widehat{R_Y}). \tag{9}$$

Define the quantities

$$f_L = \min(1(X \setminus R_X), 1(\widehat{L_Y})) + \min(0(X \setminus R_X), 0(\widehat{L_Y}))$$

and

$$f_R = \min(1(R_X), 1(\widehat{R_Y})) + \min(0(R_X), 0(\widehat{R_Y}))$$

which represent frequency-based upper bounds for the LCS terms from the right hand side of Equation (9). They are useful because Equation (4) together with Equation (9) implies that

$$\mathrm{LCS}(X, Y) \leq f_L + f_R. \tag{10}$$

We also introduce the quantity

$$Z = \max\left(\min(1(X \setminus R_X), 1(\widehat{L_Y})), \min(0(X \setminus R_X), 0(\widehat{L_Y}))\right)$$

which is the larger of the two addends defining f_L, and equal to the length of the string returned by

$$\mathrm{BestMatch}(X \setminus R_X, \widehat{L_Y}).$$

We further subdivide into cases based off the size of Z.

Case 1(a)(i): $Z > (\alpha/2 + 10\beta)\, m$

When Z is large we can combine two unary subsequences to get a good enough LCS approximation. We first show that Z is bigger than $f_L/2$ by a constant fraction of m.

By definition we have

$$f_L - Z = \min\left(\min(1(X \setminus R_X), 1(\widehat{L_Y})), \min(0(X \setminus R_X), 0(\widehat{L_Y}))\right) \leq 1(X \setminus R_X).$$

Since X has αm ones and R_X has length αm we get that

$$1(X \setminus R_X) = \alpha m - 1(R_X) = \alpha m - (\alpha m - 0(R_X)) = 0(R_X).$$

Then using the case assumptions we have

$$0(R_X) \leq (\alpha/2 + 4\beta)\, m < Z - 6\beta m.$$

Chaining these inequalities together and rearranging we deduce that

$$Z > f_L/2 + 3\beta m.$$

Now if we make a single call to the Greedy routine we obtain a string of length

$$\text{Greedy}(X \setminus R_X, R_X, Y) \geq \text{BestMatch}(X \setminus R_X, \widehat{L_Y}) + \text{BestMatch}(R_X, \widehat{R_Y}). \tag{11}$$

From our earlier discussion we have

$$\text{BestMatch}(X \setminus R_X, \widehat{L_Y}) \geq Z > f_L/2 + 3\beta m.$$

Moreover

$$\text{BestMatch}(R_X, \widehat{R_Y}) = \max(\min(1(R_X), 1(\widehat{R_Y})), \min(0(R_X), 0(\widehat{R_Y}))) \geq f_R/2$$

since we are taking the maximum over two addends that sum to f_R.

Finally, if we substitute the above inequalities into Equation (11) and apply Equation (10) we get that

$$\begin{aligned}\text{Greedy}(X \setminus R_X, R_X, Y) &\geq (f_L + f_R)/2 + 3\beta m \geq \text{LCS}(X, Y)/2 + 3\beta m \\ &\geq (1/2 + 3\beta)\, \text{LCS}(X, Y).\end{aligned}$$

Thus running Greedy gives us a better than $1/2$ approximation in this case.

Case 1(a)(ii): $Z \leq (\alpha/2 + 10\beta)\, m$

Intuitively, in this case Z is too small for us to ensure a good approximation using frequency guarantees alone. However, because Z is so small, the LCS also cannot be too large. Because of this, we will be able to get a good approximation by combining a common subsequence of the (balanced) right ends of the strings with a common subsequence of the strings with the right ends removed.

More concretely, we leverage the ApproxED algorithm from Definition 7. From our previous observation we have $f_L \leq 2Z \leq (\alpha + 20\beta)\, m$. So via Equation (9) we can bound the LCS by

$$\text{LCS}(X, Y) \leq f_L + \text{LCS}(R_X, \widehat{R_Y}) \leq (\alpha + 20\beta)\, m + \text{LCS}(R_X, R_Y) \tag{12}$$

where in the last step we also used the fact that $\widehat{R_Y} \subseteq R_Y$.

We will get an approximation by returning a unary subsequence from the left parts of the strings, and using edit distance approximation on the right ends. First, we can make a call to BestMatch and get a common subsequence of length at least

$$\text{BestMatch}(X \setminus R_X, Y \setminus R_Y) \geq \text{Match}(X \setminus R_X, Y \setminus R_Y, 0) \geq (\alpha/2 - 4\beta)\, m.$$

This last inequality above follows from combining the case 1 assumptions about the frequencies of characters in $0(R_X)$ and $1(R_Y)$, together with the facts that $0(Y) \approx 1(X) = \alpha m$ and $|R_X| = |R_Y| = \alpha m$.

Now, since R_X and R_Y are $(4\beta/\alpha)$-balanced we can apply Lemma 10 with $n = \alpha m$ as long as we take β sufficiently small in terms of α. Note that for fixed alphabet size and ρ, the parameter β remains $\Omega(1)$. This ensures that in subquadratic time we can compute a common subsequence of R_X and R_Y with length at least

$$(1/2 + \gamma)\, \text{LCS}(R_X, R_Y)$$

where $\gamma < 1$ is the constant from Lemma 10, which is larger for smaller values of β. By the frequency assumptions on R_X and R_Y, we know that

$$\text{LCS}(R_X, R_Y) \geq (\alpha/2 - 4\beta)\, m$$

so in fact the subsequence of R_X and R_Y returned by Lemma 10 has length at least

$$(1/2 + \gamma)\, \text{LCS}(R_X, R_Y) \geq \text{LCS}(R_X, R_Y)/2 + \gamma(\alpha/2 - 4\beta)m.$$

Now if we combine these two subsequences together, we get a common subsequence of X and Y of length at least

$$\text{LCS}(R_X, R_Y)/2 + (\alpha/2 - 4\beta)\, m + \gamma(\alpha/2 - 4\beta)m$$

which can be written in the form

$$((\alpha + 20\beta)m + \text{LCS}(R_X, R_Y))\,/2 + (\gamma\alpha/2 - 14\beta - 4\gamma\beta)\, m.$$

By applying Equation (12) and the fact that $\gamma < 1$ we see that this is at least

$$\text{LCS}(X, Y)/2 + (\gamma\alpha/2 - 18\beta)\, m.$$

Finally, by picking β small enough this expression is at least

$$\text{LCS}(X, Y)/2 + (\gamma\alpha/3)\, m \geq (1/2 + \gamma/6)\, \text{LCS}(X, Y)$$

where in the last step we have used Equation (4) together with $1(X) = \alpha m$ and $0(Y) \approx \alpha m$. This proves that we can attain a better than 1/2 approximation for the LCS as claimed.

Case 1(b): $1(R_Y) < (\alpha/2 - 4\beta)\, m$ and $0(R_X) \leq (\alpha/2 + 2\beta)\, m$

In this case the right ends of X and Y each do not contain too much their respective strings' most common characters. We show that this implies the LCS of both strings must be so small that simply returning a unary string yields a better than 1/2 approximation.

As in case 1(a), consider an optimal alignment between X and Y. Now define the substring $\widehat{R_Y}$ to be the minimal suffix of Y which contains all characters of Y that R_X is aligned to. Let $\widehat{L_Y} = Y \setminus \widehat{R_Y}$ be what remains of Y after R_Y is removed. Since the alignment is optimal, Equation (9) holds.

We now subdivide into further case based off how the right ends of strings are aligned.

Case 1(b)(i): every character of R_X is matched to some character of R_Y in the alignment

In this case $\widehat{R_Y}$ is a substring of R_Y. From Equation (4) and the case assumptions we get that

$$\mathrm{LCS}(X \setminus R_X, \widehat{L_Y}) \leq 1(X \setminus R_X) + 0(\widehat{L_Y}) \leq (\alpha/2 + 2\beta)\, m + 0(\widehat{L_Y})$$

and

$$\mathrm{LCS}(R_X, \widehat{R_Y}) \leq 1(\widehat{R_Y}) + 0(\widehat{R_Y}) \leq (\alpha/2 - 4\beta)\, m + 0(\widehat{R_Y}).$$

Note that in this second inequality we are using the inequality $1(\widehat{R_Y}) \leq 1(R_Y)$ which follows from the earlier observation that $\widehat{R_Y} \subseteq R_Y$. By adding these inequalities together and substituting the result into Equation (9) we deduce that

$$\mathrm{LCS}(X, Y) \leq (\alpha - 2\beta)\, m + \left(0(\widehat{L_Y}) + 0(\widehat{R_Y})\right) \leq (\alpha - 2\beta)\, m + \alpha m = (2\alpha - 2\beta)\, m$$

where we have used the fact that 0 occurs in $Y \approx \alpha m$ times.

Case 1(b)(ii): some character of R_X is matched outside R_Y in the alignment

In this case R_Y is a substring of $\widehat{R_Y}$.

Using Equation (4) again we find that

$$\mathrm{LCS}(X \setminus R_X, \widehat{L_Y}) \leq 1(X \setminus R_X) + 0(\widehat{L_Y}) \tag{13}$$

From $0(R_X) \leq (\alpha/2 + 2\beta)\, m$ we know that $1(R_X) \geq (\alpha/2 - 2\beta)\, m$ since R_X has length αm. It follows that

$$1(X \setminus R_X) = 1(X) - 1(R_X) \leq (\alpha/2 + 2\beta)\, m$$

since $1(X) = \alpha m$.

Similarly, since $1(R_Y) \leq (\alpha/2 - 4\beta)\, m$ and R_Y has length αm we know that $0(R_Y) > (\alpha/2 + 4\beta)\, m$. Since $R_Y \subseteq \widehat{R_Y}$ it must be the case that $0(\widehat{R_Y}) > (\alpha/2 + 4\beta)\, m$ which means that

$$0(\widehat{L_Y}) = 0(Y) - 0(\widehat{R_Y}) \approx \alpha m - 0(\widehat{R_Y}) < (\alpha/2 - 4\beta)\, m.$$

Adding these two inequalities and substituting into Equation (13) proves that

$$\mathrm{LCS}(X \setminus R_X, \widehat{L_Y}) \leq (\alpha - 2\beta)\, m.$$

Then applying Equation (9) and using the fact that R_X has length αm proves that

$$\mathrm{LCS}(X, Y) = \mathrm{LCS}(X \setminus R_X, \widehat{L_Y}) + \mathrm{LCS}(R_X, \widehat{R_Y}) \leq (\alpha - 2\beta)m + \alpha m = (2\alpha - 2\beta)\, m.$$

Thus in both this subcase and the previous one, the LCS is at most $(2\alpha - 2\beta)\, m$. Hence, returning the string of $\approx \alpha m$ zeros obtained by calling $\mathrm{Match}(X, Y, 0)$ gives a better than $1/2$ approximation as desired.

Case 1(c): $1(R_Y) \leq (\alpha/2 + 2\beta)\, m$ and $0(R_X) < (\alpha/2 - 4\beta)\, m$

This case is symmetric to case 1(b) and similar reasoning handles it.

Case 2: $1(L_Y), 0(L_X) \leq (\alpha/2 + 2\beta)\, m$
Combining cases 1(a), 1(b), and 1(c) resolves the situation where

$$1(R_Y), 0(R_X) \leq \alpha/2 + 2\beta.$$

Consequently, case 2 is symmetric to case 1. In particular, we can flip the strings (by replacing X with the string X' which consists of the symbols of X read in reverse from right to left, and replacing Y with its analogous reverse string Y') and apply the arguments from case 1 to handle this case.

Case 3: $1(L_Y), 1(R_Y) \leq (\alpha/2 + \beta)\, m$ and $0(L_X), 0(R_X) > (\alpha/2 + 2\beta)\, m$
In this case both ends of the inputs have many 0s, so repeated calls to Match will be enough to guarantee a better than 1/2 approximation.

Since L_Y has length αm and at most $(\alpha/2 + \beta)\, m$ instances of 1, it must have at least

$$0(L_Y) \geq \alpha m - (\alpha/2 + \beta)m = (\alpha/2 - \beta)m$$

occurrences of 0. Since $1(R_Y) \leq (\alpha/2 + \beta)\, m$ the same reasoning shows that

$$0(R_Y) \geq (\alpha/2 - \beta)\, m.$$

Consequently we can get common subsequences of the left and right ends of the strings consisting entirely of 0s with lengths

$$\text{Match}(L_X, L_Y, 0) = \min(0(L_X), 0(L_Y)) \geq (\alpha/2 - \beta)\, m$$

and

$$\text{Match}(R_X, R_Y, 0) = \min(0(R_X), 0(R_Y)) \geq (\alpha/2 - \beta)\, m.$$

Since the ends have many 0s, we expect the middle substrings to have many 1s. Indeed, since the left end of X

$$1(L_X) = |L_X| - 0(L_X) < \alpha m - (\alpha/2 + 2\beta)m = (\alpha/2 - 2\beta)\, m$$

does not have many 1s and similar reasoning shows that

$$1(R_X) < (\alpha/2 - 2\beta)\, m$$

this holds for the right end as well, the middle of X has only

$$1(M_X) = 1(X) - 1(L_X) - 1(R_X) > \alpha m - 2(\alpha/2 - 2\beta)m = 4\beta m$$

appearances of 1.

To argue that M_Y has enough 1s we will need to appeal to the fact that Y is not balanced. From Equation (8) we know that

$$1(Y) = m - 0(Y) > m - (1/2 - 10\beta)m = (1/2 + 10\beta)\, m.$$

We can use the case assumptions together $\alpha < 1/2$ to then deduce

$$1(M_Y) = 1(Y) - 1(L_Y) - 1(R_Y) > (\alpha + 10\beta)m - 2(\alpha/2 + \beta)m = 8\beta m.$$

Consequently the largest all 1s subsequence between the middle substrings has length at least

$$\text{Match}(M_X, M_Y, 1) = \min(1(M_X), 1(M_Y)) \geq 4\beta m.$$

Thus, by combining these three subsequences from three calls to match we can recover in linear time a common subsequence of size

$$\text{Match}(L_X, L_Y, 0) + \text{Match}(M_X, M_Y, 1) + \text{Match}(R_X, R_Y, 0) > 2(\alpha/2 - \beta)m + 4\beta m = (\alpha + 2\beta)m.$$

By Equation (7) the true LCS has length at most $\approx 2\alpha m$, so a string of length $(\alpha + 2\beta)\, m$ is a $1/2 + \epsilon$ approximation for some constant $\epsilon < 2\beta/\alpha$ as desired.

Case 4: $0(L_X), 0(R_X) \leq (\alpha/2 + \beta)\, m$ and $1(L_Y), 1(R_Y) > (\alpha/2 + 2\beta)\, m$

This case is symmetric to case 3 and similar reasoning handles it.

Case 5: $0(L_X), 1(R_Y) > (\alpha/2 + \beta)\, m$

In this case, the ends of the strings have unusually large instances of either 0 or 1. This will enable us to combine two calls to Match to get the desired approximation.

We can check that the right end of Y does not have many 0s

$$0(R_Y) = |R_Y| - 1(R_Y) < \alpha m - (\alpha/2 + \beta)m = (\alpha/2 - \beta)\, m.$$

It follows that the remainder of the string $Y \setminus R_Y = L_Y \cup M_Y$ has many zeros

$$0(Y \setminus R_Y) = 0(Y) - 0(R_Y) \approx \alpha m - 0(R_Y) > (\alpha/2 + \beta)\, m.$$

Similar reasoning shows that

$$1(L_X) = \alpha m - 0(L_X) < (\alpha/2 - \beta)\, m$$

so that the remaining portion of X has many ones

$$1(X \setminus L_X) = 1(X) - 1(L_X) = \alpha m - 1(L_X) > (\alpha/2 + \beta)\, m.$$

It follows that

$$\text{Match}(L_X, Y \setminus R_Y, 0) + \text{Match}(X \setminus L_X, R_Y, 1) > (\alpha + 2\beta)\, m$$

yields a better than $1/2$ approximation since by Equation (7) the LCS is at most $\approx 2\alpha m$.

Case 6: $1(L_Y), 0(R_X) > (\alpha/2 + \beta)\, m$

This is symmetric to case 5 and a similar argument proves the result holds in this situation.

By inspection or by referring to Table 1 of [21], we can verify that these cases handle all possible input strings satisfying the conditions of the lemma. Since in every case we obtain a better than $1/2$ approximation for the LCS in subquadratic time, we have proven the claim. ◀

5 Open Problems

Theorem 2 of our work shows how to obtain better than $1/|\Sigma|$ approximations for the longest common subsequence of *equal-length strings* over an alphabet Σ. It remains an open problem to get such approximations in the setting where the input strings have different length (and by Corollary 5, to solve this problem it suffices to obtain an improvement in the setting of binary alphabets, where $|\Sigma| = 2$).

Additionally, although our algorithm beats the longstanding trivial approximation ratio for LCS, it does so only by a modest amount. It would be interesting to get better approximation ratios, both in terms of their concrete value for small $|\Sigma|$ and in terms of their growth as a function of the alphabet size.

References

1. Amir Abboud and Arturs Backurs. Towards hardness of approximation for polynomial time problems. In *8th Innovations in Theoretical Computer Science Conference, ITCS 2017, January 9-11, 2017, Berkeley, CA, USA*, volume 67 of *LIPIcs*, pages 11:1–11:26, 2017.
2. Amir Abboud, Arturs Backurs, and Virginia Vassilevska Williams. Tight hardness results for LCS and other sequence similarity measures. In *IEEE 56th Annual Symposium on Foundations of Computer Science, FOCS 2015, Berkeley, CA, USA, 17-20 October, 2015*, pages 59–78. IEEE Computer Society, 2015.
3. Amir Abboud, Thomas Dueholm Hansen, Virginia Vassilevska Williams, and Ryan Williams. *Simulating Branching Programs with Edit Distance and Friends: Or: A Polylog Shaved is a Lower Bound Made*, page 375–388. Association for Computing Machinery, New York, NY, USA, 2016. doi:10.1145/2897518.2897653.
4. Amir Abboud and Aviad Rubinstein. Fast and deterministic constant factor approximation algorithms for LCS imply new circuit lower bounds. In Anna R. Karlin, editor, *9th Innovations in Theoretical Computer Science Conference, ITCS 2018, January 11-14, 2018, Cambridge, MA, USA*, volume 94 of *LIPIcs*, pages 35:1–35:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2018.
5. Alex Andoni. Simpler constant-factor approximation to edit distance problems, 2019.
6. Alexandr Andoni, Robert Krauthgamer, and Krzysztof Onak. Polylogarithmic approximation for edit distance and the asymmetric query complexity. In *51th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2010, October 23-26, 2010, Las Vegas, Nevada, USA*, pages 377–386. IEEE Computer Society, 2010.
7. Alexandr Andoni and Negev Shekel Nosatzki. Edit distance in near-linear time: it's a constant factor. *CoRR*, abs/2005.07678, 2020. arXiv:2005.07678.
8. Alexandr Andoni and Krzysztof Onak. Approximating edit distance in near-linear time. *SIAM J. Comput.*, 41(6):1635–1648, 2012.
9. Arturs Backurs and Piotr Indyk. Edit distance cannot be computed in strongly subquadratic time (unless SETH is false). *SIAM J. Comput.*, 47(3):1087–1097, 2018.
10. Ziv Bar-Yossef, T. S. Jayram, Robert Krauthgamer, and Ravi Kumar. Approximating edit distance efficiently. In *45th Symposium on Foundations of Computer Science (FOCS 2004), 17-19 October 2004, Rome, Italy, Proceedings*, pages 550–559. IEEE Computer Society, 2004.
11. Tugkan Batu, Funda Ergün, and Süleyman Cenk Sahinalp. Oblivious string embeddings and edit distance approximations. In *Proceedings of the Seventeenth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2006, Miami, Florida, USA, January 22-26, 2006*, pages 792–801. ACM Press, 2006.
12. Mahdi Boroujeni, Soheil Ehsani, Mohammad Ghodsi, Mohammad Taghi Hajiaghayi, and Saeed Seddighin. Approximating edit distance in truly subquadratic time: Quantum and mapreduce. In Artur Czumaj, editor, *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2018, New Orleans, LA, USA, January 7-10, 2018*, pages 1170–1189. SIAM, 2018.

13 Joshua Brakensiek and Aviad Rubinstein. Constant-factor approximation of near-linear edit distance in near-linear time. In *Proceedings of the 52nd Annual ACM SIGACT Symposium on Theory of Computing*, STOC 2020, page 685–698, New York, NY, USA, 2020. Association for Computing Machinery. doi:10.1145/3357713.3384282.

14 Karl Bringmann and Marvin Künnemann. Multivariate fine-grained complexity of longest common subsequence. In *SODA*, 2018.

15 Diptarka Chakraborty, Debarati Das, Elazar Goldenberg, Michal Koucký, and Michael E. Saks. Approximating edit distance within constant factor in truly sub-quadratic time. In Mikkel Thorup, editor, *59th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2018, Paris, France, October 7-9, 2018*, pages 979–990. IEEE Computer Society, 2018.

16 Lijie Chen, Shafi Goldwasser, Kaifeng Lyu, Guy N. Rothblum, and Aviad Rubinstein. Fine-grained complexity meets IP = PSPACE. In Timothy M. Chan, editor, *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2019, San Diego, California, USA, January 6-9, 2019*, pages 1–20. SIAM, 2019.

17 MohammadTaghi Hajiaghayi, Masoud Seddighin, Saeed Seddighin, and Xiaorui Sun. Approximating LCS in linear time: Beating the $\sqrt{n}$ barrier. In *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2019, San Diego, California, USA, January 6-9, 2019*, pages 1181–1200. SIAM, 2019.

18 Michal Koucký and Michael Saks. Constant factor approximations to edit distance on far input pairs in nearly linear time. In *Proceedings of the 52nd Annual ACM SIGACT Symposium on Theory of Computing*, STOC 2020, pages 699–712, New York, NY, USA, 2020. Association for Computing Machinery. doi:10.1145/3357713.3384307.

19 Gad M. Landau, Eugene W. Myers, and Jeanette P. Schmidt. Incremental string comparison. *SIAM Journal on Computing*, 27(2):557–582, 1998. doi:10.1137/s0097539794264810.

20 William J. Masek and Mike Paterson. A faster algorithm computing string edit distances. *J. Comput. Syst. Sci.*, 20(1):18–31, 1980.

21 Aviad Rubinstein and Zhao Song. Reducing approximate longest common subsequence to approximate edit distance. In *Proceedings of the Thirty-First Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '20, page 1591–1600, USA, 2020. Society for Industrial and Applied Mathematics.

Efficient Splitting of Necklaces

Noga Alon
Department of Mathematics, Princeton University, NJ, USA
Schools of Mathematics and Computer Science, Tel Aviv University, Israel

Andrei Graur
Department of Management Science and Engineering, Stanford University, CA, USA

Abstract

We provide efficient approximation algorithms for the Necklace Splitting problem. The input consists of a sequence of beads of n types and an integer k. The objective is to split the necklace, with a small number of cuts made between consecutive beads, and distribute the resulting intervals into k collections so that the discrepancy between the shares of any two collections, according to each type, is at most 1. We also consider an approximate version where each collection should contain at least a $(1-\varepsilon)/k$ and at most a $(1+\varepsilon)/k$ fraction of the beads of each type. It is known that there is always a solution making at most $n(k-1)$ cuts, and this number of cuts is optimal in general. The proof is topological and provides no efficient procedure for finding these cuts. It is also known that for $k=2$, and some fixed positive ε, finding a solution with n cuts is PPAD-hard.

We describe an efficient algorithm that produces an ε-approximate solution for $k=2$ making $n(2+\log(1/\varepsilon))$ cuts. This is an exponential improvement of a $(1/\varepsilon)^{O(n)}$ bound of Bhatt and Leighton from the 80s. We also present an online algorithm for the problem (in its natural online model), in which the number of cuts made to produce discrepancy at most 1 on each type is $\tilde{O}(m^{2/3}n)$, where m is the maximum number of beads of any type. Lastly, we establish a lower bound showing that for the online setup this is tight up to logarithmic factors. Similar results are obtained for $k>2$.

2012 ACM Subject Classification Theory of computation → Approximation algorithms analysis

Keywords and phrases necklace splitting, necklace halving, approximation algorithms, online algorithms, discrepancy

Digital Object Identifier 10.4230/LIPIcs.ICALP.2021.14

Category Track A: Algorithms, Complexity and Games

Related Version *Full Version*: https://arxiv.org/abs/2006.16613

Funding *Noga Alon*: Research supported in part by NSF grant DMS-1855464, BSF grant 2018267 and the Simons Foundation.

1 Introduction

1.1 The problems

The Necklace Splitting problem deals with a fair partition of a necklace with beads of n colors among k agents. The objective is to cut the necklace into intervals and distribute them to the agents in an equitable way. Before adding more on the background, we give the formal definition of the problem.

▶ **Definition 1** (Necklace Splitting). *An instance of Necklace Splitting for n colors and k agents consists of a set of beads ordered along a line, where each bead is colored by a color $i \in [n] = \{1, 2, \ldots, n\}$. The goal is to split the necklace, via at most $n(k-1)$ cuts made between consecutive beads, into intervals and distribute them to the k agents so that for each color i, every agent gets either $\lceil \frac{m_i}{k} \rceil$ or $\lfloor \frac{m_i}{k} \rfloor$ beads of color i, where m_i is the number of beads of color i.*

Note that this definition is slightly broader than the one given in [1], where it is assumed that m_i is divisible by k for all $i \in [n]$. However, as shown in [4], these two forms of the Necklace Splitting problem are equivalent. We call the special case $k = 2$ of two agents the Necklace Halving problem. A related problem is the ε-Consensus Splitting problem. Its formal definition is the following:

▶ **Definition 2** (ε-Consensus Splitting)**.** *An instance $I_{n,k}$ of ε-Consensus Splitting with n measures and k agents consists of n non-atomic probability measures on the interval $[0, 1]$, which we denote by μ_i, for $i \in [n] = \{1, 2, , \ldots, n\}$. The goal is to split the interval, via at most $n(k-1)$ cuts, into subintervals and distribute them to the k agents so that for every two agents $a, b \in [k]$ and every measure $i \in [n]$, we have $|\mu_i(U_a) - \mu_i(U_b)| \leq \frac{2\varepsilon}{k}$, where U_a, U_b are the unions of all intervals a, b receive, respectively.*

The ε-Consensus Splitting problem can be viewed as a continuous variant of the Necklace Splitting problem. Furthermore, as will be shown in the proofs of our results, every instance of Necklace Splitting can be converted into an instance of ε-Consensus Splitting. We also consider the **ε-approximate version of Necklace Splitting**, where the goal is to split the necklace so that the difference between the shares of any two agents, according to each type i, is at most $2\varepsilon m_i/k$.

The existence of a solution for the Necklace Splitting problem using at most $n(k-1)$ cuts, a bound which is tight in general, was proved, using topological arguments, first for $k = 2$ agents in [19] (see also [5] for a short proof and [20] for an earlier continuous version), and then for the general case of k agents in [1]. A more recent proof of this existence result appears in [21]. However, the proofs are non-constructive. The Necklace Halving problem is first discussed in [10]. The problem of finding an efficient algorithmic proof of Necklace Splitting is mentioned in [2].

Recently, there have been several results regarding the hardness of the Necklace Halving problem. These are discussed in the next subsection. These suggest pursuing the challenge of finding efficient approximation algorithms, as well as that of proving non-conditional hardness in restricted models.

1.2 Hardness and Approximation

PPA and PPAD are two complexity classes introduced in the seminal paper of Papadimitriou, [22]. Both of these are contained in the class TFNP, which is the complexity class of *total search* problems, consisting of all problems in NP where a solution exists for every instance. A problem is PPA-complete if and only if it is polynomially equivalent to the canonical problem LEAF, described in [22]. Similarly, a problem is PPAD-complete if and only if it is polynomially equivalent to the problem END-OF-THE-LINE. A problem is PPA-hard or PPAD-hard if the respective canonical problem is polynomially reducible to it. A number of important problems, such as several versions of Nash Equilibrium [14] and Market Equilibrium [13], have been proved to be PPAD-complete. It is known that PPAD $\subseteq$ PPA. Hence, PPA-hardness implies PPAD-hardness. Filos-Ratsikas and Goldberg showed that the Necklace Halving problem, as well as the ε-Consensus Halving problem, is PPA-hard [16], see also [17], [15]. Additionally, in [18] it is shown that for a fixed constant $\delta > 0$, and ε inversely polynomial in n, obtaining a solution to the ε-Consensus Halving with fewer than $n + n^{1-\delta}$ is PPA-hard. Our main objective here is to find efficient approximation algorithms for the problems. Although not directly related to our results, it is worth mentioning that in [11] it is proved that for $k = 2$ agents it is NP-hard to minimize the number of cuts for instances where the optimal number is less than n, even with 2 beads of each type.

1.3 Our contribution

We consider approximation algorithms for two versions of the problem, namely the online and the offline versions. We allow the algorithms to make more than n cuts, and expect either a *proper* solution or an ε-approximate one. A *proper* solution is a finite set of cuts and a distribution of the resulting intervals to the k agents so that the absolute discrepancy is at most 1. The absolute discrepancy here and in what follows is the maximum discrepancy, over all types, between the shares of beads of this type allocated to any two agents. An ε-approximate solution is a relaxation in which the discrepancy in any type is at most a fraction $2\varepsilon/k$ of the number of beads of this type. The objective is to minimize the number of cuts the algorithm makes. This problem for the ε-approximate version has been considered earlier in [9] and [12].

In addition to approximation, we also consider hardness in the online model, discussed in the next subsection. In the online model, the hardness is measured by the minimum number of cuts needed to produce a proper solution. Lower bounds on the number of cuts needed in this model provide a barrier for what online algorithms can achieve.

Some of our ideas for finding deterministic approximation algorithms are inspired by papers in Discrepancy Minimization, such as [3], [7], [6] and [8]. In [3], the terminology refers to the **Balancer** as the entity with the designated task of minimizing the absolute discrepancy between agents. We adopt the same terminology here. Thus, the Balancer has the role of an algorithm that makes cuts and assigns the resulting intervals to agents in order to achieve a proper solution for Necklace Splitting.

Our main algorithmic results are summarized in the theorems below. The upper and lower bounds for the number of cuts obtained for the online model appear in the table at the end of this subsection. Throughout the paper, for Necklace Halving, we use the notation $m = \max_{i\in[n]} m_i$ where m_i is the number of beads of color i, and n is the number of types (=colors).

▶ **Theorem 1.** *There exists an efficient, deterministic, offline algorithm that provides a proper solution to the Necklace Halving problem, making at most* $n(\log m + O(1))$ *cuts.*

Here and in what follows an efficient algorithm means an algorithm whose running time is polynomial in the length of the input necklace.

In [9] and [12] the authors describe offline algorithms for the ε-approximate version of Necklace Halving, making $O((\frac{1}{\varepsilon})^{\Theta(n)})$ cuts. Our techniques here provide an algorithm that requires only $n(\log(1/\varepsilon) + O(1))$ cuts for the problem, yielding an exponential improvement for the number of cuts.

▶ **Theorem 2.** *There exists an efficient, deterministic, online algorithm that provides a proper solution to the Necklace Halving problem, making at most* $O(m^{2/3} \cdot n(\log n)^{1/3})$ *cuts.*

The algorithmic results in the online model, and the nearly matching lower bounds we establish appear in the table below. Note that the algorithms are optimal up to constant factors for any fixed constant $n \geq 3$. In the lower bounds for Online Necklace Halving, we always assume that $m_i = m$ for all $i \in [n]$.

Problem	$n = 2$ colors	$n \geq 3$, $n = O(1)$ colors	n colors (general case)
Upper bound	$O(m^{2/3})$	$O(m^{2/3})$	$O(m^{2/3} \cdot n(\log n)^{1/3})$
Lower bound	$\Omega(\sqrt{m})$	$\Omega(m^{2/3})$	$\Omega(n \cdot m^{2/3})$

1.4 Computational model and online version

The offline computational model considered here is natural. The input for Necklace Splitting, for an instance with k agents and n colors, consists of a series of indices, each one taking a value in $[n]$, which represents the color of the respective bead. The runtime is, as usual, the number of basic operations the algorithm makes to provide a solution.

Next, we describe the online model. The parameters k, n and m_i for $i \in [n]$ are given in advance. We refer to *time* t, $0 \leq t \leq \sum_{i\in[n]} m_i - 1$ as the state after the first t beads were revealed and decisions about cutting before any of these have already been made. The beads are revealed one by one in the following way: for integral $t, 0 \leq t \leq \sum_{i\in[n]} m_i - 1$, at time t the Balancer receives the color of bead number $t+1$ and is given the opportunity to make a cut between beads t and $t+1$, where this decision is irreversible. If a cut is made, and J is the newly created interval, the Balancer also has to choose immediately the agent that gets J, before advancing to time $t+1$.

1.5 Techniques

The proofs in the paper combine combinatorial and probabilistic ideas with linear algebra and geometric tools. Theorem 1 is proved by converting the instance of Necklace Halving into a continuous instance, which can be considered an instance of ε-Consensus Halving, where the $[0, 1]$ interval is colored by n colors. We reason about finding a solution to this ε-Consensus Halving instance, for a suitable ε, and then adapt the algorithm to obtain a valid solution for the discrete Necklace Halving instance. The algorithm for the continuous instance is based on Carathéodory's Theorem for cones, and involves linear algebra manipulations. To obtain a solution for the discrete instance from the solution to the continuous instance, we describe how to shift cuts at the end to ensure they are not made in the interior of (intervals corresponding to) beads.

The online algorithm discussed in Theorem 2 is inspired by known techniques used in online algorithms for discrepancy minimization. The idea here is to cut the necklace into pieces, each having a sufficiently small number of beads of each color. The problem then becomes an online discrepancy problem, where one can use a derandomization of a natural randomized algorithm that proceeds by using an appropriate potential function motivated by the method of conditional expectations. Obtaining discrepancy ≤ 1 at the end of the necklace traversal requires a modification to the potential function technique, that handles beads of certain colors in a more careful manner once the remaining beads of these colors become scarce. The lower bound showing that the online algorithm is optimal up to logarithmic factors is proved in two steps. The first one is an argument showing that if anytime during the process the discrepancy between the shares allocated so far to the two agents according to one of the colors is relatively large, while according to another color both shares are 0, then a large number of cuts is required to ensure an appropriate solution at the end. In the second step, it is proved that in order to keep the discrepancy according to each color sufficiently small during the process, a large number of cuts is needed. This is shown by introducing and analyzing appropriate potential functions, where the challenge here is to define functions that enable the adversary to ensure they will keep growing for any choice of a place to cut, and any allocation of the resulting interval, provided that the interval created is not too short. One of the lemmas in the proof here is based on the fact that a certain matrix is totally unimodular. The full details appear in the following sections.

1.6 Structure

The structure of the rest of the paper is as follows: in Section 2 we present the approximation algorithm for the offline version of the problem. Section 3 contains the algorithm for the online version. Section 4 contains the lower bounds for the online model. The final Section 5 contains several extensions and open problems. To simplify the presentation we omit all floor and ceiling signs throughout the paper whenever these are not crucial. All logarithms are in base 2, unless otherwise specified.

2 An offline algorithm

Proof of Theorem 1

Proof. Given a necklace with m_i beads of color i for $1 \leq i \leq n$, where $m = \max m_i$, construct an instance of ε-Consensus Halving as follows. Replace each bead of color i by an interval of i-measure $1/m_i$ and j-measure 0 for all $j \neq i$. These intervals are placed next to each other according to the order in the necklace, and their lengths are chosen so that altogether they cover $[0,1]$. We first give a marking procedure that splits the continuous necklace so that the absolute discrepancy is at most ε, with $\varepsilon = \frac{1}{2m}$. Then, we show how to modify the solution from the continuous instance to the discrete necklace so that the cuts are made between consecutive beads and we obtain a proper solution.

Given n non-atomic measures μ_i on the interval $[0,1]$ we describe an efficient algorithm that cuts the interval in at most $n(2 + \lceil \log_2 \frac{1}{\varepsilon} \rceil)$ places and splits the resulting intervals into two collections C_0, C_1 so that $\mu_i(C_j) \in [\frac{1}{2} - \frac{\varepsilon}{2}, \frac{1}{2} + \frac{\varepsilon}{2}]$ for all $i \in [n], 0 \leq j \leq 1$. Note, first, that if the collection C_1 has the right amount according to each of the measures μ_i, so does the collection C_0. For each interval $I \subset [0,1]$ denote $\mu(I) = \mu_1(I) + \ldots + \mu_n(I)$. Thus $\mu([0,1]) = n$. Using $2n-1$ cuts split $[0,1]$ into $2n$ intervals $I_1, I_2, \ldots, I_{2n}$ so that $\mu(I_r) = 1/2$ for all r. Note that it is easy to find these cuts efficiently, since each measure μ_i is uniform on its support.

For each interval I_r let v_r denote the n-dimensional vector $(\mu_1(I_r), \mu_2(I_r), \ldots, \mu_n(I_r))$.

By a simple linear algebra argument, which is a standard fact about the properties of basic solutions for Linear Programming problems, one can write the vector $(1/2, 1/2, \ldots, 1/2)$ as a linear combination of the vectors v_r with coefficients in $[0,1]$, where at most n of them are not in $\{0,1\}$. This follows from Carathédory's Theorem for cones. For completeness, we include the proof, which also shows that one can find coefficients as above efficiently. Start with all coefficients being $1/2$. Call a coefficient which is not in $\{0,1\}$ *floating* and one in $\{0,1\}$ *fixed*. Thus at the beginning all $2n$ coefficients are floating. As long as there are more than n floating coefficients, find a nontrivial linear dependence among the corresponding vectors and subtract a scalar multiple of it which keeps all floating coefficients in the closed interval $[0,1]$ shifting at least one of them to the boundary $\{0,1\}$, thus fixing it.

This process clearly ends with at most n floating coefficients. The intervals with fixed coefficients with value 1 are now assigned to the collection C_1 and those with coefficient 0 to C_0. The rest of the intervals remain. Split each of the remaining intervals into two intervals, each with μ-value $1/4$. We get a collection $J_1, J_2, \ldots, J_m$ of $m \leq 2n$ intervals, each of them has the coefficient it inherits from its original interval. Each such interval defines an n-vector as before, and the sum of these vectors with the corresponding coefficients (in $(0,1)$) is exactly what the collection C_1 should still get to have its total vector of measures being $(1/2, \ldots, 1/2)$.

As before, we can shift the coefficients until at most n of them are floating, assign the intervals with $\{0,1\}$ coefficients to the collections C_0, C_1 and keep at most n intervals with floating coefficients. Split each of those into two intervals of μ-value $1/8$ each and proceed as before, until we get at most n intervals with floating coefficients, where the μ-value of each of them is at most $\varepsilon/2$. This happens after at most $\lceil \log_2(1/\varepsilon) \rceil$ rounds. In the first one, we have made $2n-1$ cuts and in each additional round at most n cuts. Thus the total number of cuts is at most $n(2+\lceil \log_2(1/\varepsilon) \rceil)-1$.

From now on we add no additional cuts, and show how to allocate the remaining intervals to C_0, C_1. Let $\mathcal{I}$ denote the collection of intervals with floating coefficients. Then $|\mathcal{I}| \leq n$ and $\mu(I) \leq \varepsilon/2$ for each $I \in \mathcal{I}$. This means that

$$\sum_{i=1}^{n} \sum_{I \in \mathcal{I}} \mu_i(I) \leq n\varepsilon/2$$

It follows that there is at least one measure μ_i so that

$$\sum_{I \in \mathcal{I}} \mu_i(I) \leq \varepsilon/2.$$

We can think of the remaining floating coefficients as the fraction of each corresponding interval that agent 1 owns. Observe that for any assignment of the intervals $I \in \mathcal{I}$ to the two collections C_0, C_1, the total μ_i measure of C_1 (and hence also of C_0) lies in $[1/2-\varepsilon/2, 1/2+\varepsilon/2]$, as this measure with the floating coefficients is exactly $1/2$ and any allocation of the intervals with the floating coefficients changes this value by at most $\varepsilon/2$. We can thus ignore this measure, for ease of notation assume it is measure number n, and replace each measure vector of the members in $\mathcal{I}$ by a vector of length $n-1$ corresponding to the other $n-1$ measures. If $|\mathcal{I}| > n-1$ (that is, if $|\mathcal{I}| = n$), then it is now possible to shift the floating coefficients as before until at least one of them reaches the boundary, fix it assigning its interval to C_1 or C_0 as needed, and omit the corresponding interval from $\mathcal{I}$ ensuring its size is at most $n-1$. This means that for the modified $\mathcal{I}$ the sum

$$\sum_{i=1}^{n-1} \sum_{I \in \mathcal{I}} \mu_i(I) \leq (n-1)\varepsilon/2.$$

Hence there is again a measure i, $1 \leq i \leq n-1$ so that

$$\sum_{I \in \mathcal{I}} \mu_i(I) \leq \varepsilon/2.$$

Again, we may assume that $i = n-1$, observe that measure $n-1$ will stay in its desired range for any future allocation of the remaining intervals, and replace the measure vectors by ones of length $n-2$. This process ends with an allocation of all intervals to C_1 and C_0, ensuring that at the end $\mu_i(C_j) \in [1/2-\varepsilon/2, 1/2+\varepsilon/2]$ for all $1 \leq i \leq n$, $0 \leq j \leq 1$. These are the desired collections. It is clear that the procedure for generating them is efficient, requiring only basic linear algebra operations.

The intervals separated by the marks are partitioned by the algorithm into two collections forming a solution of the continuous problem. Note that the continuous solution would give discrepancy at most $\max_{i \in [n]} m_i \cdot \varepsilon \leq 1/2$ in terms of beads if we were allowed to cut at the marked points. The only subtle point is that some of the marks may be in the interior of small intervals corresponding to beads, and we wish to cut only between beads.

Call a mark between two consecutive beads *fixed* and call the other marks *floating*. We first show how to shift each of the floating marks so that the absolute discrepancy does not increase beyond $1/2$ and all but at most one mark for each color are made between two consecutive beads. To do so, if there exists a floating mark between two intervals assigned to the same agent eliminate it and merge the two intervals. If there is no such mark and there are at least two floating marks in the interior of intervals corresponding to color i, we shift both of them by the same amount in the appropriate way until at least one of them becomes fixed. If during this simultaneous shift one of the two marks arrives in a spot occupied by a different mark, we stop the shift and discard one of the duplicate marks. Note that the quantities the two agents receive do not change.

This procedure reduces the number of floating marks until there is at most one floating mark for each color. If there is such a floating mark, round it to the closest boundary between beads noting that this can increase the absolute discrepancy by at most 1. Therefore, once all marks are fixed, the absolute discrepancy is $\leq 3/2$. Since all the cuts are between consecutive beads, this discrepancy has to be an integer, and thus it is at most 1, as desired. The number of cuts made is $\leq n(2 + \lceil \log_2 \frac{1}{\varepsilon} \rceil) = n(3 + \lceil \log_2 m \rceil) = n(\log m + O(1))$. ◀

▶ Remarks.

- The argument can be extended to splitting into k nearly fair collections of intervals. See section 5 for more details.
- The ε-approximate Necklace Halving problem can be solved with $n(\log(\frac{1}{\varepsilon}) + O(1))$ cuts by using the above algorithm for the continuous instance with the required value of ε.
- The proof can be adapted to obtain a solution with $n(\log(\frac{1}{\varepsilon}) + O(1))$ cuts to the ε-Consensus Halving problem, with the appropriate natural assumptions about the way the measures are presented.
- In [18] the authors give an efficient algorithm for solving a special case of the ε-Consensus Halving problem that works for probability measures each of which is uniform on a single interval. The algorithm provides a solution making at most n cuts for this special case.

3 An online algorithm

Proof of Theorem 2

Proof. We describe an efficient online algorithm that achieves absolute discrepancy at most 1. The algorithm makes $O(m^{2/3}n(\log n)^{1/3})$ cuts. It is worth mentioning that the main part of the algorithm is a derandomization of a simple randomized algorithm which cuts the necklace into pieces each of which has a sufficiently small number of beads of each color and then assigns them randomly and uniformly to the two agents.

Note first that if, say, $\log n > m/1000$, the result is trivial, as less than nm cuts suffice to split the necklace into single beads, hence we may and will assume that $m \geq 1000 \log n$. Throughout the algorithm we call the beads that have not yet been revealed the *remaining beads*. This definition makes sense as in the online model the beads of the necklace are revealed one by one. We provide a cutting rule and a distribution rule. During the algorithm, we call a color i *critical* if the number of remaining beads of this color is smaller than $20\frac{m_i}{m^{1/3}}(\log n)^{1/3}$, otherwise it is *normal*. When encountering a bead of a critical color i while traversing the necklace, the algorithm makes a cut before and after it, allocating that bead to the agent with a smaller number of beads of this type, where ties are broken arbitrarily. We call such cuts that are made right before or after beads of a critical color *forced*.

In addition to the rule about forced cuts, we provide a rule determining when to stop traversing the necklace and make a cut when no beads of a critical color are seen. Define $g = \frac{100}{8m^{2/3}(\log n)^{1/3}}$, and for every $i \in [n]$, $g_i = m_i g = \frac{100m_i}{8m^{2/3}(\log n)^{1/3}}$. Whenever after the last cut made after bead number x we reach a bead number y so that $[x, y]$ (the interval containing beads $x+1, x+2, ..., y$) contains at most g_i beads of color i for every i that is normal at that time and exactly g_j beads of some normal color j, we make a cut. As explained above, the exception to this rule is when we encounter a bead of a color i that is critical before the portion following the last cut has enough beads of some normal color. If $g_i \leq 1$ for some color i, then we cut before and after each bead of color i, essentially treating color i as critical from the beginning.

To decide about the allocation of the intervals created we define, for each color $i \in [n]$, a potential function $\phi_i(t)$, and a function $\psi_i(t)$ that is an upper bound of ϕ_i and is computable efficiently. The variable t here will denote, throughout the algorithm, the index of the last cut made.

The functions ϕ_i, ψ_i are defined by considering an appropriate probabilistic process. For each $i \in [n]$, let X_i be the random variable whose value is the difference between the number of beads of color i belonging to agent 1 and that belonging to agent 2 if after each cut the interval created is assigned to a uniform random agent. Let ε_k be 1 if the k'th interval is assigned to agent 1 and -1 otherwise. Therefore $X_i = \sum_{j=1}^{p} \varepsilon_j a_j$, where $p-1$ is the total number of cuts made and a_j the number of beads of color i on interval I_j, the j'th created interval. The distribution defining X_i is the one where each ε_j is 1 or -1 randomly, uniformly and independently. The function $\phi_i(t)$ is defined as follows

$$\phi_i(t) = \mathbb{E}\left[\frac{e^{\lambda X_i/m_i} + e^{-\lambda X_i/m_i}}{2} | \varepsilon_1, \varepsilon_2, ..., \varepsilon_t\right]$$

This is a conditional expectation, where the conditioning is on the allocation of the first t intervals represented by $\varepsilon_1, \ldots, \varepsilon_t$, and where $\lambda = \frac{4m^{1/3}(\log n)^{2/3}}{10}$. (This choice of λ will become clear later). The purpose of the division by m_i is to normalize the exponent of the potential functions to ensure maintaining a relatively small discrepancy for all colors i simultaneously. Since $X_i = \sum_j \varepsilon_j a_j$, where a_j is the number of beads on the j'th interval of color i, we have that

$$\phi_i(t) = \mathbb{E}\left[\frac{e^{\lambda \sum_j \varepsilon_j a_j/m_i} + e^{-\lambda \sum_j \varepsilon_j a_j/m_i}}{2} | \varepsilon_1, \varepsilon_2, ..., \varepsilon_t\right]$$

The function $\psi_i(t)$ is defined in a way ensuring it upper bounds the function $\phi_i(t)$. It is convenient to split each $\phi_i(t)$ into

$$\frac{1}{2}\mathbb{E}\left[e^{\lambda \sum_j \varepsilon_j a_j/m_i} | \varepsilon_1, .., \varepsilon_t\right] + \frac{1}{2}\mathbb{E}\left[e^{-\lambda \sum_j \varepsilon_j a_j/m_i} | \varepsilon_1, .., \varepsilon_t\right].$$

For simplicity, denote the first term ϕ_i' and the second term ϕ_i''. Therefore

$$\phi_i'(t) = \frac{1}{2}\mathbb{E}\left[e^{\lambda \sum_j \varepsilon_j a_j/m_i} | \varepsilon_1, .., \varepsilon_t\right] = \frac{1}{2} e^{\lambda \sum_{j=1}^{t} \varepsilon_j a_j/m_i} \cdot \prod_{j \geq t+1} \left(\frac{e^{\lambda a_j/m_i} + e^{-\lambda a_j/m_i}}{2}\right)$$

$$= \frac{1}{2} e^{\lambda \sum_{j=1}^{t} \varepsilon_j a_j/m_i} \cdot \prod_{j \geq t+1} \cosh(\lambda a_j/m_i)$$

A similar expression exists for ϕ_i''. Define $s_t = \sum_{j=1}^{t} a_j/m_i$ and $u_t = \sum_{j=1}^{t} \varepsilon_j a_j/m_i$. By the discussion above

$$\phi_i(t) = \frac{e^{\lambda u_t} + e^{-\lambda u_t}}{2} \prod_{j \geq t+1} \cosh(\lambda a_j/m_i).$$

Using the well-known inequality that $\cosh(x) \leq e^{x^2/2}$, it follows that

$$\phi_i(t) \leq \frac{e^{\lambda u_t} + e^{-\lambda u_t}}{2} e^{\lambda^2 \sum_{j \geq t+1} (a_j/m_i)^2/2}.$$

By the way the cuts are produced $a_j \leq g_i$ for all j, and hence

$$\sum_{j=t+1} (a_j/m_i)^2 \leq \max_{j \geq t+1} (|a_j/m_i|) \cdot (\sum_{j \geq t+1} a_j/m_i) \leq g \cdot (\sum_{j \geq t+1} a_j/m_i) = g(1 - s_t).$$

Therefore

$$\phi_i(t) \leq \frac{e^{\lambda u_t} + e^{-\lambda u_t}}{2} e^{\lambda^2 g(1-s_t)/2}.$$

Define $\psi_i(t)$ to be the above upper bound for $\phi_i(t)$, that is

$$\psi_i(t) = \frac{e^{\lambda u_t} + e^{-\lambda u_t}}{2} e^{\lambda^2 g(1-s_t)/2}.$$

Note that $\psi_i(t)$ can be easily computed efficiently at time t, since s_t and u_t (as well as g and λ) are known at this point.

Having defined the potential functions ϕ_i and their upper bounds ψ_i, we are now ready to describe the allocation rule following cuts that create intervals with no beads of any critical color. (The rule for allocating intervals consisting of a single bead of a critical color has already been described). Initialize $\phi(0) = \sum_{i \in [n]} \phi_i(0), \psi(0) = \sum_{i \in [n]} \psi_i(0)$, where by convention $\psi_i(0) = e^{g\lambda^2/2}$. After each cut t during the process, we define $\phi(t) = \sum_{i \text{ normal}} \phi_i(t)$ and $\psi(t) = \sum_{i \text{ normal}} \psi_i(t)$. In other words, once a color i becomes critical, the terms ϕ_i and ψ_i are dropped from the respective expressions.

Having allocated the first t intervals, at cut $t+1$, we choose ε_{t+1}, which corresponds to a choice of the agent who gets the interval, in order to minimize $\psi(t+1)$. To show that this algorithm produces a proper solution, where the absolute discrepancy at the end is at most 1, we prove the following two claims:

▷ Claim 1. The upper bound $\psi(t)$ is (weakly) decreasing in the variable t.

▷ Claim 2. For each i, after each cut made before the color becomes critical, the discrepancy in color i is at most $10\frac{m_i}{m^{1/3}}(\log n)^{1/3}$ (in absolute value).

Claim 2 implies that after the first cut that causes color i to become critical, the discrepancy on i is at most $10\frac{m_i}{m^{1/3}}(\log n)^{1/3} + g_i < 20\frac{m_i}{m^{1/3}}(\log n)^{1/3} - g_i$. Hence, it follows from the way the algorithm deals with subsequent beads of color i, that the process will end with a balanced partition of the beads of each color i between the agents, allocating to each of them either $\lfloor m_i/2 \rfloor$ or $\lceil m_i/2 \rceil$ of these beads. As this argument works for every color, the algorithm produces a proper solution. Next, we prove the claims.

Proof of Claim 1. Note that whenever some color i becomes critical, the term ψ_i that we drop from ψ is positive. Hence, it is enough to prove that $\psi(t) \geq \frac{\psi(t+1|\varepsilon_{t+1}=1)+\psi(t+1|\varepsilon_{t+1}=-1)}{2}$, where $\psi(t+1|\varepsilon_{t+1} = \chi)$ denotes the value of $\psi(t+1)$ if we choose $\varepsilon_{t+1} = \chi \in \{-1, 1\}$. It suffices to show that for every i, $\psi_i(t) \geq \frac{1}{2}[\psi_i(t+1|\varepsilon_{t+1} = 1] + \frac{1}{2}[\psi_i(t+1|\varepsilon_{t+1} = -1]$.

We proceed with the proof of this inequality. To do so, note that

$$\psi_i(t+1|\varepsilon_{t+1}=1) = \frac{e^{\lambda(u_t+a_{t+1}/m_i)} + e^{-\lambda(u_t+a_{t+1}/m_i)}}{2} e^{\lambda^2 g(1-s_t-a_{t+1}/m_i)/2},$$

and

$$\psi_i(t+1|\varepsilon_{t+1}=-1) = \frac{e^{\lambda(u_t-a_{t+1}/m_i)} + e^{-\lambda(u_t-a_{t+1}/m_i)}}{2} e^{\lambda^2 g(1-s_t-a_{t+1}/m_i)/2}.$$

Therefore

$$\frac{\psi_i(t+1|\varepsilon_{t+1}=1) + \psi_i(t+1|\varepsilon_{t+1}=-1)}{2} =$$

$$\frac{e^{\lambda u_t} + e^{-\lambda u_t}}{2} \cdot \frac{e^{\lambda a_{t+1}/m_i} + e^{-\lambda a_{t+1}/m_i}}{2} e^{\lambda^2 g(1-s_t-a_{t+1}/m_i)/2}$$

$$\leq \frac{e^{\lambda u_t} + e^{-\lambda u_t}}{2} \cdot e^{\lambda^2 g(a_{t+1}/m_i)/2} e^{\lambda^2 g(1-s_t-a_{t+1}/m_i)/2} = \frac{e^{\lambda u_t} + e^{-\lambda u_t}}{2} \cdot e^{\lambda^2 g(1-s_t)/2} = \psi_i(t),$$

as needed. ◁

Proof of Claim 2. Let t be a cut made while color i is normal. To prove that the discrepancy on color i in absolute value is at most $10\frac{m_i}{m^{1/3}}(\log n)^{1/3}$, it suffices to prove $\psi_i(t) \leq \frac{1}{2} e^{\lambda \cdot 10(\frac{\log n}{m})^{1/3}} e^{\lambda^2 g(1-s_t)} = \frac{1}{2} e^{2\log n} e^{\lambda^2 g(1-s_t)}$. By Claim 1, $\psi(t) \leq \psi(0) = ne^{g\lambda^2/2}$. Hence, it is enough to prove that $ne^{g\lambda^2/2} \leq \frac{1}{2} e^{2\log n} e^{\lambda^2 g(1-s_t)}$. This is equivalent to $\lambda^2 g s_t/2 + \log 2 \leq 4\log n$. Since $s_t \leq 1$, we get $\lambda^2 g s_t/2 + \log 2 \leq \log n + \log 2 \leq 2\log n$, as needed. ◁

Lastly, we prove that the total number of cuts is $O(n(\log n)^{1/3} \cdot m^{2/3})$. The number of forced cuts cannot exceed $2n \cdot 20\frac{m_i}{m^{1/3}}(\log n)^{1/3} = O(m^{2/3} n (\log n)^{1/3})$. To bound the number of non-forced cuts, note that whenever we make such a cut, there is a color j such that the number of beads of this color on the interval created is exactly g_j. We call the cut j-tight for that respective color. It is easy to see that for every color i there are most $O(m^{2/3}(\log n)^{1/3})$ i-tight cuts. Hence, the total number of non-forced cuts is at most $O(m^{2/3} n (\log n)^{1/3})$. This completes the proof. ◀

4 Lower bounds

In this section we present the lower bounds for Necklace Halving in the online model.

4.1 A preliminary bound

We provide a $\Omega(\sqrt{m})$ lower bound for the number of cuts required in any online algorithm when the number of colors is $n = 2$ and there are m beads of each color. We need the following simple lemma, which is a special case of a more general elegant result of Tijdeman [23]. Since this special case is much simpler, we include its proof, for completeness.

▶ **Lemma 1.** *For every real $\gamma \in [0,1]$ there is an infinite binary sequence $a_1, a_2, a_3, \ldots$ so that in every prefix of it $a_1, a_2, \ldots, a_j$ the number of elements a_i which are 1 deviates from γj by less than 1.*

Proof. By compactness it suffices to prove the existence of such a sequence of any finite length r. Consider the following system of linear inequalities in the variables $x_1, x_2, \ldots, x_r$: $0 \leq x_i \leq 1$ for all $1 \leq i \leq r$, and for every $j \leq r$, $\lfloor \gamma j \rfloor \leq \sum_{i=1}^{j} x_i \leq \lceil \gamma j \rceil$. This system has a real solution $x_i = \gamma$ for every i and the matrix of coefficients of the constraints is totally unimodular. Hence there is an integral solution $x_i = a_i \in \{0, 1\}$ providing the required sequence. ◀

We use the following notation. During the algorithm let t denote the number of beads revealed so far. If a cut is made at this point, let x_t be the difference between the number of beads of color 1 allocated to agent 1 and the number of beads of color 1 allocated to agent 2. Define y_t similarly for beads of color 2. Let α_t, β_t denote the number of remaining beads of colors 1 and 2, respectively.

▶ **Lemma 2.** *Let Δ be a positive integer. Suppose that a cut is made at point t and $|x_t| = \Delta$ and assume that no bead of color 2 appeared so far. Then there exists an adversarial input that forces the Balancer to make at least $\Delta/4 = \Omega(\Delta)$ cuts.*

Proof. Without loss of generality assume that $x_t = \Delta > 0$. Note that by assumption $\beta_t = m$ and $\alpha_t < m$. Put $\gamma = \frac{m}{\alpha_t + m}$ and note that $\gamma > 1/2$. By Lemma 1 it is possible to choose an ordering of the remaining $\alpha_t + m$ beads of the necklace so that in every prefix of it of any length j, the number of beads of color 2 deviates from γj by less than 1. Since our online model allows the Balancer to see the next bead before the decision to make a cut preceding it we may have to change the first bead in this ordering, this still ensures that in any interval of length ℓ in the remainder of the necklace, the number of beads of color 2 deviates from $\gamma\ell$ by at most 2.

Suppose the Balancer cuts the remainder of the necklace and allocates the resulting intervals $R_1, ..., R_u$ to agent 1 and $T_1, ..., T_v$ to agent 2 to obtain a balanced allocation. For each one of these intervals I let $\ell(I)$ denote its length. By assumption at time t agent 1 has exactly Δ more beads than agent 2. Since at the end each agent has half of the beads (for simplicity we assume that m is even), $\sum_{i=1}^{v} \ell(T_i) - \sum_{j=1}^{u} \ell(R_j) = \Delta$.

By construction, the total number of beads of color 2 in all intervals T_i deviates from $\gamma \sum_{i=1}^{v} \ell(T_i)$ by at most $2v$. Similarly, the total number of beads of color 2 in all intervals R_j deviates from $\gamma \sum_{j=1}^{u} \ell(R_j)$ by at most $2u$. As these two numbers should be equal it follows that

$$\gamma\Delta = \gamma\left(\sum_{i=1}^{v} \ell(T_i) - \sum_{j=1}^{u} \ell(R_j)\right) \leq 2u + 2v$$

This implies that $2(u + v) \geq \gamma\Delta > \Delta/2$ and as the number of cuts is at least $u + v$ the desired result follows. ◀

The last lemma easily implies the following.

▶ **Theorem 3.** *There exists an adversarial input that forces any deterministic algorithm for Online Necklace Halving with $n = 2$ colors to make $\Omega(\sqrt{m})$ cuts in order to obtain a proper solution.*

Proof. Put $\Delta = \sqrt{m}$ and proceed by revealing only beads of color 1. By Lemma 2, if after a cut at some t, $|x_t| > \sqrt{m}$, the desired result follows. Otherwise it is clear the number of beads between any two consecutive cuts is less than $2\sqrt{m}$, implying that the total number of cuts made by the Balancer is $\Omega(\sqrt{m})$. ◀

4.2 A nearly tight bound

▶ **Theorem 4.** *An adversary can force any deterministic algorithm for Online Necklace Halving with $n = 3$ colors and m beads of each color to make $\Omega(m^{2/3})$ cuts.*

Proof. As in the previous subsection, let x_t denote the discrepancy between the number of beads of color 1 allocated to agent 1 and that allocated to agent 2 after cut t, and let y_t denote the corresponding discrepancy for color 2, where color 3 will be kept as a potential threat. We proceed by revealing only beads of the first two colors. By Lemma 2 with $\Delta = m^{2/3}$ the Balancer needs to maintain $|x_t|, |y_t| \leq m^{2/3}$, since otherwise the adversary can force $\Omega(m^{2/3})$ cuts, using beads of the third color. Hence, we assume that during the process of revealing the initial $m + 4m^{2/3}$ beads of the necklace x_t, y_t stay in the above range after each cut.

Define a potential function

$$M(x, y) = x^2 + y^2 + 5m^{2/3}(x - y)$$

After a cut with $v_t = (x_t, y_t) = (x, y)$ define $\gamma = \frac{10m^{2/3} - 4y}{20m^{2/3} + 4(x-y)}$. Note that $0 < \gamma < 1$, as $|x|, |y| \leq m^{2/3}$. By Lemma 1 it is possible to order the remaining part of the first $m + 4m^{2/3}$ beads of the necklace so that in each prefix of any length j of this remaining part the number of beads of color 1 deviates from γj by less than 1 and the number of beads of color 2 deviates by less than 1 from $(1 - \gamma)j$. As the first bead of this remaining part has been observed already by the Balancer we may need to change one bead in this ordering, getting a deviation of less than 2 in each prefix. This means that if the next cut will be made after some j additional beads, the vector $p = (p_1, p_2)$ of additional beads of colors 1 and 2, respectively, can be written as a sum of the vector $p' = (\gamma j, (1 - \gamma)j)$ and an error vector $\delta = (\delta_1, \delta_2)$ of ℓ_∞-norm smaller than 2. We get that

$$M(v_t + p') - M(v) = p_1'^2 + p_2'^2 + 2xp_1' + 2yp_2' + 5m^{2/3}p_1' - 5m^{2/3}p_2' =$$

$$= p_1'^2 + p_2'^2 + \frac{1}{2}[p_1' \cdot (10m^{2/3} + 4x) - p_2' \cdot (10m^{2/3} - 4y)] = p_1'^2 + p_2'^2 \geq \frac{1}{2}j^2$$

and similarly,

$$M(v - p') - M(v) = p_1'^2 + p_2'^2 + \frac{1}{2}[-p_1' \cdot (10m^{2/3} + 4x) + p_2' \cdot (10m^{2/3} - 4y)] = p_1'^2 + p_2'^2 \geq \frac{1}{2}j^2$$

A simple computation using the fact that $|x|, |y| \leq m^{2/3}$ and that a similar bound holds after adding or subtracting the vector p' shows that adding or subtracting the vector δ can decrease the value of M by less than $15m^{2/3}$. Therefore, we get

$$M(v_t \pm p) - M(v_t) \geq j^2/2 - 15m^{2/3}$$

which implies $M(v_{t+1}) - M(v_t) \geq j^2/2 - 15m^{2/3}$, with a cut of j beads.

Suppose that we have r cuts among the first $m + 4m^{2/3}$ beads of the necklace, and the lengths of the resulting intervals are $j_1, j_2, \ldots, j_r$. Since throughout the process $|x_t|, |y_t| \leq m^{2/3}$, it follows that $M(x_t, y_t) \leq 12m^{2/3}$. On the other hand by the above discussion the value of M at the end is at least $\sum_{i=1}^{r} \frac{j_i^2}{2} - 15m^{2/3}r$. Since $\sum_{i=1}^{r} j_i \geq m$ (as we cannot have $4m^{2/3}$ consecutive beads with no cut among them), it follows, by Cauchy-Schwartz, that $\sum j_i^2 \geq \frac{m^2}{r}$. This implies that

$$\frac{1}{2}\frac{m^2}{r} - 15rm^{2/3} \leq 12m^{4/3}$$

showing that $r = \Omega(m^{2/3})$, as needed. ◀

▶ Remark. For $n > 3$ colors with m beads of each color one can consider a necklace consisting of $\lfloor n/3 \rfloor$ segments with at least 3 colors in each of them. The above argument shows that it is possible to force $\Omega(m^{2/3})$ cuts in each segment, implying an $\Omega(nm^{2/3})$ lower bound. Thus, for n colors, the gap between our lower and upper bounds for the number of cuts required is only a factor of $\Theta((\log n)^{1/3})$.

5 Extensions and open problems

We conclude with some generalizations of the algorithms presented and the lower bounds obtained, and with comments on some of the questions that remain open.

5.1 Generalizations

In this section, we present our online and offline results for the general case of k agents.

▶ **Theorem 5.** *There exists an efficient, deterministic, offline algorithm that provides a proper solution to the Necklace Splitting problem, making at most* $n(k-1)\lceil 4 + \log_2(3km)\rceil$ *cuts.*

Proof. As in the proof of Theorem 1, we first convert the Necklace Splitting instance into a continuous instance J, and obtain a solution with absolute discrepancy at most $\frac{\varepsilon}{2k} = \frac{1}{2km}$, possibly making some floating cuts. Then, to obtain a proper solution for the discrete instance, we shift the floating cuts by solving a network flow problem.

To obtain the solution to the continuous instance J, we recursively apply a modified version of the algorithm that makes cuts on the continuous necklace from Theorem 1. Define $\varepsilon' = \varepsilon/3k = \frac{1}{3km}$, and divide the k players into two disjoint groups A, B, with $\lfloor k/2 \rfloor$ agents and $\lceil k/2 \rceil$ agents respectively. Think of A, B as two agents and split the continuous necklace among them. By following the algorithm in the proof of Theorem 1, one can make $\leq n(2+\lceil \log_2 \frac{1}{\varepsilon'} \rceil)$ cuts and split the interval so that A gets $\frac{\lfloor k/2 \rfloor}{k} \pm \varepsilon'/2$ of each measure i. We can do so by starting with all floating coefficients equal to $\frac{\lfloor k/2 \rfloor}{k}$ instead of $\frac{1}{2}$ and by following the proof of Theorem 1. Repeat the same procedure for the groups A and B recursively, splitting the share of A among its $|A|$ members and doing the same for B. In the end, the error can be bounded by $\varepsilon' + \frac{2}{3}\varepsilon' + \frac{2}{3} \cdot \frac{4}{7}\varepsilon' + \ldots < 3\varepsilon'$. If we denote by $T(k)$ the number of cuts made to obtain absolute discrepancy $\leq \varepsilon'$ for a continuous instance with n types and k agents, then $T(2) = n[\log_2 \frac{1}{\varepsilon'} + 2]$, and $T(k) = T(\lfloor k/2 \rfloor) + T(\lceil k/2 \rceil) + n[\log_2 \frac{1}{\varepsilon'} + 2]$, which gives that the number of cuts made for this split is $T(k) = n(k-1)\lceil 2 + \log_2(3km)\rceil)$.

Hence, we have obtained a proper solution for the continuous instance J, making $n(k-1)\lceil 2 + \log_2(3km)\rceil)$ cuts, yet we have to handle floating cuts. We categorize each floating cut by the color of the interval in whose interior it lies. For each color i, we handle the corresponding floating cuts. First, note that if $k > m_i$, we can shift each floating cut on color i to one of the ends of the i-interval in such a way that no agent gets more than one bead of color i and this will provide discrepancy at most 1 on color i without creating any additional cuts. Hence, we may assume $m_i \geq k$.

We use a network flow algorithm to decide, for each bead of color i that does not fully belong to one agent, to whom it should be allocated. Define a directed graph G_i, with vertices s, the source, t, the sink, V_i, representing the set of beads of color i, and H, the set of vertices representing the agents. Let E be the set of edges with

$$E = \{(s, v), v \in V_i\} \cup \{(h, t), h \in H\} \cup \{(v, h), \text{agent h owns a share of bead v}\}$$

All edges $\{(s,v), v \in V_i\}$ have capacity 1 and lower bound 1. Each edge (v,h) has capacity 1 and lower bound 0. Finally, for each edge (h,t), set the capacity to be $\lceil x_h \rceil$ and the lower bound to be $\lfloor x_h \rfloor$, where x_h is the quantity of type i allocated to agent h in the solution to the continuous instance. Now, if we assign each edge (v,h) a value equal to the share of bead v allocated to agent h in the continuous solution and each edge (h,t) the value x_h, this is a legal flow. Hence, there exists an integral legal flow in the network, and it is well known that one can find such a flow efficiently. Note that an integral flow corresponds to a distribution of the beads of color i where no additional cut is made and the absolute discrepancy is at most 1 if $k \nmid a_i$ and at most 2 if $k|a_i$. Thus, the integral flow determines which agent gets each of the contested beads of color i, corresponding to a shift of each floating cut to one of the ends of the bead it crosses.

If $k|a_i$, the continuous solution could give some agent a a share of $x_a = a_i/k - \varepsilon_1$ and some agent b a share $x_b = a_i/k + \varepsilon_2$, for small positive values $\varepsilon_1, \varepsilon_2$. In this case, the integral network flow solution could give agent a $a_i/k - 1$ beads and agent b $a_i/k + 1$ beads of color i. As a_i/k is an integer, the number of agents receiving $a_i/k + 1$ beads is the same as the number of agents receiving $a_i/k - 1$. Hence, we can make at most $2k$ cuts after the shift is done to obtain discrepancy 0. We perform the shifting procedure for every color i, and obtain a proper solution with at most $nk + n(k-1)\lceil 2 + \log_2(3km) \rceil < n(k-1)\lceil 4 + \log_2(3km) \rceil$, as needed. This completes the proof. The network flow argument follows the approach in [4]. ◀

▶ **Theorem 6.** *There exists an efficient, deterministic, online algorithm that provides a proper solution for the Necklace Splitting problem, making at most $\tilde{O}(nk^{1/3} \cdot m^{2/3})$ cuts.*

Proof. Note that the result is trivial for $k > m$. For $k \leq m$, we again use the idea of defining a potential function ϕ and a function ψ that is an upper bound for ϕ and is computable efficiently. Instead of having one pair of functions ϕ_i, ψ_i for each color i, we now have $\binom{k}{2}$ such functions, one for each pair of agents. For each color i and agents $p \neq q$, define $\phi_i^{p,q} = \mathbb{E}\left[\frac{e^{\lambda X_{p,q,i}/m_i} + e^{-\lambda X_{p,q,i}/m_i}}{2}\right]$, where $X_{p,q,i}$ is the random variable of the difference between the number of beads of color i given to agent p and that of agent q. The relevant random distribution here assigns every newly created interval to one of the k agents with equal probability which is $1/k$. The quantity $g = g(n,k,m)$ is defined here as $g = \frac{1}{m^{2/3}k(\log(nk))^{1/3}}$, and each g_i, the maximum number of beads of color i allowed between two consecutive cuts as $g_i = m_i g$. We say color i is *critical* when the number of remaining beads of this color is at most $20k^{1/3}m^{2/3}$.

The function $\psi_i^{p,q}$ is defined by

$$\psi_i^{p,q}(t) = \frac{e^{\lambda x_{t,i}^{p,q}} + e^{-\lambda x_{t,i}^{p,q}}}{2} \cdot e^{2\lambda^2 g(1-s_t)/k}$$

where s_t is, as before, the proportion of beads of color i allocated already, and $x_{t,i}^{p,q}$ is the discrepancy between p and q on color i after cut t divided by m_i.

The main difference required here is the replacement of the inequality $\cosh(\lambda a) \leq e^{\lambda^2 a^2/2}$ by the following inequality which holds whenever, say, $\lambda a \leq 1$:

$$\frac{k-2}{k}e^{\lambda \cdot 0} + \frac{1}{k}e^{\lambda a} + \frac{1}{k}e^{-\lambda a} = 1 + \frac{2}{k}(\cosh(\lambda a) - 1)$$

$$\leq 1 + \frac{2}{k}(e^{\lambda^2 a^2/2} - 1) \leq 1 + \frac{2}{k}\frac{2\lambda^2 a^2}{2} = 1 + \frac{2\lambda^2 a^2}{k} \leq e^{2\lambda^2 a^2/k}.$$

Each $\phi_i^{p,q}$ is bounded using the fact that each of the intervals created has at most $g_i = m_i g$ beads of color i for every i. By the inequality applied with $a \leq g$ and $\lambda = \frac{(k/m)^{1/3}}{4g}$ (ensuring that indeed $\lambda a \leq \frac{(k/m)^{1/3}}{4} < 1/2$), it follows that if every interval generated is allocated to an agent in order to minimize $\psi = \sum_{p,q\in[k],\ p\neq q,\ i\in[n]} \psi_i^{p,q}$, then the function ψ never increases during the algorithm. As

$$\psi(0) < nk^2 e^{2\lambda^2 g/k} = nk^2 e^{\varepsilon^2/8gk} < \frac{e^{\lambda\varepsilon/k}}{2}$$

the computation shows that at the end the absolute discrepancy is $\leq \varepsilon/k$. We omit the details. ◀

Next, we present two simple special cases where we obtain proper solutions efficiently with the optimal number of cuts, $n(k-1)$. In the first case, the number of beads of each color is equal to k, the number of agents. In the second case, we set the number of colors to be $n = 2$.

▶ **Proposition 1.** *There exists an efficient algorithm that solves any instance of Necklace Splitting for n colors and k agents where there are exactly k beads of each color, making at most $n(k-1)$ cuts.*

Proof. Traverse the necklace once bead by bead and cut between any pair of consecutive beads unless the second one is the first appearance of a bead of color i for some $i \in [n]$. After each cut made, if S is the set of colors present in the newly created interval J, we allocate J to an agent that has not received up to that point any beads of any color in S. To show that after each cut such an agent exists, first note that by the description above, no agent receives two beads of the same color. If J contains only one bead and its color is i, there must exist an agent who has not received any bead of color i up to that point, as there are as many agents as beads of color i. If J has $p \geq 2$ beads, of colors $c_1, ..., c_p \in [n]$ appearing in this order, we can still give it to an agent that has not received any bead of color c_1, since each of the other beads in J has a color that has not appeared before.

It thus follows that with this allocation rule each agent gets exactly 1 bead of each color. To prove the upper bound on the number of cuts, note that for each $i \in [n]$, we never cut right before the first bead of color i that appears on the necklace. Hence, there are exactly $n-1$ beads (besides the very first one) with no cut right before them. Since there are $kn-1$ points between consecutive beads the algorithm makes exactly $kn - 1 - (n-1) = n(k-1)$ cuts. ◀

▶ **Proposition 2.** *There exists an efficient algorithm that solves any instance of Necklace Splitting for $n = 2$ colors and k agents, making at most $2(k-1)$ cuts.*

Proof. We first consider the case when k divides both m_1, m_2, where m_i is the number of beads of color i. Given a necklace with m_1 beads of color 1 and m_2 beads of color 2 consider it as a circular necklace. By the discrete intermediate value theorem there is a circular arc of $(m_1 + m_2)/k$ beads containing exactly m_1/k beads of color 1 (and hence also exactly m_2/k beads of color 2). Cut in the ends of this circular arc, assign it to the first agent, and continue inductively. Clearly, every agent gets the same number of beads of each color.

To extend the proof for general m_1, m_2, write $m_1 = kp + r$ and $m_2 = kq + s$. We look for a circular arc of $\lceil \frac{m_1}{k} \rceil + \lceil \frac{m_2}{k} \rceil$ beads containing exactly $\lceil \frac{m_1}{k} \rceil$ beads of color 1 (and hence also exactly $\lceil \frac{m_2}{k} \rceil$ beads of color 2). If $r \neq 0$, the agent to whom we distribute the arc gets $p+1$ beads of color 1. Similarly, if $s \neq 0$, the agent gets $q+1$ beads of color 2. Hence, by

inductively finding a suitable arc and cutting it from the necklace, at the end of the process, the first r agents will get $p+1$ beads of color 1 and the rest p. Similarly, the first s agents will get $q+1$ beads of color 2 and the rest q. ◀

5.2 Connections to ε-Consensus Splitting

Our results easily extend to the ε-Consensus Splitting problem with non-atomic probability measures whose density functions are piecewise linear. This is stated in the next two theorems whose detailed proofs are provided in the full version.

▶ **Theorem 7.** *There exists an efficient, deterministic, offline algorithm that provides a solution to the ε-Consensus Splitting problem, making at most $n(k-1)\lceil 4+\log_2(3km)\rceil$ cuts, provided that the density functions of the probability measures are piecewise linear.*

▶ **Theorem 8.** *There exists an efficient, deterministic, online algorithm that provides a solution for the ε-Consensus Splitting problem, making at most $O(\frac{kn\log(nk)}{\varepsilon^2})$ cuts, provided that the density functions of the probability measures are piecewise linear.*

Note that for $k=2$ agents, the number of cuts resulting from the algorithm corresponding to Theorem 8 is $O(\frac{n\log n}{\varepsilon^2})$. The proof of Theorem 2 relies on using this algorithm for $k=2$ agents with $\varepsilon=\Theta((\frac{\log n}{m})^{1/3})$.

5.3 Open questions

Theorem 1 provides a proper solution to the offline version for $k=2$ agents by making a number of cuts that depends logarithmically on m, the maximum number of beads of a color. It would be interesting to see if this dependency can be improved asymptotically.

Another open question arises in the context of the Online Necklace Halving problem for $n=2$ colors, where the lower bound for the number of cuts is only $\Omega(\sqrt{m})$, whereas the upper bound for the number of cuts produced by our algorithm is $O(m^{2/3})$. Lastly, for the general case of n colors for the online version of Necklace Halving there is a $\Theta((\log n)^{1/3})$ gap between the lower bound and the algorithm we provided. It will be interesting to close these gaps.

References

1 Noga Alon. Splitting necklaces. *Advances in Mathematics*, 63(3):247–253, 1987.
2 Noga Alon. Non-constructive proofs in Combinatorics. *Proceedings of the International Congress of Mathematicians (ICM)*, 63:1421–1429, 1990.
3 Noga Alon, Michael Krivelevich, Joel H. Spencer, and Tibor Szabó. Discrepancy Games. *The Electronic Journal of Combinatorics*, 12(1):R51, 2005.
4 Noga Alon, Dana Moshkovitz, and Muli Safra. Algorithmic construction of sets for k-restrictions. *ACM Transactions on Algorithms*, 2(2):153–177, 2006.
5 Noga Alon and Douglas B West. The Borsuk-Ulam Theorem and Bisection of Necklaces. *Proceedings of the American Mathematical Society*, 98(4):623–628, 1986.
6 Nikhil Bansal. Constructive Algorithms for Discrepancy Minimization. *Proc. 51st Symposium on Foundations of Computer Science (IEEE)*, pages 3–10, 2010.
7 Nikhil Bansal and Joel H. Spencer. Deterministic Discrepancy Minimization. *Algorithmica*, 67(4):451–471, 2013.
8 Nikhil Bansal and Joel H. Spencer. On-line Balancing of Random Inputs. *Random Structures and Algorithms*, 57(4):879–891, 2020.

9 Sandeep N. Bhatt and Frank T. Leighton. A Framework For Solving VLSI Graph Layout Problems. *Journal of Computer and System Sciences*, 28(2):300–343, 1984.

10 Sandeep N. Bhatt and Charles E. Leiserson. How to assemble tree machines. *Proceedings of the 14th Symposium on the Theory of Computing, San Francisco*, pages 99–104, 1981.

11 Paul Simon Bonsma, Thomas Epping, and Winfried Hochstättler. Complexity results on restricted instances of a paint shop problem for words. *Discrete Appl. Math*, 154(9):1335–1343, 2006.

12 Steven J. Brams and Alan D. Taylor. Fair division: From cake-cutting to dispute resolution. *Cambridge University Press*, 1996.

13 Bruno Codenotti, Amin Saberi, Kasturi Varadarajan, and Yinyu Ye. The complexity of equilibria: Hardness results for economies via a correspondence with games. *Theoretical Computer Science*, 408(2-3):188–198, 2008.

14 Constantinos Daskalakis, Paul W. Goldberg, and Christos H. Papadimitriou. The Complexity of Computing a Nash Equilibrium. *Theoretical Computer Science*, 39(1):195–259, 2009.

15 Aris Filos-Ratsikas, Soren Kristoffer Stiil Frederiksen, Paul W. Goldberg, and Jie Zhang. Hardness Results for Consensus Halving. *43rd International Symposium on Mathematical Foundations of Computer Science (MFCS)*, pages 24:1–24:16, 2018.

16 Aris Filos-Ratsikas and Paul W. Goldberg. Consensus Halving is PPA-Complete. *Proceedings of the 50th Annual ACM Symposium on Theory of Computing (STOC)*, pages 51–64, 2018.

17 Aris Filos-Ratsikas and Paul W. Goldberg. The Complexity of Splitting Necklaces and Bisecting Ham Sandwiches. *Proceedings of the 51st Annual ACM Symposium on Theory of Computing (STOC)*, pages 638–649, 2019.

18 Aris Filos-Ratsikas, Alexandros Hollender, Katerina Sotiraki, and Manolis Zampetakis. Consensus Halving: Does it Ever Get Easier? *Proceedings of the 21st ACM Conference on Economics and Computation*, pages 381–399, 2020.

19 Charles H. Goldberg and Douglas B. West. Bisection of circle colorings. *SIAM J. Algebraic Discrete Methods*, 6(1):93–106, 1985.

20 Charles R. Hobby and John R. Rice. A moment problem in L_1 approximation. *Proceedings of the American Mathematical Society*, 16(4):665–670, 1965.

21 Frédéric Meunier. Simplotopal maps and necklace splitting. *Discrete Mathematics*, 323:14–26, 2014.

22 Christos H. Papadimitriou. On the complexity of the parity argument and other inefficient proofs of existence. *Journal of Computer and System Sciences*, 48(3):498–532, 1994.

23 Robert Tijdeman. On a distribution problem in finite and countable sets. *Journal of Combinatorial Theory, Series A*, 15(2):129–137, 1973.

Comparative Design-Choice Analysis of Color Refinement Algorithms Beyond the Worst Case

Markus Anders
TU Kaiserslautern, Germany
TU Darmstadt, Germany

Pascal Schweitzer
TU Kaiserslautern, Germany
TU Darmstadt, Germany

Florian Wetzels
TU Kaiserslautern, Germany

Abstract

Color refinement is a crucial subroutine in symmetry detection in theory as well as practice. It has further applications in machine learning and in computational problems from linear algebra.

While tight lower bounds for the worst case complexity are known [Berkholz, Bonsma, Grohe, ESA2013] no comparative analysis of design choices for color refinement algorithms is available.

We devise two models within which we can compare color refinement algorithms using formal methods, an online model and an approximation model. We use these to show that no online algorithm is competitive beyond a logarithmic factor and no algorithm can approximate the optimal color refinement splitting scheme beyond a logarithmic factor.

We also directly compare strategies used in practice showing that, on some graphs, queue based strategies outperform stack based ones by a logarithmic factor and vice versa. Similar results hold for strategies based on priority queues.

2012 ACM Subject Classification Theory of computation → Design and analysis of algorithms; Theory of computation → Online algorithms

Keywords and phrases Color refinement, Online algorithms, Graph isomorphism, Lower bounds

Digital Object Identifier 10.4230/LIPIcs.ICALP.2021.15

Category Track A: Algorithms, Complexity and Games

Related Version *Full Version*: `https://arxiv.org/abs/2103.10244`

Funding Received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (EngageS: grant agreement No. 820148).

1 Introduction

Color refinement, also known as 1-dimensional Weisfeiler-Leman algorithm, is a crucial cornerstone of symmetry detection in theory as well as practice. It emerged as a subroutine for algorithms solving the graph isomorphism problem and its efficiency remains to date one of the determining factors for the running time of practical isomorphism solvers. Modern, highly efficient implementations are based on Hopcroft's algorithm for automata minimization [8], which was first adapted to color refinement by McKay in his widely used tool NAUTY [11]. A more recent but also in the meantime large application area of color refinement can be found in machine learning. Specifically, color refinement is used in the Weisfeiler-Leman Kernel for graph classifications as a measure for similarity [15] and as the foundation of graph neural networks [13]. The algorithm can also be applied to effectively reduce the size of linear equation systems [7].

Given a graph, color refinement iteratively recolors the vertices producing increasingly fine partitions of vertices into color classes. Starting with an initial, usually monochromatic coloring, in each iteration the colors of the vertices are chosen to depend on the colors of the neighbors and their multiplicities. If vertices differ in the number of neighbors they have in some color class, the algorithm *splits* up the vertices accordingly by assigning them distinct colors. This is done exhaustively until no further splits are possible.

The applications mentioned above depend on highly engineered implementations of the algorithm. This is the reason why modern implementations meticulously optimize the color refinement subroutine treating many special cases with tailored code [1, 9, 12]. Especially in machine learning applications it is crucial to achieve scalability for big data inputs [15]. Overall, demand for fast implementations of color refinement is high. Since color refinement has a quasilinear worst case running time, even small logarithmic or constant factors can have a crucial impact.

Indeed, the best known implementation of color refinement runs in time $\mathcal{O}(m \log(n))$ (see [4, 10]). Remarkably, within a model with modest assumptions, a tight lower bound construction matching this upper bound was given in 2015 [4]. This result tells us that there are graphs for which color refinement, no matter how it is implemented, runs in $\Omega(m \log(n))$. However, the result does not make any comparative statements between various ways to implement color refinement. In fact, there are dramatic differences in the various implementations of color refinement. While all color refinement algorithms depend on performing the aforementioned splits, there is a lot of freedom as to which order we perform the splits in. A *worklist* is usually employed to determine in what order these splits are performed. Common choices include a stack, queue, priority queue or combinations of these.

So far however, there has been no rigorous analysis as to whether one worklist choice is superior over another – or how significant the order of splits actually is. Going one step further, a natural question is whether there are efficient optimal solutions. If not the case, maybe there are at least solutions that are competitive with all other methods.

Contribution. This paper performs an in-depth comparative analysis of design choices for color refinement algorithms. The first challenge is to actually find a model within which we can compare color refinement algorithms with formal methods. We employ a two-pronged approach. We distinguish (1) algorithms that may only use information realistically collected during the color refinement process itself, and (2) algorithms that are allowed to compute additional information about the underlying graph. Remarkably, our results in the two orthogonal models concur in their conclusion. Namely, that there is no design choice that is competitive beyond a logarithmic factor.

More specifically, in (1) we model algorithms that may only access information explored during the color refinement process itself. For this we define a formal online model within which, in fact, all practical algorithms operate. In this model, the algorithmic decisions of when to refine with respect to what may solely depend on this information. We prove that this information does not suffice to make optimal or even competitive choices, no matter the amount of computational power used. Specifically, we show no online algorithm is within a logarithmic factor of the offline optimum. We also investigate the direct relationship between practical (online) color refinement strategies. Each of the strategies stack, queue, and priority queue, is outperformed by another of the strategies by a logarithmic factor on some graphs.

For (2), we define an "offline" version of the problem, which is essentially to compute an optimal split order for a given graph. Through a reduction from the set cover problem we prove an approximation hardness result. Specifically, unless $\mathsf{P} = \mathsf{NP}$, no approximation factor

Algorithm 1 A typical rendition of color refinement.

```
1  function ColorRefinement(G, π)
       Input  : graph G, coloring π
       Output : refined coloring π
2      initialize empty worklist W;
3      put all cells of π into W;
4      while W is non-empty do
5          take a cell C from W;
6          for each cell X containing a neighbor of a vertex in C do
7              for each vertex in X count its neighbors in C ;
8              split X into X_1, ..., X_k in π, according to neighbor counts;
9              let X_i be one of the largest cells of X_1, ..., X_k;
10             put all sets X_1, ..., X_k except X_i into W;
11             if X ∈ W then replace X in W with X_i;
12     return π
```

in $o(\log(n))$ can be achieved by polynomial-time algorithms. This proves that unless $\mathsf{P} = \mathsf{NP}$, even when collecting more information about the underlying graph than current algorithms actually do, computing a competitive let alone optimal order of splits is intractable.

Overall, our results demonstrate that while the choice of worklist can indeed make a crucial difference, there is no clear optimal color refinement strategy. We conclude that users need to adapt color refinement algorithms to the specific type of graphs encountered in the algorithmic application area in mind.

2 Color Refinement

All graphs in this paper are finite, simple, undirected graphs, unless stated otherwise. The neighborhood of a vertex v is denoted $N(v)$. For a set of vertices $V' \subseteq V(G)$ the *neighborhood* is the set $N[V'] \coloneqq (\cup_{v \in V'} N(v)) \setminus V'$. A *coloring* of a graph G is a map $\pi \colon V(G) \to \mathcal{C}$ from the vertices to some set of colors. A (color) *class* is a set $\pi^{-1}(c)$ of vertices of the same color.

We begin with a discussion of the color refinement algorithm itself. Algorithm 1 describes a typical rendition of color refinement. The basic idea is as follows. If two vertices in some class X have a different number of neighbors in some class C then X can be split by partitioning it according to neighbor counts in C. Whenever we split up a class X according to its connections to another class C in such a fashion (see Line 7 and Line 8) we say that we *refine X with respect to C*. Specifically this means that after the split, two vertices have the same color precisely if they had the same color before the split and they have the same number of neighbors in C. We repeatedly split classes with respect to other classes until no further splits are possible. A partition not admitting further splits is called *equitable*.

Algorithm 1 maintains the classes with respect to which refinements still have to be performed in a worklist W. Note that the algorithm does not fully specify the internals of the worklist. Specifically, it does not state in Line 5 which cell is extracted from the worklist next. We should emphasize that the final partition into color classes is independent of the choices of cells that are extracted, however the overall running time may depend on it. Typical implementations use a stack, queue, priority queue or a similar data structure. All of these choices result in the same worst case running time of $\Theta((n+m)(\log n))$ (see [4]). To achieve this running time it is crucial to prevent one largest cell (Line 9) from being added to the worklist. Splits with respect to this class are already covered by the other classes.

Overall, the main design choice of the algorithm is the choice of when to split which class with respect to which other class. To describe a general framework for the possible strategies of what to split when, we first need to understand what information is available to the algorithm for making its decision.

2.1 Partial Quotient Graphs

For an equitable partition, quotient graphs capture the information of how many neighbors vertices from one class have in another class. They are used in individualization refinement algorithms as pruning invariants (see [12]). Typically, the quotient graph is computed on the fly during the execution of a color refinement algorithm.

We now introduce the concept of *partial quotient graphs*. These graphs are a tool to formalize the information gathered up to a certain point during the execution of color refinement algorithms. As we cannot precisely say which information an algorithm collects, the quotient graphs give an overapproximation of the available information and model all information that could have possibly been gathered. For the purpose of our lower bounds, overapproximating can only strengthen the conclusions.

The partial quotient graph of a colored graph (G, π) is denoted by $P(G, \pi)$. Quotient graphs are directed and contain self-loops. They include vertex labels l_V as well as edge labels l_E. The vertex set of $P(G, \pi)$ is the set of all sets of colors of (G, π), i.e., $V(P(G, \pi)) := 2^{\pi(V(G))}$. A set of colors also represents the union of the respective color classes.

Vertices of the partial quotient graph are labeled with the size of their corresponding set of vertices in G, i.e., for all sets of colors $c \in 2^{\pi(V(G))}$ we define $l_V(P(G, \pi))(c) := |\pi^{-1}(c)|$, where by $\pi^{-1}(c)$ we denote the vertices whose color is in c. The edge set contains all connections between (unions of) color classes that would not cause a split. Thus there is an edge from c_1 to c_2 if $\pi^{-1}(c_2)$ does not split $\pi^{-1}(c_1)$. Formally, this means

$$E(P(G, \pi)) := \{(c_1, c_2) \mid c_1, c_2 \in 2^{\pi(V(G))}, \forall v, w \in \pi^{-1}(c_1) : \ d_{\pi^{-1}(c_2)}(v) = d_{\pi^{-1}(c_2)}(w)\}.$$

Edges only exist whenever the connection between unions of color classes are regular on one side, so we can label each edge with the corresponding degree, i.e., $l_E(P(G, \pi))((c_1, c_2)) := d_{\pi^{-1}(c_2)}(v)$, where $v \in \pi^{-1}(c_1)$ is arbitrary.

Let us justify the definition with an example. Suppose we split in a monochromatic graph the class of all vertices with itself. Then the new coloring partitions the vertices precisely by degree. That is, classes contain vertices of the same degree. An algorithm would know this degree, since it has counted the edges incident with each vertex, but it would not know how many neighbors a vertex has within a current color class. In the partial quotient graph, there is an edge from each new color classes to the union of all color classes.

The definition of partial quotient graphs contains many more vertices and edges and information on these than would truly be available while executing color refinement. In fact, partial quotient graphs grow exponentially in size, since all possible unions of color classes are considered. Common color refinement algorithms clearly gather much less information. Firstly, only connections of classes that are involved in a refinement are actually considered. Secondly, only information about unions of colors that occurred as a color class in a previous step of the refinement is known. Thus, usually color refinement algorithms only uncover a small, polynomial-sized portion of the partial quotient graphs defined above.

However, for our lower bounds, we assume that algorithms have access to the entire partial quotient graphs. We show that even if we generously allow such access, the information is not sufficient to derive a strategy with constant competitive ratio. For upper bounds, we only use information of the aforementioned polynomial-sized portion of partial quotient graphs. In fact, the upper bounds are based on a stack-based approach akin to Algorithm 1.

Algorithm 2 Corresponding color refinement for a strategy W.

```
1 function ColorRefinement(G, π)
    Input  : graph G, coloring π
    Output : refined coloring π
2   create list S containing P(G, π);
3   while π is not equitable do
4     (C, X) := W(S);
5     for each vertex in X count its neighbors in C;
6     split X into X_1, ..., X_k in π, according to neighbor counts;
7     append P(G, π) to S;
8   return π
```

2.2 Online Model

We now define a model that bases the choice of which color classes to use for the next refinement solely on the information available through partial quotient graphs. Practical implementations such as a queue or a stack are naturally captured by this, but the model even allows for much more powerful choices. The goal is then to prove that no strategy based solely on information of partial quotient graphs is sufficient to make optimal choices.

Let us start by defining the concept of a *strategy* $W : \mathcal{P}^* \to (2^{\mathbb{N}})^2$. A strategy is a function mapping a sequence of quotient graphs $P = P_1 \cdots P_k \in \mathcal{P}^*$ to two vertices of the last quotient graph $(C, X) \in V(P_k)^2$, that is, two unions of color classes. The sequence of graphs P denotes all partial quotient graphs observed during execution of the algorithm up to step k. The pair (C, X) denotes the choice of colors with which the algorithm continues in the next step: in step $k+1$, the algorithm refines X with respect to C.

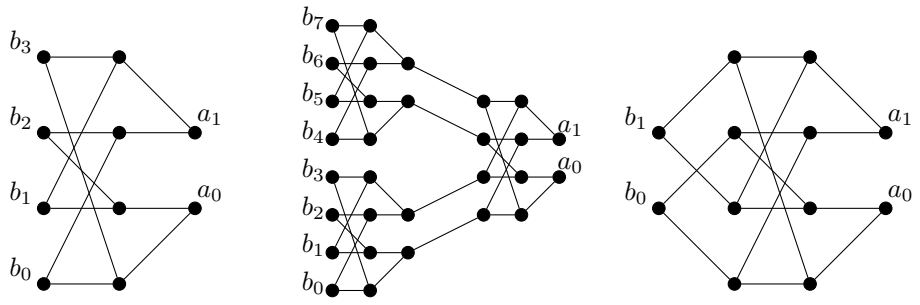
For a strategy W we now define a *corresponding color refinement implementation*. Assume we are working on G and have already refined up to a coloring π_k within k steps. Furthermore, let $P_1, \ldots, P_k$ denote the partial quotient graphs corresponding to the execution. Next, we compute $(C, X) = W(P_1 \cdots P_k)$ and refine X with respect to C. The algorithm terminates whenever π_k is equitable. A formal definition is given in Algorithm 2. We call W a *valid* strategy if the corresponding color refinement implementation is correct, i.e., if it terminates with an equitable partition in finite time on all finite graphs.

Throughout this paper, we measure the *cost* of the strategy W, denoted $\mathrm{cost}(W, G)$, in terms of the number of edges that need to be considered to execute the refinements. Specifically, when refining X with respect to C, we charge the algorithm the number of edges connecting X with C. This is the same model as used in [4] reflecting the actual running time of practical implementations (see [11, 12]). We use the terms cost and time interchangeably.

3 Graph Gadgets

Throughout the paper we construct graphs that cause color refinement to behave in particular manners. These graphs are mostly built using three types of graph gadgets, described next.

And gadgets. Let us first discuss the AND_i gadgets as used by Berkholz et al. [4]. There is set B of 2^i in-vertices that come in pairs and 2 out-vertices. The goal of the gadget is that whenever all pairs of *in-vertices* have been split, a split of two *out-vertices* a_0 and a_1 is induced, but not before.



(a) An AND_2 gadget. **(b)** An AND_3 gadget. **(c)** The unidirectional gadget.

Figure 1 Basic gadget constructions as used throughout the paper. Vertices labeled with b_i always denote in-vertices, while a_i denotes out-vertices.

The AND_2 gadget (see Figure 1) is the well known CFI-gadget [5], where two gates form the in-vertices B and the third one the out-vertices a_0, a_1.

The AND_i gadget is constructed recursively using AND_2 gadgets. For $i > 2$, the AND_i gadget is constructed by taking the union of two AND_{i-1} and one AND_2 gadget. The four out-vertices of the AND_{i-1} gadgets are then connected to the four in-vertices of the AND_2 gadget. Figure 1 shows how the AND_3 gadget can be constructed using three AND_2 gadgets.

The important property is that in an AND_i gadget, all pairs b_{2j}, b_{2j+1} with $j \in \{0, ..., 2^{i-1}\}$ need to be distinguished to induce a split of a_0 and a_1. We should also record a property for the opposite direction: if a_0 and a_1 are distinguished, no split on B should be induced.

Unidirectional gadgets. We now describe the *undirectional gadget*. As the name suggests, it blocks the continuation of a split of pairs in one direction but allows it in the opposite direction. Figure 1 illustrates the gadget.

The gadget behaves as follows. Consider in-vertices b_0, b_1 and out-vertices a_0 and a_1. Distinguishing b_0 and b_1 should induce a split of a_0 and a_1. However, distinguishing a_0 and a_1 should *not* cause a split of b_0 and b_1. The gadget is obtained through a modification of the AND_2 gadget. We use the fact that a split of out-vertices in AND_2 does not cause a split of the pairs of in-vertices. Therefore, by connecting the in-vertices to new vertices a_0 and a_1, such that the AND_2 gadget is activated by any of the two singletons, we get the desired property.

Interestingly, the unidirectional gadget has also been used as a crucial building block in [3] and [6] to study the complexity of various problems closely related to color refinement.

Concealer gadgets. We conclude our discussion of gadgets with the *concealer gadgets*. Similar to the AND_i gadget, a concealer gadget C_i of level i has 2^i in-vertices B and 2 out-vertices a_0, a_1. Whereas in the AND gadget, *all* input pairs need to be distinguished, the concealer gadget only includes *one* specific pair that causes a split of the out-vertices. We call the pair causing the split of out-vertices the *correct pair*, while all other pairs not causing the split are called *dead end pairs*.

The idea is that the correct pair can not be located easily by color refinement algorithms. Hence, the gadget *conceals* where refinement can be continued.

To achieve this behavior, the gadget consists of 2^{i-1} unidirectional gadgets and the out-vertices a_0, a_1. We modify all but one of the unidirectional gadgets so that the connection of the in-gate agrees with the one of the out-gate. This causes these gadgets to become dead

ends – activating any of these gadgets has no effect on the out-vertices. The last, unmodified unidirectional gadget is the only one that can actually split the out-vertices and is therefore the only correct gadget.

The out-vertices of the entire concealer gadget are then connected to the out-vertices of all the unidirectional gadgets so that activating the correct pair causes a split of the out-vertices. Figure 2 shows a concealer gadget C_3.

Since we did not specify which of the pairs is the correct pair, there are several concealer gadgets for each $i \in \mathbb{N}$. Abusing notation we denote all of them by C_i. The concealer gadgets have two crucial properties. First, as long as the correct pair has not been split (and the neighbors of a correct pair have not been split) the partial quotient graphs of two concealer gadgets on the same size are isomorphic. Second, the correct pair can only be split from outside the gadget. We formalize these properties in the following.

Consider two colored concealer gadgets $(C_i, \pi), (C_i', \pi')$ of the same order. Suppose $\{b_s, b_{s+1}\}$ is the correct pair in (C_i, π) and $\{b_t, b_{t+1}\}$ is the correct pair in (C_i', π'). We say the two graphs still *concur* if the colors for the vertices agree (note that the two graphs have the same vertex set) and in both graphs neither the correct pairs nor their neighbors have been split. Specifically, we require that

- the vertex colorings agree, (i.e., $\pi(v) = \pi'(v)$ for every $v \in V(C_i) = V(C_i')$),
- the correct pairs have not been distinguished (i.e., $\pi(b_s) = \pi(b_{s+1})$ and $\pi'(b_t) = \pi'(b_{t+1})$),
- the neighbors of the correct pairs have not been distinguished (i.e., $\pi(v) = \pi(v')$ for all $v, v' \in N_{C_i}(b_s) \cup N_{C_i}(b_{s+1})$ and $\pi(v) = \pi(v')$ for all $v, v' \in N_{C_i'}(b_t) \cup N_{C_i'}(b_{t+1})$).

▶ **Lemma 1.** *Suppose (C_i, π) and (C_i', π') are colored concealer gadgets that concur. Then the graphs have the same partial quotient graphs, i.e., $P(C_i, \pi) = P(C_i', \pi')$.*

Proof. Suppose for a vertex v we want to count the number of neighbors that v has in a union of color classes X. We claim that this number is the same in C_i and C_i'. Indeed, we only need to consider edges incident with v that have one endpoint in $M = \{b_s, b_{s+1}, b_t, b_{t+1}\}$ and one endpoint in $N[M]$ (the neighborhood of M). Let E' be the set of these edges and let E_v' be the set of these edges incident with v.

Note that for each of the four sets $\{b_s, b_{s+1}\}$, $\{b_t, b_{t+1}\}$, $N[\{b_s, b_{s+1}\}]$, and $N[\{b_t, b_{t+1}\}]$ either X contains the set entirely or not at all.

If v is in M then either all edges of E_v' have an endpoint in X or no such edge does.

Likewise if v is in $N[M]$ then either all edges of E_v' have an endpoint in X or no such edge does.

Moreover, in either case, whether all such edges are or no such edge is contained does not depend on whether we consider C_i or C_i'.

This implies that the number of edges counted in the refinement (i.e., those incident with v and having an endpoint in X) is the same in C_i and C_i'. ◀

▶ **Lemma 2.** *For concealer gadgets (C_i, π) and (C_i', π') suppose $\pi = \pi'$ so that*

- *vertices in an input pair that is correct in one of the graphs have the same color and*
- *all vertices that are not in an input pair have the same color.*

Then (C_i, π) and (C_i', π') concur. After an arbitrary sequence of splits to both graphs the resulting graphs still concur and neither correct input pairs nor the out pair are split.

Proof. This follows by induction on the number of steps observing that the functionality of the unidirectional gadget ensures that the output pair is never split, and thus vertices inside correct gadgets are never split. ◀

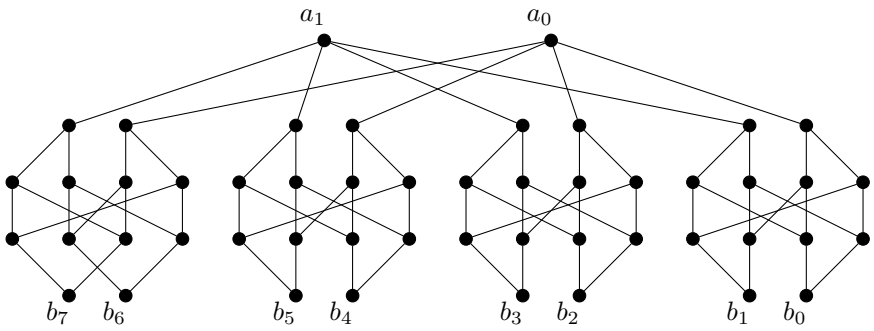


Figure 2 A concealer gadget C_3. Vertices b_6, b_7 form the correct pair; other pairs are dead ends.

The two lemmas show that unless a correct pair is split, the gadgets always concur and an algorithm in the online model will have to perform splits consistently on both graphs. Moreover, the output pair is never split.

Intuitively this means that in the online model, an algorithm can only guess which pair is the correct pair. Therefore, when faced with a concealer gadget, the algorithm potentially has to try all input pairs.

4 Competitive Ratio

We prove the non-existence of a c-competitive strategy in the online model. In particular, in this section, we prove the following theorem:

▶ **Theorem 3.** *For every strategy W of the online model, there is an infinite family of graphs G_k ($k \in \mathbb{N}$) such that* $\text{cost}(W, G_k) \in \Omega(\text{opt}(G_k) \cdot \log(\text{opt}(G_k)))$*, where* $\text{opt}(G_k) \in \Theta(|G_k|)$ *is the minimal cost of a strategy on G_k.*

The theorem implies that the information provided by partial quotient graphs is not sufficient to make competitive let alone optimal choices in color refinement algorithms.

Towards this goal, we first define the class of *concealer graphs*, which we denote with $\mathcal{G}_k$ ($k \in \mathbb{N}$). Concealer graphs resemble the graphs of the lower bound construction in [4] closely. Essentially, we swap out AND_i gadgets in the original construction for concealer gadgets C_i. A concealer graph of $\mathcal{G}_4$ is illustrated in Figure 3.

The main idea is that we can then speed-up or slow-down particular strategies by changing the position of the correct pairs within the concealer gadgets. This forces one strategy to extensively search for the correct pairs, while another strategy finds them immediately.

In the rest of this section we provide formal arguments for the above claims. We start with a precise description of concealer graphs. Then, we show that for every concealer graph there exists a fast strategy. Contrarily, we then provide a slow concealer graph for every strategy. Together these two statements prove Theorem 3.

4.1 Concealer Graphs

The first ingredient for the concealer graphs is a "splitting scheme" that results in the worst case running time of $\Omega(m \log(n))$. Consider a vertex set of size $n = 2^k$, on which the following refinements are performed. First, we split the set in halves, then quarters, then eighths and so on, until all vertices have their own distinct color. This gives us $\log(n)$ rounds of refinements, each with a cost of $\Omega(n)$. This results in total costs of $\Omega(n \log(n))$. By ensuring that sufficiently many edges are involved, the running time can be increased to $\Omega(m \log(n))$.

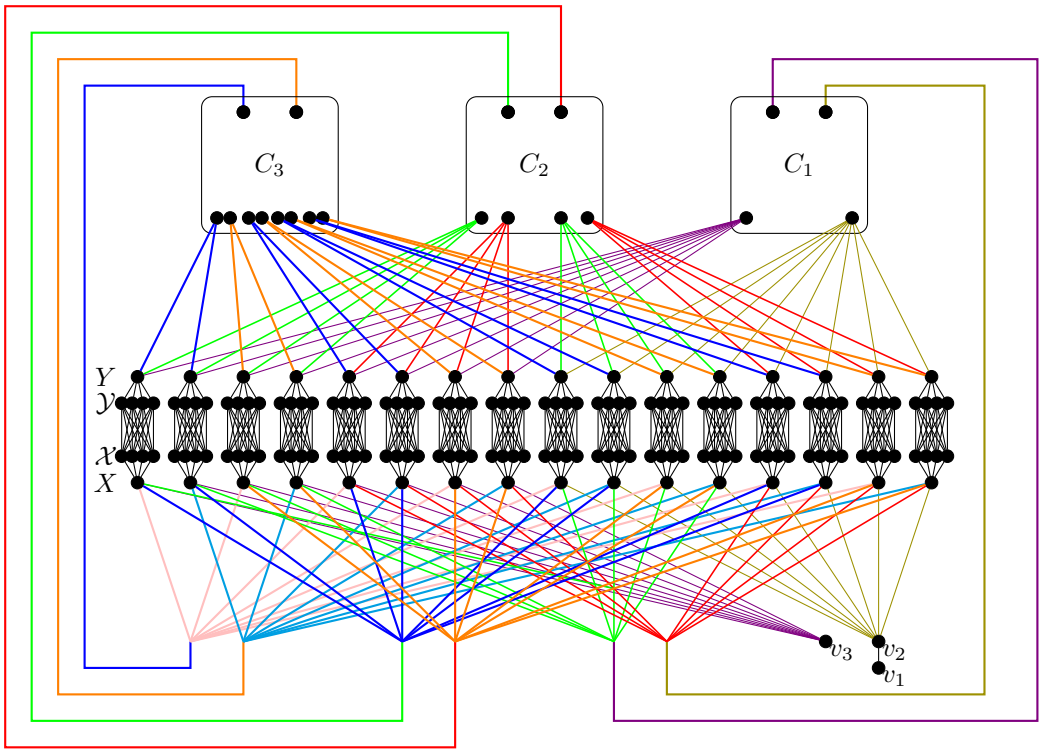


Figure 3 A concealer graph from the class $\mathcal{G}_4$.

Concealer graphs can be used to cause the splitting scheme just described. The graphs contain *middle layers* $(X, \mathcal{X}, \mathcal{Y}, Y)$ (see Figure 3) in which the splitting scheme can be forced. The graph is constructed in a way such that splitting Y into halves, quarters, eighths and so on, causes the next halving refinement on X. The edge colors in Figure 3 indicate the splitting scheme. While the halves (yellow and purple) of Y lead to a split of X into quarters (red and green), the quarters of Y lead to eighths (blue and orange) of X and so on. By initially splitting X in halves, any color refinement algorithm needs to cycle through these layers until X is fully discrete.

The core idea of the general lower bound construction in [4] is that the AND_i gadget enforces refinements with respect to *every* block of level i, which in turn ensures costs of $2^k \cdot k^2 \in \Omega(m)$ for every level.

We modify the construction to suit our purposes as follows. In the concealer graphs, we swap for each i the AND_i gadget for a concealer gadget C_i. On a particular graph, the worst case behavior is therefore not enforced for all refinement strategies anymore. However, a deterministic online algorithm cannot choose for *all* possible concealer gadgets the correct pair in level i to allow it to continue with level $i+1$. Hence, an adversary can construct a graph that makes a specific color refinement slow, while keeping a "shortcut" for other algorithms that choose the correct pair directly.

We now formally define the class $\mathcal{G}_k$ of concealer graphs. Note that for every $k \in \mathbb{N}$, we define a set of graphs $\mathcal{G}_k$. Essentially, we describe a graph $G_k \in \mathcal{G}_k$ based on concealer gadgets, and the set $\mathcal{G}_k$ then simply consists of all possible instantiations (i.e., positions of the correct pairs) for the included concealer gadgets.

At its core, a graph $G_k \in \mathcal{G}_k$ consists of the four middle layers of vertices $(X, \mathcal{X}, \mathcal{Y}, Y)$, that are interconnected using additional gadgets. Formally, the vertex set of G_k includes $X = \{x_0, ..., x_{2^k-1}\}$, $\mathcal{X} = \{x_i^j \mid 0 \leq i < 2^k, 0 \leq j < k\}$, $\mathcal{Y} = \{y_i^j \mid 0 \leq i < 2^k, 0 \leq j < k\}$,

$Y = \{y_0, ..., y_{2^k-1}\}$, a simple starting gadget induced by only three vertices v_1, v_2, v_3 and $k-1$ concealer gadgets. For $0 \leq l \leq k$ and $0 \leq q \leq 2^l - 1$ let $\mathcal{B}_q^l = \{q2^{k-l}, ..., (q+1)2^{k-l} - 1\}$ be the q-th binary block of level l. We use this notation on all sets of size 2^k for some $k \in \mathbb{N}$.

Every x_i is connected to a corresponding y_i via a complete bipartite graph of size k consisting of vertices in $\mathcal{X}$ and $\mathcal{Y}$ (see Figure 3). Formally, each x_i is connected to all x_i^j, y_i to all y_i^j and x_i^j to all $y_i^{j'}$. For each level $l \in \{1, ..., k-1\}$, the i-th binary block of level l is connected to the i-th in-vertex of the l-th concealer gadget. Furthermore, for each gadget C_l, we connect a_0 to all X_i^l with i even and a_1 to all X_i^l with i odd. The starting construction splits X into the blocks X_0^0 and X_1^0. We refer to the i-th in-vertex of the l-th concealer gadget as b_i^l and to the i-th out-vertex as a_i^l.

Let us generally consider how a refinement strategy has to operate on G_k. The algorithm starts with the monochromatic coloring of G_k. The first refinement always distinguishes vertices by their degree, meaning we get the individualized starting gadget $\{v_1\}, \{v_2\}, \{v_3\}$, the distinct layers in the middle $X, \mathcal{X} \cup \mathcal{Y}, Y$, the in- and out-vertices of the concealer gadgets $\bigcup_{l \in \{1,...,k-1\}} \{b_i^l, a_j^l \mid i \in \{0, ..., 2^l\}, j \in \{0, 1\}\}$, and the union of the inner vertices of the concealer gadgets. Next the middle layers are split in half. From this point onwards the splits that are possible depend on finding the correct pair in the gadgets. This can lead to fast or slow refinements, as discussed next.

4.2 A Fast Strategy for Every Concealer Graph

We now show that for every *fixed* concealer graph $G_k \in \mathcal{G}_k$ we can define a linear time strategy. We show this by providing an appropriate sequence of refinements.

For each concealer gadget C_l in G_k, let $b_{i_l}^l, b_{i_l+1}^l$ be the correct pair. Now consider an online refinement strategy on such a graph. After the first (and fixed) refinement, we refine X with respect to $\{v_2\}$ or $\{v_3\}$. We choose one half of $X_{i_1}^1$ for the next refinement and then $\mathcal{X}_{i_1}^1$, $\mathcal{Y}_{i_1}^1$ and $Y_{i_1}^1$ while propagating the split through the middle layers. The important property is that $Y_{i_1}^1$ always splits the correct pair of the next concealer gadget. The concealer gadget then in turn splits X into quarters. Now, we continue with the quarters $X_{i_2}^2$, $\mathcal{X}_{i_2}^2$, $\mathcal{Y}_{i_2}^2$ and $Y_{i_2}^2$, such that the second concealer gadget is activated. This splits X in eighths.

We now repeat this scheme, such that for each level we only propagate the blocks corresponding to correct pairs through the layers and immediately continue with the next level after activating the concealer gadget. When X is discrete, we get the equitable coloring by refining with respect to each level k block of X, $\mathcal{X}$, $\mathcal{Y}$ and Y.

Now consider the cost of this strategy. While cycling through the layers, the most expensive refinements are those with respect to the blocks of $\mathcal{X}$ and $\mathcal{Y}$. On level l, they have cost $2^{k-l} \cdot k^2$, which means the total cost for all levels is $2^k \cdot k^2 = \Theta(m)$. Once X is discrete the cost of the final refinements of $\mathcal{X}$, $\mathcal{Y}$ and Y is also in $\Theta(m)$.

Overall, the cost for an optimal solution for G_k is linear, i.e., $\text{opt}(G_k) \in \Theta(m)$. Note that since refinement is always continued with color classes that have just been created, the scheme actually follows a depth-first approach and can be implemented using a stack.

4.3 A Slow Concealer Graph for Every Strategy

For a *fixed* strategy W, we now provide an infinite family of concealer graphs G_k on which this strategy is slow, i.e., incurs super-linear cost. The family is constructed by choosing for every $k \in \mathbb{N}$ one specific concealer graph $G_k \in \mathcal{G}_k$.

We start with an arbitrary graph $G_k \in \mathcal{G}_k$. We run W on G_k and observe which color classes are split within the concealer gadgets. Say we are looking at concealer gadget C_i. If W distinguishes the correct pair in G_k, but there are still dead ends that have not been

distinguished, then we replace G_k by the graph $G'_k \in \mathcal{G}_k$ obtained from G_k by replacing the gadget C_i with another one so that a dead end not yet investigated becomes the correct pair. Due to Lemma 1 and Lemma 2 we know that up until the point where W finds the correct pair in C_i for graph G_k, the strategy W performs the same sequence of splits when executed on G'_k as on G_k. Thus, by doing these transformations exhaustively, we ensure W distinguishes all correct pairs in all the concealer gadgets last. This causes $2^k \cdot k^2$ cost per level and hence $2^k \cdot k^3 = \Theta(m \log(n))$ total cost.

Since the optimal solution for fixed G_k only has linear cost (see Section 4.2), we in turn get that $\text{cost}(W, G_k) \in \Omega(\text{opt}(G_k) \cdot \log(\text{opt}(G_k)))$. This in turn proves Theorem 3. Further details and a more formal reasoning can be found in the full version [2].

5 Comparison of Practical Worklists

We now compare specific, practical worklist data structures. First, we compare stacks with queues. We show that either of the two can asymptotically outperform the other. We can also show the same result for priority queues (see the full version [2]).

5.1 Stack Advantage over Queue

To see how a stack worklist might outperform a queue, recall the fast strategies for concealer graphs of Section 4. The specific, fast split scheme discussed there is realized by a worklist maintained as a stack. Indeed, whenever possible we continue with a "newest" class.

We conclude from Theorem 3 that there is a class of graphs on which a stack based worklist asymptotically outperforms a queue based worklist by a logarithmic factor.

We should remark that it is possible to prove the same result with a simpler construction that does not rely on concealer gadgets. We should also remark that the construction does not apply to all stack based worklists. However, it is possible to modify the construction such that a particular stack based worklist is optimal. For example this can be done for the worklist that choses smallest color classes first (see the full version [2]).

5.2 Queue Advantage over Stack

Now, we construct a graph class, called the *queue graphs*, for which a queue based worklist outperforms a stack based one by a logarithmic factor. This complements the result of the previous section. The construction is also based on the graph class of Berkholz et al. [4]. It is an extension of these graphs, which allows queue worklists to finish quickly but maintains the slow behavior for stacks. We provide an intuitive description. A formal definition and a detailed analysis is given in the full version [2].

The starting gadget. We use a starting gadget (see Figure 4) that forces a stack worklist to perform certain splits before others, while a queue worklist behaves differently.

Consider the gadget together with the coloring indicated in the figure. Any color refinement eventually splits the pairs $\{p_1^{1,3}, p_1^{2,3}\}$, $\{p_2^{1,3}, p_2^{2,3}\}$ and $\{a_0, a_1\}$. However, a stack based worklist splits $\{p_1^{1,3}, p_1^{2,3}\}$ or $\{p_2^{1,3}, p_2^{2,3}\}$ *before* $\{a_0, a_1\}$, while a queue based one splits $\{a_0, a_1\}$ before the other two pairs.

Graph class construction. We start with the graphs from [4] as a main building block. Recall that these graphs are the graphs from Section 4 where the concealer gadgets are replaced by AND gadgets. As argued in [4] a worst case behavior is enforced for every refinement strategy: any refinement on these graphs has a cost of $\Omega(2^k \cdot k^3) = \Omega(m \log(n))$.

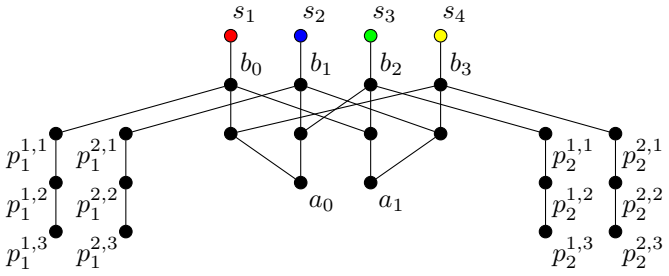


Figure 4 An extension of AND_2 gadget, which will be used as the starting gadget of the new construction. The starting vertices $s_1, ..., s_4$ have been individualized.

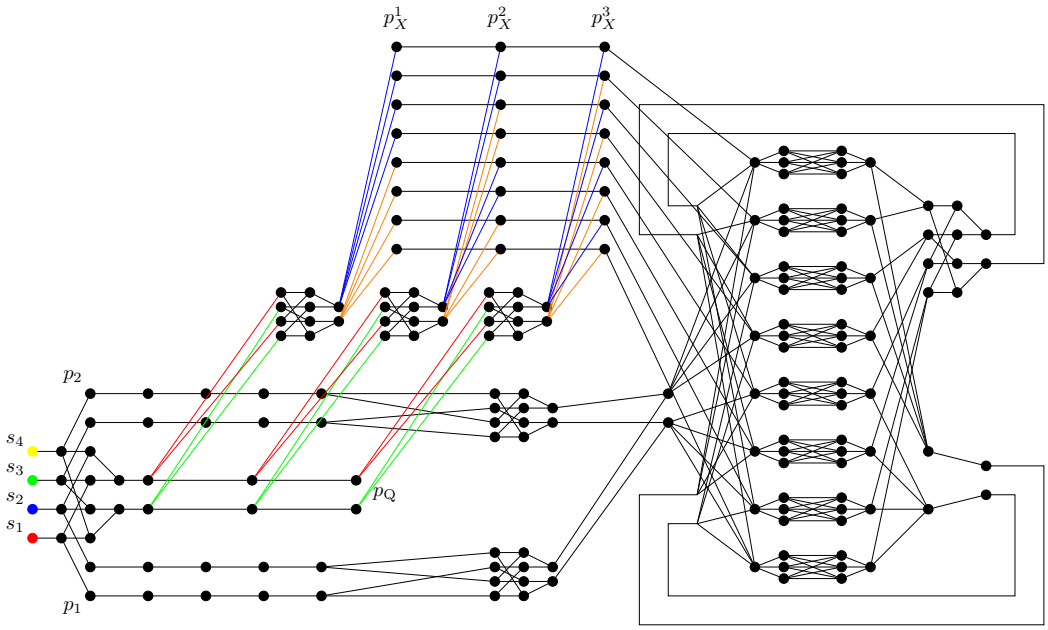
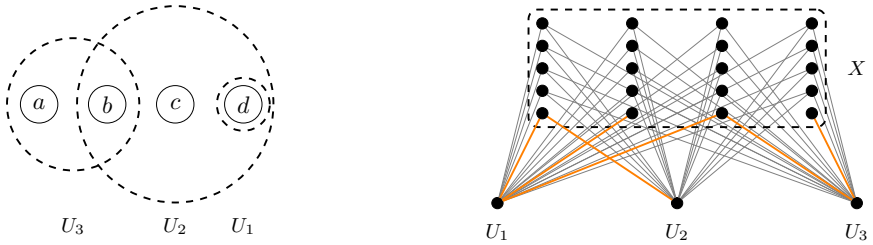


Figure 5 The graph G_3^{Q} for the queue advantage.

We now add "shortcuts" that allow queue based algorithms to bypass the construction. The core idea is to ensure the queue algorithm refines the set X into a discrete set within a single level of its breadth first behavior. This causes $\mathcal{X}$, $\mathcal{Y}$ and Y to completely split in subsequent rounds, thereby preventing the cycling behavior that causes superlinear cost. Indeed, if $\mathcal{X}$ and $\mathcal{Y}$ are handled only once, then the total cost is in $\mathcal{O}(2^k \cdot k^2) = \mathcal{O}(m)$.

Simultaneously we force the stack into the typical cycling behavior. We do so by forcing it to make the same splits of X as the simple starting gadget from Section 4 would.

We apply the following changes to define queue graphs G_k^{Q} (see Figure 5): we connect each vertex in X to a path of length k. We add the new starting gadget described above. We extend the paths p_1 and p_2 to a length of $k+2$ and connect the ends to the old starting vertices through unidirectional gadgets. We also attach a third path p_{Q} of length k to a_0 and a_1 and connect the i-th pair to the level-i blocks of the i-th vertices of the X-paths, again through unidirectional gadgets. Note that the graph has still a size of $\mathcal{O}(2^k \cdot k^2)$.



(a) Set cover instance. **(b)** Result of the reduction.

Figure 6 Reduction of the set cover instance $S = \{a, b, c, d\}$ and $\mathcal{U} = \{\{d\}, \{b, c, d\}, \{a, b\}\}$. Orange lines indicate connections to elements of S, all other edges are connections to dummy elements.

Queue Behavior. Consider a queue based color refinement. It splits the pairs within the paths p_1, p_2, p_Q layer by layer. The splits of the i-th pair of p_Q induce a split of the i-th vertices of the X-paths into the binary blocks of level i. After $k + 7$ many rounds, this leads to a split of X into the blocks of level k. Note that at the same time $\{p^0_{\text{end}}\}$ or $\{p^1_{\text{end}}\}$ will be able to split X. Therefore, we know that X will be fully discrete before any subset of $\mathcal{X}$ can be considered by the worklist.

Stack Behavior. Any stack based color refinement running on G^Q_k splits one of the paths p_1, p_2 in the starting gadget before splitting the path p_Q. This induces the worst case cycling behavior of the construction from [4]. The unidirectional gadgets and depth first strategy hinder the algorithm from distinguishing anything else in the starting gadget before the rest of the graph has been distinguished. Therefore, no "shortcut" can be applied and a stack based color refinement on G^Q_k has costs of at least $\Omega(m \log(n))$.

6 Approximation Hardness

Complementing our previous results, we now provide an approximation hardness result for computing optimal color refinement strategies. We begin by defining the optimal refinement worklist problem:

▶ **Problem** (Refinement Worklist Problem). *Given a colored graph (G, π), compute a minimal cost sequence of pairs of color classes $W = (C_1, X_1), \ldots, (C_t, X_t)$ such that:*

1. *Refining with respect to W results in the stable coloring π^∞.*
2. *For all prefixes $(C_1, X_1), \ldots, (C_s, X_s)$, the partial quotient graph obtained after refining C_i w.r.t. X_i for $i = 1, \ldots, s-1$ contains C_s and X_s (as unions of color classes).*

The cost of a sequence W is the sum of the costs for refining with respect to all $(C_i, X_i) \in W$.

The approximation hardness result is based on a reduction from the set cover problem. The set cover problem takes a finite universe S and a set of subsets of S, i.e., $\mathcal{U} \subseteq 2^S$. The decision variant then asks whether there exists a selection of k subsets in $\mathcal{U}$ whose union equals S. For simplicity, we assume $\bigcup_{U \in \mathcal{U}} U = S$. Set cover is well-known to be NP-complete.

The optimization variant requires a minimal selection of subsets that cover S, i.e., a solution that minimizes k. This problem is known to be NP-hard. More specifically, it is known that unless P = NP, polynomial-time algorithm can only reach an approximation factor of $\Omega(\log(n))$ [14].

▶ **Theorem 4.** *Unless* P = NP*, polynomial-time algorithms may only reach an approximation factor of* $\Omega(\log(n))$ *for the refinement worklist problem.*

Proof. We reduce the optimization variant of the set cover problem to the refinement worklist problem. More specifically, we reduce it in a manner which allows control of the parameters, so that the approximation hardness result of set cover immediately transfers to refinement worklists. The reduction is illustrated in Figure 6.

Given a set cover instance $(S,\mathcal{U})$ we define a related colored graph (G,π). We create one large color class X containing all elements of the universe S, as well as n^2 *dummy elements* (where n is the size of the set cover instance). Hence, the size of X is $n^2+|S|$.

We add a singleton color class for each subset $U \in \mathcal{U}$, i.e., we add vertex U with color U. We connect the vertex U with all vertices of X *except* for the elements that are contained in U. Formally, we define the edges $E(G) := \{\{U,x\} \mid x \in X \wedge x \notin U\}$. Note that U has $n^2+|S|-|U|$ connections to X.

In the constructed graph, all elements of the universe are eventually distinguished from the dummy elements in X. Refining X with respect to X is not productive, since there are no edges present and no splits occur. The only way to distinguish elements of X is to refine X with respect to an element of $\mathcal{U}$. Doing so always distinguishes all the elements contained in $U \in \mathcal{U}$ from the dummy elements and other remaining elements of X. Overall, we need to refine X with a subset of $\mathcal{U}$ that forms a set cover of S.

After that, assuming all elements of S have been distinguished from the dummy elements, it might be possible to split the resulting classes further through their connections to $\mathcal{U}$. However, the total cost for these further refinements is bounded by $c \cdot n^2$ for some fixed constant c.

The cost for refining X with respect to U is n^2+m, where m is the number of remaining elements of S in X *after* the elements of U have been removed. Since we need to choose at most $|S|$ subsets in a reasonable solution (otherwise we could remove redundant elements from the solution), and each time X gets smaller by at least one element, the cost incurred by m over all subsets is at most $|S|^2 \leq n^2$. Ignoring the cost of m, we get that each subset incurs additional cost of n^2 through the dummy elements.

Hence, the final cost is upper bounded by $c \cdot n^2 + (N_U+1) \cdot n^2 = (N_U+c+1)n^2$ and lower bounded by $N_U n^2$, where N_U is the number of chosen subsets.

We finish our arguments with a proof by contradiction. Assume there is a polynomial-time algorithm with an approximation factor in $o(\log(n))$. Given a set cover instance $(S,\mathcal{U})$, we apply the polynomial-time reduction stated above. Assume now we get an approximate solution with cost $x \cdot n^2$. We know that this implies a set cover solution with cost at most x.

The optimal set cover solution with cost x' would imply a worklist solution with cost at most $(x'+c)n^2$ (for a fixed c). Hence, we know that the worklist solution also approximates the optimal solution of the original set cover instance with a factor in $o(\log(n))$.

The set cover instance has a size in the 3rd root of the size of the refinement worklist problem. But since $o(\log(n)) = o(\log(\sqrt[3]{n}))$, we get a contradiction to the approximation hardness result of set cover. ◀

References

1 Markus Anders and Pascal Schweitzer. Engineering a fast probabilistic isomorphism test. In *2021 Proceedings of the Symposium on Algorithm Engineering and Experiments (ALENEX)*, pages 73–84. SIAM, 2021. doi:10.1137/1.9781611976472.6.

2 Markus Anders, Pascal Schweitzer, and Florian Wetzels. Comparative design-choice analysis of color refinement algorithms beyond the worst case. *CoRR*, abs/2103.10244, 2021. full version of the paper. arXiv:2103.10244.

3 Vikraman Arvind, Frank Fuhlbrück, Johannes Köbler, Sebastian Kuhnert, and Gaurav Rattan. The parameterized complexity of fixing number and vertex individualization in graphs. In *41st International Symposium on Mathematical Foundations of Computer Science, MFCS 2016, August 22-26, 2016 - Kraków, Poland*, volume 58 of *LIPIcs*, pages 13:1–13:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2016. doi:10.4230/LIPIcs.MFCS.2016.13.

4 Christoph Berkholz, Paul S. Bonsma, and Martin Grohe. Tight lower and upper bounds for the complexity of canonical colour refinement. *Theory Comput. Syst.*, 60(4):581–614, 2017. doi:10.1007/s00224-016-9686-0.

5 Jin-yi Cai, Martin Fürer, and Neil Immerman. An optimal lower bound on the number of variables for graph identifications. *Comb.*, 12(4):389–410, 1992. doi:10.1007/BF01305232.

6 Martin Grohe. Equivalence in finite-variable logics is complete for polynomial time. In *37th Annual Symposium on Foundations of Computer Science, FOCS '96, Burlington, Vermont, USA, 14-16 October, 1996*, pages 264–273. IEEE Computer Society, 1996. doi:10.1109/SFCS.1996.548485.

7 Martin Grohe, Kristian Kersting, Martin Mladenov, and Erkal Selman. Dimension reduction via colour refinement. In *Algorithms - ESA 2014 - 22th Annual European Symposium, Wroclaw, Poland, September 8-10, 2014. Proceedings*, volume 8737 of *Lecture Notes in Computer Science*, pages 505–516. Springer, 2014. doi:10.1007/978-3-662-44777-2_42.

8 J.E. Hopcroft. An n log n algorithm for minimizing states in a finite automaton. In *Theory of Machines and Computations*, pages 189–196. Academic Press, 1971.

9 Tommi A. Junttila and Petteri Kaski. Engineering an efficient canonical labeling tool for large and sparse graphs. In *Proceedings of the Nine Workshop on Algorithm Engineering and Experiments, ALENEX 2007, New Orleans, Louisiana, USA, January 6, 2007*. SIAM, 2007. doi:10.1137/1.9781611972870.13.

10 Sandra Kiefer and Brendan D. McKay. The iteration number of colour refinement. In *47th International Colloquium on Automata, Languages, and Programming, ICALP 2020, July 8-11, 2020, Saarbrücken, Germany (Virtual Conference)*, volume 168 of *LIPIcs*, pages 73:1–73:19. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPIcs.ICALP.2020.73.

11 Brendan D. McKay. Practical graph isomorphism. In *10th. Manitoba Conference on Numerical Mathematics and Computing (Winnipeg, 1980)*, pages 45–87, 1981.

12 Brendan D. McKay and Adolfo Piperno. Practical graph isomorphism, II. *J. Symb. Comput.*, 60:94–112, 2014. doi:10.1016/j.jsc.2013.09.003.

13 Christopher Morris, Martin Ritzert, Matthias Fey, William L. Hamilton, Jan Eric Lenssen, Gaurav Rattan, and Martin Grohe. Weisfeiler and Leman go neural: Higher-order graph neural networks. In *The Thirty-Third AAAI Conference on Artificial Intelligence, AAAI 2019, The Thirty-First Innovative Applications of Artificial Intelligence Conference, IAAI 2019, The Ninth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2019, Honolulu, Hawaii, USA, January 27 - February 1, 2019*, pages 4602–4609. AAAI Press, 2019. doi:10.1609/aaai.v33i01.33014602.

14 Ran Raz and Shmuel Safra. A sub-constant error-probability low-degree test, and a sub-constant error-probability PCP characterization of NP. In *Proceedings of the Twenty-Ninth Annual ACM Symposium on the Theory of Computing, El Paso, Texas, USA, May 4-6, 1997*, pages 475–484. ACM, 1997. doi:10.1145/258533.258641.

15 Nino Shervashidze, Pascal Schweitzer, Erik Jan van Leeuwen, Kurt Mehlhorn, and Karsten M. Borgwardt. Weisfeiler-lehman graph kernels. *J. Mach. Learn. Res.*, 12:2539–2561, 2011. URL: http://dl.acm.org/citation.cfm?id=2078187.

Search Problems in Trees with Symmetries: Near Optimal Traversal Strategies for Individualization-Refinement Algorithms

Markus Anders
TU Kaiserslautern, Germany
TU Darmstadt, Germany

Pascal Schweitzer
TU Kaiserslautern, Germany
TU Darmstadt, Germany

Abstract

We define a search problem on trees that closely captures the backtracking behavior of all current practical graph isomorphism algorithms. Given two trees with colored leaves, the goal is to find two leaves of matching color, one in each of the trees. The trees are subject to an invariance property which promises that for every pair of leaves of equal color there must be a symmetry (or an isomorphism) that maps one leaf to the other.

We describe a randomized algorithm with errors for which the number of visited nodes is quasilinear in the square root of the size of the smaller of the two trees. For inputs of bounded degree, we develop a Las Vegas algorithm with a similar running time.

We prove that these results are optimal up to logarithmic factors. For this, we show a lower bound for randomized algorithms on inputs of bounded degree that is the square root of the tree sizes. For inputs of unbounded degree, we show a linear lower bound for Las Vegas algorithms. For deterministic algorithms we can prove a linear bound even for inputs of bounded degree. This shows why randomized algorithms outperform deterministic ones.

Our results explain why the randomized "breadth-first with intermixed experimental path" search strategy of the isomorphism tool Traces (Piperno 2008) is often superior to the depth-first search strategy of other tools such as nauty (McKay 1977) or bliss (Junttila, Kaski 2007). However, our algorithm also provides a new traversal strategy, which is theoretically near optimal and which has better worst case behavior than traversal strategies that have previously been used.

2012 ACM Subject Classification Theory of computation → Design and analysis of algorithms; Theory of computation → Online algorithms

Keywords and phrases Online algorithms, Graph isomorphism, Lower bounds

Digital Object Identifier 10.4230/LIPIcs.ICALP.2021.16

Category Track A: Algorithms, Complexity and Games

Funding Received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (EngageS: grant agreement No. 820148).

1 Introduction

We define a new search problem involving trees with symmetries. In this problem, two unknown trees are given as input and they can be gradually explored. The leaves of the trees are colored and the task is to find a pair of leaves, one in each tree, with matching colors or determine that such a pair does not exist. The crucial element that distinguishes our model from standard exploration tasks is that the color of the leaves allows us to draw conclusions about the local surroundings of the leaf. More precisely, there is an invariance

axiom guaranteed to always hold. It says that if two leaves are of the same color then there is a symmetry (an automorphism or an isomorphism depending on the leaves being in the same tree or not) that maps one leaf to the other.

The invariance property guarantees that the local neighborhood around the leaf is structurally the same as the neighborhood around other leaves of the same color. This allows us to discard unexplored parts of the search tree thereby opening the possibility of having algorithms that explore only a sublinear number of the nodes of the trees.

Our motivation behind the model lies in the desire for a theoretical analysis of practical solvers for the graph isomorphism problem, automorphism group computations, and graph canonization. While the best theoretical algorithms, such as Babai's quasipolynomial time algorithm [2], are based on algorithmic group theory, practical solvers [4, 6, 7, 11, 15] exclusively follow the individualization-refinement (IR) paradigm. First introduced into the realm of practical isomorphism testing and canonization by McKay in 1977 [10], the basic principle has remained unchanged to date. Indeed, all algorithms in this paradigm perform some form of backtracking, implicitly creating a recursion tree for each input graph. The tools solve the graph isomorphism problem by finding two leaves in this recursion tree that correspond to each other. The number of nodes in the recursion tree that are actually called during the execution is closely linked to the running time of the overall algorithm. Despite being simple, our search problem and the exploration model capture precisely the task needed to be solved and assess correctly the running times of solutions.

Traditionally, IR tools have followed a depth-first search approach to traverse the search tree. However, in 2008, Piperno [15] introduced his tool Traces, which broke away from this principle, performing a form of breadth-first search that is intermixed with random traversal of the tree (so-called experimental paths). The traversal strategy is at the very heart of the underlying algorithm. Significantly outperforming all the other tools on most practical inputs [12], Traces revealed that the traversal strategy is arguably the most important design choice in IR algorithms. This immediately raises the question whether there are theoretical, structural reasons why this traversal strategy is favorable. Going one step further, we can ask for optimal traversal strategies.

However, so far there has been no rigorous justification as to why one traversal strategy should be superior and in particular there is no research into optimal traversal strategies.

Contribution. The introduction of our particular search problem in trees with symmetry allows us to strip away all the other design choices that have to be made in the creation of an IR algorithm and isolates the core issue of the traversal strategy in the search tree. The ultimate goal is twofold. Firstly, to provide a more rigorous, theoretical foundation for the design of practical graph isomorphism tools. Secondly, to design novel, near optimal strategies to be adopted in future generations of practical solvers.

An input consists of two trees without vertices that only have one child. Let n denote the size of the smaller one of the two trees and N the size of the larger one. The cost of our algorithms is measured in the number of nodes that are explored. For our algorithms, the terms "running time" or "complexity" refer to this cost measure.

Regarding *upper bounds*, we provide a simple randomized Monte Carlo algorithm with an upper bound of $\mathcal{O}(\sqrt{n}\log n)$ explored nodes. For trees of bounded degree we design a more complicated algorithm achieving an upper bound of $\mathcal{O}(\sqrt{n}\log N)$ for Las Vegas algorithms (i.e., randomized algorithms without errors).

Table 1 This table summarizes lower and upper bounds for the isomorphism problem implied by the results of this paper. Here $n = \min\{n_1, n_2\}$ and $N = \max\{n_1, n_2\}$, where the sizes of the trees are n_1, n_2 and d gives the maximum degree of the two input trees. We state separate lower bounds for trees with bounded (d-adic) and unbounded degree.

Setting	Lower Bound	Lower Bound (d-adic)	Upper Bound
Monte Carlo	$\Omega(\sqrt{n})$	$\Omega(\sqrt{n})$	$\mathcal{O}(\log(n)\sqrt{n})$
Las Vegas	$\Omega(n)$	$\Omega(\sqrt{n})$	$\mathcal{O}(d\log(N)\sqrt{n})$
Deterministic	$\Omega(n)$	$\Omega(n)$	$\mathcal{O}(n)$

These algorithms are accompanied by nearly matching *lower bounds*, showing that $\Omega(\sqrt{n})$ nodes need to be explored for randomized Monte Carlo algorithms even on bounded degree trees and that for unbounded degree inputs, Las Vegas algorithms need to visit $\Omega(n)$ nodes in expectation. For deterministic algorithms we get a lower bound of $\Omega(n)$ even for inputs of bounded degree. Table 1 gives an overview of the lower and upper bounds we prove.

Overall this shows that the new traversal strategies are optimal up to logarithmic factors. However, it also shows that randomized traversal strategies, even those without error, asymptotically outperform deterministic ones.

The algorithms for proving the upper bounds immediately imply IR algorithms with the same runtimes (up to almost linear factors in the graph order for non-recursive work). We should emphasize that our new upper bounds asymptotically outperform the traversal strategies that are currently being used in practice in the worst case. In fact, we implemented the Monte Carlo algorithm in a new graph isomorphism solver DEJAVU which runs significantly faster than existing tools on most graphs [1].

2 A Model for Tree Exploration with Symmetries

This section presents the exploration model that is used throughout this paper. The model enables us to perform a focused analysis of the traversal strategies used in the search trees of IR algorithms. We state the model independent of any discussion regarding IR.

2.1 Black Box Search Trees

In our search problems the input consists of one or two hidden trees of which certain information is to be discovered. We first explain how the trees can be explored.

Exploration Model. We consider rooted trees in which there is a priori no bound on the degree of the vertices. However, we require that no vertex has exactly one child. Furthermore, the leaves of the tree are colored.

Our exploration model for the trees restricts access of algorithms to the trees themselves and how they can be explored. We think of new information as being provided by an oracle to the exploration algorithm. During execution, a node of the tree is either *explored* or *unexplored*. Whenever a new node is explored, the algorithm learns the number of children of the node. In particular the algorithm knows whether the node is a leaf or not. Furthermore, in case the node is a leaf, it learns its color. At the beginning of an execution, everything except the root is deemed unexplored. The algorithm can only ever access previously explored nodes. The degree (i.e., the number of children) of an explored node v, which we denote by $\deg(v)$, is always known. To explore further nodes, the algorithm can explore a child

Figure 1 Example exploration in the black box search tree model. Starting from the root, the algorithm only ever knows explored nodes and their degrees. Through the use of an oracle, random children of explored nodes may then be queried.

of a previously explored node. Specifically, the algorithm can request the i-th child of v with $i \in \{1, \dots, d(v)\}$, which thereby becomes explored. For this, the input has an arbitrary but fixed ordering for the children of each vertex.

The *cost* of the exploration is measured in the number of oracle accesses, i.e., the number of nodes that are ever visited by the algorithm. (In particular there is no cost for traversing previously explored parts of the tree.)

Figure 1 illustrates such an exploration of a tree. Note that while the algorithm always knows the degree of explored nodes, it is essentially unable to chose a new specific child to explore since in another input the ordering of the children may be different.

More formally, a black box search tree $T = (V, E, \mathrm{col})$ consists of a rooted tree with colored leaves and for each node an ordering of the children. We omit the orderings from the notation. Of course all choices of orderings lead to proper search trees. The function $\mathrm{col}\colon L(T) \to \mathbb{N}$ maps the *leaves of the tree* $L(T)$ to natural numbers, referred to as colors.

In our algorithms, we use the procedure `NewChild`$\colon V \to V \cup \{\bot\}$ to explore the tree, which agrees with the previous description as follows. For an explored vertex v the algorithm chooses the smallest index of a previously unexplored child of v and queries the oracle for that child. If no unexplored child exists, the function returns $\bot$.

In the description of randomized algorithms, we also use the function `RandomChild`$\colon V \to V \cup \{\bot\}$, which returns a child chosen uniformly at random among all children of v, which means that it can in particular return previously explored children.

Isomorphism Invariance. So far we are lacking the crucial part of the model, namely symmetries. The core property of our trees is that the presence of leaves with equal colors implies the existence of symmetries of the trees. More specifically, they imply *color-preserving isomorphisms*, defined as follows. An isomorphism φ between two trees T_1 and T_2 is a bijection on vertices $\varphi\colon V(T_1) \to V(T_2)$, such that v is a child of v' if and only if $\varphi(v)$ is a child of $\varphi(v')$. A color-preserving isomorphism furthermore requires that $\mathrm{col}(l) = \mathrm{col}(\varphi(l))$ holds for all leaves $l \in L(V_1)$. This implies that leaves of a color can only be mapped to leaves of that same color. If $T_1 = T_2$ we also call φ an *automorphism* or a symmetry.

The crucial property that we require for all black box search trees is that whenever two leaves have the same color, we can derive an isomorphism:

▶ **Axiom** (Complete Isomorphism Invariance). *If $l_1 \in T_1, l_2 \in T_2$ and $\mathrm{col}(l_1) = \mathrm{col}(l_2)$, then there exists a color-preserving isomorphism $\varphi\colon V(T_1) \to V(T_2)$ such that $\varphi(l_1) = l_2$.*

We should highlight that the axiom in particular has to hold for the case $T_1 = T_2$, yielding automorphisms (possibly the identity if $l_1 = l_2$).

The crucial consequence of the axiom is that it allows us to draw conclusions about the structure of unexplored parts of the search tree. For example, applying this knowledge enables us to conclude that the last remaining node of Figure 1 is blue.

Throughout the paper, we assume that all inputs, may they consist of one or two trees, adhere to this axiom. Also, all exploration algorithms operate in our exploration model.

Algorithm 1 Random Walk of the Search Tree.

```
1 function RandomWalk(v)
    Input  : vertex v of a black box search tree
    Output : a random leaf of the search tree rooted at v
2   while deg(v) ≠ 0 do  v := RandomChild(v) ;
3   return v;
```

Isomorphism Exploration Problem. We are now ready to state our problem of interest for black box search trees: the isomorphism exploration problem.

▶ **Problem** (Isomorphism Exploration). *Given two search trees T_1 and T_2, compute leaves $l_1 \in T_1$ and $l_2 \in T_2$ with $\mathrm{col}(l_1) = \mathrm{col}(l_2)$, if they exist and return $\perp$ otherwise.*

For simplicity we will always assume that the trees are disjoint, that is $V(T_1) \cap V(T_2) = \varnothing$. This way we do not need to specify for oracle queries what tree they relate to.

There are other interesting problems, such as finding two leaves of the same color within one tree, that can be defined within the model. These and their relationship are discussed in Section 5.

While the tree model and the exploration problem can be defined it their own right, we have a concrete motivation behind the definitions. Indeed, the motivation behind the specifics of our model lies in so-called *individualization-refinement* algorithms, the prevailing method to solve the graph isomorphism problem in practice. In fact, the isomorphism exploration problem captures very closely the runtime of these algorithms. A more detailed explanation of this is deferred to the end of the paper in Section 5.

However, let us briefly remark that in our context the requirement that inner nodes may not have exactly one child is not only natural for our intended application but also crucial. If we drop this requirement the nature of the problem changes dramatically and ray searching as well as doubling techniques become relevant (see [3] for further pointers).

3 Upper Bounds

We provide upper bounds for the isomorphism exploration problem by developing appropriate algorithms. Of course by querying the oracle for the input trees alternatingly, there is a simple deterministic algorithm with a complexity of $\mathcal{O}(\min\{|T_1|, |T_2|\})$. For randomized algorithms, we start with a Monte Carlo algorithm. Subsequently we argue that there is also a Las Vegas algorithm, (i.e., an algorithm that always answers correctly) which still has a good expected runtime if there is a modest bound on the maximum degree of the tree.

3.1 Probabilistic Bidirectional Search

The central idea of the probabilistic isomorphism test discussed in this section is to perform *random root to leaf walks* in the black box search trees. The recursive procedure for such walks is described in Algorithm 1 and simply works as follows: a random walk is performed by starting in the root node and repeatedly choosing uniformly at random a child of the current node, until a leaf is reached.

Repeatedly executing random walks, the probabilistic isomorphism test exploits the following observation: assume we have two isomorphic trees T_1, T_2. Further assume we fix some leaf $l \in T_1$. We call all leaves l' with $\mathrm{col}(l) = \mathrm{col}(l')$ *occurrences* of l. The algorithm tries to find occurrences of l through random walks of the trees. Towards finding l, we

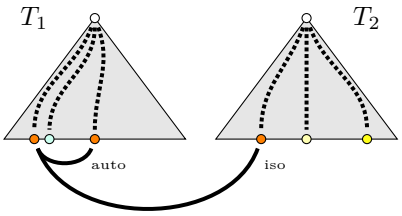


Figure 2 The probabilistic bidirectional search algorithm simultaneously samples leaves in both trees using random walks. It then tests for automorphisms within a tree and isomorphisms across trees to perform the probabilistic test.

always perform two random walks, one in T_1 and one in T_2. Since we assumed the trees *are isomorphic*, we are *equally likely* to find an occurrence of l in T_1 or in T_2. But if the trees *are not isomorphic*, we can find occurrences of l *only* in T_1 (otherwise, due to the isomorphism invariance axiom, T_1 and T_2 would be isomorphic).

Algorithm 2 describes a procedure based on this observation. Instead of using just a single leaf l however, it uses two sets of leaves L_1 and L_2 for comparison. Whenever an entirely new leaf is found (that is not an occurrence of a previously found leaf), it is added to the respective set of leaves and used for subsequent testing.

If a leaf is an occurrence of a previously discovered leaf, it either reveals an isomorphism between the trees or an automorphism (possibly the identity) of one of the trees. This is again a consequence of the isomorphism invariance axiom. If an isomorphism[1] is discovered, the algorithm has found two equally colored leaves in both trees and terminates. Otherwise, after a certain number of automorphisms have been found (the number depends on the desired error bound), the algorithm concludes that the trees are probably non-isomorphic within the given error bound. If the trees are isomorphic, we find automorphisms and isomorphisms with equal probability. Hence, we are highly unlikely to discover many automorphisms without also discovering an isomorphism. Figure 2 illustrates this key concept underlying the algorithm. The following lemma provides a correctness and running time analysis.

▶ **Lemma 1.** *Given black box search trees T_1, T_2 of heights h_1, h_2 and a desired error probability ϵ, Algorithm 2 solves the isomorphism exploration problem with probability at least $1-\epsilon$ with an expected worst-case runtime bounded by $\mathcal{O}\left(\lceil \log_2(\frac{1}{\epsilon})\rceil \cdot \max(h_1, h_2) \cdot \min\{\sqrt{|T_1|}, \sqrt{|T_2|}\}\right)$.*

Proof. *(Correctness.)* First, observe that whenever a pair of leaves is returned their color is checked for equality. This ensures that if the algorithm returns a pair of leaves, the answer is always correct. The algorithm can therefore only fail to produce the correct output by not finding a suitable pair of equally colored leaves despite the fact that they exist. In particular, this implies that if the trees are non-isomorphic, the algorithm cannot err.

To bound the error probability, we view the computation as a sequence of *tests*. A test repeatedly performs random walks of the search trees until one automorphism (possibly the identity) or one isomorphism is found. Hence, each test can be described as a sequence of j iterations. In each iteration $j' < j$, neither l_1 nor l_2 produced an isomorphism or automorphism. During a test, the algorithm neither terminates, nor is c incremented. In

[1] Slightly abusing terminology we often use the term isomorphism to mean an isomorphism between the two trees rather than an isomorphism that is an automorphism of one tree.

Algorithm 2 Probabilistic Bidirectional Search.

```
1  function Isomorphism(T_1, T_2, ε)
       Input  : black box search trees T_1, T_2 and probability ε
       Output : two leaves l_1 ∈ T_1, l_2 ∈ T_2 such that col(l_1) = col(l_2) with probability
                at least 1 − ε if such leaves exist, ⊥ otherwise
2      c := 0;
3      L_1 := L_2 := ∅;
4      while c ≤ ⌈−log_2(ε)⌉ do
5          f_(aut,1) := f_(aut,2) := false;   // f_(aut,i) signals automorphism found in
            T_i
6          l_1 := RandomWalk(root of T_1) ;
7          l_2 := RandomWalk(root of T_2) ;
8          if col(l_1) = col(l_2) then return (l_1, l_2);
9          for i ∈ {1, 2} do
10             for l' ∈ L_(3−i) do
11                 if col(l_i) = col(l') then return (l_i, l') ;
12                 if col(l_{3−i}) = col(l') then f_(aut,(3−i)) := true ;
13         if ¬f_(aut,1) then L_1 := L_1 ∪ {l_1};
14         if ¬f_(aut,2) then L_2 := L_2 ∪ {l_2};
15         if f_(aut,1) ∨ f_(aut,2) then c := c + 1;
16     return ⊥;
```

iteration j of the test, an automorphism or isomorphism is found. Now, note that when T_1 and T_2 are isomorphic, leaves contained in L_i can equally likely be found in T_1 or T_2. Hence, finding an automorphism or isomorphism in a test is equally likely. In particular, the probability is $\frac{1}{2}$ for finding an isomorphism for each $i \in \{1, 2\}$ rather than an automorphism. Anytime we find an automorphism but no isomorphism, we increment c by 1. We terminate when c reaches $e := \lceil -\log_2(\epsilon) \rceil$. Assuming the trees are isomorphic, the probability of this outcome is therefore bounded by $(\frac{1}{2})^e$.

(Runtime.) We will calculate the expected number of leaves explored before termination. We may consider the number of leaves instead of nodes by adding the multiplicative factor $\max(h_1, h_2)$ for the maximum length of a root to leaf walk in the search trees to our runtime. (We explain subsequently how to improve this factor.)

We may assume that the input trees are non-isomorphic and thus that the algorithm terminates because the condition $c > e = \lceil -\log_2(\epsilon) \rceil$ was met. This suffices to give an upper bound since earlier termination due to the discovery of isomorphisms clearly only leads to a smaller expected running time.

Consider running $2\sqrt{|T_i|}$ iterations of the algorithm. We may assume that in the j-th iteration L_1 and L_2 each contain at least j leaves: otherwise, some previous iteration already discovered an automorphism or an isomorphism. Furthermore, we may assume that the probability to find a leaf is uniform across all leaves: if probabilities are non-uniform, the chance for finding some leaves repeatedly only increases (see [13]). The probability of finding an automorphism in L_i (with $i \in \{1, 2\}$) within j iterations is therefore at least $\frac{j}{|T_i|}$. After $\sqrt{|T_i|}$ iterations, the probability for finding an automorphism in T_i is then at least $\frac{\sqrt{|T_i|}}{|T_i|} = \frac{1}{\sqrt{|T_i|}}$. Hence, the probability of finding no automorphism after $2\sqrt{|T_i|}$ many steps is at most

$$\left(\frac{\sqrt{|T_i|}-1}{\sqrt{|T_i|}}\right)^{\sqrt{|T_i|}} \leq \frac{1}{\mathbf{e}} < \frac{1}{2}, \quad \text{where } \mathbf{e} \text{ is the Euler constant.}$$

We view the computation as a series of batches consisting of $2\sqrt{|T_i|}$ iterations each. For each of them, the probability for finding an automorphism is at least $1/2$. For termination, we need to find e many automorphisms. The expected number of batches is thus in $\mathcal{O}(e)$, which shows that the overall number of iterations is in $\mathcal{O}(e \cdot 2\sqrt{|T_i|})$. ◀

We can improve the bound on the running time replacing the factor $\max\{h_1, h_2\}$ with the factor $\log_2(\min\{\sqrt{|T_1|}, \sqrt{|T_2|}\})$. To do so we alter the algorithm to take into account that the trees may be of very different sizes and also the trees may be quite unbalanced. To compensate for this we employ a doubling technique. However, we first need a bound for the expected length of the random root to leaf walks used in our algorithm.

▶ **Lemma 2.** *In an n-node black box search tree the expected length of a random root to leaf walk (i.e., the running time of Algorithm 1) is in $\mathcal{O}(\log n)$.*

Proof. Let $g(T)$ be the expected length of a random root to leaf walk in tree T. Note that the number of leaves t of a black box search tree is in $\Theta(n)$ for an n-node tree. We will argue that among the trees T with t leaves the value $g(T)$ is maximal if T is a binary tree in which all leaves are located on two consecutive levels. Since in such a tree even the maximum root to leaf distance is $\mathcal{O}(\log n)$, this proves the theorem.

First let T be a tree which has a vertex v with more than two children $u_1, \ldots, u_j$. Let T_i be the subtree of T rooted at u_i and assume without loss of generality that $g(T_i) < g(T_j)$ for $i < j$. Alter the tree T into a new tree T' by inserting a new node w as a child of v and then relocating the trees T_1 and T_2 so that their roots are now children of w instead of v. Then, conditional on the event that the random walk reaches v, the expected length of the walk has increased. Thus $g(T') > g(T)$. Since there are only finitely many trees with t leaves, by induction it suffices now to consider binary trees.

Let T be a binary tree and suppose there are leaves ℓ_1 and ℓ_2 whose height differs by more than 1. Say ℓ_1 is on the level furthest from the root. There must be another leaf ℓ_3 whose parent p is also the parent of ℓ_1. Alter the tree to obtain a new tree T' by assigning ℓ_2 as the new parent of ℓ_3 and ℓ_2. Note that p is further away from the root than ℓ_2. Thus, the tree being binary, the probability of a random walk reaching ℓ_2 is larger than that of reaching p. Therefore $g(T') > g(T)$. By induction this proves the theorem. ◀

▶ **Theorem 3.** *There is an algorithm that solves the isomorphism exploration problem with probability at least $1 - \epsilon$ and expected worst-case runtime bounded by*
$\mathcal{O}\left(\lceil \log_2(\frac{1}{\epsilon}) \rceil \cdot \log_2(\min\{\sqrt{|T_1|}, \sqrt{|T_2|}\}) \cdot \min\{\sqrt{|T_1|}, \sqrt{|T_2|}\}\right)$.

Proof. Set $n = \min\{|T_1|, |T_2|\}$. For an integer s, we run the algorithm with a budget $2s$ that limits the number of walks that can be performed in each tree to s. Furthermore, we limit the length of the random walks by $h = c \log_2(s)$ for some suitable constant determined later. Whenever a random walk exceeds the length h, we abort the walk and ignore it. If the algorithm does not terminate within the alloted budget then we double s and restart. This guarantees that the number of queries does not exceed $\mathcal{O}(s \log s)$ when we run it with integer s.

At least in the smaller of the two trees, automorphisms are found with high probability whenever s exceeds $\sqrt{n}$. Indeed, by Lemma 2 the average length of a random walk in the smaller tree is in $\mathcal{O}(\log n) = \mathcal{O}(\log \sqrt{n})$. Thus, by Markov's bound with probability $1/2$, the

random walks end in a leaf of height at most $\mathcal{O}(\log n)$. Thus, by the Chernoff bound, for sufficiently large s, with probability $1/2$ at least $1/4$ of the random walks end in a leaf of height at most $\mathcal{O}(\log n)$. We choose c so that this height is at most $c \log_2(n)$.

In case the graphs are isomorphic, automorphisms and isomorphisms are still found with equal probability. Thus our arguments for the probabilities remain in place since we essentially perform the same algorithm in pruned subtrees. Regarding the running time, note that the probability that the algorithm does not terminate with budget s decreases exponentially with s. That is, the probability is in $\mathcal{O}(a^{s/\min\{\sqrt{|T_1|},\sqrt{|T_2|}\}})$ for some constant $a < 1$ once $s > 2\min\{\sqrt{|T_1|}, \sqrt{|T_2|}\}$. ◀

We should remark that the collision problem was previously exploited in the context of the group isomorphism problem [16], but in that context structural information on the corresponding trees is known. Also the idea of sampling with random walks was used for the isomorphism algorithm in [17], but that algorithm only uses a single leaf in the search tree and thus cannot achieve sublinear running time guarantees.

3.2 Las Vegas Bidirectional Search

The major drawback of the probabilistic bidirectional search algorithm is that it makes errors. Considering trees of height 1 it is not difficult to see that a non-erring algorithm, even a randomized one, needs to query a linear fraction of the leaves to distinguish non-isomorphic trees. However, if the degree of the input graphs is restricted, we can beat this bound.

To do this, we basically strive to choose a specific set of nodes in both trees that ensures a "collision" of leaves. This guarantees that we find equally colored leaves, if they exist.

We refer to the maximum degree among the considered trees as d. Our main new idea is to split search trees in a balanced manner, followed by techniques to exploit isomorphism invariance. We want to note that the techniques for exploiting isomorphism invariance are inspired by techniques described in [12, 18], which essentially also perform splits. However, rather than heuristically applying them, here we perform them in a balanced and systematic way. Towards this goal we need the notion of a *split* (v, h), which is a node $v \in T_i$ at level h in one of the input trees. We define the *cost* of a split as a pair of numbers (s_1, s_2) as follows: s_1 is the size of the tree T_{3-i} truncated at level h (i.e., the ball of radius h around the root). If the tree T_i truncated at level h is non-isomorphic to the tree T_{3-i} truncated at level h then $s_2 := s_1$, *otherwise* s_2 is the size of the subtree rooted in $v \in T_i$ at level h.

The intuition for our exploration strategy is that s_1 bounds the size of the subtree to be explored in T_{3-i}, while s_2 bounds the size of the subtree to be explored in T_i (up to logarithmic factors). While the special treatment of the case when the trees disagree on the first h levels may seem cumbersome at first, the idea is that if trees already differ in the first h levels, we can decide non-isomorphism by exploring all nodes in the subtree of T_i consisting of the first h levels and then at most as many vertices within the first h levels of T_{3-i}.

We call a split (v, h) a *balanced split* whenever its cost (s_1, s_2) satisfies $\max\{s_1, s_2\} \leq 4d \cdot \min\{\sqrt{|T_1|}, \sqrt{|T_2|}\}$. Note that slightly abusing terminology in a balanced split the subtree with root v can be unproportionally large, as long as the two trees truncated at level h are non-isomorphic and s_1 is sufficiently small.

At this point it might neither be clear how to find a balanced split nor that a balanced split always exists. However, assume for now that we are given a balanced split. In that case we can efficiently solve isomorphism (even deterministically) as follows. We perform breadth-first search up to level h in both trees T_1 and T_2, visiting all nodes $N_1 \subseteq V(T_1)$ and $N_2 \subseteq V(T_2)$ up to and including level h. We can conclude non-isomorphism immediately

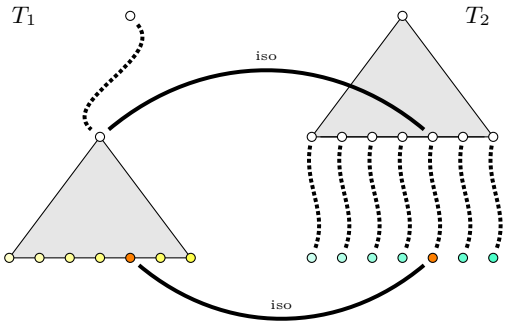


Figure 3 State of the search trees after termination of Algorithm 4. If trees are isomorphic, a node v_1 in T_1 at some level h must be mapped to some node v_2 in T_2 at level h by an isomorphism. But if that is the case, then a leaf below v_2 is isomorphic to some leaf below v_1.

whenever the breadth-first search has finished level h and the two trees truncated at level h are non-isomorphic. We can thus assume now that these trees are isomorphic. By exploring all nodes up to level h (which is the level containing v), we surely explore the node v in one of the trees. Without loss of generality assume in the following that $v \in V(T_1)$.

In T_1, we explore *all* leaves L_v of the subtree rooted at *one* fixed node, namely v from the balanced split. Let $N_2' \subseteq N_2$ denote the set of nodes at level h in T_2. Then, we explore for each node v' in N_2' *one* arbitrary leaf $l_{v'}$ in the subtree rooted at v'. If the trees are isomorphic, there must exist some $v' \in N_2'$ that can be mapped to v with an isomorphism. Since we explored all leaves of v, the leaf $l_{v'}$ with ancestor v' must be isomorphic (equally colored) to one of the leaves in L_v. Figure 3 illustrates how the collision of leaves is enforced through the exploration strategy.

The procedure for exploring, within our model, the first h levels of the subtree rooted at a particular node v is described in Algorithm 3. Starting from a given node v, it performs breadth-first traversal until only leaves are left, or h levels have been explored. The algorithm is also given a cost limit s and the algorithm aborts if this limit is reached.

Using Algorithm 3 as a subroutine, Algorithm 4 gives an implementation in the exploration model of the entire algorithm just described.

Let us argue an upper bound for the runtime of Algorithm 4 (still assuming that we are given a balanced split). From the definition of a balanced split, we can conclude that $|L_v|$ and $|N_2|$ are bounded by $\mathcal{O}(d \cdot \min\{\sqrt{|T_1|}, \sqrt{|T_2|}\})$. Since exploration up to level h in T_1 (Line 3) may only explore as many nodes as the exploration in T_2, we ensure that $|N_1| \leq |N_2|$ holds. Now, the last phase probes at most $\mathcal{O}(d \cdot \min\{\sqrt{|T_1|}, \sqrt{|T_2|}\})$ many paths, giving an overall upper bound of $\mathcal{O}\left(d \cdot h(T_2) \cdot \min\{\sqrt{|T_1|}, \sqrt{|T_2|}\}\right)$.

However, note that the factor $h(T_2)$ can be excessively large because the paths from level h to the leaves can be of length $\Theta(|T_2|)$. To prevent this, we alter the algorithm as follows. In T_2 we allocate a total budget of $c'\sqrt{|T_2|}\log(|T_2|)$ for some constant c' for all the level-h-to-leaves paths. By Lemma 2 the expected length of one such path is in $\mathcal{O}(\log|T_2|)$. Thus, by linearity of expectation, the expected total cost for the paths is $\mathcal{O}(\sqrt{|T_2|}\log|T_2|)$. By Markov's inequality for a suitable choice of c', with probability $1/2$ the total cost is in $\mathcal{O}(\sqrt{|T_2|}\log|T_2|)$. If the total cost exceeds this bound we simply restart the process.

Overall we can replace the factor $h(T_2)$ by $\mathcal{O}(\log(|T_2|))$, giving $\mathcal{O}(d \cdot \log_2(\max\{|T_1|, |T_2|\}) \cdot \min\{\sqrt{|T_1|}, \sqrt{|T_2|}\})$. More generally, it is easy to see that given a split of cost (s_1, s_2), the modification of Algorithm 4 runs in $\mathcal{O}(\log_2(\max\{|T_1|, |T_2|\}) \cdot \max\{s_1, s_2\})$. There is an interesting analogy to the runtime of the probabilistic bidirectional search algorithm. A main difference is that the runtime directly depends on the maximum degree of the trees.

Algorithm 3 Breadth-first Search: Computing a Subtree of the Search Tree.

```
1  function Subtree(v, h, s)
       Input  : start node v, height limit h, cost limit s
       Output : (L, s') where L is the set of leaves of the subtree under v up to level h
                and s' is the number of explored nodes, or (⊥, ⊥) if cost limit did not
                suffice
2      if h = 0 then return {v};
3      N := {(v, 0)};
4      L := {};
5      s' := 0;
6      while N ≠ ∅ do
7          (v, h') := Some(N) ;                    // pick arbitrary element of N
8          N := N ∖ {(v, h')};
9          h' := h' + 1;
10         c := NewChild(v);
11         while c ≠ ⊥ do
12             s' := s' + 1;
13             if s' > s then return (⊥, ⊥) ;
14             if h' = h ∨ deg(c) = 0 then L := L ∪ {c} ;
15             else N := N ∪ {(c, h')} ;
16             c := NewChild(v);
17     return (L, s');
```

The crucial question remains whether balanced splits always exist and whether they can be found efficiently. We first address the question of existence of balanced splits.

▶ **Lemma 4.** *Let T_1, T_2 be black box search trees with maximum degree d. Then there exists a balanced split for search trees T_1 and T_2.*

Moreover, if h' is the maximal level for which the tree T_1 truncated at level h' is smaller than $4d \cdot \min\{\sqrt{|T_1|}, \sqrt{|T_2|}\}$, the two subtrees up to level h' are isomorphic, and there are no leaves up to level h', then at least $\frac{3}{4}$ of the nodes at level h' in the smaller tree constitute balanced splits with cost $s_2 \leq 2\min\{\sqrt{|T_1|}, \sqrt{|T_2|}\}$.

Proof. We can assume w.l.o.g. that $|T_1| \leq |T_2|$. Let h' be the maximal level of T_2 where the size of the subtree up to level h' is smaller than or equal to $4d \cdot \min\{\sqrt{|T_1|}, \sqrt{|T_2|}\}$.

If the subtrees up to level h' in T_1 and T_2 differ, we have found a balanced split. Furthermore, if there are leaves in the trees up to level h', we have found a balanced split as well. Hence, we assume that subtrees are isomorphic and no leaves are present.

We now argue that at least $\frac{3}{4}$ of the nodes at level h' in T_1 constitute balanced splits. Consider level h' of T_1 and T_2. Let $s_{h'} \leq 4d \cdot \min\{\sqrt{|T_1|}, \sqrt{|T_2|}\}$ be the size of the subtree up to and including level h' in T_1. By assumption, the respective subtree of T_2 is of equal size. Furthermore, by assumption there are no leaves up to level h', implying that the tree contains at least $n_{h'} \geq \frac{1}{2} \cdot s_{h'}$ nodes at level h'.

Towards a contradiction, we assume that $s_{h'} \leq 4 \cdot \min\{\sqrt{|T_1|}, \sqrt{|T_2|}\}$. But then we can increment h': since $s_{h'} \leq 4 \cdot \min\{\sqrt{|T_1|}, \sqrt{|T_2|}\}$, it holds that $s_{h'+1} \leq 4d \cdot \min\{\sqrt{|T_1|}, \sqrt{|T_2|}\}$. This is a contradiction to the assumption that h' is maximal. Hence, we know $4 \cdot \min\{\sqrt{|T_1|}, \sqrt{|T_2|}\} < s_{h'} \leq 4d \cdot \min\{\sqrt{|T_1|}, \sqrt{|T_2|}\}$.

Algorithm 4 Bidirectional Search.

```
1  function Isomorphism(T_1, T_2, v, h)
       Input  : black box search trees T_1, T_2 and a split v, h with v ∈ V(T_1)
       Output : two leaves l_1 ∈ T_1, l_2 ∈ T_2 such that col(l_1) = col(l_2) if they exist, ⊥
                otherwise
2      (N_2, s) := Subtree(root of T_2, h, ∞) ;
3      (N_1, _) := Subtree(root of T_1, h, s) ;                    // explores v
4      if N_1 = ⊥ or T_1 and T_2 up to level h non-isomorphic then
5          return ⊥;
6      (L_v, _) := Subtree(v, ∞, ∞);
7      for n ∈ N_2 do
8          l := RandomWalk(n);      // can be an arbitrary, even determinsitic
9          for l' ∈ L_v do
10             if col(l) = col(l') then return (l, l') ;
11     return ⊥;
```

We can immediately conclude $n_{h'} \geq 2 \cdot \min\{\sqrt{|T_1|}, \sqrt{|T_2|}\}$. Naturally, there can be at most $\frac{1}{2} \cdot \min\{\sqrt{|T_1|}, \sqrt{|T_2|}\}$ corresponding subtrees which have a size greater than $2 \cdot \min\{\sqrt{|T_1|}, \sqrt{|T_2|}\}$ with roots at level h' in T_1 (since $|T_1| \leq |T_2|$). Consequently, there must be at least $n_{h'} - \frac{1}{2} \cdot \min\{\sqrt{|T_1|}, \sqrt{|T_2|}\} \geq \frac{3}{4} n_{h'}$ subtrees rooted at level h' with a size smaller than $2 \cdot \min\{\sqrt{|T_1|}, \sqrt{|T_2|}\}$. This concludes the proof for both claims of the lemma. ◀

Thus, for all search trees there exist balanced splits. We now explain how to find balanced splits efficiently. As shown by the lower bound in the next section, it is impossible to do this deterministically in an adequate running time. We thus need a randomized procedure for finding balanced splits. We will show that the following method is suitable. Rather than pseudocode we give a high level description in Algorithm 5.

It turns out that when the cost limit is sufficiently high the algorithm finds balanced splits with good probability. The intuition behind this is based on Lemma 4: once the algorithm reaches a sufficiently high level of the tree with breadth-first search, the majority of nodes constitute balanced splits.

From the description of the algorithm, the following corollary follows readily:

▶ **Corollary 5.** *If Algorithm 5 chooses in some iteration an element v at some level h in Step 3 so that (v, h) constitutes a split with cost (s_1, s_2) and at this point $s \geq s_2$, then the algorithm terminates after this iteration and returns a split with cost (s'_1, s'_2) where $s'_1 = s_1, s'_2 \leq s_2$.*

Proof. By assumption, (v, h) constitutes a split with cost (s_1, s_2). This implies that the entire subtree below v is smaller than s_2. Consequently, since s is large enough, Algorithm 5 explores the entire subtree below v in Step 4, unless probing the other way in parallel terminates first: this only contradicts our claim if it results in a more expensive split. However, since a more costly split necessitates more steps to explore its respective subtree, running the search in parallel ensures that the cheaper split is found first. Note that $s_1 = s'_1$ holds for all nodes at level h, concluding the proof. ◀

Using this, we can prove that Algorithm 5 terminates with a balanced split with good probability, giving what we need for our isomorphism test:

▶ **Lemma 6.** *Lemma If Algorithm 5 terminates with a split, it constitutes a balanced split with a probability of at least $\frac{3}{4}$.*

Algorithm 5 Las Vegas Balanced Splits.

function `SplitOrNotIsomorphic`(T_1, T_2)

Input : Black box search trees T_1 and T_2.

Output : Split (v, h) or conclude T_1 and T_2 are non-isomorphic

Step 1. Set cost limit $s \leftarrow 1$.

Step 2. Perform breadth-first search in T_1 and T_2, limiting the size of the traversed subtree to s nodes (each). If after any level the breadth-first search trees for T_1 and T_2 are non-isomorphic, terminate concluding non-isomorphism. Let h denote the level reached so far. If breadth-first search discovers a leaf v at or below level h, the algorithm terminates with the split (v, h).

Step 3. For each $i \in \{1, 2\}$, uniformly and independently at random choose a node v_i at level h in both trees. Compute breadth-first search starting from the node v_i in T_i, until one of the following conditions is met: (1) Breadth-first search finishes exploring the entire subtree of v_i, constituting the split (v_i, h). (2) Breadth-first search explored s nodes. This step is performed in parallel for both $i \in \{1, 2\}$ (i.e., each step alternates between the two), until one method succeeds in finding a split or both finish unsuccessfully. If at any point a split is found, the algorithm terminates immediately returning the split.

Step 4. Set $s \leftarrow 2s$ and jump to Step 2.

Proof. We can assume w.l.o.g. that $|T_1| \leq |T_2|$. Let h' be the maximal level of T_2 where the size of the subtree up to level h' is smaller than or equal to $4d \cdot \min\{\sqrt{|T_1|}, \sqrt{|T_2|}\}$.

First, we observe that we may always assume that the breadth-first trees explored in T_1 and T_2 up to level h' are isomorphic, since otherwise Algorithm 5 terminates immediately with no split (Step 2). Furthermore, since Algorithm 5 terminates when discovering leaves within the first h' levels in the breadth-first exploration (Step 2) and this result in balanced splits, we may assume that Algorithm 5 explores no leaves in the breadth-first search. We note that if Algorithm 5 finds no leaves, each doubling of s can only increase the level h reached by breadth-first search by at most 1.

Consider now Step 3 in the algorithm once level h' is reached. The algorithm picks a node of level h' uniformly at random. We now argue that with probability at least $\frac{3}{4}$ a node that is the root of a *small subtree* is chosen, i.e., a subtree that is smaller than $2 \cdot \min\{\sqrt{|T_1|}, \sqrt{|T_2|}\}$. This however follows readily from Lemma 4: since all subtrees at level h' are chosen for exploration with uniform probability, we can conclude that choosing a node that is the root of such a small subtree in T_1 has a probability of at least $\frac{3}{4}$. From the maximality of h' we can conclude that $s \geq 4 \cdot \min\{\sqrt{|T_1|}, \sqrt{|T_2|}\}$ (see proof of Lemma 4). Hence, Corollary 5 ensures that the algorithm terminates with a balanced split when choosing a node that is the root of a small subtree.

Furthermore, note that before level h' is reached, it is not possible for Algorithm 5 to return a split that is not a balanced split since the cost of probing is smaller than the bound for balanced splits. ◀

At level h' Algorithm 5 terminates with probability $\frac{3}{4}$. Careful inspection of the proof of Lemma 4 and Lemma 6 reveals that Algorithm 5 also terminates with probability at least $\frac{3}{4}$ after every consecutive doubling of s. While the cost (and therefore runtime) doubles, the probability of not terminating quarters, which defines a geometric series: this results in an expected runtime of Algorithm 5 bounded by $\mathcal{O}(d \cdot \min\{\sqrt{|T_1|}, \sqrt{|T_2|}\})$.

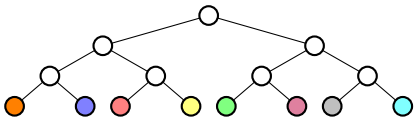


Figure 4 A search tree from the class $\mathcal{M}_3$.

We now run Algorithm 5 and Algorithm 4 in series, which results in the desired algorithm: if Algorithm 5 terminates with non-isomorphism we are done and otherwise Algorithm 4 tests isomorphism with the provided split. We observe that whenever Algorithm 5 terminates with a split, the costs of the split are also bounded by s: the execution time cannot be larger than the cost of the returned split. Running the previously described modification of Algorithm 4 with a split of cost s incurs expected cost bounded by $\mathcal{O}(\log_2(\max\{\sqrt{|T_1|}, \sqrt{|T_2|}\}) \cdot s)$. Using this, the following theorem follows.

▶ **Theorem 7.** *Let T_1, T_2 be black box search trees with maximum degree d. There exists an algorithm for the isomorphism exploration problem with no error that has an expected worst-case runtime bounded by $\mathcal{O}(d \cdot \log_2(\max\{\sqrt{|T_1|}, \sqrt{|T_2|}\}) \cdot \min\{\sqrt{|T_1|}, \sqrt{|T_2|}\})$.*

4 Lower Bounds

We prove lower bounds within the confines of the model. Easy lower bounds can be obtained by considering input trees of height 1, however, we are interested in bounds that also apply to trees of bounded degree. We utilize the search tree family $\mathcal{M}_h$ for this purpose (see Figure 4). A tree is in $\mathcal{M}_h$, if it is a complete binary tree of height h such that leaves have pair-wise distinct colors, i.e., for all $(l_1, l_2) \in L(V(\mathcal{M}_h))^2$ with $l_1 \neq l_2$ it holds that $\text{col}(l_1) \neq \text{col}(l_2)$.

We remark that shrunken multipedes (see [14]) are graphs that produce search trees very similar to those in $\mathcal{M}_h$ when used as input for IR algorithms.

Generally, due to their uniformity, trees from $\mathcal{M}_h$ can only be distinguished or proven isomorphic by considering leaves. A traversal strategy must either conclude – with good probability $(\frac{1}{2})$– that the set of leaves of the trees are entirely disjoint or equal. In the case when trees are isomorphic, the traversal strategy must provide two leaves with equal colors.

4.1 Randomized Lower Bound

We prove lower bounds for the isomorphism problem for randomized algorithms that err.

We will use a particular type of exploration algorithm for our purposes. We call an algorithm *unadaptive* on a class of inputs, if on each input from the class, the number of queries is always the same (in particular independent of randomness involved in the algorithm) and the queries performed by the algorithms on inputs from the class are independent of the answers given by the oracle. (The queries may still depend on the randomness involved in the algorithm.) This means in particular that even when matching leaves have been found, the algorithm will simply continue to run, possibly making further queries, and at some later point make a decision about the output.

▶ **Lemma 8.** *If some (possibly randomized) algorithm A solves the isomorphism exploration problem with expected run-time $f(n)$ and error-probability ϵ then for each $h \in \mathbb{Z}$ there is a randomized algorithm B that is unadaptive on the class of inputs $\mathcal{M}_h$ with a run-time in $\mathcal{O}(f(n))$ and error-probability ϵ.*

Proof. If an algorithm A solving the problem with expected run-time $f(n)$ and error probability ϵ is given, then by repeating the algorithm and using Markov's inequality we can design an algorithm A' with a run-time bounded by $\mathcal{O}(f(n))$ (not just in expectation) that still has an error probability of ϵ. For this note that even if the trees have been partially explored, it is possible to simulate the algorithm from scratch by pretending that explored nodes of the tree are unexplored.

To obtain the algorithm B we alter algorithm A' by simply pretending all discovered leaves have a randomly chosen previously unused color. More precisely, when a leaf is discovered, we pretend it has a color in $\{1, \ldots, 2^h\}$ drawn independently and uniformly at random from the colors that have not been used yet. We continue the simulation until A' halts. We then claim the input to be a yes instance if we found matching leaves and a no instance otherwise. This can only decrease the error probability in comparison to A'. ◀

For our lower bound we define a combinatorial problem of trees. Let M_h be the complete binary tree of height h, so that trees in $\mathcal{M}_h$ are colored versions of M_h. For two rooted trees U, S let $\mathrm{Inj}(U, S)$ be the set of root respecting injective homomorphisms from U to S. That is, the set contains the injective maps from $V(U)$ to $V(S)$ that map the root of U to the root of S and that map an edge of U to an edge of S.

From now on fix a height h and consider the tree M_h. Let $\mathcal{U}_h$ be the set of trees U for which $\mathrm{Inj}(U, M_h)$ is non-empty. This set contains exactly the trees isomorphic to a subtree of M_h. For two trees $U_1, U_2 \in \mathcal{U}_h$ we let $P(h, U_1, U_2)$ be the probability that for uniformly chosen $\alpha_1 \in \mathrm{Inj}(U_1, M_h)$ and independently, uniformly chosen $\alpha_2 \in \mathrm{Inj}(U_2, M_h)$ the set $L(M_h) \cap \alpha_1(V(U_1)) \cap \alpha_2(V(U_2))$ is non-empty. For integers a, b define

$$P(h, a, b) = \max \{P(U_1, U_2) \mid |L(U_1)| = a \wedge |L(U_2)| = b\}.$$

Let $P(h, m) = \max\{P(h, a, b) \mid a + b \leq m\}$. We will argue that $P(h, m)$ constitutes an upper bound on the probability of success for a randomized algorithm for isomorphism exploration that queries at most m nodes.

▶ **Lemma 9.** *Let B be an algorithm that is unadaptive on the class of inputs from $\mathcal{M}_h$. Suppose on inputs from $\mathcal{M}_h$ algorithm B makes m queries and has error probability ϵ. Then $1 - \epsilon \leq P(h, m)$.*

Proof. Consider the behavior of algorithm B on inputs from $\mathcal{M}_h$ with the colors $\{1, \ldots, 2^h\}$ being randomly assigned bijectively to the leaves. The algorithm B explores subtrees T_1' and T_2', one in each of the input trees. Since the algorithm makes m queries, together these trees can have at most m leaves. Our argument groups the possibilities in which B can query the oracle according to the topology of the two subtrees.

For two trees U_1 and U_2 consider the event E_{U_1,U_2} that T_1' is isomorphic to U_1 and T_2' is isomorphic to U_2. The event can of course only occur if $|U_1| + |U_2| \leq m$. Recall that algorithm B, being unadaptive, does not use the information on colors of the leaves provided by the oracle until the very end. Thus, the probability that B finds matching leaves on isomorphic inputs conditional to event E_{U_1,U_2} is $P(h, U_1, U_2)$.

We conclude that the probability that B finds matching leaves[2] is at most $P(h, m)$. ◀

We show that the trees need to have sufficiently many leaves for $P(h, T_1, T_2)$ to be large.

[2] In our problem definition the algorithm has to find two leaves of the same color. If the task only asked to decide whether the graphs are isomorphic, the algorithm could still guess, which would incur another factor of 1/2.

▶ **Lemma 10.** $P(h,a,b) \leq \frac{ab}{2^h}$.

Proof. For two trees T_1 and T_2 let $E(T_1,T_2)$ be the expected number of elements contained in the set $L(\mathcal{M}_h) \cap \alpha_1(V(T_1)) \cap \alpha_2(V(T_2))$, where α_1 and α_2 are taken independently and uniformly from $\mathrm{Inj}(T_1,\mathcal{M}_h)$ and $\mathrm{Inj}(T_2,\mathcal{M}_h)$, respectively. We define $E(h,a,b)$ in analogy to $P(h,a,b)$ as the maximum $E(T_1,T_2)$ over all choices of T_1 and T_2 with $|L(T_1)| = a$ and $|L(T_2)| = b$. By the Markov inequality it suffices to show that $E(h,a,b) \leq \frac{ab}{2^h}$.

Only vertices that are of distance h from the root in T_i can be mapped to a vertex in $L(\mathcal{M}_h)$. The automorphism group of $\mathcal{M}_h$ can map each leaf to every other leaf (i.e., acts transitively on the leaves). The graph $\mathcal{M}_h$ has 2^h leaves. Thus, for vertices $v_1 \in T_1$ and $v_2 \in T_2$ both of distance h from the root, the probability that $\alpha(v_1) = \alpha(v_2)$ is at most $\frac{1}{2^h}$.

By linearity of expectation the expected number of pairs (v_1,v_2) for which $\alpha(v_1) = \alpha(v_2) \in L(\mathcal{M}_h)$ is at most $\frac{1}{2^h} \cdot a \cdot b$. ◀

▶ **Theorem 11** (randomized lower bound)**.** *In the black box search tree model, a (possibly randomized making errors) traversal strategy runs in* $\Omega(\min\{\sqrt{|T_1|}, \sqrt{|T_2|}\})$ *worst-case cost for the isomorphism exploration problem, even on binary trees.*

Proof. By Lemma 8, it suffices to show the statement for an unadaptive algorithm B on $\mathcal{M}_h$. By Lemma 10, if B queries less than $\frac{1}{2}\sqrt{|\mathcal{M}_h|}$ nodes then both trees T_1 and T_2 uncovered by B have at most $\frac{1}{2}\sqrt{|\mathcal{M}_h|}$ leaves. But by the previous lemma we know that $P(h, \frac{1}{2}\sqrt{|\mathcal{M}_h|}, \frac{1}{2}\sqrt{|\mathcal{M}_h|}) \leq \frac{1}{4}$, which shows that the probability that B finds matching leaves in the two trees is at most $\frac{1}{4}$. This shows that B cannot find matching leaves with probability $\frac{1}{2}$. ◀

4.2 Deterministic Lower Bound

We exploit the randomized lower bound to obtain a strengthened deterministic one.

▶ **Theorem 12** (deterministic lower bound)**.** *In the black box search tree model, a deterministic traversal strategy runs in* $\Omega(\min\{|T_1|, |T_2|\})$ *worst-case cost for the isomorphism exploration problem, even on binary trees.*

Proof. Consider a deterministic algorithm on inputs from $\mathcal{M}_{2h}$, where $h = \log(n)$ and n is a power of 2. By Theorem 11 there are instances consisting of pairs of trees T_1, T_2 on which the algorithm makes $\Theta(\sqrt{2^{2\log(n)}}) = \Theta(n)$ queries in total. We know from the proof that the trees T_i can be chosen from $\mathcal{M}_{2h}$. For each $i \in \{1,2\}$, remove from T_i all non-root vertices whose parents have not been explored (and thus who have not been explored either). Let T_i' be the resulting tree, respectively for each i. On the input pair (T_1',T_2') the algorithm behaves exactly the same as on (T_1,T_2) and thus also makes $\Theta(n)$ queries in total, however T_i' has at most $\mathcal{O}(n)$ vertices. This shows that on (T_1',T_2') the algorithm makes $\Omega(\min\{|T_1'|, |T_2'|\})$ queries. ◀

Note that balanced splits for the trees of $\mathcal{M}_h$ can be found almost trivially: after finding out the height h through a single walk, an arbitrary node at level $\frac{h}{2}$ will induce a balanced split. This shows that while $\mathcal{M}_h$ constitutes worst-case examples for probabilistic algorithms, this is not true for deterministic algorithms. And indeed, our deterministic lower bound applies to subtrees of trees in $\mathcal{M}_h$ which have leaves on different levels.

5 Motivation Behind the Model

The motivation behind the specifics of our model lies in so-called *individualization-refinement* (IR) algorithms, the prevailing method to solve the graph isomorphism problem in practice. We explain that and why the isomorphism exploration problem captures the runtime of these algorithms, but refer to [11, 12] for a formal definition of IR algorithms.

Currently all practical state-of-the-art tools are based on the IR paradigm. Invariably, these algorithms perform a type of backtracking procedure to explore the structure of input graphs. Naturally, this leads to a *search tree*. While work is performed by the algorithms in each node of the search tree, the dominating factor for the running time is the size of the tree itself. Specifically, the running time per node is almost linear. However, the size of the search tree is exponential in the worst case [14].

When solving for isomorphisms of two graphs we get two search trees, one for each of the graphs. The leaves of the search trees correspond to complete invariants (described in more detail below), i.e., colors in our terminology. The task of finding isomorphisms in the graphs translates to finding pairs of leaves of equal color in the trees.

As mentioned earlier, nowadays there are various software packages implementing the paradigm in different flavors. These packages differ in many details (see below), of which the most crucial aspect is the traversal strategy through the tree.

The Search Tree. While the problem definition, algorithms, and lower bounds in this paper do not require further knowledge on how IR algorithms operate, we want to give some intuition about the composition of the search tree and in particular where the axiom regarding complete isomorphism invariance comes from.

The IR search tree is the recursion tree of a backtracking procedure whose goal it is to analyze the structure of an input graph (two input graphs in case of the isomorphism problem). Initially, forming the root of the tree, the algorithm distinguishes the vertices of the input graph using readily computed invariants. For example the vertices are easily distinguished by their degree, but other information is also used. The algorithm typically used for this is the *color refinement algorithm* (also known as vertex classification or 1-dimensional Weisfeiler-Leman algorithm). In case all vertices are distinguished from another, isomorphism can easily be checked, and no recursion is needed. Otherwise the algorithm starts to pick a class of indistinguishable vertices and for each vertex in this class, one at a time, artificially alters the vertex to make it distinguishable. This process is called *individualization* (hence the name individualization-refinement algorithm). Each individualization causes a recursive call, which corresponds to a child of the current node in the search tree. In the recursion, the refinement algorithm is called again to check if new information propagates through the graph and can be used to distinguish vertices further. As the entire procedure proceeds recursively it allows us to distinguish more and more vertices from each other. Recursion continues until all vertices have been distinguished from one another, i.e., the partition of the vertices into different types is *discrete*. The leaves of the IR search tree therefore correspond to discrete partitions. To the leaves, because all vertices have been distinguished, we can associate an invariant that completely describes the structure. We call this a complete invariant. This invariant corresponds to the color of the leaf in our exploration model.

The defining property of these algorithms is that all computations are made in an isomorphism-invariant fashion. This is precisely what leads to our invariance axiom. Indeed, checking isomorphism between two leaves l_1, l_2 in the backtracking tree then becomes trivial since in each leaf all vertices have been distinguished from one another. In particular there

can be only one possible isomorphism, which would have to map vertices of the same type to each other. If an isomorphism exists this implies that the individualization choices made to get to l_1 can be mapped to the choices made to get to l_2. The isomorphism-invariance of the entire procedure guarantees that for matching leaves there is an automorphism (isomorphism when considering trees from different graphs) mapping one leaf to the other.

While, depending on the input, the backtracking trees can come in many shapes and sizes, we want to record several properties. For starters internal vertices cannot have only one child since there is no reason to individualize vertices in singleton classes. Furthermore the number of children a node in the search tree can have is certainly bounded by the number of vertices of the input graph. However, in practice most nodes have significantly fewer children.

The size of the search tree is the dominating factor of the running time. In fact, for most implementations the non-recursive work can be polynomially bounded. Thus, up to a polynomial factor, the running time agrees with the number of vertices of the search tree that are traversed (see for example [17, Theorem 9] and [15]).

Practical Heuristics. A closer look at the practical tools reveals that for most graphs they do not traverse the entire tree. Two main heuristics are commonly applied to prune the search tree, called *invariant pruning* and *automorphism pruning*. The reader familiar with IR tools may worry that our model omits these two crucial aspects. However, in the following, we explain why both of these pruning techniques are captured by the model.

First of all, by using *invariant pruning*, IR algorithms are sometimes able to cut off parts of the search tree at inner nodes. This is done using invariants which yield different values at different parts of the tree. In principle this process could be emulated in the model by adding colors to the inner nodes. When doing so, it is crucial that the invariance axiom does not apply to the inner nodes, since only for leaves the associated invariants are complete. We could extend all results from this paper to the adapted model. In any case, it turns out that to some extent this kind of mechanism is already captured in our model since it allows inner vertices to have different degrees. Degrees of vertices can serve to distinguish inner nodes exactly in the same fashion as invariants do.

Second of all, by using discovered automorphisms of the graph, *automorphism pruning* is a method to skip branches that we already know are symmetric. Another way of seeing this is that automorphisms allow us to form quotients of the search tree. In our setting, we can simply define the trees to be the quotients of the original search trees. This simulates perfect automorphism pruning. It does not explain how to find one or all automorphisms in one of the trees, but these kinds of problems are closely linked to finding isomorphisms and can also be expressed in our search tree model (see below).

In summary, both types of heuristics are captured by our model.

Traversal Strategies in Practical Tools. There are only two main traversal strategies used by competitive practical tools. In fact, with the exception discussed below, all competitive tools essentially traverse the search tree using depth-first search [5, 6, 8, 12].

However, the practical solver Traces introduced a radically different strategy, which turns out to be much more effective in most practical cases: breadth-first traversal is combined with random walks of the search trees [12, 15]. The idea is that breadth-first traversal maximizes applicability of pruning rules, while random walks are used to discover automorphisms for pruning. While crucially exploiting randomization, the tool does not make errors.

Related Problems. Practical algorithms mostly do not decide the graph isomorphism problem, but rather solve one of two strongly related types of problems. In the following, we discuss these related problems and how they can be modeled using black box search trees.

The *automorphism group problem* requires computation of the entire automorphism group of a graph. This problem is closely related to the graph isomorphism problem and there are polynomial-time Turing reductions from each problem to the other [9]. Regarding IR search trees, in our model the problem corresponds to finding all leaves of an arbitrary color.

Alternatively, we can solve the *asymmetry problem* or equivalently the problem of searching for a *non-trivial graph automorphism*. In our model this translates to finding two distinct leaves of the same color if they exist. In IR algorithms, it suffices to solve the asymmetry problem. When non-trivial automorphisms are discovered, a wrapper algorithm may then remove the symmetry from the search tree and repeat the task on a quotient of the search tree. Such a wrapper algorithm has to repeat the problem at most a number of times that is quasi-linear in the order of the graph (repetitions are directly tied to the length of the longest subgroup series). Overall, our results on traversal strategies immediately carry over to the various tasks regarding automorphism group computations.

Another important problem in practice is *canonization*. The goal here is essentially to find a normal form for graphs by finding a canonical ordering of the vertices. This way isomorphism testing reduces to equality testing of the normal forms. Hence, graph isomorphism reduces to canonization in polynomial-time, but we currently do not know whether graph isomorphism and canonization are polynomial-time equivalent.

In IR algorithms, this problem can be modeled as follows. The solvers are now executed on a single graph, yielding a single search tree. The goal is to return a particular leaf in the given input trees. The requirement is that the output has to be consistent across different but isomorphic inputs. It is not clear to us whether any of the techniques for sublinear exploration developed in this paper can be transferred to the canonization problem.

6 Conclusion and Future Work

We designed an abstract model that captures the backtracking behavior of IR algorithms and proved bounds for various scenarios. We want to stress the fact that the class of trees $\mathcal{M}_h$ used throughout the paper for lower bound constructions models actual recursion trees of the IR algorithms. In fact the trees closely resemble those arising form so-called shrunken multipede graphs of [14], which form worst case inputs for all IR algorithms. These recursion trees are in particular of degree at most 4 and have exponential size. In other words, our worst case lower bounds apply to instances stemming from true inputs to IR algorithms.

Using our new insights we can explain why some of the strategies used by the currently fastest practical solver Traces turn out to be highly efficient. As discussed previously, Traces uses breadth-first search intertwined with random walks of the search tree. In particular, this is often done in a cost balancing manner, such that the number of random walks is proportional to the cost of breadth-first search. This in turn often leads to the automorphism group being found in time proportional to the square root of the search tree size. For sophisticated pieces of software such as Traces, the traversal strategy is of course not the only deciding factor when it comes to running time. However, generally, the experimental paths often enable Traces to discover automorphisms much earlier than solvers solely utilizing depth-first traversal. Hence automorphisms are available more quickly for pruning. Overall, in some sense, Traces emulates some of the techniques described in our Monte Carlo algorithm.

Interestingly, Traces also sometimes uses some techniques of the Las Vegas algorithm we describe. Specifically it performs splits in its "special traversal" strategy for automorphism groups (see [12]). When Traces detects a leaf on level h with parent v, in our terminology it executes the split $(v, h-1)$. Since many graphs in the benchmark suite of [12] have search trees of height 2 or 3, Traces in practice turns out to frequently perform splits that are fairly balanced. This results in significant speedups over other solvers (e.g., see runtime on combinatorial graphs with switched edges in [12]).

In subsequent work, we were able to show that an implementation of the Monte Carlo approach indeed outperforms state-of-the-art solutions for isomorphism testing in practice. Interestingly, besides superior worst-case guarantees, the approach has further practical advantages that simplify its implementation over state-of-the-art tools [1].

Regarding future work, a theoretical question that remains is whether sublinear traversal strategies for the graph canonization problem are possible. Furthermore, the challenge remains to close the gap of logarithmic factors between our upper and lower bounds. Also one might want to address the fact that the size of the larger tree rather than the size of the smaller tree appears in the upper bound of the Las Vegas algorithm.

References

1 Markus Anders and Pascal Schweitzer. Engineering a fast probabilistic isomorphism test. In *ALENEX'21: Proceedings of the Symposium on Algorithm Engineering and Experiments*, 2021. to appear.

2 László Babai. Graph isomorphism in quasipolynomial time [extended abstract]. In *Proceedings of the 48th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2016, Cambridge, MA, USA, June 18-21, 2016*, pages 684–697. ACM, 2016. doi:10.1145/2897518.2897542.

3 Marek Chrobak and Claire Kenyon-Mathieu. SIGACT news online algorithms column 10: competitiveness via doubling. *SIGACT News*, 37(4):115–126, 2006. doi:10.1145/1189056.1189078.

4 Paul T. Darga, Hadi Katebi, Mark Liffiton, Igor L. Markov, and Karem Sakallah. Saucy3. http://vlsicad.eecs.umich.edu/BK/SAUCY/.

5 Paul T. Darga, Mark H. Liffiton, Karem A. Sakallah, and Igor L. Markov. Exploiting structure in symmetry detection for CNF. In *Proceedings of the 41st Annual Design Automation Conference*, DAC '04, pages 530–534, New York, NY, USA, 2004. ACM. doi:10.1145/996566.996712.

6 Tommi Junttila and Petteri Kaski. Engineering an efficient canonical labeling tool for large and sparse graphs. In *ALENEX'07: Proceedings of the Ninth Workshop on Algorithm Engineering and Experiments*, pages 135–149, New Orleans, USA, 2007. SIAM.

7 José Luis López-Presa, Antonio Fernández Anta, and Luis N. Chiroque. Conauto2. https://sites.google.com/site/giconauto/.

8 José Luis López-Presa, Luis F. Chiroque, and Antonio Fernández Anta. Novel techniques to speed up the computation of the automorphism group of a graph. *J. Applied Mathematics*, 2014:934637:1–934637:15, 2014. doi:10.1155/2014/934637.

9 Rudolf Mathon. A note on the graph isomorphism counting problem. *Inf. Process. Lett.*, 8(3):131–132, 1979. doi:10.1016/0020-0190(79)90004-8.

10 Brendan D. McKay. Practical graph isomorphism. In *10th. Manitoba Conference on Numerical Mathematics and Computing (Winnipeg, 1980)*, pages 45–87, 1981.

11 Brendan D. McKay and Adolfo Piperno. Nauty and traces user guide. https://cs.anu.edu.au/people/Brendan.McKay/nauty/nug25.pdf.

12 Brendan D. McKay and Adolfo Piperno. Practical graph isomorphism, II. *Journal of Symbolic Computation*, 60(0):94–112, 2014. doi:10.1016/j.jsc.2013.09.003.

13 A. G. Munford. A note on the uniformity assumption in the birthday problem. *Amer. Statist.*, 31(3):119, 1977. doi:10.2307/2682958.

14 Daniel Neuen and Pascal Schweitzer. Benchmark graphs for practical graph isomorphism. In *25th Annual European Symposium on Algorithms, ESA 2017, September 4-6, 2017, Vienna, Austria*, volume 87 of *LIPIcs*, pages 60:1–60:14. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2017. doi:10.4230/LIPIcs.ESA.2017.60.

15 Adolfo Piperno. Search space contraction in canonical labeling of graphs (preliminary version). *CoRR*, abs/0804.4881, 2008. arXiv. arXiv:0804.4881.

16 David J. Rosenbaum. Breaking the $n^{\log n}$ barrier for solvable-group isomorphism. In *Proceedings of the Twenty-Fourth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2013, New Orleans, Louisiana, USA, January 6-8, 2013*, pages 1054–1073. SIAM, 2013. doi:10.1137/1.9781611973105.76.

17 Pascal Schweitzer. *Problems of unknown complexity: graph isomorphism and Ramsey theoretic numbers*. Phd. thesis, Universität des Saarlandes, Saarbrücken, Germany, 2009.

18 Stoicho D. Stoichev. New exact and heuristic algorithms for graph automorphism group and graph isomorphism. *ACM Journal of Experimental Algorithmics*, 24(1):1.15:1–1.15:27, 2019. doi:10.1145/3333250.

Breaking the Barrier Of 2 for the Competitiveness of Longest Queue Drop

Antonios Antoniadis
University of Twente, The Netherlands

Matthias Englert
University of Warwick, Coventry, UK

Nicolaos Matsakis
Athens, Greece

Pavel Veselý
Computer Science Institute of Charles University, Prague, Czech Republic

Abstract

We consider the problem of managing the buffer of a shared-memory switch that transmits packets of unit value. A shared-memory switch consists of an input port, a number of output ports, and a buffer with a specific capacity. In each time step, an arbitrary number of packets arrive at the input port, each packet designated for one output port. Each packet is added to the queue of the respective output port. If the total number of packets exceeds the capacity of the buffer, some packets have to be irrevocably rejected. At the end of each time step, each output port transmits a packet in its queue and the goal is to maximize the number of transmitted packets.

The Longest Queue Drop (LQD) online algorithm accepts any arriving packet to the buffer. However, if this results in the buffer exceeding its memory capacity, then LQD drops a packet from the back of whichever queue is currently the longest, breaking ties arbitrarily. The LQD algorithm was first introduced in 1991, and is known to be 2-competitive since 2001. Although LQD remains the best known online algorithm for the problem and is of practical interest, determining its true competitiveness is a long-standing open problem. We show that LQD is 1.707-competitive, establishing the first $(2-\varepsilon)$ upper bound for the competitive ratio of LQD, for a constant $\varepsilon > 0$.

2012 ACM Subject Classification Theory of computation → Online algorithms

Keywords and phrases buffer management, online scheduling, online algorithms, longest queue drop

Digital Object Identifier 10.4230/LIPIcs.ICALP.2021.17

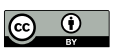
Category Track A: Algorithms, Complexity and Games

Related Version *Full Version*: https://arxiv.org/abs/2012.03906

Funding *Antonios Antoniadis*: Work done in part while the author was at Saarland University and Max-Planck-Institute for Informatics and supported by DFG grant AN 1262/1-1.
Pavel Veselý: Work done while the author was at University of Warwick. Partially supported by European Research Council grant ERC-2014-CoG 647557, by GA ČR project 19-27871X, and by Charles University project UNCE/SCI/004.

1 Introduction

The fact that communication networks are omnipresent highlights the significance of improving their performance. A natural way to achieve such performance improvements is to develop better algorithms for buffer management of shared-memory switches which form the lower levels of network communication. We study a fundamental model of such switches.



Consider a shared-memory network switch consisting of a buffer of size $M \in \mathbb{N}$, an input port, and $N \in \mathbb{N}$ output ports. Furthermore, consider a slotted time model. In each time step, an arbitrary number of unit-valued packets arrive to the input port. Each packet comes with a label specifying the output port that it has to be forwarded to. A buffer management algorithm has to make a decision for each packet: either irrevocably reject it, or accept it while ensuring that the buffer capacity M is respected, which may mean that a previously accepted packet has to be evicted. At the end of the time step, each output port with at least one packet in the buffer destined to it transmits a packet. The goal of the buffer management algorithm is to accept/reject incoming packets or evict already accepted packets, so as to maximize the throughput, i.e., the total number of transmitted packets, while ensuring that at most M packets in total are stored for all output ports at any time.

Given the inherently online nature of buffer management problems, a standard approach is to design online algorithms for them and evaluate the algorithm's performance using its competitive ratio. More specifically, an online algorithm ALG is c-competitive (where $c \geq 1$), if the number of packets transmitted by an optimal offline algorithm OPT (that has full knowledge of the incoming packet sequence a priori) is at most c times the number of packets transmitted by ALG. There exists an extensive body of research dedicated to designing competitive online algorithms with the aim of improving the performance of networking devices that incorporate buffers (see e.g. [20, 32]).

Since packets have unit value, we can assume without loss of generality that the packets destined to a specific output port are transmitted in an earliest-arrival (FIFO) fashion and thus, it is helpful to associate each output port with a queue.

Intuitively speaking, to maximize throughput, one would like to maintain a flow of packet transmissions for as many queues in parallel as possible. It is therefore desirable to prioritize accepting packets for queues that do not have many incoming packets in the near future. Unfortunately, an online algorithm does not know which queues these are, and in order to be insured against an adversarial input it seems reasonable to try to keep the queue lengths as balanced as possible in every step. This is exactly the idea behind the online algorithm *Longest Queue Drop (LQD)*, introduced in 1991 by Wei, Coyle, and Hsiao [35]: The incoming packet is always accepted and if this causes the buffer to exceed its capacity then one packet from the longest queue, breaking ties arbitrarily, is evicted (this could be the incoming packet).[1]

The LQD algorithm, apart from being a natural online algorithm to derive, remains the only known competitive algorithm for this problem. Since the algorithm is simple and can be used, for instance, to achieve a fair distribution of the bandwidth, it is of some practical interest; see e.g. [10, 11, 12, 13, 31, 33].

Previous Results

Hahne, Kesselman, and Mansour [21] provided the first formal analysis of LQD, showing that it is 2-competitive (see also Aiello, Kesselman, and Mansour [1]). The proof follows from a simple procedure that charges the extra profit of OPT to the profit of LQD. Furthermore, they demonstrate that LQD is at least $\sqrt{2}$-competitive, and also showed a general lower bound of 4/3 for the competitive ratio of any deterministic online algorithm.

[1] Wei, Coyle, and Hsiao proposed the LQD algorithm for the problem of shared-memory switches, consisting of N input ports and N output ports. Rather than assuming N input ports, each of which may receive at most one packet per time step, we more generally assume that there is a single input port of infinite capacity. Furthermore, we do not put any restrictions on the number of output ports, i.e., we allow N to be arbitrarily large. Again, this only makes the problem more general.

The analysis of LQD in [1, 21] was then refined by Kobayashi, Miyazaki, and Okabe [28] who showed that the LQD competitive ratio is at most $2 - \min_{k=1,\dots,N}(\lfloor M/k \rfloor + k - 1)/M$. However, for $N > \sqrt{M}$, this bound becomes $2 - O(1/\sqrt{M})$ and therefore does not establish a $2 - \varepsilon$ upper bound for a constant $\varepsilon > 0$ in general. Additionally, for the case of $N = 2$ output ports, LQD is exactly $\frac{4M-4}{3M-2}$-competitive [28] (we note that although this result holds for an even buffer size, the argument unfortunately breaks down when the buffer size is odd). For the case of $N = 3$ output ports, Matsakis shows that LQD is 1.5-competitive [30].

More recently, Bochkov, Davydow, Gaevoy, and Nikolenko [9] improved the lower bound on the competitiveness of LQD from $\sqrt{2}$ to approximately 1.44 (using a direct simulation of LQD and also independently, by solving a linear program). Moreover, they show that any deterministic online algorithm is at least $\sqrt{2}$-competitive, using a construction inspired by the LQD specific lower bound from [1, 21]. To the best of our knowledge, so far, no randomized algorithms for this problem have been studied.

Our Contribution

Although LQD is the best known online algorithm for buffer management in shared-memory switches, determining its true competitiveness remains an elusive problem and has been described as a significant open problem in buffer management [20, 32]. After the initial analysis which showed that LQD is 2-competitive and not better than $\sqrt{2}$-competitive [1, 21] progress on the upper bound has been limited to special cases (e.g., with restrictions on the number of output ports or memory size) [28, 30]. In this paper, we make the first major progress in almost twenty years on upper bounding the competitive ratio of LQD. Namely, we prove the first $(2 - \varepsilon)$ upper bound for a constant $\varepsilon > 0$ without restrictions on the number of ports or the size of the buffer:

▶ **Theorem 1.** *LQD is* 1.707*-competitive.*

We remark that Theorem 1 applies to LQD with any tie-breaking rule, even if tie-breaking is under control of the adversary, and that our upper bound is strictly smaller than $1 + 1/\sqrt{2}$.

Our Techniques

The proof of 2-competitiveness of LQD in [1, 21] uses the following general approach. If an optimal offline algorithm OPT currently stores more packets for a queue than LQD does, these excess packets present potential extra profit for OPT. Each such potential extra packet p in OPT is then matched to a packet that is transmitted by LQD at some point before packet p can be transmitted by OPT.

Our approach is different in that we (for the most part) do not match specific packets to one another. Instead, the idea is to take the total profit of LQD in each step and distribute it evenly among all potential extra packets that exist at the time. As such, the scheme is less discrete than the previous one. We then carefully calculate that, for each queue, *on average* each potential extra packet in that queue receives a profit strictly larger than one.

As described here, this approach does not quite work yet. Two additional types of charging concepts have to be combined with this first idea: One involves not splitting the LQD profit completely evenly and instead slightly favoring queues with relatively few potential extra packets, and the other involves matching some of the potential extra packets of OPT to extra packets that LQD transmits. Another difficulty is that the lengths of two queues, from which packets are rejected or evicted in the same time step, may differ by one packet. This makes our proof more intricate. To deal with this, we introduce a potential function that

will amortize the LQD profit in a suitable way. Then, the main challenge is to obtain useful lower bounds on the profit assigned to each queue, for which we introduce a novel scheme that relates the buffers of LQD and of OPT.

Further Related Work

We refer the reader to the survey by Goldwasser [20] for an overview of online algorithms for buffer management problems. Additionally, the survey of Nikolenko and Kogan [32] incorporates some more recent work. In the following, we discuss some of the results related to online buffer management for switches. In general, buffer management algorithms can be partitioned into *preemptive* ones, i.e., algorithms that allow for the eviction of already accepted packets from the buffer (eviction is also referred to as preemption), and *non-preemptive* ones that never evict a packet after it has been accepted.

Kesselman and Mansour [25] study buffer management in shared-memory switches in the non-preemptive setting in which a packet has to be transmitted once it is stored in the buffer and can no longer be evicted. They introduce the Harmonic online algorithm, which tries to maintain the length of the i^{th} longest queue as roughly proportional to a $1/i$ fraction of the memory. They show that this algorithm is $(\ln(N) + 2)$-competitive and give a general lower bound of $\Omega(\log N / \log\log N)$ for the performance of any deterministic non-preemptive online algorithm. Considering the non-constant lower bound that they establish, it follows that preemption provides a significant advantage.

Eugster, Kogan, Nikolenko, and Sirotkin [19] generalize the same problem in the following two ways: First, they study unit-valued packets labeled with an output port and a processing requirement (in our case, we have a unit processing cycle per packet). Packets accepted to the same queue have the same processing requirement. They introduce the preemptive Longest-Work-Drop algorithm: If the buffer is not full, the incoming packet is accepted; otherwise, a packet is preempted from a queue that has the largest total processing requirement. They show that this algorithm is 2-competitive and at least $\sqrt{2}$-competitive and that the competitive ratio of LQD for this more general problem is at least $(\sqrt{k} - o(\sqrt{k}))$, where k is the maximum processing time of any packet. Second, they address the problem of different packet values when all packets have unit processing requirements. They show that LQD is at least $(\sqrt[3]{k} - o(\sqrt[3]{k}))$-competitive in this case, where k is the maximum packet value. They also introduce a new algorithm which they conjecture to have a constant competitive ratio.

Azar and Richter [6] study switches with multiple input queues. More specifically, they consider one output port and N input ports and assume that each input port has an independent buffer of size M. At each time step, one packet can be sent from a single input port to the output port. For $M = 1$, they prove a lower bound of $1.46 - \Theta(1/N)$ for the competitive ratio of any randomized online algorithm and a lower bound of $2 - 1/N$ for deterministic online algorithms. They also give a randomized $\frac{e}{e-1} \approx 1.582$-competitive algorithm for $M > \log N$. For $M > 1$, Albers and Schmidt [3] design a deterministic 1.889-competitive algorithm for this problem and show a deterministic lower bound of $\frac{e}{e-1} \approx 1.582$ when $N \gg M$. Azar and Litichevskey [5] give a deterministic online algorithm matching this bound for large M.

A lot of research has been dedicated to the natural single input and single output port model. The model is trivial for unit packet values, but challenging if packets can have different values and the goal is to maximize the total value of transmitted packets. There exists a single queue for the accepted packets and one of the most studied versions of this problem requires packet transmission in the FIFO order. Kesselman, Lotker, Mansour, Patt-Shamir, Schieber, and Sviridenko [24] show that a simple greedy algorithm is exactly

$(2-1/(M+1))$-competitive when preemption is allowed. A series of works gradually improved the analysis of a better online algorithm from 1.983 [26], over 7/4 [7], to $\sqrt{3}$ [17]. Kesselman, Mansour, and van Stee [26] also show a general lower bound of 1.419 for the competitive ratio of any preemptive deterministic online algorithm.

The authors of [24] introduce the bounded-delay model of single output port switches. In this model, the buffer has unlimited size and allows for packets to be transmitted in any order, however, each packet has a deadline after which it needs to be dropped from the buffer. Once again, the problem is only interesting if packets can have different values. Any deterministic online algorithm is at least $\phi \approx 1.618$-competitive [4, 15, 22, 36], and after a sequence of gradual improvements [16, 18, 29], Veselý, Chrobak, Jeż, and Sgall [34] recently gave a ϕ-competitive algorithm. The competitive ratio of randomized algorithms is still open, with the best upper bound of $\frac{e}{e-1} \approx 1.582$ [8, 14, 23] (that holds even against the adaptive adversary), while the lower bounds are 1.25 against the oblivious adversary [8] and 4/3 against the adaptive adversary [15].

Lastly, we mention the model of Combined Input and Output Queued (CIOQ) Switches, in which the switch has N input ports and N output ports. Each input and output port has its own buffer and each input port can transfer a packet to any output port; however, at most one packet can be sent from any input port and at most one packet can be accepted by any output port, during one transfer cycle of the switch. A parameter S called *speedup* equals the number of transfer cycles of the switch taking place per one time step. For the unit-value case, Kesselman and Rosén [27] provide a 2-competitive non-preemptive online algorithm for $S = 1$, which becomes 3-competitive for any S. A faster algorithm with the same competitive ratio is given by Al-Bawani, Englert, and Westermann [2].

2 Setup of the Analysis

We fix an arbitrary instance I. Let OPT and LQD be the optimal offline algorithm and the Longest Queue Drop algorithm, respectively. In a slight abuse of notation, we also denote the profit that the optimal offline algorithm gains on input instance I as OPT and the profit that the Longest Queue Drop algorithm gains as LQD. Our goal is to give an upper bound on $\mathsf{OPT}/\mathsf{LQD}$.

For a time step t and a queue q, we say that OPT transmits an OPT-extra packet from q if OPT transmits a packet from q in step t but LQD does not. Equivalently, queue q is non-empty in OPT's buffer but empty in LQD's buffer at t. Similarly, we say that LQD transmits an LQD-extra packet from a queue q in step t if LQD transmits a packet from q at t but OPT does not.

Let $\mathsf{OPT}_{\mathsf{EXTRA}}$ and $\mathsf{LQD}_{\mathsf{EXTRA}}$ be the total number of transmitted OPT-extra and LQD-extra packets, respectively, over all time steps and queues. Then $\mathsf{OPT} - \mathsf{OPT}_{\mathsf{EXTRA}} = \mathsf{LQD} - \mathsf{LQD}_{\mathsf{EXTRA}}$ and hence $\frac{\mathsf{OPT}}{\mathsf{LQD}} = 1 + \frac{\mathsf{OPT}_{\mathsf{EXTRA}} - \mathsf{LQD}_{\mathsf{EXTRA}}}{\mathsf{LQD}}$. Therefore, if we show that $\varrho \cdot (\mathsf{OPT}_{\mathsf{EXTRA}} - \mathsf{LQD}_{\mathsf{EXTRA}}) \leq \mathsf{LQD}$ for some $\varrho > 1$, it will imply a competitive ratio of $1 + 1/\varrho < 2$ for the Longest Queue Drop algorithm.

Let e_q denote the total number of transmitted OPT-extra packets from queue q over all time steps. Then we have $\mathsf{OPT}_{\mathsf{EXTRA}} = \sum_q e_q$ and we will show

$$\varrho \cdot \left(\sum_q e_q - \mathsf{LQD}_{\mathsf{EXTRA}} \right) \leq \mathsf{LQD} \ . \tag{1}$$

We now give a high-level overview of the proof of Equation (1), which consists of two parts: (i) splitting the LQD profit among queues q with $e_q > 0$, and (ii) mapping transmitted LQD-extra packets to queues q with $e_q > 0$.

For (ii), we use the term $\mathsf{LQD}_{\mathsf{EXTRA}}$ in (1) to "cancel out" some transmitted OPT-extra packets. To this end, we will define how each transmitted LQD-extra packet p is mapped to a queue q (which is different from the one p is transmitted from). Let m_q be the number of transmitted LQD-extra packets which are mapped to q. The mapping will be such that $\sum_q m_q \leq \mathsf{LQD}_{\mathsf{EXTRA}}$ and that $m_q \leq e_q$. Define $\hat{e}_q = e_q - m_q \geq 0$ as the number of OPT-extra packets transmitted from queue q which are not canceled out.

We have $\left(\sum_q e_q - \mathsf{LQD}_{\mathsf{EXTRA}}\right) \leq \sum_q (e_q - m_q) = \sum_q \hat{e}_q$. Hence, it is sufficient for each q to receive a profit of at least $\varrho \cdot \hat{e}_q$, from which it follows that $\varrho \cdot \sum_q \hat{e}_q \leq \mathsf{LQD}$, implying (1).

We describe splitting the LQD profit, enhanced with a suitable potential, in Section 3 and introduce useful quantities for bounding the profit assigned to a particular queue in Section 4. Then, in Section 5, we introduce the mapping of transmitted LQD-extra packets to queues and derive a relation between the buffers of LQD and of OPT. Finally, we put the bounds together and optimize the value of ϱ in Section 6, which will yield our upper bound on the LQD competitive ratio.

3 Splitting the LQD Profit

In this section, we explain how the LQD profit is split. Before we proceed, we introduce some notation and terminology and define a key time step for a queue. When we refer to the state of a queue at time step t under some algorithm, we refer to the state after all new packets of step t have arrived and after all possible rejections/preemptions of packets by the algorithm, but before any packet is transmitted by the algorithm at the end of step t. (Note that by preemption we mean an eviction of an already accepted packet.) We use the following notation:

$s^t_{\mathsf{OPT}}(q)$: the number of packets in queue q in the OPT buffer in step t,
$s^t_{\mathsf{LQD}}(q)$: the number of packets in queue q in the LQD buffer in step t,
$s^t_{\max} = \max_q s^t_{\mathsf{LQD}}(q)$: the maximal size of a queue in the LQD buffer in step t.

We say that a queue q is *active* in a time step t if $s^t_{\mathsf{OPT}}(q) \geq 1$ or $s^t_{\mathsf{LQD}}(q) \geq 1$. Otherwise, if q is empty in both buffers at t, we say that q is *inactive* at t. See Figure 1 for an illustration.

Assumptions on the Instance

First, we assume without loss of generality (w.l.o.g.) that the longest queue in LQD's buffer has at least two packets in every step before the last step when packets are transmitted by LQD.

(A1) We assume that once $s^t_{\max} \leq 1$, no packet arrives to any queue in any step $t' > t$. Consequently, $s^t_{\max} \geq 2$ for any step t starting from the step in which the first packets are stored in the buffer and before the last step when the LQD buffer is not empty.

To see that this assumption is w.l.o.g., note that if $s^t_{\max} \leq 1$ then after packets are transmitted in step t, the LQD buffer is empty. Hence, LQD's processing of possible packets arriving after t is essentially independent of its behavior up to step t, and the adversary may postpone the arrival of future packets by an arbitrary number of steps, which may only help to increase the throughput of OPT but does not change the throughput of LQD.

We also make the following assumption w.l.o.g., which will greatly reduce the additional notation required.

(A2) For any queue q and step t, we assume that if $s^t_{\mathsf{LQD}}(q) \leq 1$ but at least one packet arrived to q at or before time step t, then no packet arrives to queue q after step t. (If $s^t_{\mathsf{LQD}}(q) = 1$ then the last packet is transmitted from q in step t.)

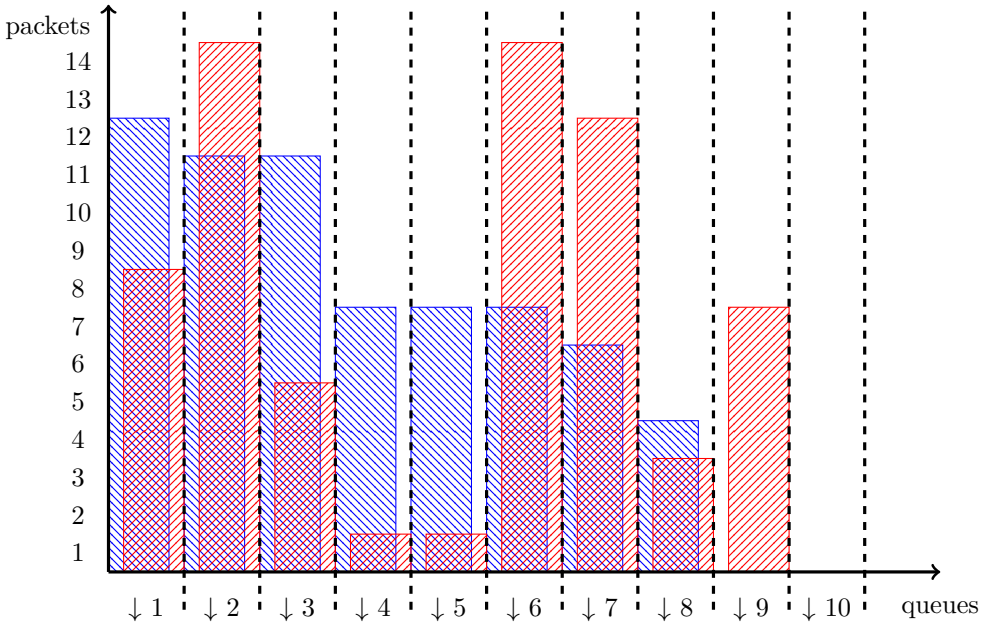


Figure 1 An example of the buffer configuration for LQD and OPT at some time step t. The blue, north-west shaded areas (aligned to the left) correspond to the packets in queues of LQD and the red, north-east shaded areas (aligned to the right) to the queues of OPT. For instance, we have $s^t_{\mathsf{OPT}}(6) = 14$, and $s^t_{\mathsf{LQD}}(6) = 7$. Furthermore, $s^t_{\max} = 12$ is the maximal size of a queue for LQD. Note that an OPT-extra packet is going to be transmitted from queue 9 in step t, and as queue 9 is empty for LQD, no further packet arrives to this queue by assumption (A2). All the queues with an index ≥ 10 are inactive (i.e., empty in both buffers). According to Definition 2, queues $1, 2$, and 3 overflow. As an example, assume that further 3 packets arrive into queue 8. Then LQD would first preempt a packet from queue 1 and then select two of the queues 1, 2 or 3, dropping one packet from each selected queue.

To see that this assumption is w.l.o.g., we iteratively modify the instance under consideration as follows: Let q be any queue q that does not satisfy this assumption and let t be the first time step such that $s^t_{\mathsf{LQD}}(q) \leq 1$ and there is a packet arriving to q at t or before. As q does not satisfy the assumption, there is a packet arriving to q after step t; let p be the first such packet. In the modified instance, p and all later packets for queue q are instead sent to a new queue which is not used in the instance otherwise. Observe that the profit of LQD does not change after redirecting these packets to a new queue, while the profit of OPT cannot decrease when we make this change. We remark that the new queue is always available as the number of output ports N is not restricted and can be arbitrarily large. Note that we only make this assumption to simplify our notation and it does not affect the generality of our analysis. Indeed, if the number of output ports used in the original instance is bounded by N_0, then after applying this transformation, there are always at most N_0 queues non-empty for LQD at any one time.

For instance, under assumption (A2), if an OPT-extra packet is transmitted from a queue q in some step t (as q is empty for LQD), then no packet arrives to q in any step $t' > t$.

Overflowing Queues

Intuitively, if a packet destined to q is rejected or preempted by LQD at t, then we say that q *overflows*. Furthermore, in such a case, the LQD buffer is full in step t and q has $s^t_{\max} - 1$ or $s^t_{\max}$ packets at t (see the example in Figure 1). This possible difference of 1 in the lengths

of two different overflowing queues makes our analysis substantially more involved.[2] For technical reasons, we also call a queue q' containing at least $s^t_{\max} - 1$ packets at t overflowing, provided that the LQD buffer is full and $s^t_{\mathsf{LQD}}(q') \geq 1$, even though there may be no packet for q' that is rejected or preempted at time t (it may even happen that no packet destined to any queue gets rejected or preempted at t but there are still some overflowing queues).

▶ **Definition 2.** *We say that a queue q* overflows *in step t if the LQD buffer is full in step t, $s^t_{LQD}(q) \geq s^t_{\max} - 1$, and $s^t_{LQD}(q) \geq 1$.*

Assumption (A2) also implies that once $s^t_{\mathsf{LQD}}(q) \leq 1$, then queue q does not overflow after t (as after step t, it is empty in the LQD buffer).

Key Time Step

Based on assumption (A2), we give a definition of a key time step t_q for queue q. For each queue q, we define:

t_q: the last time step in which queue q overflows; if q does not overflow in any step, we define $t_q = -1$ (we index time steps starting from 0).

Some important properties follow directly from the definition of t_q: No packet is ever preempted by LQD from q after t_q and no packet arriving to q after t_q is rejected by LQD, since a preemption or rejection in some step t implies that the queue overflows at t. We remark that we define $t_q = -1$ for queues q that do not overflow in any step in order to have the property that for such queues, $t_q < t$ for all time steps t.

We would like to keep track of how many OPT-extra packets are yet to be transmitted from a queue, for which the following notation is useful.

e^t_q: the number of OPT-extra packets transmitted from q in step t or later.

$\hat{e}^t_q = \max\{e^t_q - m_q, 0\}$: that is, e^t_q adjusted for the packets that are canceled out by transmitted LQD-extra packets. Note that m_q will be specified in Section 5.

Note that $e_q = e^0_q = e^{t_q}_q$ as no OPT-extra packet is transmitted before time t_q by assumption (A2). Thus, e^t_q is constant up to time t_q. After that, it further remains constant until q becomes empty for LQD, and then it decreases by one in each step until it becomes equal to zero. The same property holds for $\hat{e}^t_q$. The definition of t_q gives us a useful observation:

▶ **Observation 3.** *For any step t and queue q with $t \geq t_q$ (i.e., that does not overflow after t), it holds that* $\max\left\{s^t_{OPT}(q) - s^t_{LQD}(q), 0\right\} \geq e^t_q$.

Phases

It will be convenient in certain parts of the analysis to consider time phases instead of time steps. More specifically, let $\tau_1 < \tau_2 < ... < \tau_\ell$ be the time steps in which at least one queue overflows for the last time, i.e., for each $1 \leq i \leq \ell$ there is a queue q such that $\tau_i = t_q \geq 0$. Note that it has to be $\ell > 0$ so that OPT gains extra profit (equivalently, $\ell = 0$ only if $\mathsf{OPT}_{\mathsf{EXTRA}} = 0$). We call the time interval $[\tau_i, \tau_{i+1})$ the *i-th phase*; for $i = \ell$, we define $\tau_{\ell+1} = \infty$. We remark that time steps before τ_1 do not belong to any phase, because there

[2] A less sophisticated version of our proof, which deals with this scenario in a less careful way, only gives an upper bound of about 1.906 on the competitive ratio. Nevertheless, this analysis still requires the majority of concepts, lemmas, and calculations developed in the paper.

are no OPT-extra packets transmitted before step τ_1. Finally, observe that for any queue q that overflows at least once (i.e., $t_q \geq 0$), there has to exist an i such that $t_q = \tau_i$ as this queue overflows at t_q for the last time.

In the remainder of the paper, our focus will be mainly on steps $\tau_1, \ldots, \tau_\ell$. For simplicity and to avoid double indexing, we shall write $s^i_{\mathsf{LQD}}(q)$ instead of $s^{\tau_i}_{\mathsf{LQD}}(q)$, and similarly, we use index i instead of τ_i in other notations. Throughout the paper, i will be used solely to index phases and time steps $\tau_1, \ldots, \tau_\ell$.

Intuition for Splitting the LQD Profit

The key ingredient of our analysis is the splitting of the LQD profit such that we assign a profit of at least $\varrho \cdot \hat{e}_q$ to each q. To keep track of how much profit we assigned to a queue q, we use counter Φ_q. In particular, $\Delta^i \Phi_q$ will be the LQD profit assigned to q in phase i and $\Phi_q = \sum_{i=1}^{\ell} \Delta^i \Phi_q$ will be the LQD profit assigned to q over all phases. We will ensure that $\mathsf{LQD} \geq \sum_q \Phi_q$. The crucial part will be to show that $\Phi_q \geq \varrho \cdot \hat{e}_q$, which, together with $\sum_q (e_q - \hat{e}_q) = \sum_q m_q \leq \mathsf{LQD}_{\mathsf{EXTRA}}$, implies (1) using

$$\mathsf{LQD} \geq \sum_q \Phi_q \geq \sum_q \varrho \cdot \hat{e}_q \geq \varrho \cdot \left(\sum_q e_q - \mathsf{LQD}_{\mathsf{EXTRA}} \right) .$$

Let LQD^i be the profit of LQD in phase i, i.e., the total number of packets transmitted by LQD in all time steps in $[\tau_i, \tau_{i+1})$. A first idea is to split LQD^i among queues q satisfying $t_q \leq \tau_i$ proportionally to $\hat{e}^i_q$, meaning that we assign a profit of $\mathsf{LQD}^i \cdot \hat{e}^i_q / \hat{e}^i$ to a queue q with $\tau_i \geq t_q$, where $\hat{e}^i = \sum_{q:\tau_i \geq t_q} \hat{e}^i_q$. Such a scheme is useful because we can relate $\hat{e}^i$ to a certain fraction of the LQD profit; this is elaborated in Section 5.

Unfortunately, this simple idea fails for "short" queues, by which we mean queues for which $\hat{e}_q$ is relatively small compared to the number of packets that LQD transmits from q starting from time t_q. In particular, the total profit assigned to such a short queue q may be very close to $\hat{e}_q$, which would only be sufficient for $\varrho = 1$, thus proving 2-competitiveness.

To give a higher profit to a short queue q, we choose a parameter $\alpha \in (0, 1)$ and directly assign to q a $(1 - \alpha)$-fraction of the profit LQD gains by transmitting packets from q itself starting at time step t_q, whereas the remaining α-fraction of these packets is split proportionally to $\hat{e}^i_q$. The parameter $\alpha \approx 0.58$ is chosen at the very end of the analysis, so as to minimize the competitive ratio upper bound.

Potential

Before describing how exactly we split the LQD profit, we introduce a potential that will help us to deal with the fact that some queues overflowing in step t may only have $s^t_{\max} - 1$ packets and not $s^t_{\max}$ packets. On an intuitive level, this potential amortizes the profit assignment by moving some profit from phases with a slack to phases in which our lower bounds on the profit assigned are tight; we develop these bounds in the subsequent sections.

Namely, at any phase i, let $\mathcal{A}^i$ be the set of queues q that are active in step τ_i and satisfy $t_q > \tau_i$ (i.e., will overflow after the beginning of phase i). Thus, for any such queue q we have $t_q = \tau_j$ for some $j > i$ and consequently, q is non-empty for LQD in every step during phase i by assumption (A2) (as otherwise, q would not overflow at τ_j).

Then, using the aforementioned parameter α, we define potential $\Psi^i := \alpha \cdot |\mathcal{A}^i|$. Note that the potential at the beginning is $\Psi^1 \leq \alpha \cdot M$ and after the last packet of the input instance is transmitted, the potential equals $\Psi^{\ell+1} = 0$. We define two quantities which express the change of this potential in phase i:

u^i = the number of queues active in step τ_{i+1} that were inactive in step τ_i and will overflow after τ_{i+1}, i.e., the number of "new" active queues that will overflow after τ_{i+1}; and
v^i = the number of queues that are active in step τ_i and overflow at τ_{i+1} for the last time, i.e., $\tau_{i+1} = t_q$ for any such queue q.

Let $\Delta^i\Psi := \Psi^{i+1} - \Psi^i$ be the change of the potential in phase i; observe that $\Delta^i\Psi = \alpha \cdot (u^i - v^i)$.

Splitting the LQD Profit

We now formally define our scheme of splitting the LQD profit. Consider phase i. Let o^i be the number of packets that LQD transmits in the i-th phase from queues q with $\tau_i \geq t_q$ and $e_q > 0$, and let n^i be the number of packets transmitted by LQD in phase i from all other queues. Note that $\mathsf{LQD}^i = o^i + n^i$.

Apart from parameter $\alpha \in (0,1)$, we use another parameter $\beta \in (0,1)$ such that $\alpha + \beta < 1$; namely we will set $\beta = \alpha^2/(8 \cdot (1-\alpha))$ (we will require that α is not too close to 1 so that $\alpha + \beta < 1$). Given the two parameters, in each phase i, we assign an LQD profit of

$$\Delta^i\Phi_q := \underbrace{\frac{\hat{e}^i_q}{\hat{e}^i} \cdot (n^i + \alpha \cdot o^i - \Delta^i\Psi) + \beta \cdot o^i_q}_{\text{L-increase}} + \underbrace{(1-\alpha-\beta) \cdot o^i_q}_{\text{S-increase}} \tag{2}$$

to each queue q with $\tau_i \geq t_q$ and $\hat{e}^i_q > 0$, where o^i_q is the number of packets that LQD transmits from q during the i-th phase.

We call the first two terms in Equation (2) (i.e., $(\hat{e}^i_q/\hat{e}^i) \cdot (n^i + \alpha \cdot o^i - \Delta^i\Psi) + \beta \cdot o^i_q$) the *L-increase* for q as they will be mainly useful for "long" queues (with relatively high $\hat{e}_q$). We call the last term, $(1-\alpha-\beta) \cdot o^i_q$, the *S-increase* for q as it works well for "short" queues. Note that we only assign profit to queues that already have overflown for the last time. Furthermore, once a queue q with $\tau_i \geq t_q$ is empty in both the LQD and OPT buffers at the start of a phase, it does not get any profit as $\hat{e}^i_q \leq e^i_q = 0$ and $o^i_q = 0$.

To ensure feasibility of our scheme, we show that in total over all queues q with $\tau_i \geq t_q$ and $\hat{e}^i_q > 0$ we assign a profit of at most $\mathsf{LQD}^i - \Delta^i\Psi$. Indeed, using $\hat{e}^i = \sum_{q:\tau_i \geq t_q} \hat{e}^i_q$ and $\sum_{q:\tau_i \geq t_q \text{ and } \hat{e}^i_q > 0} o^i_q \leq o^i$, we have

$$\begin{aligned}
\sum_{q:\tau_i \geq t_q \text{ and } \hat{e}^i_q > 0} \Delta^i\Phi_q &= \sum_{q:\tau_i \geq t_q \text{ and } \hat{e}^i_q > 0} \left(\frac{\hat{e}^i_q}{\hat{e}^i} \cdot (n^i + \alpha \cdot o^i - \Delta^i\Psi) + \beta \cdot o^i_q + (1-\alpha-\beta) \cdot o^i_q \right) \\
&\leq n^i + \alpha \cdot o^i - \Delta^i\Psi + \beta \cdot o^i + (1-\alpha-\beta) \cdot o^i \\
&= n^i + o^i - \Delta^i\Psi = \mathsf{LQD}^i - \Delta^i\Psi\,.
\end{aligned}$$

While the scheme to split the LQD profit is relatively simple to define, showing $\Phi_q \geq \varrho \cdot \hat{e}_q$ brings technical challenges, namely, in obtaining suitable lower bounds on the profits assigned proportionally to $\hat{e}^i_q$ and in summing up these lower bounds over all phases. We get our lower bound based on a novel scheme that relates the buffers of LQD and of OPT, which is introduced in the next two sections.

4 Live and Let Die

Analyzing the S-increases is relatively easy. Most of the remainder of the proof is focused on analyzing the L-increases. We start by deriving a helpful lower bound on $n^i + \alpha \cdot o^i - \Delta^i\Psi$. For this, we first introduce the notion of live and dying queues, which are defined with respect to a fixed queue q with $t_q \leq \tau_i$ and $\hat{e}_q > 0$. For this fixed queue, we need to define live and

dying queues up until the first phase that comes after OPT transmits the last packet from q not canceled out by an LQD-extra packet. Let $j_q := \min\{j : \hat{e}_q^j = 0\}$ be the index j of the earliest step τ_j in which all remaining OPT-extra packets to be transmitted from q are canceled out.

▶ **Definition 4.** *Fix a queue q with $\hat{e}_q > 0$, and consider a phase i with $t_q \leq \tau_i$ and $i \leq j_q$. Let q' be a queue for which LQD stores at least one packet at time step τ_i. Queue q' is called* live *with respect to (w.r.t.) queue q at time step τ_i if*

(i) *$\tau_i < t_{q'}$, i.e., q' overflows at some time step after the i-th phase, or*

(ii) *$e_{q'} = 0$ and $s_{LQD}^{j_q}(q') \geq 1$,*

or both. Otherwise, q' is called dying *with respect to queue q at time τ_i.*

Note that step τ_{j_q} referred to in $s_{\mathsf{LQD}}^{j_q}(q')$ is after t_q, since $\hat{e}_q^t = \hat{e}_q > 0$ in any step t before the first OPT-extra packet is transmitted from q. Furthermore, $e_{q'} > 0$ implies that q' becomes empty in the LQD buffer before it becomes empty in the OPT buffer, by Observation 3. Intuitively, and assuming that $\hat{e}_q = e_q$, at time step τ_i, a queue q' is dying with respect to q if (i) it no longer overflows and (ii) either LQD runs out of packets to send from q' before the time OPT does or LQD runs out of packets to send from q' before the beginning of phase j_q.

The definition of live and dying queues implies the following property about transitions between these two types. This follows since the only property in Definition 4 (for a fixed q) that may change with increasing i is whether or not $\tau_i < t_{q'}$.

▶ **Observation 5.** *If a queue q' is dying (w.r.t. queue q) in time step τ_i, it will never be live (w.r.t. queue q) in step τ_j for any $j > i$. If q' is live (w.r.t. queue q) at τ_i, it can become dying (w.r.t. queue q) in time step τ_{i+1} only if it overflows in time step τ_{i+1} for the last time.*

For any phase i, we denote the set of queues that are live in step τ_i w.r.t. q as $\mathcal{L}_q^i$ and the set of queues dying in step τ_i w.r.t. q as $\mathcal{D}_q^i$. For a fixed phase i and queue q, the sets $\mathcal{L}_q^i$ and $\mathcal{D}_q^i$ partition all queues in which LQD stores packets at time τ_i. Let d_q^i be the number of packets transmitted from queues in $\mathcal{D}_q^i$ during phase i. We now relate $n^i + \alpha \cdot o^i$ to $|\mathcal{L}_q^i|$ and d_q^i.

▶ **Observation 6.** *It holds that $n^i \geq |\mathcal{L}_q^i| \cdot (\tau_{i+1} - \tau_i)$ and also $n^i + \alpha \cdot o^i \geq |\mathcal{L}_q^i| \cdot (\tau_{i+1} - \tau_i) + \alpha \cdot d_q^i$.*

Proof. Recall that o^i is the number of packets that LQD transmits in phase i from queues q' satisfying $\tau_i \geq t_{q'}$ and $e_{q'} > 0$, and that $n^i = \mathsf{LQD}^i - o^i$ (i.e., n^i is the number of packets that LQD sends in the i-th phase from queues q' that will overflow after τ_i or that satisfy $e_{q'} = 0$). As packets sent from queues that are live in step τ_i are accounted for in n^i, it holds that $n^i \geq |\mathcal{L}_q^i| \cdot (\tau_{i+1} - \tau_i)$, which proves the first claim.

Since every queue q' with $\tau_i \geq t_{q'}$ and $e_{q'} > 0$ is dying w.r.t. queue q at time step τ_i, we have that $o^i \leq d_q^i$. It holds that $|\mathcal{L}_q^i| \cdot (\tau_{i+1} - \tau_i) + d_q^i \leq \mathsf{LQD}^i = n^i + o^i$, and this inequality implies the second claim by using $o^i \leq d_q^i$ and $\alpha \leq 1$. ◀

Fix a queue q. We would like to lower bound the number d_q^i of packets transmitted from dying queues during the i-th phase in some way. Note that the LQD buffer is full at times τ_i and τ_{i+1}. Suppose for a moment that the set of live queues (w.r.t. queue q) does not change between step τ_i and step τ_{i+1}, i.e., $\mathcal{L}_q^{i+1} = \mathcal{L}_q^i$. Now, if the number of packets that LQD stores in live queues $\mathcal{L}_q^i$ increases by m between step τ_i and step τ_{i+1}, then we know that $d_q^i \geq m$. This is because the buffer is full, so if the live queues gain m packets, then dying queues must have lost at least m packets (possibly more if there are new dying queues in step τ_{i+1}). Since dying queues do not overflow, the only possible way to reduce the number of packets stored by LQD in dying queues is to transmit them.

We now formalize this intuition and handle cases where the set of live queues changes from one phase to the next. For the fixed queue q and each phase i such that $\tau_i \geq t_q$ and $i \leq j_q$, we define

$$\sigma_q^i = \begin{cases} \left(\sum_{q' \in \mathcal{L}_q^i} s_{\mathsf{LQD}}^i(q')\right) / |\mathcal{L}_q^i| & \text{if } |\mathcal{L}_q^i| \geq 1, \\ 1 & \text{otherwise}. \end{cases} \tag{3}$$

In words, σ_q^i equals the average number of LQD packets in live queues (w.r.t. queue q) in step τ_i, provided that there is at least one such queue. Later, in Lemma 13, we show that $\hat{e}_q^i > 0$ implies $|\mathcal{L}_q^i| \geq 1$ for any phase i (i.e., that there is at least one live queue w.r.t. queue q) and in most cases, we will only need σ_q^i in phases i with $\hat{e}_q^i > 0$. Since live queues are non-empty for LQD, it holds that $\sigma_q^i \geq 1$. Furthermore, as the average is at most the maximum and as the maximum is an integer, this gives us the following observation.

▶ **Observation 7.** *In any phase i such that $\tau_i \geq t_q$ and $i \leq j_q$, it holds that $\lceil \sigma_q^i \rceil \leq s_{\max}^i$.*

The following lower bound on σ_q^i is useful for making β as small as possible.

▶ **Observation 8.** *Assuming $|\mathcal{L}_q^i| \geq 1$, it holds that $\sigma_q^i \geq 2$ for any i with $\tau_i \geq t_q$ and $i < j_q$.*

Proof. We show that any live queue q' (w.r.t. queue q) has at least two packets in the LQD buffer in any step $\tau_i \geq t_q$ with $i < j_q$, which is sufficient as σ_q^i is the average size of live queues, provided that $|\mathcal{L}_q^i| \geq 1$. For a live queue q', consider two cases (as in Definition 4): First, if $\tau_i < t_{q'}$ then indeed $s_{\mathsf{LQD}}^i(q') \geq 2$ by assumption (A2) (if we had $s_{\mathsf{LQD}}^i(q') \leq 1$, then q' would be empty at $t_{q'}$ and would not overflow in that step). Second, if $e_{q'} = 0$ and $s_{\mathsf{LQD}}^{j_q}(q') \geq 1$, where $j_q = \min\{j : \hat{e}_q^j = 0\}$, then we have that $s_{\mathsf{LQD}}^i(q') \geq 2$, using assumption (A2) again together with $i < j_q$. ◀

Packets Transmitted from Dying Queues

We can now formally state our lower bound on the number of packets transmitted from dying queues, taking into account the change of the potential as well. As a byproduct (by rearranging the bound on d_q^i below), we obtain an upper bound on u^i, the number of "new" active queues that will overflow after τ_{i+1}, which captures the increase of the potential. Recall that v^i equals the number of queues that are active in step τ_i and overflow at τ_{i+1} for the last time.

▶ **Lemma 9.** *Consider any queue q with $\hat{e}_q > 0$. For each phase i with $\tau_i \geq t_q$ and $\hat{e}_q^i > 0$, $d_q^i \geq (\sigma_q^{i+1} - \sigma_q^i) \cdot |\mathcal{L}_q^i| + \sigma_q^{i+1} \cdot u^i - v^i$.*

Proof. As q is fixed, we consider live and dying queues w.r.t. queue q only. For simplicity, let $\tau = \tau_i$ and $\tau' = \tau_{i+1}$, thus the i-th phase is $[\tau, \tau')$. By the definition of σ_q^i, live queues contain $\sigma_q^i \cdot |\mathcal{L}_q^i|$ packets in the LQD buffer in step τ and thus, dying queues $\mathcal{D}_q^i$ have $M - \sigma_q^i \cdot |\mathcal{L}_q^i|$ packets in total in step τ, since the LQD buffer is full in step τ. As dying queues do not overflow in any step after τ, it is sufficient to show that queues $\mathcal{D}_q^i$ altogether contain at most $M - \sigma_q^i \cdot |\mathcal{L}_q^i| - (\sigma_q^{i+1} - \sigma_q^i) \cdot |\mathcal{L}_q^i| - \sigma_q^{i+1} \cdot u^i + v^i = M - \sigma_q^{i+1} \cdot |\mathcal{L}_q^i| - \sigma_q^{i+1} \cdot u^i + v^i$ packets in LQD's buffer in step τ'. Let x be the number of LQD packets in queues $\mathcal{D}_q^i$ in step τ', so our goal is to show

$$x \leq M - \sigma_q^{i+1} \cdot |\mathcal{L}_q^i| - \sigma_q^{i+1} \cdot u^i + v^i \,. \tag{4}$$

To this end, we analyze the LQD buffer in step τ'. By Observation 5, any live queue in $\mathcal{L}_q^i$ is also live in step τ' or overflows in step τ' (possibly both). Let $\mathcal{L}' = \mathcal{L}_q^i \cup \mathcal{L}_q^{i+1}$ be

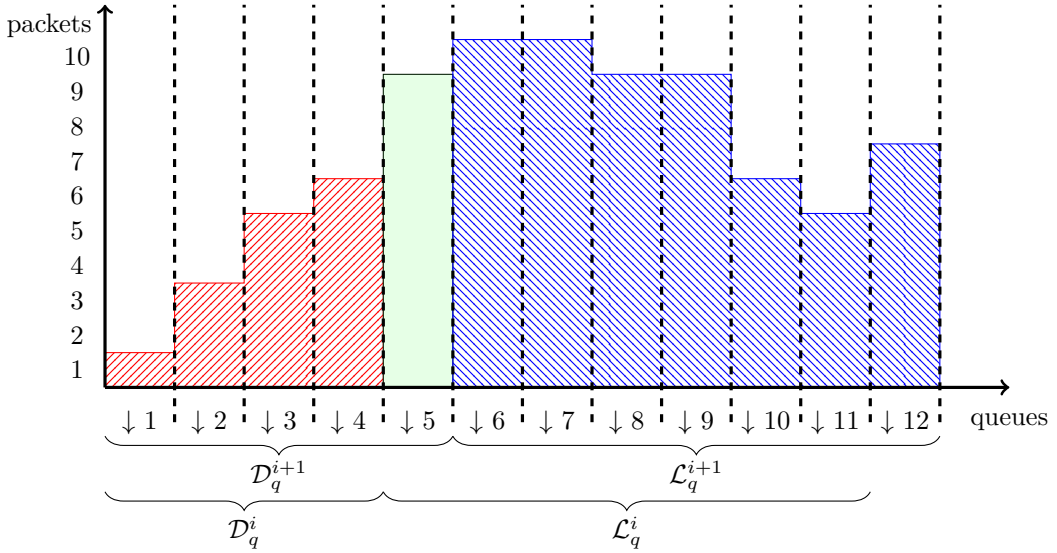


Figure 2 An example of the LQD buffer in step $\tau' = \tau_{i+1}$ for illustrating the proof of Lemma 9. Note that $s_{\max}^{i+1} = 10$ and that queues $5-9$ overflow. However, only queue 5 becomes dying as it overflows for the last time at τ', i.e., $t_5 = \tau'$. Moreover, queue 12 was empty in step τ_i and queue 1 will become empty for LQD just after packets are transmitted in step τ' (note that no further packets will arrive to queue 1 after step τ' by assumption (A2)). We have that $\mathcal{L}' = \{5, 6, \ldots, 12\}$. Finally, $\sigma_q^{i+1} = 56/7 = 8$, since there are 56 packets in 7 live queues $\mathcal{L}_q^{i+1}$.

the set of queues that are live in step τ or in step τ'. Observe that $\mathcal{L}' \cap \mathcal{D}_q^i = \emptyset$, since by Observation 5 dying queues may only become empty in LQD's buffer but not live. It follows that queues in $\mathcal{L}' \setminus \mathcal{L}_q^i$ must be inactive in step τ. Next, no queue that is live in step τ is empty for LQD in step $\tau' = \tau_{i+1}$ by Definition 4, using $i < j_q$, which follows from $\hat{e}_q^i > 0$. Finally, note that $\mathcal{L}'$ may contain some dying queues in $\mathcal{D}_q^{i+1}$, but all of them must overflow at τ', and that $\mathcal{L}' \setminus \mathcal{L}_q^{i+1} \subseteq \mathcal{D}_q^{i+1} \setminus \mathcal{D}_q^i$ (however, equality is not necessarily true). Concluding, set $\mathcal{L}'$ consists of three disjoint types of queues:

- **(i)** live queues in step τ that remain live in step τ',
- **(ii)** live queues in step τ that become dying in step τ' – these are queues in $\mathcal{L}' \setminus \mathcal{L}_q^{i+1}$ and we have that $|\mathcal{L}' \setminus \mathcal{L}_q^{i+1}| = v^i$, which follows from Observation 5 and from the definition of v^i, and
- **(iii)** queues inactive in step τ that are live in step τ' – these are queues in $\mathcal{L}' \setminus \mathcal{L}_q^i$ and there are at least u^i many of them (some live queues may not overflow in any step, so they are not accounted for in u^i).

See Figure 2 for an illustration.

Any queue in $\mathcal{L}' \setminus \mathcal{L}_q^{i+1}$ must overflow at τ', so it has at least $s_{\max}^{i+1} - 1 \geq \sigma_q^{i+1} - 1$ LQD packets at τ', where we use $\sigma_q^{i+1} \leq s_{\max}^{i+1}$ by Observation 7. It follows that queues in $\mathcal{D}_q^{i+1}$ have at least $x + |\mathcal{L}' \setminus \mathcal{L}_q^{i+1}| \cdot (\sigma_q^{i+1} - 1)$ packets in total in step τ'. By the definition of σ_q^{i+1} and since LQD's buffer is full at τ', queues in $\mathcal{D}_q^{i+1}$ contain $M - |\mathcal{L}_q^{i+1}| \cdot \sigma_q^{i+1}$ packets and thus $x + |\mathcal{L}' \setminus \mathcal{L}_q^{i+1}| \cdot (\sigma_q^{i+1} - 1) \leq M - |\mathcal{L}_q^{i+1}| \cdot \sigma_q^{i+1}$. Rearranging and using $|\mathcal{L}' \setminus \mathcal{L}_q^{i+1}| = v^i$, we get $x \leq M - |\mathcal{L}'| \cdot \sigma_q^{i+1} + v^i$. Using $|\mathcal{L}'| = |\mathcal{L}_q^i| + |\mathcal{L}' \setminus \mathcal{L}_q^i| \geq |\mathcal{L}_q^i| + u^i$, we obtain $x \leq M - (|\mathcal{L}_q^i| + u^i) \cdot \sigma_q^{i+1} + v^i$, implying (4). This concludes the proof as explained above. ◀

We now give two more upper bounds on u^i, the number of "new" active queues that will overflow after τ_{i+1}. The advantage of the following bound over the one from Lemma 9 is that it does not use σ_q^{i+1}.

▶ **Lemma 10.** *Consider any queue q with $\hat{e}_q > 0$. For each phase i with $\tau_i \geq t_q$ and $\hat{e}_q^i > 0$, $u^i \leq \frac{1}{2} \cdot \left(|\mathcal{L}_q^i| \cdot (\sigma_q^i - 1) + d_q^i\right)$.*

Proof. As q is fixed, we consider live and dying queues w.r.t. queue q only. Let x be the number of LQD packets in queues $\mathcal{D}_q^i$ in step τ_{i+1}. Similarly as in the proof of Lemma 9, we show that

$$x \leq M - |\mathcal{L}_q^i| - 2 \cdot u^i \,. \tag{5}$$

This equation implies the lemma, since dying queues $\mathcal{D}_q^i$ have $M - \sigma_q^i \cdot |\mathcal{L}_q^i|$ packets in total in step τ_i and thus, $d_q^i = M - \sigma_q^i \cdot |\mathcal{L}_q^i| - x \geq M - \sigma_q^i \cdot |\mathcal{L}_q^i| - (M - |\mathcal{L}_q^i| - 2 \cdot u^i) = 2 \cdot u^i - (\sigma_q^i - 1) \cdot |\mathcal{L}_q^i|$ by (5), from which the lemma follows by rearranging. To justify (5), queues accounted for in u^i are empty for LQD at τ_i and overflow after τ_{i+1}, so LQD must store at least two packets in each of them in step τ_{i+1} by assumption (A2). Moreover, no queue that is live in step τ_i is empty for LQD in step τ_{i+1} by Definition 4, using $i < j_q$, which follows from $\hat{e}_q^i > 0$. Hence, there can be at most $M - |\mathcal{L}_q^i| - 2 \cdot u^i$ LQD packets in queues $\mathcal{D}_q^i$ in step τ_{i+1}. ◀

Finally, we give a third upper bound on u^i that is incomparable to those in Lemmas 9 and 10 and is useful for phases with a relatively small number of steps.

▶ **Lemma 11.** *For any phase i with $\tau_{i+1} - \tau_i \leq \lceil \sigma_q^i \rceil - 2$, it holds that $u^i \leq \frac{1}{2} \cdot (n^i + o^i)$.*

Proof. Recall that the new active queues accounted for in u^i are empty in both buffers in step τ_i and will overflow after τ_{i+1}; let $\mathcal{U}$ be the set of these u^i queues. We show that the number of packets in queues $\mathcal{U}$ in step τ_{i+1} is at most $n^i + o^i$, i.e., the number of packets LQD transmits during the i-th phase. This is sufficient, since any queue in $\mathcal{U}$ has at least two LQD packets at τ_{i+1} (otherwise, if there was a queue in $\mathcal{U}$ with at most one LQD packet in step τ_{i+1}, no packets would arrive after τ_{i+1} to this queue by assumption (A2), so it would not overflow after τ_{i+1}).

Since the LQD buffer is full in step τ_i, it is sufficient to observe that any queue q' has at least $s_{\mathsf{LQD}}^i(q') - (\tau_{i+1} - \tau_i)$ packets in step τ_{i+1} in the LQD buffer; in other words, that packets present in q' at τ_i are not preempted till step τ_{i+1}. Suppose for a contradiction that $s_{\mathsf{LQD}}^{i+1}(q') \leq s_{\mathsf{LQD}}^i(q') - (\tau_{i+1} - \tau_i) - 1$. Thus, there must be a step $t \in (\tau_i, \tau_{i+1}]$ such that a packet is preempted from q' and $s_{\mathsf{LQD}}^t(q') \leq s_{\mathsf{LQD}}^i(q') - (t - \tau_i) - 1$. As $s_{\mathsf{LQD}}^i(q') \leq s_{\max}^i$, it holds that

$$s_{\mathsf{LQD}}^t(q') \leq s_{\max}^i - (t - \tau_i) - 1 \,. \tag{6}$$

Recall that there is a queue $\bar{q}$ with $\tau_i = t_{\bar{q}}$, i.e., which overflows for the last time at τ_i. At τ_i, queue $\bar{q}$ has at least $s_{\max}^i - 1$ packets and thus,

$$s_{\mathsf{LQD}}^t(\bar{q}) \geq s_{\max}^i - 1 - (t - \tau_i) \,. \tag{7}$$

Combining this with (6), we obtain $s_{\mathsf{LQD}}^t(\bar{q}) \geq s_{\mathsf{LQD}}^t(q')$. It holds that $t - \tau_i \leq \tau_{i+1} - \tau_i \leq \lceil \sigma_q^i \rceil - 2 \leq s_{\max}^i - 2$, where the second inequality is by the assumption of the lemma and the third inequality by Observation 7. Plugging this into (7), we obtain $s_{\mathsf{LQD}}^t(\bar{q}) \geq 1$. As a packet is preempted from q' at t, queue q' overflows at t, i.e., it has at least $s_{\max}^t - 1$ LQD packets. Thus, $s_{\mathsf{LQD}}^t(\bar{q}) \geq s_{\max}^t - 1$ and by Definition 2, $\bar{q}$ overflows at t (here, we also use that the

LQD buffer is full at t as q' overflows and that $s^t_{\mathsf{LQD}}(\bar{q}) \geq 1$). However, this contradicts $t_{\bar{q}} = \tau_i < t$. Hence, any queue q' has at least $s^i_{\mathsf{LQD}}(q') - (\tau_{i+1} - \tau_i)$ LQD packets in step τ_{i+1} and the number of LQD packets in queues $\mathcal{U}$ in step τ_{i+1} is at most $n^i + o^i$, which concludes the proof. ◀

5 Mapping Transmitted LQD-extra Packets

So far we derived a lower bound on $n^i + \alpha \cdot o^i$ for a phase i which depends, among other things, on the number of queues $|\mathcal{L}^i_q|$ which are live w.r.t. a queue q at time τ_i. To make this bound useful, we now would like to relate $|\mathcal{L}^i_q|$ to $\hat{e}^i$.

The underlying idea behind establishing a relationship between $|\mathcal{L}^i_q|$ and $e^i = \sum_{q':\tau_i \geq t_{q'}} e^i_{q'}$ is simple. By Observation 3, quantity e^i is bounded by the number of packets that are stored in OPT's buffer, but not in LQD's buffer at time τ_i. Recall that the LQD buffer is full in step τ_i. Intuitively, for each packet that OPT has in its buffer but LQD has not, there must be a packet that LQD has in its buffer but OPT has not. Suppose for a moment that the latter packets are all located in queues in $\mathcal{L}^i_q$. Then there can be no more than $\sigma^i_q \cdot |\mathcal{L}^i_q|$ of them and so, we would have $e^i \leq \sigma^i_q \cdot |\mathcal{L}^i_q|$. Unfortunately, things are more complicated because not all packets of the latter type may be located in live queues. We address this problem by introducing the earlier mentioned careful mapping of transmitted LQD-extra packets to cancel out some of the packets counted in e^i. The following notation will be useful for brevity:

- $\mathcal{Q}^i$: the set of queues q with $\tau_i \geq t_q$ and $e^i_q > 0$. In words, $\mathcal{Q}^i$ is the set of queues that have overflowed for the last time by step τ_i and there are still some OPT-extra packets to be transmitted from them in phase i or later.

To describe our specific mapping, we apply the procedure specified in Algorithm 1 on the solutions of LQD and OPT on the fixed instance I. Our values m_q are given as the final values of m'_q after the procedure has been run.

Algorithm 1 Mapping Procedure.

foreach *queue* q **do**
  Initialize $m'_q := 0$ `// counter for packets assigned to` q
foreach *phase* i **do**
  foreach *LQD-extra packet* p *transmitted in phase* i **do**
    if *there is a queue* $q \in \mathcal{Q}^i$ *with* $m'_q < e^i_q$ **then**
      $q' := \operatorname{arg\,min}_{q:q\in\mathcal{Q}^i \text{ and } m'_q < e^i_q} \{t_q\}$ `// breaking ties arbitrarily`
      $m'_{q'} := m'_{q'} + 1$ `// assign packet` p `to queue` q'
    `// Otherwise, packet` p `is not assigned`
foreach *queue* q **do**
  $m_q := m'_q$ `// the final value of` m'_q

We now show a lower bound on the $m_{q'}$ values for a phase i. The bound is specific to a particular queue $q \in \mathcal{Q}^i$ with $m_q < e_q$; in the following, live and dying queues are w.r.t. queue q. For technical reasons, we prove a lower bound on the following quantity: Let $\overline{m}^i_{q'}$ be the number of LQD-extra packets transmitted in a phase $j \geq i$ that are mapped to q'. The point is that the constraint $m'_{q'} < e^i_{q'}$ for assigning an LQD-extra packet to q' in Algorithm 1 implies that $e^i_{q'} \geq \overline{m}^i_{q'}$, even though it may happen that $e^i_{q'} < m_{q'}$.

For simplicity, let $z_q^i := \sum_{q' \in \mathcal{L}_q^i} [s_{\mathsf{OPT}}^i(q') > 0]$ be the number of live queues $\mathcal{L}_q^i$ that are non-empty in OPT. In words, the first term of the bound in Lemma 12 below, i.e., $\sum_{q' \in \mathcal{D}_q^i} \max\left\{s_{\mathsf{LQD}}^i(q') - s_{\mathsf{OPT}}^i(q'), 0\right\}$, equals the number of packets that LQD stores in excess of OPT in dying queues $q' \in \mathcal{D}_q^i$ with $s_{\mathsf{LQD}}^i(q') > s_{\mathsf{OPT}}^i(q')$ in step τ_i, while the second term, i.e., $|\mathcal{L}_q^i| - z_q^i$, equals the number of live queues $\mathcal{L}_q^i$ that are empty in the OPT buffer in step τ_i.

▶ **Lemma 12.** *For any phase i and queue $q \in \mathcal{Q}^i$ with $m_q < e_q^i$ (i.e., with $\hat{e}_q^i > 0$) we have* $\sum_{q' \in \mathcal{Q}^i} \overline{m}_{q'}^i \geq \sum_{q' \in \mathcal{D}_q^i} \max\left\{s_{LQD}^i(q') - s_{OPT}^i(q'), 0\right\} + \left(|\mathcal{L}_q^i| - z_q^i\right)$.

Proof. Recall from Definition 4 that $j_q = \min\{j : \hat{e}_q^j = 0\}$ is the index j of the earliest step τ_j in which all remaining OPT-extra packets to be transmitted from q are canceled out. Note that $i < j_q$ by the assumption of the lemma and that $m_q < e_q^j$ and $q \in \mathcal{Q}^j$ for any $j \in [i, j_q)$.

Consider a dying queue $q' \in \mathcal{D}_q^i$ with $s_{\mathsf{LQD}}^i(q') - s_{\mathsf{OPT}}^i(q') > 0$. It holds that $e_{q'} = 0$ by Observation 3. By Definition 4, q' will be empty for LQD in step τ_{j_q}. Since q' does not overflow after time τ_i (and hence LQD will accept all packets which may arrive to q' after time τ_i), at least $s_{\mathsf{LQD}}^i(q') - s_{\mathsf{OPT}}^i(q')$ LQD-extra packets are transmitted from q' from time τ_i until time $\tau_{j_q} - 1$, i.e., in phases $j \in [i, j_q)$. Using $m_q < e_q^j$ and $q \in \mathcal{Q}^j$ for any $j \in [i, j_q)$, all these LQD-extra packets are allocated to queues $\overline{q}$ that satisfy $t_{\overline{q}} \leq t_q \leq \tau_i$ and $e_{\overline{q}}^i > 0$; the second property holds as $t_{\overline{q}} \leq t_q \leq \tau_i$ and as $\overline{q} \in \mathcal{Q}^j$ for a phase $i \leq j < j_q$ in which the LQD-extra packet assigned to it is transmitted, by Algorithm 1. Thus, such queues $\overline{q}$ are part of the set $\mathcal{Q}^i$.

In addition, there are $|\mathcal{L}_q^i| - z_q^i$ live queues in step τ_i that are empty in the OPT buffer. Hence, LQD transmits $|\mathcal{L}_q^i| - z_q^i$ LQD-extra packets in step τ_i from such queues, and these packets are assigned to queues $\overline{q} \in \mathcal{Q}^i$ by Algorithm 1, using again that $m_q < e_q^i$. ◀

Finally, we bound the number of OPT-extra packets which are not canceled out by LQD-extra packets. Equivalently, for a queue q, we show a lower bound on $|\mathcal{L}_q^i|$ in terms of $\hat{e}^i$. The lemma below in particular implies that if $\hat{e}_q^i > 0$ then also $|\mathcal{L}_q^i| \geq 1$, i.e., there is at least one live queue w.r.t. queue q.

▶ **Lemma 13.** *For any phase i and queue $q \in \mathcal{Q}^i$ with $\hat{e}_q^i > 0$, we have that $\hat{e}^i \leq (\sigma_q^i - 1) \cdot |\mathcal{L}_q^i|$.*

Proof. First note that

$$\hat{e}^i = \sum_{q' \in \mathcal{Q}^i} \hat{e}_{q'}^i = \sum_{q' \in \mathcal{Q}^i} \max\{e_{q'}^i - m_{q'}, 0\} \leq \sum_{q' \in \mathcal{Q}^i} \max\{e_{q'}^i - \overline{m}_{q'}^i, 0\} = \sum_{q' \in \mathcal{Q}^i} (e_{q'}^i - \overline{m}_{q'}^i) \,,$$

where the inequality holds by $\overline{m}_{q'}^i \leq m_{q'}$ and the last step follows from $e_{q'}^i \geq \overline{m}_{q'}^i$, by the definition of $\overline{m}_{q'}^i$ and Algorithm 1. Using Lemma 12, we obtain

$$\hat{e}^i \leq \sum_{q' \in \mathcal{Q}^i} (e_{q'}^i) - \sum_{q' \in \mathcal{D}_q^i} \max\left\{s_{\mathsf{LQD}}^i(q') - s_{\mathsf{OPT}}^i(q'), 0\right\} - \left(|\mathcal{L}_q^i| - z_q^i\right) \tag{8}$$

By the definition of σ_q^i in (3) and since the LQD buffer is full in step τ_i, we have

$$M = \sigma_q^i \cdot |\mathcal{L}_q^i| + \sum_{q' \in \mathcal{D}_q^i} s_{\mathsf{LQD}}^i(q') \,. \tag{9}$$

Regarding the OPT buffer, we have

$$\begin{aligned} M \geq \sum_{q'} s^i_{\mathsf{OPT}}(q') &\geq \sum_{q' \in \mathcal{Q}^i} \max\left\{s^i_{\mathsf{OPT}}(q') - s^i_{\mathsf{LQD}}(q'), 0\right\} + \sum_{q'} \min\{s^i_{\mathsf{LQD}}(q'), s^i_{\mathsf{OPT}}(q')\} \\ &\geq \sum_{q' \in \mathcal{Q}^i} \left(e^i_{q'}\right) + \sum_{q'} \min\{s^i_{\mathsf{LQD}}(q'), s^i_{\mathsf{OPT}}(q')\} \\ &\geq \sum_{q' \in \mathcal{Q}^i} \left(e^i_{q'}\right) + \sum_{q' \in \mathcal{D}^i_q} \min\{s^i_{\mathsf{LQD}}(q'), s^i_{\mathsf{OPT}}(q')\} + z^i_q\,, \end{aligned} \tag{10}$$

where the third inequality uses Observation 3. Combining (9) and (10), we obtain

$$\sum_{q' \in \mathcal{Q}^i} \left(e^i_{q'}\right) + \sum_{q' \in \mathcal{D}^i_q} \min\{s^i_{\mathsf{LQD}}(q'), s^i_{\mathsf{OPT}}(q')\} + z^i_q \leq \sigma^i_q \cdot |\mathcal{L}^i_q| + \sum_{q' \in \mathcal{D}^i_q} s^i_{\mathsf{LQD}}(q')$$

After rearranging, we get

$$\sum_{q' \in \mathcal{Q}^i} \left(e^i_{q'}\right) - \sum_{q' \in \mathcal{D}^i_q} \max\left\{s^i_{\mathsf{LQD}}(q') - s^i_{\mathsf{OPT}}(q'), 0\right\} + z^i_q \leq \sigma^i_q \cdot |\mathcal{L}^i_q|\,. \tag{11}$$

Finally, plugging (11) into Equation (8), we get $\hat{e}^i \leq \sigma^i_q \cdot |\mathcal{L}^i_q| - |\mathcal{L}^i_q|$, as desired. ◀

6 Putting It All Together

In this section, we complete the proof of 1.707-competitiveness for LQD. First, we use the lemmas developed in previous sections to show a lower bound on the L-increase in phase i for a queue q. This will be divided into two cases, according to whether the value of σ^i_q decreases or not (w.r.t. variable i). Next, we sum these lower bounds over all phases and derive a lower bound for this sum. Finally, we optimize the parameters α and β to maximize ϱ (and thus, minimize the competitive ratio upper bound) subject to $\Phi_q \geq \varrho \cdot \hat{e}_q$ for any queue q.

6.1 Lower Bounds on the L-Increase

In this section, for a queue q, we show lower bounds on the L-increase for a phase i with $\tau_i \geq t_q$ and $\hat{e}^i_q > 0$. In such a phase, Lemma 13 implies that $\hat{e}^i \leq (\sigma^i_q - 1) \cdot |\mathcal{L}^i_q|$. We consider two main cases, depending on whether or not σ_q decreases. We first deal with the case $\sigma^{i+1}_q \geq \sigma^i_q$.

▶ **Lemma 14.** *Consider any phase i and queue q with $\tau_i \geq t_q$, $\hat{e}^i_q > 0$, and $\sigma^{i+1}_q \geq \sigma^i_q$. Then the L-increase in phase i for queue q satisfies*

$$\frac{\hat{e}^i_q}{\hat{e}^i} \cdot (n^i + \alpha \cdot o^i - \Delta^i \Psi) + \beta \cdot o^i_q \geq \hat{e}^i_q \cdot \frac{\alpha \cdot \left(\sigma^{i+1}_q - \sigma^i_q\right) + (\tau_{i+1} - \tau_i)}{\sigma^i_q - 1}\,. \tag{12}$$

Proof. Recall that d^i_q is the number of packets transmitted from queues in $\mathcal{D}^i_q$ during phase i. Using Lemma 9 and $\sigma^{i+1}_q \geq 1$, we get

$$d^i_q \geq (\sigma^{i+1}_q - \sigma^i_q) \cdot |\mathcal{L}^i_q| + \sigma^{i+1}_q \cdot u^i - v^i \geq (\sigma^{i+1}_q - \sigma^i_q) \cdot |\mathcal{L}^i_q| + u^i - v^i\,. \tag{13}$$

Multiplying (13) by $\alpha \in (0, 1)$, adding $(\tau_{i+1} - \tau_i) \cdot |\mathcal{L}^i_q|$ to both sides, and rearranging, we obtain

$$(\tau_{i+1} - \tau_i) \cdot |\mathcal{L}^i_q| + \alpha \cdot d^i_q - \alpha \cdot (u^i - v^i) \geq \left(\alpha \cdot \left(\sigma^{i+1}_q - \sigma^i_q\right) + (\tau_{i+1} - \tau_i)\right) \cdot |\mathcal{L}^i_q|.$$

Using $n^i + \alpha \cdot o^i \geq (\tau_{i+1} - \tau_i) \cdot |\mathcal{L}_q^i| + \alpha \cdot d_q^i$ by Observation 6 and $|\mathcal{L}_q^i| \geq \hat{e}^i/(\sigma_q^i - 1)$ by Lemma 13, we get $n^i + \alpha \cdot o^i - \Delta^i\Psi \geq \dfrac{\alpha \cdot \left(\sigma_q^{i+1} - \sigma_q^i\right) + (\tau_{i+1} - \tau_i)}{\sigma_q^i - 1} \cdot \hat{e}^i$, and multiplying this by $\hat{e}_q^i/\hat{e}^i$ proves (12). ◀

Next, we deal with the (most involved) case when $\sigma_q^{i+1} < \sigma_q^i$. The following technical lemma also captures the case of the last phase $i = j_q - 1$ in which we assign some LQD profit to queue q (in this phase, we have $\hat{e}_q^{i+1} = 0$). We omit its proof due to space constraints.

▶ **Lemma 15.** *Consider any phase i and queue q with $\tau_i \geq t_q$, $\hat{e}_q^i > 0$, and $\sigma_q^{i+1} < \sigma_q^i$. Then, for $\beta = \alpha^2/(8 \cdot (1-\alpha))$ and any $0 < \alpha \leq 8/9$, it holds that*

$$\frac{\hat{e}_q^i}{\hat{e}^i} \cdot (n^i + \alpha \cdot o^i - \Delta^i\Psi) + \beta \cdot o_q^i \geq \sum_{t=\tau_i}^{\tau_{i+1}-1} \left(\frac{\hat{e}_q^t}{\sigma_q^i - 1}\right) - \frac{g_q^i}{2(\sigma_q^i - 1)} - \hat{e}_q^{i+1} \cdot \left(\frac{1}{\sigma_q^{i+1} - 1} - \frac{1}{\sigma_q^i - 1}\right), \tag{14}$$

where g_q^i is the number of steps $t \in [\tau_i, \tau_{i+1})$ with $\hat{e}_q^t > 0$ and $s_{LQD}^t(q) = 0$.

6.2 Total LQD Profit Assigned to a Queue

Fix a queue q with $\hat{e}_q > 0$, i.e., with transmitted OPT-extra packets that are not canceled out by transmitted LQD-extra packets. We now show a lower bound on $\sum_i \Delta^i\Phi_q$. Recall that $\Delta^i\Phi_q > 0$ only for phases i with $\tau_i \geq t_q$ and $\hat{e}_q^i > 0$.

First, we bound the sum of S-increases. Note that as q overflows at t_q, LQD stores for this queue at least $s_{\max}^{t_q} - 1$ packets in step t_q, and since it does not overflow after t_q, LQD transmits at least $s_{\max}^{t_q} - 1 \geq \left\lceil \sigma_q^{t_q} \right\rceil - 1$ packets from q at or after t_q, where the inequality is from Observation 7 (recall that $t_q = \tau_i$ for some phase i). Thus, the sum of S-increases is at least $(1 - \alpha - \beta) \cdot \left(\left\lceil \sigma_q^{t_q} \right\rceil - 1\right)$. The next lemma shows a bound on the total L-increase assigned to queue q (its proof is omitted due to space constraints).

▶ **Lemma 16.** *Assuming $\alpha \leq 0.6$ and using $b_0 = \left\lceil \sigma_q^{t_q} \right\rceil - 1$ for simplicity, the sum of L-increases assigned to a queue q with $\hat{e}_q > 0$ over all phases is at least*

$$\alpha \cdot \hat{e}_q \cdot \left(1 + \frac{b_0}{\hat{e}_q}\right) \cdot \ln\left(1 + \frac{\hat{e}_q}{b_0}\right) + \alpha \cdot \hat{e}_q \cdot \ln\left(\frac{1}{\alpha}\right).$$

Summing up the lower bound on the total S-increase with the lower bound on the total L-increase from Lemma 16, we obtain the following lower bound (where $b_0 = \left\lceil \sigma_q^{t_q} \right\rceil - 1$):

$$\sum_i \Delta^i\Phi_q \geq (1 - \alpha - \beta) \cdot b_0 + \alpha \cdot \hat{e}_q \cdot \left(1 + \frac{b_0}{\hat{e}_q}\right) \cdot \ln\left(1 + \frac{\hat{e}_q}{b_0}\right) + \alpha \cdot \hat{e}_q \cdot \ln\left(\frac{1}{\alpha}\right). \tag{15}$$

6.3 Calculation of the Competitive Ratio Upper Bound

According to the following lemma, we can have $\varrho = 1.41478$ in (1), which implies that the competitive ratio of LQD is at most $1 + 1/\varrho < 1.707$, according to the discussion in Section 2. Thus, the following lemma concludes the proof of Theorem 1.

▶ **Lemma 17.** *Consider a queue q with $\hat{e}_q > 0$. For any values of $\sigma_q^{t_q}$ and $\hat{e}_q$, it holds that $\sum_i \Delta^i\Phi_q \geq \varrho \cdot \hat{e}_q$ for $\varrho = 1.41478$.*

Proof sketch. As before, let $b_0 = \left\lceil \sigma_q^{t_q} \right\rceil - 1$. Using inequality (15) as a lower bound on $\sum_i \Delta^i \Phi_q$, it is sufficient to show

$$(1-\alpha-\beta)\cdot b_0 + \alpha\cdot\hat{e}_q\cdot\left(1+\frac{b_0}{\hat{e}_q}\right)\cdot\ln\left(1+\frac{\hat{e}_q}{b_0}\right)+\alpha\cdot\hat{e}_q\cdot\ln\left(\frac{1}{\alpha}\right)\geq\varrho\cdot\hat{e}_q\,.$$

Dividing by $\hat{e}_q$ and defining $x=\frac{\hat{e}_q}{b_0}$ gives us that the optimal choice of ϱ satisfies,

$$\varrho := \sup_{0<\alpha<0.6}\inf_{x>0}\left((1-\alpha-\beta)\cdot\frac{1}{x}+\alpha\cdot\left(1+\frac{1}{x}\right)\cdot\ln(1+x)+\alpha\cdot\ln\left(\frac{1}{\alpha}\right)\right)\,. \tag{16}$$

(Here, we also take into account that we require $\alpha < 0.6$ in the analysis.) Using routine calculations and optimizing through mathematical software, we get that the optimal choice for α is approximately 0.57635 for which $\varrho \geq 1.41478$ and therefore, the competitive ratio of LQD is at most $1 + 1/\varrho \leq 1.70683$. ◀

References

1. W. Aiello, A. Kesselman, and Y. Mansour. Competitive buffer management for shared-memory switches. *ACM Transactions on Algorithms*, 5(1):3:1–3:16, 2008.
2. K. Al-Bawani, M. Englert, and M. Westermann. Online packet scheduling for CIOQ and buffered crossbar switches. *Algorithmica*, 80(12):3861–3888, 2018.
3. S. Albers and M. Schmidt. On the performance of greedy algorithms in packet buffering. *SIAM Journal on Computing*, 35(2):278–304, 2005.
4. N. Andelman, Y. Mansour, and A. Zhu. Competitive queueing policies for QoS switches. In *Proceedings of the 14th ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 761–770, 2003.
5. Y. Azar and A. Litichevskey. Maximizing throughput in multi-queue switches. *Algorithmica*, 45(1):69–90, 2006.
6. Y. Azar and Y. Richter. Management of multi-queue switches in QoS networks. *Algorithmica*, 43(1-2):81–96, 2005.
7. N. Bansal, L. Fleischer, T. Kimbrel, M. Mahdian, B. Schieber, and M. Sviridenko. Further improvements in competitive guarantees for QoS buffering. In *Proceedings of the 31st International Colloquium on Automata, Languages and Programming (ICALP)*, pages 196–207, 2004.
8. M. Bienkowski, M. Chrobak, and Ł Jeż. Randomized competitive algorithms for online buffer management in the adaptive adversary model. *Theoretical Computer Science*, 412(39):5121–5131, 2011.
9. I. Bochkov, A. Davydow, N. Gaevoy, and S. I. Nikolenko. New competitiveness bounds for the shared memory switch. *CoRR*, abs/1907.04399, 2019. `arXiv:1907.04399`.
10. J. L. Bruno, B. Özden, A. Silberschatz, and H. Saran. Early fair drop: a new buffer management policy. *Multimedia Computing and Networking*, 3654:148–161, 1998.
11. S. Chamberland and B. Sansò. Overall design of reliable ip networks with performance guarantees. In *Proceedings of the IEEE International Conference on Communications: Global Convergence Through Communications (ICC)*, pages 1145–1151, 2000.
12. H. J. Chao and X. Guo. *Quality of Service Control in High-Speed Networks*. Wiley-IEEE Press, 2001.
13. H. J. Chao and B. Liu. *High Performance Switches and Routers*. Wiley-IEEE Press, 2007.
14. F. Y. L. Chin, M. Chrobak, S. P. Y. Fung, W. Jawor, J. Sgall, and T. Tichý. Online competitive algorithms for maximizing weighted throughput of unit jobs. *Journal of Discrete Algorithms*, 4(2):255–276, 2006.
15. F. Y. L. Chin and S. P. Y. Fung. Online scheduling with partial job values: Does timesharing or randomization help? *Algorithmica*, 37(3):149–164, 2003.

16 M. Chrobak, W Jawor, J. Sgall, and T. Tichý. Improved online algorithms for buffer management in QoS switches. *ACM Transactions on Algorithms*, 3(4):50, 2007.

17 M. Englert and M. Westermann. Lower and upper bounds on FIFO buffer management in QoS switches. *Algorithmica*, 53(4):523–548, 2009.

18 M. Englert and M. Westermann. Considering suppressed packets improves buffer management in quality of service switches. *SIAM Journal on Computing*, 41(5):1166–1192, 2012.

19 P. Eugster, K. Kogan, S. Nikolenko, and A. Sirotkin. Shared memory buffer management for heterogeneous packet processing. In *Proceedings of the 34th IEEE International Conference on Distributed Computing Systems (ICDCS)*, pages 471–480, 2014.

20 M. H. Goldwasser. A survey of buffer management policies for packet switches. *SIGACT News*, 41(1):100–128, 2010.

21 E. L. Hahne, A. Kesselman, and Y. Mansour. Competitive buffer management for shared-memory switches. In *Proceedings of the 13th ACM Symposium on Parallelism in Algorithms and Architectures (SPAA)*, pages 53–58, 2001.

22 B. Hajek. On the competitiveness of on-line scheduling of unit-length packets with hard deadlines in slotted time. In *Proceedings of the 35th Conference on Information Sciences and Systems*, pages 434–438, 2001.

23 Ł. Jeż. A universal randomized packet scheduling algorithm. *Algorithmica*, 67(4):498–515, 2013.

24 A. Kesselman, Z. Lotker, Y. Mansour, B. Patt-Shamir, B. Schieber, and M. Sviridenko. Buffer overflow management in QoS switches. *SIAM Journal on Computing*, 33(3):563–583, 2004.

25 A. Kesselman and Y. Mansour. Harmonic buffer management policy for shared memory switches. *Theoretical Computer Science*, 324(2-3):161–182, 2004.

26 A. Kesselman, Y. Mansour, and R. van Stee. Improved competitive guarantees for QoS buffering. *Algorithmica*, 43(1-2):63–80, 2005.

27 A. Kesselman and A. Rosén. Scheduling policies for CIOQ switches. *Journal of Algorithms*, 60(1):60–83, 2006.

28 K. M. Kobayashi, S. Miyazaki, and Y. Okabe. A tight bound on online buffer management for two-port shared-memory switches. In *Proceedings of the 19th ACM Symposium on Parallelism in Algorithms and Architectures (SPAA)*, pages 358–364, 2007.

29 F. Li, J. Sethuraman, and C. Stein. Better online buffer management. In *Proceedings of the 18th ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 199–208, 2007.

30 N. Matsakis. *Approximation Algorithms for Packing and Buffering problems*. PhD thesis, University of Warwick, UK, 2015.

31 M. Nabeshima and K. Yata. Performance improvement of active queue management with per-flow scheduling. *IEE Proceedings-Communications*, 152(6):797–803, 2005.

32 S. I. Nikolenko and K. Kogan. Single and multiple buffer processing. In *Encyclopedia of Algorithms*, pages 1988–1994. Springer, 2016.

33 B. Suter, T. V. Lakshman, D. Stiliadis, and A. K. Choudhury. Design considerations for supporting TCP with per-flow queueing. In *Proceedings of the 17th IEEE Conference on Computer Communications (INFOCOM)*, pages 299–306, 1998.

34 P. Veselý, M. Chrobak, Ł. Jeż, and J. Sgall. A ϕ-competitive algorithm for scheduling packets with deadlines. In *Proceedings of the 30th ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 123–142, 2019.

35 S. X. Wei, E. J. Coyle, and M. T. Hsiao. An optimal buffer management policy for high-performance packet switching. In *Proceedings of the Global Communication Conference (GLOBECOM)*, pages 924–928, 1991.

36 A. Zhu. Analysis of queueing policies in QoS switches. *Journal of Algorithms*, 53(2):137–168, 2004.

Relaxed Locally Correctable Codes with Improved Parameters

Vahid R. Asadi
Simon Fraser University, Burnaby, Canada

Igor Shinkar
Simon Fraser University, Burnaby, Canada

Abstract

Locally decodable codes (LDCs) are error-correcting codes $C\colon \Sigma^k \to \Sigma^n$ that admit a local decoding algorithm that recovers each individual bit of the message by querying only a few bits from a noisy codeword. An important question in this line of research is to understand the optimal trade-off between the query complexity of LDCs and their block length. Despite importance of these objects, the best known constructions of constant query LDCs have super-polynomial length, and there is a significant gap between the best constructions and the known lower bounds in terms of the block length.

For many applications it suffices to consider the weaker notion of *relaxed LDCs (RLDCs)*, which allows the local decoding algorithm to abort if by querying a few bits it detects that the input is not a codeword. This relaxation turned out to allow decoding algorithms with constant query complexity for codes with *almost linear* length. Specifically, [2] constructed a q-query RLDC that encodes a message of length k using a codeword of block length $n = O_q(k^{1+O(1/\sqrt{q})})$ for any sufficiently large q, where $O_q(\cdot)$ hides some constant that depends only on q.

In this work we improve the parameters of [2] by constructing a q-query RLDC that encodes a message of length k using a codeword of block length $O_q(k^{1+O(1/q)})$ for any sufficiently large q. This construction matches (up to a multiplicative constant factor) the lower bounds of [14, 23] for constant query *LDCs*, thus making progress toward understanding the gap between LDCs and RLDCs in the constant query regime.

In fact, our construction extends to the stronger notion of relaxed locally *correctable* codes (RLCCs), introduced in [13], where given a noisy codeword the correcting algorithm either recovers each individual bit of the codeword by only reading a small part of the input, or aborts if the input is detected to be corrupt.

2012 ACM Subject Classification Theory of computation → Error-correcting codes

Keywords and phrases Algorithmic coding theory, consistency test using random walk, Reed-Muller code, relaxed locally decodable codes, relaxed locally correctable codes

Digital Object Identifier 10.4230/LIPIcs.ICALP.2021.18

Category Track A: Algorithms, Complexity and Games

Related Version *Full Version*: `https://eccc.weizmann.ac.il/report/2020/142/` [1]

Acknowledgements We are thankful to Tom Gur for many fruitful discussions on this topic, and we are grateful to the anonymous referees for their valuable suggestions that helped improve the presentation of the paper.

1 Introduction

Locally decodable codes (LDCs) are error-correcting codes that admit a decoding algorithm that recovers each specific symbol of the message by reading a small number of locations in a possibly corrupted codeword. More precisely, a locally decodable code $C\colon \mathbb{F}^k \to \mathbb{F}^n$ with local decoding radius $\tau \in [0, 1]$ is an error-correcting code that admits a local decoding

algorithm $\mathcal{D}_C$, such that given an index $i \in [k]$ and a corrupted word $w \in \mathbb{F}^n$ which is τ-close to an encoding of some message $C(M)$, reads a small number of symbols from w, and outputs M_i with high probability. Similarly, we have the notion of *locally correctable codes (LCCs)*, which are error-correcting codes that not only admit a local algorithm that decodes each symbol of the message, but are also required to correct an arbitrary symbol from the entire codeword. Indeed, note that for systematic codes the notion of LCC is strengthening of LDC. Locally decodable and locally correctable codes have many applications in different areas of theoretical computer science, such as complexity theory, coding theory, property testing, cryptography, and construction of probabilistically checkable proof systems. For details, see the surveys [26, 18] and the references within.

Despite the importance of LDCs and LCCs, and the extensive amount of research studying these objects, the best known construction of constant query LDCs has super-polynomial length $n = \exp(\exp(\log^{\Omega(1)}(k)))$, which is achieved by the highly non-trivial constructions of [25] and [9]. For constant query LCCs, the best known constructions are of exponential length, which can be achieved by some parameterization of Reed-Muller codes. It is important to note that there is a huge gap between the best known lower bounds for the length of constant query LDCs and the length of best known constructions. Currently, the best known lower bound on the length of LDCs says that for $q \geq 3$ it must be at least $k^{1+\Omega(1/q)}$, where q stands for the query complexity of the local decoder. See [14, 17, 23] for the best general lower bounds for constant query LDCs.

Motivated by applications to probabilistically checkable proofs (PCPs), Ben-Sasson, Goldreich, Harsha, Sudan, and Vadhan introduced in [2] the notion of *relaxed locally decodable codes (RLDCs)*. Informally speaking, a relaxed locally decodable code is an error-correcting code which allows the local decoding algorithm to abort if the input codeword is corrupt, but does not allow it to err with high probability. In particular, the decoding algorithm should always output the correct symbol, if the given word is not corrupted. Formally, a code $C\colon \mathbb{F}^k \to \mathbb{F}^n$ is an RLDC with decoding radius $\tau \in [0, 1]$ if it admits a relaxed local decoding algorithm $\mathcal{D}_C$ which given an index $i \in [k]$ and a possibly corrupted codeword $w \in \mathbb{F}^n$, makes a small number of queries to w, and satisfies the following properties.

Completeness: If $w = C(M)$ for some $M \in \mathbb{F}^k$, then $\mathcal{D}_C^w(i)$ should output M_i.

Relaxed decoding: If w is τ-close to some codeword $C(M)$, then $\mathcal{D}_C^w(i)$ should output either M_i or a special *abort* symbol with probability at least $2/3$.

This relaxation turns out to be very helpful in terms of constructing RLDCs with better block length. Indeed, [2] constructed of a q-query RLDC with block length $n = k^{1+O(1/\sqrt{q})}$.

The notion of *relaxed LCCs (RLCCs)*, recently introduced in [13], naturally extends the notion of RLDCs. These are error-correcting codes that admit a correcting algorithm that is required to correct every symbol of the codeword, but is allowed to abort if noticing that the given word is corrupt. More formally, the local correcting algorithm gets an index $i \in [n]$, and a (possibly corrupted) word $w \in \mathbb{F}^n$, makes a small number of queries to w, and satisfies the following properties.

Completeness: If $w \in C$, then $\mathcal{D}_C^w(i)$ should output w_i.

Relaxed correcting: If w is τ-close to some codeword $c^* \in C$, then $\mathcal{D}_C^w(i)$ should output either c_i^* or a special *abort* symbol with probability at least $2/3$.

Note that if the code C is systematic, i.e., the encoding of any message $M \in \mathbb{F}^k$ contains M in its first k symbols, then the notion of RLCC is stronger than RLDC.

Recently, building on the ideas from [13], [3] constructed RLCCs whose block length matches the RLDC construction of [2]. For the lower bounds, the only result we are aware of is the work of Gur and Lachish [12], who proved that for any RLDC the block length must be at least $n = k^{1+\Omega(1/q^2)}$.

Given the gap between the best constructions and the known lower bounds, it is natural to ask the following question:

> What is the best possible trade-off between the query complexity and the block length of an RLDC and RLCC?

In particular, [2] asked whether it is possible to obtain a q-query RLDC whose block length is strictly smaller than the best known lower bound on the length of LDCs. A positive answer to their question would show a separation between the two notions, thus proving that the relaxation is *strict*. See paragraph *Open Problem* in the end of Section 4.2 of [2].

In this work we make progress on this problem by constructing a relaxed locally decodable code $C\colon \mathbb{F}^K \to \mathbb{F}^N$ with query complexity $O(q)$ and block length $K^{1+O(1/q)}$. In fact, our construction gives the stronger notion of a relaxed locally correctable code.

▶ **Theorem 1** (Main Theorem). *For every sufficiently large $q \in \mathbb{N}$ there exists an q-query relaxed locally correctable code $C\colon \{0,1\}^K \to \{0,1\}^N$ with constant relative distance and constant decoding radius, such that the block length of C is*

$$N = q^{O(q^2)} \cdot K^{1+O(1/q)} \ .$$

Furthermore, the code C is linear and systematic.

Therefore, our construction improves the parameters of the q-query RLDC construction of [2] with block length $N = K^{1+O(\sqrt{1/q})}$, and matches (up to a multiplicative factor in q) the lower bound of $\Omega(K^{1+\frac{1}{\lceil q/2 \rceil - 1}})$ for the block length of q-query LDCs [14, 23].

▶ Remark 2. In this paper we prove Theorem 1 for a code $C\colon \mathbb{F}^K \to \mathbb{F}^N$ over a large alphabet. Specifically, we show a code $C\colon \mathbb{F}^K \to \mathbb{F}^N$ satisfying Theorem 1, for a finite field $\mathbb{F}$ satisfying $|\mathbb{F}| \geq c_q \cdot K^{1/q}$, for some $c_q \in \mathbb{N}$ that depends only on q.

Using the techniques from [3] it is not difficult to obtain an RLCC over the binary alphabet with almost the same block length. Indeed, this can be done by concatenating our code over large alphabet with an arbitrary binary code with constant rate and constant relative distance. The concatenation we use slightly differs from how it is usually applied, as we apply it on the CTRW level, and not on each symbol of the large alphabet separately. See Section 3 for details.

1.1 Related works

RLDC and RLCC constructions. Relaxed locally decodable codes, were first introduced by [2], motivated by applications to constructing short PCPs. Their construction has a block length equal to $N = K^{1+O(1/\sqrt{q})}$. Since that work, there were no constructions with better block length, in the constant query complexity regime . Recently, [13] introduced the related notion of relaxed locally correctable codes (RLCCs), and has found applications in efficient interactive protocols, in particular, in interactive oracle proofs [21].

The work of [13] showed a construction of a q-query RLCCs with block length $N = \text{poly}(K)$. Then, [3] constructed relaxed locally correctable codes with block length matching that of [2] (up to a multiplicative constant factor that only depends on q). The construction of [3] had two main components, that we also use in the current work.

Consistency test using random walk (CTRW): Informally, given a word w, and a coordinate i we wish to correct, CTRW samples a sequence of constraints $\mathcal{C}_1, \mathcal{C}_2, \ldots, \mathcal{C}_t$ on w, such that the domains of $\mathcal{C}_i$ and $\mathcal{C}_{i+1}$ intersect, with the guarantee that if w is close

to some codeword $c^* \in C$, but $w_i \neq c^*_i$, then with high probability w will be far from satisfying at least one of the constraints. In other words, CTRW performs a random walk on the constraints graph and checks if w is consistent with c^* in the i'th coordinate. We introduce this notion in detail in Section 2.1.

Correctable canonical PCPPs (ccPCPP): These are PCPP systems for some specified language L satisfying the following properties:

- **(i)** for each $w \in L$ there is a unique proof $\pi(w)$ that satisfies the verifier with probability 1,
- **(ii)** the verifier accepts with high probability only pairs (x, π) that are close to some $(w, \pi(w))$ for some $w \in L$, i.e., only the pairs where x is close to some $w \in L$, and π is close to $\pi(w)$, and
- **(iii)** the set $\{w \circ \pi_w : w \in L\}$ is an RLCC.

Canonical proofs of proximity have been introduced in [11] and have been studied in [7, 13, 20].

Lower bounds. For lower bounds, the only bound we are aware of is that of [12], who proved that any q-query relaxed locally decodable code must have a block length $N \geq K^{1+\Omega(\frac{1}{q^2})}$.

For the strict notion of locally decodable codes, it is known by [14, 23] that for $q \geq 3$ any q-query LDC must have block length $N \geq \Omega(K^{1+\frac{1}{\lceil q/2-1 \rceil}}/\log(K))$. For $q = 3$ a slightly stronger bound of $N \geq \Omega(K^2/\log\log(K))$ is known for *linear* LDCs [23]. For $q = 2$ [17] proved an exponential lower bound of $N \geq \exp(\Omega(K))$. See also [4, 10, 19, 22, 24] for more related work on lower bounds for LDCs.

2 Proof overview

In this section we informally describe our code construction. Roughly speaking, our construction consists of two parts:

The Reed-Muller encoding: Given a message $M \in \mathbb{F}^K$, its Reed-Muller encoding is the evaluation of an m-variate polynomial of degree at most d over $\mathbb{F}$, whose coefficients are determined by the message we wish to encode.

Proofs of proximity: The second part of the encoding consists of the concatenation of PCPPs, each claiming that a certain restriction of the first part agrees with some Reed-Muller codeword.

Specifically, given a message $M \in \mathbb{F}^K$, we first encode it using the Reed-Muller encoding $\mathsf{RM}_{\mathbb{F}}(m, d)$, where m roughly corresponds to the query complexity of our RLDC, and the field is large enough so that the distance of the Reed-Muller code, which is equal to $1 - \frac{d}{|\mathbb{F}|}$, is some constant, say $3/4$. That is, the first part of the encoding corresponds to an evaluation of some polynomial $f \colon \mathbb{F}^m \to \mathbb{F}$ of degree at most d. The second part of the encoding consists of a sequence of PCPPs claiming that the restrictions of the Reed-Muller part to some carefully chosen planes in $\mathbb{F}^m$ are evaluations of some low-degree polynomial.

The planes we choose are of the form $\mathcal{P}_{\vec{a},\vec{h},\vec{h}'} = \{\vec{a} + t \cdot \vec{h} + s \cdot \vec{h}' : t, s \in \mathbb{F}\}$, where $\vec{a} \in \mathbb{F}^m$, and $\vec{h}, \vec{h}' \in \mathbb{H}^m$ for some $\mathbb{H}$ subfield of $\mathbb{F}$. We will call such planes $\mathbb{H}$-*planes*. In order to obtain the RLDC with the desired parameters, we choose the field $\mathbb{H}$ so that $\mathbb{F}$ is the extension of $\mathbb{H}$ of degree $[\mathbb{F} : \mathbb{H}] = m$. It will be convenient to think of $\mathbb{H}$ as a field and think of $\mathbb{F}$ as a vector space of $\mathbb{H}$ of dimension m (augmented with the multiplicative structure on $\mathbb{F}$). Indeed, the saving in the block length of the RLDC we obtain crucially relies on the fact that we ask for PCPPs for only a small collection of planes, and not all planes in $\mathbb{F}^m$. The actual constraints required to be certified by the PCPPs are slightly more complicated, and we describe them next.

The constraints of the first type correspond to $\mathbb{H}$-planes $\mathcal{P}$ and points $\vec{x} \in \mathcal{P}$. For each such pair $(\mathcal{P}, \vec{x})$ the code will contain a PCPP certifying that

(i) the restriction of the Reed-Muller part to $\mathcal{P}$ is close to an evaluation of some polynomial of total degree at most d,

(ii) and furthermore, this polynomial agrees with the value of the Reed-Muller part on $\vec{x}$.

In order to define it formally, we introduce the following notation.

▶ **Notation 3.** *Let $\mathbb{F}$ be a finite field of size n. Fix $f\colon \mathbb{F}^m \to \mathbb{F}$, a plane $\mathcal{P}$ in $\mathbb{F}^m$, and a point $\vec{x} \in \mathcal{P}$. Denote $f_{|\mathcal{P}}^{(\vec{x})} = f_{|\mathcal{P}} \circ (f(\vec{x}))^{n^2}$. That is, the length of $f_{|\mathcal{P}}^{(\vec{x})}$ is $2 \cdot n^2$, and it consists of $f_{|\mathcal{P}}$ concatenated with n^2 repetitions of $f(\vec{x})$.*

Given the notation above, if f is the first part of the codeword, corresponding to the Reed-Muller encoding of the message, then the PCPP for the pair $(\mathcal{P}, \vec{x})$ is expected to be the proof of proximity claiming that $f_{|\mathcal{P}}^{(\vec{x})}$ is close to the language

$$\mathsf{RM}_{|\mathcal{P}}^{(\vec{x})} = \{Q \circ (Q(\vec{x}))^{(n^2)} : Q \text{ is the evaluation of a degree-}d \text{ polynomial on } \mathcal{P}\} \subseteq \mathbb{F}^{2n^2} \quad . \quad (1)$$

Note that by repeating the symbol $Q(\vec{x})$ for n^2 times, the definition indeed puts weight $1/2$ on the constraint that the input $f_{|\mathcal{P}}$ is close to some low-degree polynomial Q, and puts weight $1/2$ on the constraint $f(\vec{x}) = Q(\vec{x})$. In particular, if $f_{|\mathcal{P}}$ is δ-close to some bivariate low degree polynomial Q for some small $\delta > 0$, but $f(\vec{x}) \neq Q(\vec{x})$, then $f_{|\mathcal{P}}^{(\vec{x})}$ is at least $(1 - \frac{d}{|\mathbb{F}|} - \delta)/2$-far from any bivariate low degree polynomial on $\mathcal{P}$.

The constraints of second type correspond to $\mathbb{H}$-planes $\mathcal{P}$ and lines $\ell \subseteq \mathcal{P}$. For each such pair $(\mathcal{P}, \ell)$ the code will contain a PCPP certifying that

(i) the restriction of the Reed-Muller part to $\mathcal{P}$ is close to an evaluation of some polynomial of total degree at most d,

(ii) and furthermore, the restriction of this polynomial to ℓ is close to $f_{|\ell}$.

(In particular, this implies that $f_{|\ell}$ is close to some low-degree polynomial.)

Next, we introduce the notation analogous to Notation 3 replacing the points with lines.

▶ **Notation 4.** *Let $\mathbb{F}$ be a finite field of size n. Fix $f\colon \mathbb{F}^m \to \mathbb{F}$, a plane $\mathcal{P}$ in $\mathbb{F}^m$, and a line $\ell \subseteq \mathcal{P}$. Denote by $f_{|\mathcal{P}}^{(\ell)} = f_{|\mathcal{P}} \circ (f_{|\ell})^n$. That is, the length of $f_{|\mathcal{P}}^{(\ell)}$ is $2 \cdot n^2$, and it consists of $f_{|\mathcal{P}}$ concatenated with n repetitions of $f_{|\ell}$.*

If f is the Reed-Muller part of the codeword, corresponding to the Reed-Muller encoding of the message, then the PCPP for the pair $(\mathcal{P}, \ell)$ is expected to be the proof of proximity claiming that $f_{|\mathcal{P}}^{(\ell)}$ is close to the language

$$\mathsf{RM}_{|\mathcal{P}}^{(\ell)} = \{Q \circ (Q_{|\ell})^n : Q \text{ is the evaluation of some degree-}d \text{ polynomial on } \mathcal{P}\} \subseteq \mathbb{F}^{2n^2} \quad . \quad (2)$$

Again, similarly to the first part, repeating the evaluation of $Q_{|\ell}$ for n times puts weight $1/2$ on the constraint that the input $f_{|\mathcal{P}}$ is a close to some low-degree polynomial Q, and puts weight $1/2$ of the constraint $f_{|\ell}$ is close to $Q_{|\ell}$.

With the proofs specified above, we now sketch the local correcting algorithm for the code. Below we only focus on correcting symbols from the Reed-Muller part. Correcting the symbols from the PCPP part follows a rather straightforward adaptation of the techniques from [3], and we omit them from the overview.

Given a word $w \in \mathbb{F}^N$ and an index $i \in [N]$ of w corresponding to the Reed-Muller part of the codeword, let $f\colon \mathbb{F}^m \to \mathbb{F}$ be the Reed-Muller part of w, and let $\vec{x} \in \mathbb{F}^m$ be the input to f corresponding to the index i. The local decoder works in two steps.

Consistency test using random walk: In the first step the correcting algorithm invokes a procedure we call *consistency test using a random walk (*CTRW*)* for the Reed-Muller code. This step creates a sequence of $\mathbb{H}$-planes of length $(m+1)$, where each plane defines a constraint checking that the restriction of w to the plane is low-degree. Hence, we get $m+1$ constraints, each depending on n^2 symbols.

Composition using proofs of proximity: Then, instead of reading the entire plane for each constraint, we use the PCPPs from the second part of the codeword to reduce the arity of each constraint to $O(1)$, thus reducing the total query complexity of the correcting algorithm to $q = O(m)$. That is, for each constraint we invoke the corresponding PCPP verifier to check that the restrictions of f to each of these planes is (close to) a low-degree polynomial. If at least one of the verifiers rejects, then the word f must be corrupt, and hence the correcting algorithm returns $\perp$. Otherwise, if all the PCPP verifiers accept, the correcting algorithm returns $f(\vec{x})$.

In particular, if f is a correct Reed-Muller encoding, then the algorithm will always return $f(\vec{x})$, and the main part of the analysis is to show that if f is close to some $Q^* \in \mathsf{RM}_{\mathbb{F}}(m,d)$, but $f(\vec{x}) \neq Q^*(\vec{x})$, then the correcting algorithm catches an inconsistency, and returns $\perp$ with some constant probability.

The key step in the analysis says that if f is close to some codeword $Q^* \in \mathsf{RM}$ but $f(\vec{x}) \neq Q^*(\vec{x})$, then with high probability f will be far from a low degree polynomial on at least one of these planes, where "far" corresponds to the notion of distances defined by the languages $\mathsf{RM}^{(\vec{x})}_{|\mathcal{P}}$ and $\mathsf{RM}^{(\ell)}_{|\mathcal{P}}$. In particular, if on one of the planes f is far from the corresponding language, then the PCPP verifier will catch this with constant probability, thus causing the correcting algorithm to return $\perp$. We discuss this part in detail below.

It is important to emphasize that the main focus of this work is constructing a correcting algorithm for the Reed-Muller part. Using the techniques developed in [3], it is rather straightforward to design the algorithm for correcting symbols from the PCPPs part of the code. See the full version for details.

2.1 CTRW on Reed-Muller codes

Below we define the notion of *consistency test using random walk (*CTRW*)* for the Reed-Muller code. This notion is a slight modification of the notion originally defined in [3] for general codes. In this paper we define it only for the Reed-Muller code. Given a word $f\colon \mathbb{F}^m \to \mathbb{F}$ and some $\vec{x} \in \mathbb{F}^m$, the goal of the test is to make sure that $f(\vec{x})$ is consistent with the codeword of Reed-Muller code closest to f. [3] describe a CTRW for the tensor power $C^{\otimes m}$ of an arbitrary codes C with good distance (e.g., Reed-Solomon). The CTRW they describe works by starting from the point we wish to correct, and choosing an axis-parallel line ℓ_1 containing the starting point. The test continues by choosing a sequence of random axis-parallel lines $\ell_2, \ell_3, \ldots \ell_t$, such that each ℓ_i intersects the previous one, ℓ_{i-1}, until reaching a uniformly random coordinate of the tensor code. That is, the length of the sequence t denotes the *mixing time of the corresponding random walk*. The predicates are defined in the natural way; namely, the test expects to see a codeword of C on each line it reads.

In this work, we present a CTRW for the Reed-Muller code, which is a variant of the CTRW described above. The main differences compared to the description above are that

(i) the test chooses a sequence of planes $\mathcal{P}_1, \mathcal{P}_3, \ldots \mathcal{P}_t$ (and not lines),

(ii) and every two planes intersect on a line (and not on a point).

Roughly speaking, the algorithm works as follows.

1. Given a point $\vec{x} \in \mathbb{F}^m$ the test picks a uniformly random $\mathbb{H}$-plane $\mathcal{P}_0$ containing $\vec{x}$.
2. Given $\mathcal{P}_0$, the test chooses a random line $\ell_1 \subseteq \mathcal{P}_0$, and then chooses another random $\mathbb{H}$-plane $\mathcal{P}_1 \subseteq \mathbb{F}^m$ containing ℓ_1.
3. Given $\mathcal{P}_1$, the test chooses a random line $\ell_2 \subseteq \mathcal{P}_1$, and then chooses another random $\mathbb{H}$-plane $\mathcal{P}_2 \subseteq \mathbb{F}^m$ containing ℓ_2.
4. The algorithm continues for some predefined number of iterations, choosing $\mathcal{P}_0, \mathcal{P}_1, \mathcal{P}_2, \ldots \mathcal{P}_t$. Roughly speaking, the number of iterations is equal to the mixing time of the corresponding Markov chain. More specifically, the process continues until a uniformly random point in $\mathcal{P}_t$ is close to a uniform point in $\mathbb{F}^m$.
5. The constraints defined for each $\mathcal{P}_i$ are the natural constraints; namely checking that the restriction of f to the entire plane $\mathcal{P}_i$ is a polynomial of degree at most d.

One of the important parameters, directly affecting the query complexity of our construction is the mixing time of the random walk. Indeed, as explained above, the query complexity of our RLDC is proportional to the mixing time of the random walk. We prove that if $[\mathbb{F} : \mathbb{H}] = m$, then the mixing time is upper bounded by m. In order to prove this we use the following claim, saying that if $\mathbb{F}$ is the field extension of $\mathbb{H}$ of degree m, and $\vec{h}_1, \ldots, \vec{h}_m \in \mathbb{H}^m$ and $t_1, \ldots, t_m \in \mathbb{F}$ are sampled uniformly, independently from each other, then $\sum_{i=1}^{m} t_i \cdot \vec{h}_i$ is close to a uniformly random point in $\mathbb{F}^m$. See the full version for the exact statement.

As explained above, the key step of the analysis is to prove that if f is close to some codeword $Q^* \in \mathsf{RM}$ but $f(\vec{x}) \neq Q^*(\vec{x})$, then with high probability at least one of the predicates defined will be violated. Specifically, we prove that with high probability the violation will be in the following strong sense.

▶ **Theorem 5** (informal, see the full version [1]). *If f is close to some codeword $Q^* \in \mathsf{RM}$ but $f(\vec{x}) \neq Q^*(\vec{x})$, then with high probability*
1. *either $f^{(\vec{x})}_{|\mathcal{P}_0}$ is $\Omega(1)$-far from $\mathsf{RM}^{(\vec{x})}_{|\mathcal{P}_0}$,*
2. *or $f^{(\ell_i)}_{|\mathcal{P}_i}$ is $\Omega(1)$-far from $\mathsf{RM}^{(\ell_i)}_{|\mathcal{P}_i}$ for some $i \in [m]$.*

Indeed, this strong notion of violation allows us to use the proofs of proximity in order to reduce the query complexity to $O(1)$ queries for each $i \in [m]$. We discuss proofs of proximity next.

2.2 PCPs of proximity and composition

The second building block we use in this work is the notion of *probabilistic checkable proofs of proximity (PCPPs)*. PCPPs were first introduced in [2] and [8]. Informally speaking, a PCPP verifier for a language L, gets an oracle access to an input x and a proof π claiming that x is close to some element of L. The verifier queries x and π in some small number of (random) locations, and decides whether to accept or reject. The completeness and soundness properties of a PCPP are as follows.

Completeness: If $x \in L$, then there exists a proof causing the verifier accept with probability 1.

Soundness: If x is far from L, then no proof can make the verifier accept with probability more than $1/2$.

In fact, we will use the slightly stronger notion of *canonical PCPP (cPCPP) systems*. These are PCPP systems satisfying the following completeness and soundness properties. For completeness, we demand that for each w in the language there is a unique *canonical* proof

$\pi(w)$ that causes the verifier to accept with probability 1. For soundness, the demand is that the only pairs (x, π) that are accepted by the verifier with high probability are those where x is close to some $w \in L$ and π is close to $\pi(w)$. Such proof system have been studied in [7, 20], who proved that such proof systems exist for every language in $\mathcal{P}$.

Furthermore, for our purposes we will demand a stronger notion of correctable canonical PCPP systems (ccPCPP). These are canonical PCPP systems where the set $\{w \circ \pi^*(w) : w \in L\}$ is a q-query RLCC for some parameter q, with $\pi^*(w)$ denoting the canonical proof for $w \in L$. It was shown in [3] how to construct ccPCPP by combining a cPCPP system with *any* systematic RLCC. Informally speaking, for every $w \in L$, and its canonical proof $\pi(w)$, we define $\pi^*(w)$ by encoding $w \circ \pi(w)$ using a systematic RLCC. The verifier for the new proof system is defined in a straightforward manner. See [3] for details.

The PCPPs we use throughout this work, are the proofs of two types, certifying that

1. $f^{(\vec{x})}_{|\mathcal{P}}$ is close to $\mathsf{RM}^{(\vec{x})}_{|\mathcal{P}}$ for some plane $\mathcal{P}$ and some $\vec{x} \in \mathcal{P}$, and
2. $f^{(\ell)}_{|\mathcal{P}}$ is close to $\mathsf{RM}^{(\ell)}_{|\mathcal{P}}$ for some plane $\mathcal{P}$ and some line $\ell \subseteq \mathcal{P}$.

Indeed, it is easy to see that the first type of proofs checks that

- **(i)** the restriction of f to $\mathcal{P}$ is close to an evaluation of some polynomial Q^* of total degree at most d,
- **(ii)** and $f(\vec{x}) = Q^*(\vec{x})$.

Similarly, the second type proof certifies that

- **(i)** the restriction of f to $\mathcal{P}$ is close to an evaluation of some polynomial Q^* of total degree at most d,
- **(ii)** and $f_{|\ell}$ is close to $Q^*_{|\ell}$.

These notions of distance go together well with the guarantees we have for CTRW in Theorem 5. This allows us to *compose* CTRW with the PCPPs to obtain a correcting algorithm with query complexity $q = O(m)$. Informally speaking, the composition theorem works as follows. We first run the CTRW to obtain a collection of $m + 1$ constraints on the planes $\mathcal{P}_0, \mathcal{P}_1, \ldots, \mathcal{P}_m$. By Theorem 5, we have the guarantee that with high probability either $f^{(\vec{x})}_{|\mathcal{P}_0}$ is $\Omega(1)$-far from $\mathsf{RM}^{(\vec{x})}_{|\mathcal{P}_0}$, or $f^{(\ell_i)}_{|\mathcal{P}_i}$ is $\Omega(1)$-far from $\mathsf{RM}^{(\ell_i)}_{|\mathcal{P}_i}$ for some $i \in [m]$. Then, instead of actually reading the values of f on all these planes, we run the PCPP verifier on $f^{(\vec{x})}_{|\mathcal{P}_0}$ to check that it is close to $\mathsf{RM}^{(\vec{x})}_{|\mathcal{P}_0}$, and running the PCPP verifier on each of the $f^{(\ell_i)}_{|\mathcal{P}_i}$ to check that they are close to $\mathsf{RM}^{(\ell_i)}_{|\mathcal{P}_i}$. Each execution of the PCPP verifier makes $O(1)$ queries to f and to the proof, and thus the total query complexity will be indeed $O(m)$. As for soundness, if $f^{(\vec{x})}_{|\mathcal{P}_0}$ is $\Omega(1)$-far from $\mathsf{RM}^{(\vec{x})}_{|\mathcal{P}_0}$, or $f^{(\ell_i)}_{|\mathcal{P}_i}$ is $\Omega(1)$-far from $\mathsf{RM}^{(\ell_i)}_{|\mathcal{P}_i}$ for some $i \in [m]$, then the corresponding verifier will notice an inconsistency with constant probability, causing the decoder to output $\perp$.

A detailed discussion of proofs of proximity and the composition is available in the full version of this work [1].

2.3 Putting it all together

Below we discuss the block length and query complexity of our code.

Query complexity: For the query complexity, the mixing time of the CTRW is upper bounded by m, and for each step of the random walk, we read $O(1)$ queries from the PCPP part. Therefore, the total query complexity is bounded by $O(m)$.

Block length: As for the block length of the code, the encoding takes a message of length K and encodes it first using the Reed-Muller code of degree d over the field $\mathbb{F}$ of size $|\mathbb{F}| = n = O(d)$, where $O(\cdot)$ depends on m, so that $K = \binom{m+d}{m} > \left(\frac{d}{m}\right)^m$. In particular, the length of the Reed-Muller part is $n^m \leq O(K)$, where, again, $O(\cdot)$ hides a constant that depends on m.

The total number of predicates (of both types) defined for the CTRW is upper bounded by $B \leq 2n^m \cdot |\mathbb{H}|^{2m} \cdot n^2 = 2n^{m+4}$, as $[\mathbb{F} : \mathbb{H}] = m$, and hence $|\mathbb{H}| = n^{1/m}$. For each such predicate, we have a PCPP part of length $\text{poly}(|\mathbb{F}|) = \text{poly}(n)$.
Therefore, the total length of our code N is upper bounded by

$$N = n^m + B \cdot \text{poly}(n) = n^{m+O(1)} \ .$$

A straightforward computation reveals that this is upper bounded by $C_m \cdot K^{1+O(1/m)}$, where C_m is a constant that depends on m (but is independent of all other parameters). See the full version for the exact computation.

2.4 Comparison to the previous work

Despite the high level similarity of our work with [3], we contribute several new and crucial ideas required in order to improve the parameters of RLCCs. To demonstrate the contribution of this work, we recall the previous best-known construction of RLCCs, due to [3]. The construction in [3] consists of two parts. First, they take a message M and encode it using tensor power of Reed–Solomon code which we denote by $C^{\otimes m}$. Then, for each axis-parallel line in $C^{\otimes m}$, they append a PCPP proof asserting that restriction of the codeword to the corresponding line is (close to) a Reed-Solomon code C. The relaxed local decoding procedure for a word f and a point $\vec{x} \in C^{\otimes m}$ works by running a CTRW as follows. First, the algorithm starts by choosing an axis-parallel line ℓ_1 that passes through $\vec{x}$, and invokes a PCPP verifier on the proof for ℓ_1 to check that $f_{|\ell_1}$ is (close to) a Reed–Solomon codeword. The algorithm continues by choosing another line ℓ_2 that intersects the ℓ_1, and checks that $f_{|\ell_2}$ is (close to) a Reed–Solomon codeword. The procedure is repeated m times by sampling $\ell_3, \ell_4, \ldots, \ell_m$, where a ℓ_{i+1} intersect ℓ_i on a uniformly random point on the line. The length of the walk m is chosen so that a uniformly random point in ℓ_m is a uniform point in the tensor code. In particular, with high probability the line ℓ_m is close to the closest global codeword.

Informally, the main idea of the analysis boils down to the following. Suppose that the line ℓ_1 is far from the closest global codeword $Q^* \in C^{\otimes m}$. That is, either (1) $f_{|\ell_1}$ is far from the Reed-Solomon code or (2) it is close to some Reed-Solomon codeword but is inconsistent with the closest global codeword $Q^*_{|\ell_1}$. In particular, using the fact that Reed-Solomon code has good distance, it follows that $\text{dist}(f_{|\ell_1}, Q^*_{|\ell_1}) = \Omega(1)$. Since after m random steps we reach a line ℓ_m that is close to the closest global codeword, i.e., $\text{dist}(f_{|\ell_1}, Q^*_{|\ell_1}) = o(1)$, then it must be the case that one of the lines $\ell_1, \ell_2, \ldots, \ell_{m-1}$ is far from a Reed-Solomon, and thus the PCPP verifier rejects with high probability.

The main barrier in improving the parameters of [3] comes from the fact that ℓ_{i+1} and ℓ_i only intersect in a single point, and since we only check that the restrictions to ℓ_i's are $\Theta(1)$-close to the Reed-Solomon code, each random step chooses a non-corrupted point with some constant probability, which causes the algorithm to fail.

A natural idea to overcome this difficulty is to replace the lines $\ell_1, \ell_2, \ldots, \ell_m$ in [3] with the planes $\mathcal{P}_1, \mathcal{P}_2, \ldots, \mathcal{P}_m$, such that two consecutive planes intersect on a line. Indeed, using the fact that a random line is a good sampler it will follow that corruption in $\mathcal{P}_1$ will propagate with high probability to $\mathcal{P}_m$.

The key difficulty in implementing this idea is deciding on the collection of the planes for which we require the PCPPs. For example, if we ask for a PCPP for *all* planes in $\mathbb{F}^m$, then the block length becomes at least $\mathbb{F}^{4m}$, which is a lot more than we can afford. On the other hand, if we ask for a PCPP for all *axis-parallel* planes, then we will not be able to sample a uniformly random line in each step of the random walk, as a uniformly random line will not be axis-parallel with high probability.

We overcome this difficulty by requiring PCPPs for all $\mathbb{H}$-planes, where $\mathbb{H}$ is a carefully chosen subfield of $\mathbb{F}$, and an $\mathbb{H}$-plane is a plane of the form $\mathcal{P}_{\vec{a},\vec{h},\vec{h}'} = \{\vec{a}+t\cdot\vec{h}+s\cdot\vec{h}' : t,s \in \mathbb{F}\}$, where $\vec{a} \in \mathbb{F}^m$, and $\vec{h},\vec{h}' \in \mathbb{H}^m$ for some $\mathbb{H}$ subfield of $\mathbb{F}$.

Specifically, we choose the subfield $\mathbb{H}$ to satisfy the following properties:

1. $\mathbb{H}$ is sufficiently small, so that the total number of planes is small, which in turn reduces the block length of the code.
2. Mixing time for CTRW on the planes converges rapidly.

Note that the two requirements are conflicting, as fast mixing implies that the collection of the planes must be somewhat large. We show a subfield $\mathbb{H}$ of $\mathbb{F}$ satisfying both of these properties, thus finding a pseudo-random collection of planes which allows us to improve the parameters of the entire construction, thus obtaining the main goal of this paper. See the full version for details.

3 Concluding remarks and open problems

In this section, we first briefly discuss how to obtain a binary RLDC using the code concatenation technique. Then, we conclude the section by reviewing some open problem which we leave for future research.

3.1 Code concatenation for binary alphabet

In this paper we constructed an $O(q)$-query RLDC $C\colon \mathbb{F}^K \to \mathbb{F}^N$ with block length $N = q^{O(q^2)} \cdot K^{1+O(1/q)}$, assuming that the field is large enough, namely, assuming that $|\mathbb{F}| \geq c_q \cdot K^{1/q}$. Using standard techniques it is possible to obtain a binary RLDC with similar parameters. This can be done by concatenating our code with an arbitrary binary code with constant rate and constant relative distance. Indeed, this transformation appears in [3, Appendix A], who showed how concatenating CTRW-based RLDC over large alphabet with a good binary code gives a binary RLDC that essentially inherits the block length and the query complexity of the RLDC over large alphabet. Below we provide the proof sketch, explaining how the concatenation works.

Proof sketch. Suppose that we want to construct a short binary RLCC. Let $C_{RLCC} : \mathbb{F}^K \to \mathbb{F}^N$ be the RLCC over some field $\mathbb{F}$ with the desired block length, and let $C_{bin} : \{0,1\}^{K'} \to \{0,1\}^{N'}$ be an error-correcting code with constant rate and constant distance. We also assume that field $\mathbb{F}$ is chosen so that $|\mathbb{F}| = 2^{K'}$. (To satisfy this condition, one can simply set $\mathbb{H}$ to be a field of characteristic 2.) This assumption will allow us to have a bijection between each symbol of $\mathbb{F}$ and binary string of length K'.

We construct the binary concatenated code $C_{concat} : \{0,1\}^{K\cdot K'} \to \{0,1\}^{N\cdot N'}$ as follows. Given a message $M \in \{0,1\}^{K\cdot K'}$, we first convert it to an string in $M' \in \mathbb{F}^K$ in the natural way. Then, we encode M' using C_{RLCC} to obtain a codeword $c^* \in C_{RLCC}$. Finally, we encode each symbol of c^* using C_{bin} to get the final codeword $c \in \{0,1\}^{N\cdot N'}$.

To prove that the concatenated code is an RLCC, Chiesa, Gur, and Shinkar proved in [3, Theorem A.4] that if C_{RLCC} admits an r-steps CTRW with some soundness guarantees, then C_{concat} admits an r-steps CTRW with related soundness guarantees. The CTRW on the concatenated code C_{concat} emulates the CTRW on C_{RLCC} by sampling planes for the CTRW on the Reed-Muller code, and instead of reading the symbols from $\mathbb{F}$, it reads the binary encodings of all symbols belonging to these planes.

Indeed, it is not difficult to see that if C_{RLCC} admits an r-steps CTRW with some soundness guarantees, then so does the concatenated code. We omit the details, and refer the interested reader to Appendix A in [3]. ◀

3.2 Open problems

We conclude the paper with several open problems we leave for future research.

1. The most fundamental open problem regarding RLDCs/RLCCs is to understand the optimal trade-off between the query complexity of LDCs and their block length in the constant query regime. It is plausible that the lower bound of [12] can be improved to $K^{1+\Omega(1/q)}$, although we do not have any evidence for this.
2. As discussed in the intoduction, [2] asked whether it is possible to prove a separation between LDCs and RLDCs. Understanding the trade-off between the query complexity and the block length is one possible way to show such separation.
3. Another interesting open problem is to construct an RLDC/RLCC with constant rate and small query complexity. In particular, it is plausible that there exist polylog(N)-query RLDCs with $N = O(K)$. It should be noted that [13] constructed constant-rate RLCCs (in fact with a rate approaching 1) and query complexity $\log(N)^{O(\log\log(N))}$.
4. Also, it would be interesting to construct RLDCs/RLCCs using high-dimensional expanders [15, 6, 5, 16]. Since there are several definitions of high-dimensional expanders, it would be interesting to state the sufficient properties of high-dimensional expanders required for RLDCs. We believe this approach can be useful in constructing constant rate RLDCs with small query complexity.

References

1 Vahid R. Asadi and Igor Shinkar. Relaxed locally correctable codes with improved parameters. *Electron. Colloquium Comput. Complex.*, 2020. URL: `https://eccc.weizmann.ac.il/report/2020/142`.

2 Eli Ben-Sasson, Oded Goldreich, Prahladh Harsha, Madhu Sudan, and Salil P. Vadhan. Robust PCPs of proximity, shorter PCPs, and applications to coding. *SIAM Journal on Computing*, 36(4):889–974, 2006.

3 Alessandro Chiesa, Tom Gur, and Igor Shinkar. Relaxed locally correctable codes with nearly-linear block length and constant query complexity. In *Proceedings of the Fourteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1395–1411, 2020.

4 A. Deshpande, R. Jain, T. Kavitha, S. V. Lokam, and J. Radhakrishnan. Better lower bounds for locally decodable codes. In *Proceedings 17th IEEE Annual Conference on Computational Complexity*, pages 184–193, 2002.

5 Yotam Dikstein, Irit Dinur, Yuval Filmus, and Prahladh Harsha. Boolean Function Analysis on High-Dimensional Expanders. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques (APPROX/RANDOM 2018)*, Leibniz International Proceedings in Informatics (LIPIcs), pages 38:1–38:20, 2018.

6 I. Dinur and T. Kaufman. High dimensional expanders imply agreement expanders. In *2017 IEEE 58th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 974–985, 2017.

7 Irit Dinur, Oded Goldreich, and Tom Gur. Every set in P is strongly testable under a suitable encoding. *Electron. Colloquium Comput. Complex.*, 2018. URL: `https://eccc.weizmann.ac.il/report/2018/050`.

8 Irit Dinur and Omer Reingold. Assignment testers: Towards a combinatorial proof of the PCP theorem. In *Proceedings of the 45th Annual IEEE Symposium on Foundations of Computer Science*, FOCS 2004, pages 155–164, 2004.

9 Klim Efremenko. 3-query locally decodable codes of subexponential length. *SIAM J. Comput.*, 41(6):1694–1703, 2012.

10 O. Goldreich, H. Karloff, L. J. Schulman, and L. Trevisan. Lower bounds for linear locally decodable codes and private information retrieval. In *Proceedings 17th IEEE Annual Conference on Computational Complexity*, pages 175–183, 2002.

11 Oded Goldreich and Madhu Sudan. Locally testable codes and pcps of almost-linear length. *J. ACM*, 53(4):558–655, 2006.

12 Tom Gur and Oded Lachish. A lower bound for relaxed locally decodable codes. In *31st ACM-SIAM Symposium on Discrete Algorithms*, SODA 2020, 2020.

13 Tom Gur, Govind Ramnarayan, and Ron D. Rothblum. Relaxed locally correctable codes. In *9th Innovations in Theoretical Computer Science Conference*, ITCS '18, pages 27:1–27:11, 2018.

14 Jonathan Katz and Luca Trevisan. On the efficiency of local decoding procedures for error-correcting codes. In *Proceedings of the Thirty-Second Annual ACM Symposium on Theory of Computing*, pages 80–86, 2000.

15 Tali Kaufman and David Mass. High Dimensional Random Walks and Colorful Expansion. In *8th Innovations in Theoretical Computer Science Conference (ITCS 2017)*, Leibniz International Proceedings in Informatics (LIPIcs), pages 4:1–4:27, 2017.

16 Tali Kaufman and Izhar Oppenheim. High Order Random Walks: Beyond Spectral Gap. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques (APPROX/RANDOM 2018)*, Leibniz International Proceedings in Informatics (LIPIcs), pages 47:1–47:17, 2018.

17 Iordanis Kerenidis and Ronald de Wolf. Exponential lower bound for 2-query locally decodable codes via a quantum argument. In *Journal of Computer and System Sciences*, pages 106–115, 2003.

18 Swastik Kopparty and Shubhangi Saraf. Local testing and decoding of high-rate error-correcting codes. *Electron. Colloquium Comput. Complex.*, 2017. URL: `https://eccc.weizmann.ac.il/report/2017/126`.

19 Kenji Obata. Optimal lower bounds for 2-query locally decodable linear codes. In *Randomization and Approximation Techniques in Computer Science*, pages 39–50, 2002.

20 Orr Paradise. Smooth and strong pcps. In *Proceedings of the 11th Innovations in Theoretical Computer Science Conference*, ITCS 2020, 2020.

21 Noga Ron-Zewi and Ron Rothblum. Local proofs approaching the witness length. *Electron. Colloquium Comput. Complex.*, 2019. URL: `https://eccc.weizmann.ac.il/report/2019/127`.

22 Stephanie Wehner and Ronald de Wolf. Improved lower bounds for locally decodable codes and private information retrieval. In *Proceedings of the 32nd International Conference on Automata, Languages and Programming*, ICALP'05, pages 1424—-1436, 2005.

23 David Woodruff. New lower bounds for general locally decodable codes. *Electron. Colloquium Comput. Complex.*, 2007. URL: `https://eccc.weizmann.ac.il/report/2007/006/`.

24 David P. Woodruff. A quadratic lower bound for three-query linear locally decodable codes over any field. In *Proceedings of the 14th International Workshop on Randomized Techniques in Computation*, RANDOM 10, pages 766–779, 2010.

25 Sergey Yekhanin. Towards 3-query locally decodable codes of subexponential length. *J. ACM*, 55(1):1:1–1:16, 2008.

26 Sergey Yekhanin. Locally decodable codes. *Foundations and Trends in Theoretical Computer Science*, 6(3):139–255, 2012.

Beating Two-Thirds For Random-Order Streaming Matching

Sepehr Assadi
Department of Computer Science, Rutgers University, Piscataway, NJ, USA

Soheil Behnezhad
Department of Computer Science, University of Maryland, College Park, MD, USA

Abstract

We study the maximum matching problem in the *random-order* semi-streaming setting. In this problem, the edges of an arbitrary n-vertex graph $G = (V, E)$ arrive in a stream one by one and in a random order. The goal is to have a single pass over the stream, use $O(n \cdot \text{polylog}(n))$ space, and output a large matching of G.

We prove that for an absolute constant $\varepsilon_0 > 0$, one can find a $(2/3 + \varepsilon_0)$-approximate maximum matching of G using $O(n \log n)$ space with high probability. This breaks the natural boundary of $2/3$ for this problem prevalent in the prior work and resolves an open problem of Bernstein [ICALP'20] on whether a $(2/3 + \Omega(1))$-approximation is achievable.

2012 ACM Subject Classification Theory of computation → Graph algorithms analysis; Theory of computation → Streaming, sublinear and near linear time algorithms

Keywords and phrases Maximum Matching, Streaming, Random-Order Streaming

Digital Object Identifier 10.4230/LIPIcs.ICALP.2021.19

Category Track A: Algorithms, Complexity and Games

Related Version *Full Version*: `https://arxiv.org/abs/2102.07011`

Funding *Sepehr Assadi*: Research supported in part by the NSF CAREER award CCF-2047061 and a gift from Google Research.
Soheil Behnezhad: Research supported by Google Ph.D. Fellowship.

Acknowledgements We thank Aaron Bernstein for helpful conversations on the random-order streaming matching problem and several insightful comments that helped us in improving the presentation of the paper. We also thank the anonymous ICALP reviewers for their valuable comments.

1 Introduction

A matching in a graph $G = (V, E)$ is any collection of vertex-disjoint edges and in the maximum matching problem, we are interested in finding a matching of largest size in G. This problem has been a cornerstone of algorithmic research and its study has led to numerous breakthrough results in theoretical computer science. In this paper, we study the maximum matching problem in the *semi-streaming* model of computation [9] defined as follows.

▶ **Definition 1.** *Given a graph $G = (V, E)$ with n vertices $V = \{1, \ldots, n\}$ and m edges in E presented in a stream $S = \langle e_1, \ldots, e_m \rangle$, a semi-streaming algorithm makes a single pass over the stream of edges S and uses $O(n \cdot \text{polylog}(n))$ space, measured in words of size $\Theta(\log n)$ bits, and at the end outputs an approximate maximum matching of G.*

The greedy algorithm for maximal matching gives a simple $1/2$-approximation algorithm to this problem in $O(n)$ space. When the stream of edges is adversarially ordered, this is simply the best result known for this problem, while it is also known that a better than

$\frac{1}{1+\ln 2} \sim 0.59$-approximation is not possible [14] (see also [11,13]). Closing the gap between these upper and lower bounds is among the most longstanding open problems in the graph streaming literature.

Going beyond this "doubly worst case" scenario, namely, an adversarially-chosen graph and an adversarially-ordered stream, there has been an extensive interest in recent years in studying this problem on **random order streams**. This line of work was pioneered in [16] who showed that the $1/2$-approximation of greedy can be broken in this case and obtained an algorithm with approximation ratio $(1/2 + 0.003)$ for this problem. Since [16], there has been two main lines of attack on this problem. Firstly, [8,10,15] followed up on the approach of [16] and improved the approximation ratio all the way to $6/11$ [8]. In parallel, [1] built on the sparsification approach of [6,7] in dynamic graphs to achieve an (almost) $2/3$-approximation but at the cost of $\widetilde{O}(n^{1.5})$ space, which is no longer semi-streaming. A beautiful work of [5] then obtained a semi-streaming (almost) $2/3$-approximation by showing how a generalization of the sparsification approach in [1] can be found in $\widetilde{O}(n)$ space.

The $2/3$-approximation ratio of the algorithm of [5] is the best possible among all prior techniques for this problem: the first line of attack in [8, 10, 15, 16] is based on finding length-3 augmenting paths and even finding *all* these paths does not lead to a better-than-$2/3$-approximation[1]. The second line in [1, 5] is based on finding an edge-degree constrained subgraph (EDCS) which hits the same exact barrier as there are graphs whose EDCS does not provide a better than $2/3$-approximation (see [6]). Finally, even for an algorithmically easier variant of this problem, the one-way communication problem, which roughly corresponds to only measuring the space of the algorithm when crossing the midpoint of the stream, the best known approximation ratio is still $2/3$ which is known to be tight for adversarial orders/partitions [11].

Given this state-of-affairs, the $2/3$-approximation ratio for random-order streaming matching has emerged as natural barrier [5,15]. In particular, [5] posed obtaining a $(2/3 + \Omega(1))$-approximation to this problem as an important open question. We resolve this question in the affirmative in our work.

1.1 Our Contributions

Our main result is a semi-streaming algorithm for maximum matching in random-order streams with approximation ratio strictly-better-than-$2/3$.

▶ **Theorem 2** (Main Result). *Let G be an n-vertex graph whose edges arrive in a random-order stream. For an absolute constant $\varepsilon_0 > 0$, there is a single-pass streaming algorithm that obtains a $(\frac{2}{3} + \varepsilon_0)$-approximate maximum matching of G using $O(n \log n)$ space w.h.p.*

Theorem 2 breaks the $2/3$-barrier of all prior work in [1, 5, 8, 10, 15, 16]. Moreover, even though the improvement over $2/3$ is minuscule in this theorem (while we did not optimize for constants, the bound on ε_0 is only $\sim 10^{-14}$ at this point), it still proves that $(2/3)$-approximation is not the "right" answer to this problem. This is in contrast to some other problems of similar flavor such as one-way communication complexity of matching (on adversarial partitions) [3,11] or the fault-tolerant matching problem [3] which are both solved using similar techniques (see the unifying framework of [3] based on EDCS) and for both $2/3$-approximation is provably best possible.

[1] The work of [8] also considers length-5 augmenting paths. However, these paths are used *instead of* length-3 paths "missed" by the algorithm *not in addition to* length-3 paths and thus the same shortcoming persists.

Beyond (2/3)-approximation. Breaking this 2/3-barrier naturally raises the question on what is the right bound on the approximation ratio of random-order streaming matching. In particular, is $(1-\varepsilon)$-approximation possible? We make progress toward settling this question as well by showing that no "completely" space-efficient algorithm exists for this latter problem: there is provably no semi-streaming algorithm for the matching problem even on bipartite graphs that can achieve a $(1-\varepsilon)$-approximation in $O(2^{(1/\varepsilon)^{0.99}} \cdot n \cdot \text{polylog}n)$ space; in other words, if one hopes for achieving a $(1-\varepsilon)$-approximation, an exponential dependence on $(1/\varepsilon)$ in the space is unavoidable. As the main focus of our work is on the algorithm in Theorem 2, we provide the details of the lower bound only in the full version [2].

1.2 Overview of Techniques

Prior work. As stated earlier, there has been two main lines of attacks on the streaming matching problem in random-order streams. The first approach aims to find a *large* matching of the graph G early on in the stream, and then spends the rest of the stream *augmenting* this matching. For instance, [16] showed that in order for the greedy algorithm to fail to find a better-than-1/2-approximation, the algorithm should necessarily pick many "wrong" edges early on in the stream. As such, in instances where greedy is not beating the 1/2-approximation itself, we already have an almost 1/2-approximation by the *middle* of the stream, and we can thus focus on augmenting this matching in the remainder half to beat 1/2-approximation. The work of [15] then improved this result further by showing that a modified greedy algorithm, when unsuccessful in obtaining a large matching itself, finds an almost 1/2-approximation when only $o(1)$-fraction of the stream has passed (as opposed to middle), which gives us more room for augmentation. Finally, [8] built on this approach and further improved the augmentation phase.

The second approach to this problem was based on obtaining an EDCS, a subgraph defined by [6, 7] and studied further in [3], that acts as a "matching sparsifier". On a high level, an EDCS is a sparse subgraph satisfying the following two constraints: (i) edge-degree of edges in the EDCS cannot be "high", while (ii) edge-degree of missing edges cannot be "low". These constraints ensure that an EDCS always contains an almost 2/3-approximate matching of the graph and has additional robustness properties. For instance, [1] proved that union of several EDCS computed on different parts of a random stream, is itself an EDCS for the entire stream. This allowed them to compute an EDCS of the input in $\widetilde{O}(n^{1.5})$ space and directly obtain their almost 2/3-approximation. Finally, [5] gave an elegant proof that weakening the requirement of EDCS allows one to still preserve the almost 2/3-approximation but now recover this subgraph in only $O(n \log n)$ space. More specifically, the algorithm of [5] first finds a subgraph only satisfying property (i) of the EDCS in the first $o(m)$ edges of the stream, and then picks *all* (potentially) necessary edges for satisfying property (ii) in the remainder; the proof then shows that this set of potentially necessary edges is of size only $O(n \log n)$.

Our work. Our approach can be seen as a natural combination of these two mostly disjoint lines of work. The first part comes from a better understanding of EDCS. We present a rough characterization of when an EDCS cannot beat the 2/3-approximation, which shows that in these instances, we can effectively ignore the second constraint of EDCS. As a result, we obtain that the only way for the algorithm of [5] to fail to achieve a better-than-2/3-approximation, is if it already picks an almost 2/3-approximation in the first $o(m)$ edges. Note that this is conceptually similar to the first line of work on random-order streaming matching, but the techniques are entirely disjoint. In particular, our proof is a deterministic property of EDCS not a randomized property of a greedy algorithm on a particular ordering.

We are now in the familiar territory of having a large matching very early on in the stream, and we can spend the remainder of the stream augmenting it. The main difference however is that starting from an almost $2/3$-approximation matching, there is essentially no length-3 paths for us to augment and we instead need to handle length-5 augmenting paths. The key challenge is to find the middle edge of these length-5 augmenting paths. Indeed, we note that the $2/3$-approximation lower bound of [11] for *adversarial* order streams gives away a $2/3$-approximate matching early on for free, yet it is provably impossible to augment it in the remainder of the stream using $\widetilde{O}(n)$ space. To get around this, we crucially use the random arrival assumption again. Particularly, we regard any length-5 augmenting path whose middle edge arrives after its two endpoint edges as a "discoverable" path and then find a constant fraction of such paths. Since the edges arrive in a random order, a constant fraction of length-5 augmenting will be discoverable and thus we are able to beat $2/3$-approximation in our setting.

2 Notation and Preliminaries

General notation. For a graph $G = (V, E)$ and $v \in V$, we use $\deg_G(v)$ to denote the degree of v in G and $N_G(v)$ to denote the neighborset of v (when clear from the context, we may drop the subscript G). For any edge $e = (u, v) \in E$, we define the edge-degree of e in G as $\deg(u) + \deg(v)$. We use $\mu(G)$ to denote the *size* (i.e., the number of edges) of the maximum matching in G.

For integer $k \geq 1$ and $p \in [0, 1]$, we use $\mathcal{B}(k, p)$ to denote the *binomial distribution* with parameters k and p. That is, $\mathcal{B}(k, p)$ is the discrete probability distribution of the number of successful experiments out of k independent experiments each with probability p of success.

Random-order streams. We consider the random-order streaming setting where the edges of G arrive one by one in an order chosen uniformly at random from all possible orderings. Let e_i be the i-th edge that arrives in the stream. For any two parameters a, b satisfying $1 \leq a < b \leq m$ we use $G[a, b]$ to denote the subgraph of G on vertex-set V and edge-set $\{e_a, \ldots, e_b\}$. We may also use $G_{<a}$ and $G_{\geq a}$ respectively for $G[1, a-1]$ and $G[a, m]$.

For the input graph G defined by the stream, we can assume w.l.o.g. that $\mu(G) \geq c \log n$ for any desirably large constant c. The reason is that any graph can be easily shown to have at most $2n \cdot \mu(G)$ edges and if $\mu(G) = O(\log n)$ then we can store the whole input in the memory and report an optimal solution using $O(n \log n)$ space. We further assume throughout the paper that the number of edges m is known by the algorithm in advance. This is a common assumption in the literature and can be removed via standard techniques by guessing m in geometrically increasing values at the expense of multiplying the space by an $O(\log n)$ factor.

2.1 Preliminaries

Hall's theorem. We use the following standard extension of the Hall's marriage theorem for characterizing maximum matching size in bipartite graphs.

▶ **Fact 3** (Extended Hall's Theorem [12]). *Let $G = (L, R, E)$ be a bipartite graph with $|L| = |R| = n$. Then, $\max\Big(|A| - |N(A)|\Big) = n - \mu(G)$, where A ranges over L or R, separately. We refer to such set A as a* ***witness set****.*

Fact 3 follows from Tutte-Berge formula for matching size in general graphs [4, 17] or a simple extension of the proof of Hall's marriage theorem itself.[2]

Alternating and augmenting paths. Given a matching M, an *alternating path* P for M is a path whose edges alternatively belong to M and do not belong to M. An *augmenting path* for M is an alternative path that starts and ends with edges that do not belong to M. Given an augmenting path P for M, we use notation $M \oplus P := (M \setminus P) \cup (P \setminus M)$ to denote the matching obtained by flipping the containment of edges of P in M. Given two matchings M and M', their *symmetric difference* $M \Delta M'$ is a graph including only the edges that belong to exactly one of M and M'.

2.2 Bernstein's Algorithm

We briefly review the parameters and guarantees of the algorithm of Bernstein [5] that we need. In the following, we slightly increase the constants in the parameters which is needed for our results.

▶ **Definition 4 (Parameters).** *For some small $\varepsilon \in (0, \frac{1}{2})$ to be determined later, let*

$$\lambda := \frac{\varepsilon}{128}, \qquad \beta_+ := 64 \cdot \lambda^{-2} \log(1/\lambda), \qquad \beta_- = (1-\lambda) \cdot \beta_+.$$

A high level overview of the algorithm of [5] is as follows:

Algorithm 1 Bernstein's Algorithm [5].

The algorithm of [5] proceeds in two phases as follows:

- Phase I terminates within the first εm edges of the stream. At the end of Phase I, the algorithm constructs a subgraph $H \subseteq G_{<\varepsilon m}$ (using the algorithm of Lemma 5) that in particular guarantees that for all $(u, v) \in H$, $\deg_H(u) + \deg_H(v) \leq \beta_+$. Also let U be the set of *all* edges in $G_{\geq \varepsilon m}$ such that $\deg_H(u) + \deg_H(v) < \beta_-$.
- In Phase II, the algorithm simply stores U in the memory and at the end of the stream returns a maximum matching of $H \cup U$.

The following lemma is all we need from [5] in our paper.

▶ **Lemma 5** (See Lemma 4.1 of [5]). *There is a way of constructing the subgraph H of $G_{<\varepsilon m}$ such that with probability at least $1 - n^{-3}$, $|H \cup U| = O(n \log(n) \cdot \text{poly}(1/\varepsilon))$.*

3 Finding an Almost (2/3)-Approximation Early On

We start by characterizing the tight instances of the algorithm of [5] (Algorithm 1). Roughly speaking, we show that the only way for Algorithm 1 to end up with a (2/3)-approximation is if in its Phase I it computes a subgraph H that already has an almost (2/3)-approximate matching. This will then be used by our algorithm in the next section to obtain a strictly better-than-(2/3)-approximation by augmenting this already-large matching. We start by presenting and proving this result for bipartite graphs which is the main part of the proof; we then extend the result to general graphs (with no considerable loss of parameters for our purpose) using the probabilistic method approach of [3] for the original EDCS.

[2] Simply add $n - \mu(G)$ vertices to each side of the graph and connect them to all the original vertices; then apply original's Hall's theorem for perfect matching to this graph as this graph now has one.

3.1 Bipartite Graphs

In this section we prove the following structural result:

▶ **Theorem 6.** *Let $\lambda \in (0, 1/2)$ and $\beta_- \leq \beta_+$ be such that $\beta_+ \geq \frac{10}{\lambda}$ and $\beta_- \geq (1-\lambda)\beta_+$. Suppose $G = (L, R, E)$ is any bipartite graph and:*

(i) *H is a subgraph of G where for all $(u, v) \in H$: $\deg_H(u) + \deg_H(v) \leq \beta_+$; and*

(ii) *U is the set of all edges (u, v) in $G \setminus H$ such that $\deg_H(u) + \deg_H(v) < \beta_-$.*

Then, for any parameter $\delta \in (0, 1)$, either:

$$\mu(H) \geq (1-4\lambda) \cdot \left(\frac{2}{3} - \delta\right) \cdot \mu(G) \quad \text{or} \quad \mu(H \cup U) \geq (1-2\lambda) \cdot \left(\frac{2}{3} + \frac{\delta^2}{18}\right) \cdot \mu(G).$$

Let us define the following (see Figure 1 for an illustration):

- Let M^* be a maximum matching of G and define $M^*_U := M^* \cap U$ and $M^*_{\bar{U}} := M^* \setminus U$.
- A is a Hall's theorem witness set in $H \cup M^*_U$ (as in Fact 3) and $B := N_{H \cup M^*_U}(A)$. Without loss of generality we assume $A \subseteq L$ and define $\bar{A} := L \setminus A$ and $\bar{B} := R \setminus B$.

We start with the following simple claim that follows easily from Fact 3. See the full version of the paper for details [2].

▷ Claim 7. For the witness set A:

(i) $|\bar{A}| + |B| \leq \mu(H \cup U)$.

(ii) There is a matching $\bar{M} \subseteq M^*_{\bar{U}}$ between A and $\bar{B}$ in G with size $|\bar{M}| = \mu(G) - \mu(H \cup M^*_U)$.

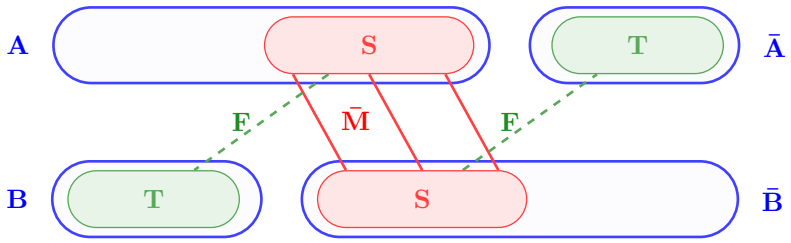


Figure 1 An illustration of the Hall's witness set and our notation in the proof of Theorem 6. Note that there are no edges between A and $\bar{B}$ in $H \cup M^*_U$, and matching $\bar{M}$ belongs entirely to $M^*_{\bar{U}}$.

Consider any edge $(u, v) \in \bar{M}$ defined in Claim 7. As $\bar{M} \subseteq M^*_{\bar{U}}$, by property (ii) of the statement of Theorem 6, we have, $\deg_H(u) + \deg_H(v) \geq \beta_-$. We arbitrarily remove the edges on u and v until the above inequality becomes tight for every edge. We let F be the remaining edges. Note that any edge in F is incident on exactly one vertex of $\bar{M}$ as there are no edges in $H \cup M^*_U$ between the endpoints of $\bar{M}$. We record these properties as follows:

$$\forall (u, v) \in \bar{M} : \ \deg_F(u) + \deg_F(v) = \beta_- \quad \text{and} \quad |F| = |\bar{M}| \cdot \beta_-. \tag{1}$$

We refer the reader to the full version of the paper [2] for illustrative examples that highlight the ideas for proving Theorem 6. Here for space constraints, we only provide the formal proof.

In Lemma 8, we prove a lower bound on $\mu(H)$. This lemma can then be used as follows: if the degrees of most edges in $\bar{M}$ are "balanced", i.e., both endpoints have degree $\approx \beta_-/2$, then $\mu(H)$ will already be of size $2 \cdot |\bar{M}|$ which is sufficient for the first condition of Theorem 6.

▶ **Lemma 8** (matching of H is large). *We have* $\mu(H) \geq \frac{\beta_-}{1+4\lambda} \cdot \sum_{(u,v)\in \bar{M}} \frac{1}{\max\{\deg_F(u)\,,\,\deg_F(v)\}}$.

Proof. For every edge $(u,v) \in \bar{M}$, define $F(u,v)$ as set of edges in F that are incident on u or v. We define the following fractional matching $x \in \mathbb{R}^F$ on edges of F:

- for any edge $e \in F(u,v)$: set $x_e := \frac{1}{1+4\lambda} \cdot \frac{1}{\max\{\deg_F(u)\,,\,\deg_F(v)\}}$.

Let us now prove this is indeed a valid fractional matching. For any vertex w matched by $\bar{M}$,

$$x_w := \sum_{e \ni w} x_e \leq \deg_F(w) \cdot \frac{1}{1+4\lambda} \cdot \frac{1}{\deg_F(w)} < 1,$$

thus satisfying the fractional matching constraint.

Now fix a vertex w not matched by $\bar{M}$. Let $u_1, \ldots, u_{\deg_F(w)}$ denote the neighbors of w in F. By definition, all these vertices are matched by $\bar{M}$. Let $v_1, \ldots, v_{\deg_F(w)}$ be the matched pairs of these vertices. We need the following simple claim proved in the full version.

▷ Claim 9. For every $i \in [\deg_F(w)]$, $\deg_F(w) \leq (1+4\lambda) \cdot \max\{\deg_F(u_i)\,,\,\deg_F(v_i)\}$.

To finalize Lemma 8, for any vertex w not matched by $\bar{M}$, we have,

$$x_w := \sum_{e=(w,u_i)} x_e = \sum_{u_i} \frac{1}{1+4\lambda} \cdot \frac{1}{\max\{\deg_F(u_i)\,,\,\deg_F(v_i)\}} \underset{\text{Claim 9}}{\leq} \sum_{u_i} \frac{1}{\deg_F(w)} = 1,$$

thus satisfying the fractional matching constraint. This implies that x is a valid fractional matching. Finally, the value of this fractional matching is:

$$\begin{aligned}\sum_{e\in F} x_e &= \sum_{(u,v)\in N} \sum_{e\in F(u,v)} x_e = \sum_{(u,v)\in N} \frac{\deg_F(u) + \deg_F(v)}{(1+4\lambda)\cdot \max\{\deg_F(u)\,,\,\deg_F(v)\}} \\ &= \frac{\beta_-}{1+4\lambda} \cdot \sum_{(u,v)\in N} \frac{1}{\max\{\deg_F(u)\,,\,\deg_F(v)\}},\end{aligned}$$

where the last equation is by Eq (1). As the integrality gap of the matching polytope on bipartite graphs is one, we obtain the desired lower bound on $\mu(H)$. ◀

We now prove that if on the other hand most edges of $\bar{M}$ are "unbalanced", then $\mu(H \cup U)$ should be sufficiently large. To continue, we need a quick definition. Let S denote the endpoints of the matching $\bar{M}$ and T be the neighborset of these vertices in F. Recall that by Eq (1), S and T are disjoint (see Figure 1).

▶ **Lemma 10** (matching of $\mu(H \cup U)$ is large). *We have* $\mu(H \cup U) \geq \frac{|\bar{M}|^2 \cdot {\beta_-}^2}{|\bar{M}| \cdot \beta_- \cdot \beta_+ - \sum_{s\in S} (\deg_F(s))^2}$.

Proof. Since $F \subseteq H$, by property (i) of Theorem 6, we have that

$$|F| \cdot \beta_+ \geq \sum_{(u,v)\in F} \deg_F(u) + \deg_F(v) = \sum_{s\in S} (\deg_F(s))^2 + \sum_{t\in T} (\deg_F(t))^2. \tag{2}$$

We can lower bound the second term of the RHS as follows. Recall that sum of quadratics is minimized over all-equal terms. As $\sum_{t\in T} \deg_F(t) = |F|$, this implies that,

$$\sum_{t\in T} (\deg_F(t))^2 \geq \sum_{t\in T} \left(\frac{|F|}{|T|}\right)^2 = |T| \cdot \left(\frac{|F|}{|T|}\right)^2 = \frac{|F|^2}{|T|}.$$

By plugging in this bound in Eq (2) and moving the terms around, we have that

$$|T| \geq \frac{|F|^2}{|F| \cdot \beta_+ - \sum_s (\deg_F(s))^2} = \frac{|\bar{M}|^2 \cdot {\beta_-}^2}{|\bar{M}| \cdot \beta_- \cdot \beta_+ - \sum_s (\deg_F(s))^2}. \qquad \text{(as } |F| = |\bar{M}| \cdot \beta_- \text{ by Eq (1))}$$

Finally, $T \subseteq \bar{A} \cup B$ (as there are no edges between A and $\bar{B}$) and thus by Claim 7, $|T| \leq \mu(H \cup U)$ which finalizes the proof. ◀

Lemma 10 can be used as follows: when degree of most edges in $\bar{M}$ are "balanced", the quantity $\sum_s (\deg_F(s))^2$ will be close to $|\bar{M}| \cdot (\beta_-)^2/2$ which implies that $\mu(H \cup U)$ will be almost $2 \cdot |\bar{M}|$; however, when degrees of edges in $\bar{M}$ are "unbalanced", the quantity $\sum_s (\deg_F(s))^2$ *cannot* decrease all the way to $|\bar{M}| \cdot (\beta_-)^2/2$ and thus we can get a higher lower bound on the value of $\mu(H \cup U)$ which breaks the (2/3)-approximation.

To finalize the proof of Theorem 6, we need the following claim for lower bounding $\sum_{s \in S} (\deg_F(s))^2$ in the RHS of Lemma 10, in the cases where RHS of Lemma 8 is small.

▷ Claim 11. Suppose $\sum_{(u,v) \in \bar{M}} \frac{\beta_-}{\max\{\deg_F(u), \deg_F(v)\}} = (2-\gamma) \cdot |\bar{M}|$ for some $\gamma \in [0,1)$; then $\sum_s (\deg_F(s))^2 \geq |\bar{M}| \cdot \left(\frac{(2+\gamma^2-2\gamma) \cdot {\beta_-}^2}{4+\gamma^2-4\gamma}\right)$.

The proof of Claim 11 is given in the full version [2]. We are now ready to prove Theorem 6.

Proof of Theorem 6. Let us pick $\gamma \in [0,1)$ such that $\sum_{(u,v) \in \bar{M}} \frac{\beta_-}{\max\{\deg_F(u), \deg_F(v)\}} = (2-\gamma) \cdot |\bar{M}|$ (as the max-term is at least $\beta_-/2$, such a γ always exist). By plugging in the bound of Claim 11 in Lemma 10, we have that,

$$\begin{aligned}
\mu(H \cup U) &\geq \frac{|\bar{M}|^2 \cdot {\beta_-}^2}{|\bar{M}| \cdot \beta_- \cdot \beta_+ - |\bar{M}| \cdot \left(\frac{(2+\gamma^2-2\gamma) \cdot {\beta_-}^2}{4+\gamma^2-4\gamma}\right)} \\
&\geq (1-2\lambda) \cdot |\bar{M}| \cdot \frac{1}{1 - \left(\frac{(2+\gamma^2-2\gamma)}{4+\gamma^2-4\gamma}\right)} && \text{(as } \beta_- \geq (1-\lambda)\beta_+\text{)} \\
&= (1-2\lambda) \cdot |\bar{M}| \cdot \frac{4+\gamma^2-4\gamma}{2-2\gamma} = (1-2\lambda) \cdot |\bar{M}| \cdot (2 + \frac{\gamma^2}{2-2\gamma}).
\end{aligned}$$

Considering $|\bar{M}| \geq \mu(G) - \mu(H \cup U)$ by Claim 7, we obtain that

$$\mu(H \cup U) \geq (1-2\lambda) \cdot \mu(G) \cdot \left(\frac{2}{3} + \frac{\gamma^2}{18 - 18\gamma + 3\gamma^2}\right) \geq (1-2\lambda) \cdot \mu(G) \cdot \left(\frac{2}{3} + \frac{\gamma^2}{18}\right).$$

Now if for the parameter δ in Theorem 6, we already have $\gamma \geq \delta$, we will obtain the second condition. Further, without loss of generality, we can assume that $|\bar{M}| \geq (\frac{1}{3} - \frac{\delta}{3}) \cdot \mu(G)$ as otherwise $\mu(H \cup M^*_U) \geq (\frac{2}{3} + \delta) \cdot \mu(G)$ by Claim 7 which is stronger than the second condition of Theorem 6.

Suppose $\gamma < \delta$ and $|\bar{M}| \geq (\frac{1}{3} - \frac{\delta}{3}) \cdot \mu(G)$ then. In this case, by the definition of γ and Lemma 8,

$$\begin{aligned}
\mu(H) &\geq \frac{1}{1+4\lambda} \cdot (2-\gamma) \cdot |\bar{M}| \geq \frac{1}{1+4\lambda} \cdot (2-\delta) \cdot \left(\frac{1}{3} - \frac{\delta}{3}\right) \cdot \mu(G) \\
&\geq (1-4\lambda) \cdot \left(\frac{2}{3} - \delta\right) \cdot \mu(G),
\end{aligned}$$

thus satisfying the first condition. This concludes the proof. ◀

We can also extend the guarantee of Theorem 6 to general (non-bipartite) graphs following the probabilistic method technique of [3] (see the full version [2] for the exact guarantee) without incurring any loss to the guarantees up to constants.

4 An Improved Algorithm via Augmentation

In this section, we show that the maximum matching of the subgraph H constructed in the early part of the stream of Algorithm 1 can be augmented well via the remaining edges. Combined with our guarantee of Section 3, we complete in this section the proof of Theorem 2. Namely, we show that for some parameter $\varepsilon_0 > 0$, there is a single-pass random-order streaming algorithm (formalized as Algorithm 2) that obtains a $(\frac{2}{3} + \varepsilon_0)$-approximate maximum matching of G using $O(n \log n)$ space with high probability of $1 - 1/\text{poly}(n)$.

4.1 The Algorithm

Our starting point is Algorithm 1. Recall that this algorithm stores two subgraphs H and U of G of size $O(n \log n)$. Subgraph H is constructed early on, after merely observing εm edges of the stream. In addition to H and U, here we store an additional subset of edges that we use to augment a matching of H with. Particularly, let M_H be an arbitrary maximum matching of H. Having matching M_H early on, in our algorithm we augment M_H using the edges that arrive in the rest of the stream (i.e., Phase II) in parallel to storing U. The augmenting paths that we find may be of size up to *five*. This is crucial since we may not have enough augmenting paths of length smaller than five to go beyond (2/3)-approximation. Now by plugging our bound of Section 3, it can be shown that either $H \cup U$ includes our desired approximation of strictly better that 2/3, or M_H is almost a (2/3)-approximate matching which coupled with the augmenting paths that we find for it in Phase II leads to our better-than-(2/3)-approximation.

To find these augmenting paths, we divide the $(1-\varepsilon)m$ edges of Phase II into Phase II.A and Phase II.B. To do this, we first draw a random variable $\tau \sim \mathcal{B}((1-\varepsilon)m, \gamma)$. Phase II.A will then proceed on the edges that arrive up to the τ-th edge of Phase II and Phase II.B proceeds on the rest of the edges. Drawing random variable τ (instead of having a fixed threshold) is particularly useful in the analysis: Conditioned on the edges that are to arrive in Phase II (but not their ordering), each edge now belongs to Phase II.A *independently* with probability γ and to Phase II.B otherwise. Note that with a fixed threshold, we do not get this independence.

For Phase II.A, let us define G_H to be the subgraph of G whose edges arrive in Phase II.A and have exactly one endpoint matched by M_H. Note that G_H is bipartite (even though G may not be) with one partition corresponding to vertices $V(M_H)$ and another to $V \setminus V(M_H)$. In Phase II.A, we only consider the edges of G_H and greedily construct a maximal $(2, b)$-matching T of G_H (for some constant $b \geq 2$). It is the vertices in part $V(M_H)$ of G_H that have maximum degree 2 in T and those in the other partition can have degree up to b. In our analysis, we show that the edges of T can be used as the two endpoint edges of many augmenting paths of length three or five for M_H (see Figure 2).

In Phase II.B, we first let $M \leftarrow M_H$ and upon arrival of each edge e, we iteratively augment M via length-up-to-five augmenting paths using the edges in $T \cup \{e\}$ until no such path is left. In our analysis, we use the edges of Phase II.B either as the middle edge of length-five augmenting paths or as the single edge of the length-one augmenting paths the algorithm may find (see Figure 2).

At the end of the stream, we return a maximum matching of $M \cup H \cup U$. The algorithm outlined above is formalized as Algorithm 2.

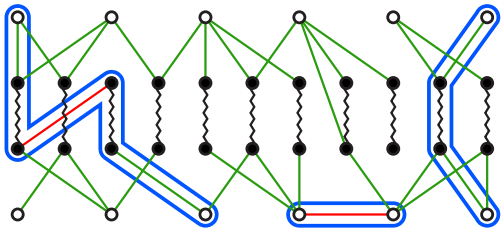


Figure 2 An example of an execution of Algorithm 2. Here the black zig-zagged edges denote M_H which is fixed by the end of Phase I and we would like to augment it. The black nodes are those matched by M_H and the white ones are those left unmatched by M_H. The edges between white and black nodes (colored green) are the edges in T. Each black node has at most two edges in T and the green nodes can have up to b. The red edges are those that arrive in Phase II.B. Three augmenting paths of length one, three, and five that are discoverable by the algorithm are highlighted by blue.

Algorithm 2 Our final random-order streaming matching algorithm with approximation ratio strictly-better-than-$2/3$.

Parameters: $\gamma = 2/3$, $b = 500$, and a sufficiently small constant $\varepsilon < 0.01$ to be fixed later.

(1) In Phase I of the algorithm, which consists of the first εm edges of the stream, we construct a subgraph H of G as in Phase I of Algorithm 1. At the end of Phase I, we fix an arbitrary maximum matching M_H of H.

(2) In Phase II, which includes all the edges that arrive after Phase II, we store subgraph U using Phase II of Algorithm 1. In addition, we store another subset of edges that we use to augment M_H. These edges are constructed in two sub-phases Phase II.A and Phase II.B.

(3) Draw random variable τ from the Binomial distribution $\mathcal{B}((1-\varepsilon)m, \gamma)$. Note that this can be done in $O(m)$ time and $O(1)$ space as we only need to count the successes.

(4) Phase II.A starts after Phase I and ends upon arrival of the τ'th edge of Phase II.

- **a.** Let $G_H(V_H, U_H, E_H)$ be a bipartite subgraph of G where $V_H := V(M_H)$ is the set of vertices matched in M_H, $U_H := V \setminus V(M_H)$ is the set of vertices left unmatched in M_H, and E_H is the edges of G between V_H and U_H that arrive in Phase II.A.
- **b.** We initialize $T \leftarrow \emptyset$ and upon arrival of an edge $e = (u, v)$ of G_H with $u \in U_H$ and $v \in V_H$, if $\deg_T(v) < 2$ and $\deg_T(u) < b$ we add e to T. That is, T is a maximal $(2, b)$-matching of G_H which requires $O(nb)$ space to store.

(5) Phase II.B starts after Phase II.A and continues to the end of the stream:

- **a.** $M \leftarrow M_H$. Upon arrival of each edge e in Phase II.B, we iteratively take an arbitrary augmenting path P for M of length up to five using the edges in $M \cup T \cup \{e\}$ and let $M \leftarrow M \oplus P$. We repeat this process until no more augmenting paths of length up to five exist in $M \cup T \cup \{e\}$; we then continue to the next edge of the stream in Phase II.B.

(6) Finally, we return a maximum matching of $M \cup H \cup U$.

Analysis of Algorithm 2

It is straightforward to verify that Algorithm 2 uses $O(|H| + |U| + nb)$ space which by Lemma 5 is $O(n \log n)$ w.h.p. Here we analyze the approximation ratio of the algorithm.

Let $M^\star$ be an arbitrary maximum matching of $G_{\geq \varepsilon m}$. Fixing an arbitrary maximum matching of G, each of its edges appears in $G_{\geq \varepsilon m}$ with probability $(1-\varepsilon)$, thus $\mathbf{E}|M^\star| \geq (1-\varepsilon)\mu(G)$. Now so long as $\mu(G) \geq 20\log(n)\varepsilon^{-2}$ and $\varepsilon < 1/2$ (which we can assume to hold as discussed in Section 2.1), we can prove a high probability lower bound on the size of $M^\star$ via a Chernoff bound on negatively associated random variables. See, e.g., [5, Lemma 2.2] for the proof of the following:

▶ **Observation 12.** *If $\mu(G) \geq 20\log(n)\varepsilon^{-2}$ and $\varepsilon < 1/2$, $\mathbf{Pr}[|M^\star| \geq (1-2\varepsilon)\mu(G)] \geq 1 - n^{-5}$.*

From now on, we condition on $G_{<\varepsilon m}$ which fixes subgraph H and matching $M^\star$. We only assume that $G_{<\varepsilon m}$ is chosen such that the high probability event of Observation 12 holds.

▶ **Assumption 13.** $|M^\star| \geq (1-2\varepsilon)\mu(G)$.

Other than Assumption 13, we do not need any other assumption on how $G_{<\varepsilon m}$ is chosen for the rest of the analysis of the approximation ratio.[3] By conditioning on the outcome of Phase I, the only randomization that will be left, is the order with which the edges of $G_{\geq \varepsilon m}$ arrive in the stream. For brevity, we do not explicitly write the conditioning on $G_{<\varepsilon m}$ for the rest of the section, but it should be noted that **all random statements are conditioned on the outcome of Phase I**.

Let $\mathcal{P}$ be the set of all augmenting paths of M_H in $S := M^\star \Delta M_H$ with length at most five. Note that since we regard H (and thus M_H) as given, the set $\mathcal{P}$ is deterministic (as it only depends on M_H and $M^\star$ and not on the order of edges in $G_{\geq \varepsilon m}$).

▶ **Observation 14.** *We have $|\mathcal{P}| \geq |M^\star| - \frac{4}{3} \cdot \mu(H)$.*

We use $G_{\text{II.A}}$ to denote the subgraph of G that arrives in Phase II.A and use $G_{\text{II.B}}$ to denote the subgraph of G that arrives in Phase II.B.

▶ **Definition 15.** *We say an augmenting path $P \in \mathcal{P}$ is "lucky" if the following holds:*

1. *If $P = \langle e_1 \rangle$ then $e_1 \in G_{II.B}$.*
2. *If $P = \langle e_1, e_2, e_3 \rangle$ then $e_1, e_3 \in G_{II.A}$.*
3. *If $P = \langle e_1, e_2, e_3, e_4, e_5 \rangle$ then $e_1, e_5 \in G_{II.A}$ and $e_3 \in G_{II.B}$.*

We denote the set of lucky augmenting paths in $\mathcal{P}$ by $\mathcal{P}_L$.

Note that the subset $\mathcal{P}_L$ of $\mathcal{P}$ is now random since it depends on the order of edges in $G_{\geq \varepsilon m}$. Lemma 16 below proves that a relatively large fraction of augmenting paths in $\mathcal{P}$ will turn out to be lucky with high probability. The proof is straightforward; see [2].

▶ **Lemma 16.** *It holds that* $\mathbf{Pr}\left(|\mathcal{P}_L| \leq \gamma^2(1-\gamma)|\mathcal{P}| - \sqrt{15\mu(G)\ln n}\right) \leq 2n^{-5}$.

Next, observe that in Phase II.B of Algorithm 2 where we iteratively discover augmenting paths, we do not have the whole subgraph $G_{\text{II.A}}$ and have stored only a subgraph T of $G_{\text{II.A}}$ in the memory. In addition, when finding augmenting paths we use only the current edge e of $G_{\text{II.B}}$ in Algorithm 2. Therefore, not all lucky paths are actually discoverable by Algorithm 2. This motivates our next definition for "discoverable paths".

[3] We note, however, that the randomization in $G_{<\varepsilon m}$ is crucial for arguing that the algorithm uses $O(n \log n)$ space. Here, however, we are only analyzing the approximation ratio.

▶ **Definition 17.** *We say an augmenting path P (not necessarily in $\mathcal{P}$) for M_H is "discoverable" if $|P| \leq 5$, all edges of P are in $M_H \cup T \cup G_{II.B}$, and P has ≤ 1 edge in $G_{II.B}$.*

The next lemma proves there are many vertex-disjoint discoverable augmenting paths, by relating them to the number of lucky augmenting paths $|\mathcal{P}_L|$.

▶ **Lemma 18.** *There is a set $\mathcal{Q}$ of vertex-disjoint discoverable augmenting paths of M_H with*

$$|\mathcal{Q}| \geq \frac{1}{2b+3}\left(|\mathcal{P}_L| - \frac{4}{b}\cdot\mu(H)\right).$$

Observe that $\mathcal{Q}$ is only a set of vertex-disjoint discoverable augmenting paths. However, since Algorithm 2 applies augmenting paths greedily and in an arbitrary order, the set of applied augmenting paths may be very different from $\mathcal{Q}$. The next claim shows that we can nonetheless relate the number of augmenting paths that Algorithm 2 applies to the size of $\mathcal{Q}$.

▷ Claim 19. Let $\mathcal{Q}$ be as in Lemma 18. Algorithm 2 applies at least $|\mathcal{Q}|/6$ augmenting paths in Phase II.B. In other words, $|M| \geq \mu(H) + \frac{1}{6}|\mathcal{Q}|$.

Next, we show that the output of Algorithm 2 is strictly larger than $\mu(H)$.

▶ **Lemma 20.** *There is an absolute constant $\varepsilon_0' > 0$ such that for any $\varepsilon < 0.01$, if $\mu(H) \leq 0.68\mu(G)$ then with probability $1 - 1/\text{poly}(n)$, we have $|M| \geq \mu(H) + \varepsilon_0' \cdot \mu(G)$.*

Proof. We have

$$|M| \overset{\text{Claim 19}}{\geq} \mu(H) + \frac{1}{6}|\mathcal{Q}| \overset{\text{Lemma 18}}{\geq} \mu(H) + \frac{|\mathcal{P}_L| - \frac{4}{b}\mu(H)}{6(2b+3)}. \tag{3}$$

On the other hand, by Lemma 16 we know that with $1 - 1/\text{poly}(n)$ probability,

$$\begin{aligned}
|\mathcal{P}_L| &> \gamma^2(1-\gamma)|\mathcal{P}| - \sqrt{15\mu(G)\ln n} && \text{(By Lemma 16)}\\
&= \frac{4}{27}|\mathcal{P}| - \sqrt{15\mu(G)\ln n} && \text{(Since } \gamma = 2/3\text{)}\\
&\geq \frac{4}{27}\left(|M^\star| - \frac{4}{3}\mu(H)\right) - \sqrt{15\mu(G)\ln n} && \text{(By Observation 14)}\\
&\geq \frac{4}{27}\left((1-2\varepsilon)\mu(G) - \frac{4}{3}\mu(H)\right) - \sqrt{15\mu(G)\ln n} && \text{(By Assumption 13)}\\
&> 0.0108\mu(G) - \sqrt{15\mu(G)\ln n} && (\varepsilon < 0.01 \text{ and } \mu(H) \leq 0.68\mu(G))\\
&> 0.01\mu(G). && \text{(Since } \mu(G) > c\log n \text{ for any desirably large constant } c.\text{)}
\end{aligned}$$

Replacing this high probability lower bound for $|\mathcal{P}_L|$ into (3) we get that w.h.p.,

$$\begin{aligned}
|M| &\geq \mu(H) + \frac{0.01\mu(G) - \frac{4}{b}\mu(H)}{6(2b+3)}\\
&> \mu(H) + 10^{-7}\mu(G). && \text{(Replacing } b = 500 \text{ and noting } \mu(H) \leq 0.68\mu(G)\text{.)}
\end{aligned}$$

This completes the proof. ◀

Lemma 20 states that the output of Algorithm 2 is strictly larger than $\mu(H)$. Moreover, the guarantee of Section 3 implies that $\mu(H)$ must be at least (almost) $\frac{2}{3}\mu(G)$ (or otherwise $\mu(H \cup U)$ is strictly larger than $2/3\mu(G)$). The combination of these implies the output of Algorithm 2 is at least $(2/3 + \Omega(1))\mu(G)$ proving Theorem 2.

References

1 Sepehr Assadi, MohammadHossein Bateni, Aaron Bernstein, Vahab S. Mirrokni, and Cliff Stein. Coresets meet EDCS: algorithms for matching and vertex cover on massive graphs. In *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2019, San Diego, California, USA, January 6-9, 2019*, pages 1616–1635, 2019.

2 Sepehr Assadi and Soheil Behnezhad. Beating two-thirds for random-order streaming matching. *CoRR*, abs/2102.07011, 2021. `arXiv:2102.07011`.

3 Sepehr Assadi and Aaron Bernstein. Towards a unified theory of sparsification for matching problems. In *2nd Symposium on Simplicity in Algorithms, SOSA@SODA 2019, January 8-9, 2019 - San Diego, CA, USA*, pages 11:1–11:20, 2019.

4 Claude Berge. *The theory of graphs*. Courier Corporation, 1962.

5 Aaron Bernstein. Improved bounds for matching in random-order streams. In *47th International Colloquium on Automata, Languages, and Programming, ICALP 2020, July 8-11, 2020, Saarbrücken, Germany (Virtual Conference)*, pages 12:1–12:13, 2020.

6 Aaron Bernstein and Cliff Stein. Fully dynamic matching in bipartite graphs. In *Automata, Languages, and Programming - 42nd International Colloquium, ICALP 2015, July 6-10, 2015, Proceedings, Part I*, pages 167–179, 2015.

7 Aaron Bernstein and Cliff Stein. Faster fully dynamic matchings with small approximation ratios. In *Proceedings of the Twenty-Seventh Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2016, January 10-12, 2016*, pages 692–711, 2016.

8 Alireza Farhadi, Mohammad Taghi Hajiaghayi, Tung Mai, Anup Rao, and Ryan A. Rossi. Approximate maximum matching in random streams. In *Proceedings of the 2020 ACM-SIAM Symposium on Discrete Algorithms, SODA 2020, Salt Lake City, UT, USA, January 5-8, 2020*, pages 1773–1785, 2020.

9 Joan Feigenbaum, Sampath Kannan, Andrew McGregor, Siddharth Suri, and Jian Zhang. On graph problems in a semi-streaming model. *Theor. Comput. Sci.*, 348(2-3):207–216, 2005.

10 Buddhima Gamlath, Sagar Kale, Slobodan Mitrovic, and Ola Svensson. Weighted matchings via unweighted augmentations. In *Proceedings of the 2019 ACM Symposium on Principles of Distributed Computing, PODC 2019, Toronto, ON, Canada, July 29 - August 2, 2019*, pages 491–500, 2019.

11 Ashish Goel, Michael Kapralov, and Sanjeev Khanna. On the communication and streaming complexity of maximum bipartite matching. In *Proceedings of the Twenty-third Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '12, pages 468–485. SIAM, 2012. URL: `http://dl.acm.org/citation.cfm?id=2095116.2095157`.

12 Philip Hall. On representatives of subsets. *Journal of the London Mathematical Society*, 1(1):26–30, 1935.

13 Michael Kapralov. Better bounds for matchings in the streaming model. In *Proceedings of the Twenty-Fourth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2013, New Orleans, Louisiana, USA, January 6-8, 2013*, pages 1679–1697, 2013. `doi:10.1137/1.9781611973105.121`.

14 Michael Kapralov. Space lower bounds for approximating maximum matching in the edge arrival model. In *Proceedings of the Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2021*, 2021.

15 Christian Konrad. A simple augmentation method for matchings with applications to streaming algorithms. In *43rd International Symposium on Mathematical Foundations of Computer Science, MFCS 2018, August 27-31, 2018, Liverpool, UK*, pages 74:1–74:16, 2018.

16 Christian Konrad, Frédéric Magniez, and Claire Mathieu. Maximum matching in semi-streaming with few passes. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques - 15th International Workshop, APPROX 2012, and 16th International Workshop, RANDOM 2012, Cambridge, MA, USA, August 15-17, 2012. Proceedings*, pages 231–242, 2012.

17 William T Tutte. The factorization of linear graphs. *Journal of the London Mathematical Society*, 1(2):107–111, 1947.

Optimal Fine-Grained Hardness of Approximation of Linear Equations

Mitali Bafna
Harvard University, Cambridge, MA, USA

Nikhil Vyas
MIT, Cambridge, MA, USA

Abstract

The problem of solving linear systems is one of the most fundamental problems in computer science, where given a satisfiable linear system (A, b), for $A \in \mathbb{R}^{n \times n}$ and $b \in \mathbb{R}^n$, we wish to find a vector $x \in \mathbb{R}^n$ such that $Ax = b$. The current best algorithms for solving dense linear systems reduce the problem to matrix multiplication, and run in time $O(n^\omega)$. We consider the problem of finding ε-approximate solutions to linear systems with respect to the L_2-norm, that is, given a satisfiable linear system $(A \in \mathbb{R}^{n \times n}, b \in \mathbb{R}^n)$, find an $x \in \mathbb{R}^n$ such that $||Ax - b||_2 \leq \varepsilon ||b||_2$. Our main result is a fine-grained reduction from computing the rank of a matrix to finding ε-approximate solutions to linear systems. In particular, if the best known $\tilde{O}(n^\omega)$ time algorithm for computing the rank of $n \times O(n)$ matrices is optimal (which we conjecture is true), then finding an ε-approximate solution to a dense linear system also requires $\tilde{\Omega}(n^\omega)$ time, even for ε as large as $(1 - 1/\text{poly}(n))$. We also prove (under some modified conjectures for the rank-finding problem) optimal hardness of approximation for sparse linear systems, linear systems over positive semidefinite matrices and well-conditioned linear systems. At the heart of our results is a novel reduction from the rank problem to a decision version of the approximate linear systems problem. This reduction preserves properties such as matrix sparsity and bit complexity.

2012 ACM Subject Classification Theory of computation → Problems, reductions and completeness

Keywords and phrases Linear Equations, Fine-Grained Complexity, Hardness of Approximation

Digital Object Identifier 10.4230/LIPIcs.ICALP.2021.20

Category Track A: Algorithms, Complexity and Games

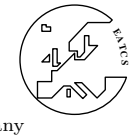
Related Version *Full Version*: `https://arxiv.org/abs/2106.13210`

Funding *Mitali Bafna*: Supported in part by a Simons Investigator Award and NSF Award CCF 1715187.
Nikhil Vyas: Supported by NSF CCF-1909429.

1 Introduction

Algorithms for solving linear equations are one of the most fundamental primitives in computer science. Formally this is the problem where, given a linear system (A, b), where $A \in \mathbb{R}^{m \times n}$ is a real matrix and $b \in \mathbb{R}^m$ is a vector in the column space of A, we need to find a vector $x \in \mathbb{R}^n$ such that $Ax = b$. Gaussian elimination running in time[1] $O(n^3)$ was one of the first algorithms for this problem. Hopcraft and Bunch [9] reduced solving linear equations to fast matrix multiplication of two $n \times n$ matrices [35, 14, 34, 37, 17, 3] which can be done in $m(n) = n^\omega$, where ω is the matrix multiplication constant. The current best known upper bound on ω is approximately 2.372.. [3]. The best known algorithms for solving linear systems reduce the problem to matrix multiplication, but there is no known reduction in the other direction. We study the complexity of finding approximate solutions to linear systems (defined more precisely later) under the following conjecture:

[1] Here we are discussing algorithms and hardness over the Real RAM, unless stated otherwise. We discuss the Word RAM in more detail in Section 1.3.



▶ **Conjecture 1.1** (Rank-Finding Conjecture over RealRAM). *Finding the rank of a matrix $A \in \mathbb{R}^{m\times n}$ with $m = O(n)$ in RealRAM is $\Omega(n^\omega)$-hard.*

The problem of finding the rank of a matrix is a central problem in linear algebra. It is known that this problem can be reduced to fast matrix multiplication [9, 20], hence there exist algorithms for the rank-finding algorithm that run in time $O(n^\omega)$. There are known faster algorithms for restricted classes of matrices. For sparse matrices we know of $O(n^2)$ algorithms [36] and for low-rank matrices the $O(n^2+\text{rank}(A)^\omega)$-time algorithms of [21, 15, 11] run in time $n^{\omega-\Omega(1)}$ when $\text{rank}(A) = n^{1-\Omega(1)}$. No improvement over the $O(n^\omega)$-runtime is known for general matrices though. We conjecture that this run time is in fact *optimal* for general matrices. The rank-finding problem is equivalent to checking whether the determinant of a matrix is 0. We get some evidence towards the truth of our conjecture by considering the computational model of arithmetic circuits[2]: In a seminal work Baur and Strassen [7] linearly reduced the the problem of matrix multiplication to the problem of computing the determinant, thus showing that the latter problem requires arithmetic circuits of size as large as those required for matrix multiplication. Furthermore, this is a central conjecture because falsifying it (getting faster algorithms for the rank-finding problem) would yield better algorithms for important problems like finding the size of a maximum matching in a graph [26, 27]. [3] For some direct evidence: there has been a line of work by Musco et al [28] that gives algorithms to approximate the Schatten p-norms in time better that $O(n^\omega)$ when $p > 0$. But at $p = 0$, the problem of finding the Schatten p-norm is the same as finding the rank of the matrix, and their algorithms run in time $\Omega(n^\omega)$. Hence they are not able to beat the runtime of $O(n^\omega)$ to approximate the rank of a matrix, let alone determine it exactly.

Conjecture 1.1 allows us to study the hardness of linear system solving and related linear algebraic problems in the style of fine-grained complexity [38]. We show that the conjecture directly implies $\tilde{\Omega}(n^\omega)$-hardness of finding exact solutions to linear systems. One could hope to get faster algorithms though when allowed to find an approximate solution to the linear system. In this paper, we consider the problem of finding approximate solutions to linear systems. Specifically, we consider the following notion of approximation:

▶ **Definition 1.2** ($\varepsilon(n)$-Approximate Linear Search). *For a function $\varepsilon : \mathbb{N} \to [0, 1]$, the ε-Approximate Linear Search problem is defined as, given a satisfiable[4] linear system $(A \in \mathbb{R}^{O(n)\times n}, b)$, find an $x \in \mathbb{R}^n$ such that $||Ax - b||_2 \leq \varepsilon(n)||b||_2$.*

Note that the all 0's vector $x = 0^n$ is a 1-approximate solution to any linear system as $||A0^n - b|| = ||b||_2$. Our main result shows that doing barely better than the trivial approximation is hard: $(1 - 1/n^{100})$-Approximate Linear Search i.e. finding an x such that $||Ax - b||_2 \leq (1 - 1/n^{100})||b||_2$. is $\tilde{\Omega}(n^\omega)$ hard under Conjecture 1.1.

Spielman and Teng [32] gave nearly linear-time algorithms ($O(n^2 \log(1/\varepsilon(n)))$-time) for finding $\varepsilon(n)$-approximate solutions to Laplacian linear-systems and this result was built upon by many works [23, 12] to give such algorithms for other restricted classes of linear systems. Our result shows that under the hardness of the rank-finding problem, these algorithms cannot be extended to general linear systems. As mentioned above, we conditionally rule out $\tilde{O}(n^2)$-time algorithms for finding $\varepsilon(n)$-approximate solutions to general linear systems even for $\varepsilon(n) = 1 - 1/n^{100}$.

---

[2] Such a reduction is unknown in the RealRAM model.

[3] This is because maximum matching algorithms has a randomized reduction to the rank-finding problem.

[4] Keeping with the convention of promise problems, we will assume that when given an unsatisfiable instance the algorithm is allowed to output an arbitrary vector.

We also extend our results to give optimal conditional hardness of approximation (under analogous conjectures for the rank-finding problem) for restricted classes of linear systems: sparse linear systems, linear systems over positive-semidefinite matrices, and well-conditioned linear systems.

Recently there has been a lot of progress in relating the exact time-complexities of various problems that have polynomial running times. Although there has been success in a variety of graph-theoretic, geometric and string problems [31, 2, 6], there are very few fine grained reductions from the assumptions therein to linear algebraic problems, an example being, the work of Musco et al [28] that showed conditional lower bounds for spectrum approximation.

The theory of probabilistically checkable proofs [5, 18] was instrumental in proving a host of NP-hardness of approximation results. Though this theory was very successful in settling the time-complexity for approximation problems in NP, there are inherent limitations to extend these techniques to problems in P. Towards this, there has been recent progress for establishing hardness of approximation results for problems in P [1, 10, 22]. Our paper makes further progress in this direction.

1.1 Our results

The table below gives a summary of our hardness results over the RealRAM.

Table 1 ε-ALS refers to ε-Approximate Linear Search Problem. Our hardness results are under different conjectures, see statements for more details. All hardness results are for $\varepsilon = 1 - 1/n^{100}$, while all algorithms are for the much (apriori) harder exact search problem. For small values of condition number better algorithms are known [19] but for our regime of either unbounded or poly(n) condition number the above algorithms are the best known.

Problem	Hardness	Algorithm
ε-ALS	$\tilde{\Omega}(n^\omega)$ (Corollary 4.2)	$O(n^\omega)$ ([9])
Sparse ε-ALS	$\tilde{\Omega}(n^2)$ (Corollary 4.4)	$O(n^2)$([19])
Well-conditioned ε-ALS (Definition 4.13)	$\tilde{\Omega}(n^\omega)$ (Corollary 4.17)	$O(n^\omega)$ ([9])
ε-ALS with PSD matrix	$\tilde{\Omega}(n^\omega)$ (Corollary 4.10)	

We consider the question: Can one get fast approximate linear system solvers for general linear systems that run in time $\tilde{o}(n^\omega)$? We answer this question in the negative, under Conjecture 1.1. In this section we discuss the hardness result for solving general linear systems approximately (first row of Table 1). We discuss the other results of the table in Section 1.2.

We refer to the decision version of the ε-approximate linear search problem as Approximate Linear Decision problem (formally defined in Definition 1.4). We prove all our hardness results by showing a reduction from the rank-finding problem to the Approximate Linear Decision problem. We now state the lemma that proves our main reduction:

▶ **Theorem 1.3** (Main reduction: Informal). *There exists a randomized Turing reduction from the rank-finding problem on $A \in \mathbb{R}^{m\times n}$ to the $(1 - 1/n^{100})$-Approximate linear decision problem*[5] *on square matrices with sparsity $\tilde{O}(nnz(A))$ and dimension $O(\max(m, n))$. The reduction runs in time $\tilde{O}(nnz(A))$ and works with high probability.*

[5] The constant 100 here is arbitrary and in fact our reduction works for all constants.

Our reduction

At the heart of our results lies an "exact to approximate" reduction for *deciding* the satisfiability of linear systems. For showing hardness of finding approximate solutions, we are able to use this philosophy of "increasing the gap" between the YES and NO instances. We consider the following natural decision analogue of the ε-approximate search problem:

▶ **Definition 1.4** ($\varepsilon(n)$-Approximate Linear Decision problem [25]). *For a function $\varepsilon : \mathbb{N} \to [0, 1]$, given a linear system $(A \in \mathbb{R}^{m\times n}, b \in \mathbb{R}^m)$, with $m = O(n)$ and the promise that it falls into one of the following two sets of instances:*
1. *YES instance: There exists an $x \in \mathbb{R}^n$ such that $Ax = b$.*
2. *NO instances: For all $x \in \mathbb{R}^n$, $||Ax - b||_2 > \varepsilon(n)||b||_2$,*

decide whether (A, b) is a YES instance or a NO instance. We will refer to $\varepsilon(n)$ as the "gap" of the instance.

We show that the rank-finding problem reduces to the $(1 - 1/n^{100})$-approximate linear decision problem described above. We also show that the rank-finding problem is equivalent to the *exact* linear decision problem i.e. the problem of deciding satisfiability of linear systems. Hence our reduction can be interpreted as increasing the gap between the YES/NO cases from almost 0 (can be arbitrarily small as we are working over the RealRAM) to $1 - 1/n^{100}$. We increase this gap in two stages, first to $\varepsilon(n) = 1/n^{O(1)}$ and then to $1 - 1/n^{100}$. This gives conditional $\tilde{\Omega}(n^\omega)$-hardness of the $(1 - 1/n^{100})$-approximate linear decision problem.

Even though the main reduction discussed here is from the rank-finding problem, we are also able to give a search to search reduction from the $1/n^{O(1)}$-approximate linear search problem to the $(1 - 1/n^{100})$-approximate linear search (see Corollary 1.6).

We will now discuss the corollaries of the main reduction outlined above. Given Theorem 1.3, we perform a standard decision to search reduction (Lemma 4.1) to get optimal hardness of approximation for the $(1-1/n^{100})$-approximate *search* problem, under Conjecture 1.1. Thus under the rank finding conjecture, this reduction rules out all $(1-1/n^{100})$-approximation algorithms that run in time $\tilde{o}(n^\omega)$.

▶ **Corollary 1.5** (Informal). *Under Conjecture 1.1, for all constants $c > 0$, the $(1 - 1/n^{100})$-approximate linear search problem is $\tilde{\Omega}(n^\omega)$-hard in the RealRAM model of computation.*

The second step of our reduction can also be used to increase the gap of the $1/n^{O(1)}$-Approximate Linear *Search* problem from $1/n^{O(1)}$ to $1 - 1/n^{100}$. This gives us the following corollary:

▶ **Corollary 1.6** (Search to Search reduction: Informal). *If for any constant a there exists an $\tilde{O}(n^a)$-time algorithm for $(1 - 1/n^{100})$-approximate linear search problem then there exists an $\tilde{O}(n^a)$-time algorithm for the $1/n^{100}$-approximate linear search problem.*

Our hardness result is tight because there exist algorithms which solve the $(1 - 1/n^c)$-Approximate Linear Search problem in $O(n^\omega)$ over the RealRAM. In fact, one can solve the more general problem of linear regression, i.e. given a (possibly unsatisfiable) linear system (A, b), find an x that minimizes $||Ax - b||_2$, in time $O(n^\omega)$.

1.2 Extensions

We prove several extensions of our main theorem using the reduction discussed above. We can modify our main reduction so that it preserves the sparsity and condition number of the original instance to get the results for sparse and well-conditioned linear systems. To get hardness for PSD linear systems we need additional ideas beyond this reduction. We get the following results:

Sparse Linear Systems

Linear equation solving has also been studied in the case of sparse linear systems. We know of $O(\text{nnz}(A)n)$ time algorithms for solving a linear system $(A \in \mathbb{R}^{O(n)\times n}, b)$ where $\text{nnz}(A)$ denotes the number of non-zero entries of A, that use Conjugate Gradient Descent [19], so that when the sparsity of A is $\tilde{O}(n)$, these algorithms run in time $\tilde{O}(n^2)$. Our reduction (discussed above), preserves the sparsity of the original matrix A and thus reduces the exact problem over sparse linear systems to the approximate problem over sparse ones. We start with an analogous conjecture to Conjecture 1.1 for finding the rank of a sparse matrix:

▶ **Conjecture 1.7** (Rank-finding Conjecture for Sparse matrices over RealRAM)**.** *Finding the rank of a matrix $A \in \mathbb{R}^{m\times n}$ with $m = O(n)$ and $nnz(A) = \tilde{O}(n)$ in RealRAM is $\Omega(n^2)$-hard.*

Under the above conjecture we show (see Corollary 4.4) that solving $(1 - 1/n^{100})$-Approximate Linear Search problems on sparse linear systems is $\tilde{\Omega}(n^2)$-hard, which is optimal up to poly-logarithmic factors.

Positive Semi-Definite Linear systems

We give optimal hardness for the $(1 - 1/n^{100})$-approximate linear search problem when the matrix A is restricted to be positive semidefinite (see Corollary 4.10). Recently there has been a lot of work for getting nearly linear-time approximation algorithms for restricted classes of matrices. For a slightly more restricted class of linear systems than PSD ones, called Strongly Diagonally Dominant (SDD) systems, Spielman and Teng gave near-linear time approximate solvers [32], leaving near-linear time approximation algorithms for PSD linear systems as the next open problem. In fact, resolving the time-complexity for PSD linear systems was in mentioned as an open problem in [4], where they gave unconditional hardness for PSD linear systems for sublinear-time algorithms. Interestingly, we show that under Conjecture 4.6, such solvers are not possible for PSD linear systems, thus giving a conditional separation of the time complexity required for approximately solving SDD linear systems versus PSD ones.

Well-conditioned linear systems

We now turn our attention to the problem of solving well-conditioned (polynomially bounded condition number) linear systems approximately. In Section 4.4, we give optimal conditional hardness of approximation for linear systems over matrices with polynomially bounded condition number under the $\tilde{\Omega}(n^\omega)$-hardness of the well-conditioned rank-finding problem.

We also prove the following analogue of Corollary 1.6 which amplifies the gap for the *search* problem on well-conditioned matrices:

▶ **Corollary 1.8.** *If there exists as $\tilde{O}(n^a)$ time algorithm for Well-conditioned $(1 - 1/n)$-Approximate Linear Search then there exists a $\tilde{O}(n^a)$-time algorithm for Well-conditioned $(1/n)$-Approximate Linear Search problem.*

1.3 Reductions over the WordRAM

Our main reduction can be modified to preserve the bit-complexity of the original matrix. In Section 5, we show analogous results for all the problems considered above over the WordRAM. We show that under analogous conjectures for the rank-finding problem over

the WordRAM, the problem of finding $(1 - 1/n^{100})$-approximate solutions to linear systems with bit-complexity $O(\log n)$ is $\tilde{\Omega}(n^\omega)$-hard over the WordRAM. Our hardness result is tight up to polylogarithmic factors because there exist $\tilde{O}(n^\omega)$-time algorithms for exactly solving linear systems on the WordRAM [33, 29, 8].

1.4 Further applications and related work

Recently, there has been a lot of progress on the algorithmic front for finding approximate solutions to restricted classes of linear systems (A, b). In a breakthrough work, Spielman and Teng [32] obtained $\tilde{O}(\text{nnz}(A)\log(1/\varepsilon(n)))$-time algorithms for finding $\varepsilon(n)$-approximate solution to Laplacian systems and Strongly Diagonally Dominant (SDD) systems. This result was followed up by algorithms for more general classes of linear systems such as Connection Laplacians [23] and Directed Laplacian systems [13]. This raised the hope that such approximation algorithms could be obtained for more general classes of matrices such as truss stiffness matrices and total variation matrices. Kyng and Zhang [25] showed that such algorithms for these slightly more general classes would imply approximation algorithms for general linear systems. Therefore, by composing our reduction with theirs, one immediate corollary we get is that solving approximately for these classes of restricted linear systems is as hard as the rank-finding problem.

In [24] the authors prove conditional hardness for the problems of Packing/Covering Linear Programs based on the hardness of approximately solving general linear equations. Prior to our work there was no evidence of hardness for approximately solving linear equations. Our results therefore imply hardness for these problems under the rank-finding problem, which is a more well-studied problem in our opinion.

Organization

In section 2 we introduce notation and basic definitions that will be used throughout the paper. In Section 3 we give a proof of our main reduction (Theorem 3.1). In Section 4, we show conditional hardness for finding approximate solutions to linear systems over the Real RAM. We then extend these conditional hardness results to restricted classes of linear systems: Section 4.2 considers sparse linear systems, Section 4.3 considers positive semidefinite linear systems, and finally Sec 4.4 considers well-conditioned linear systems. In Section 5 we show the analogues of these results over the WordRAM model of computation.

All the proofs are deferred to the full version of the paper.

2 Preliminaries

Below is some notation that will be used throughout:

Notation

We will use $\text{nnz}(A)$ for to denote the sparsity of a matrix A and we will assume that $\text{nnz}(A) \geq \max(n, m)$ for $A \in \mathbb{R}^{m\times n}$. We will call a matrix $A \in \mathbb{R}^{m\times n}$ sparse if $\text{nnz}(A) = \tilde{O}(\max(m, n))$. We will use $\kappa(A)$ to denote the condition number of a matrix A. By bit complexity of A or $\mathcal{B}(A)$ we will refer to maximum bit complexity of any entry in the matrix/vector A. We will use $\leq_T$ to denote Turing reductions. Most of our Turing reductions run in quasi-time linear in the input size. We say $\langle a, b\rangle$ to denote the inner product of a and b i.e. $\sum_i a_i b_i$. We denote $W^\perp$ to denote the subspace orthogonal to the subspace W. By $P_W(b)$ we denote the projection of b on the vector space W. For a matrix M we denote its column space by

colspace(M). By $A^\dagger$ we mean the pseudoinverse of a matrix A. For a matrix $A \in \mathbb{R}^{m\times n}$ by $\Pi_A = A(AA^T)^\dagger A^T$ we mean the linear operator such that for all $x \in \mathbb{R}^m$, $\Pi_A(x)$ is the projection of x on colspace(A). By $g = \tilde{O}(f)$ we mean $g = O(f \cdot \text{polylog}(f))$. By $g = \tilde{\Omega}(f)$ we mean $g = \Omega(f/\text{polylog}(f))$. Whenever not specified by algorithms we mean randomized algorithms. We use w.h.p. to denote a probability of $1 - 1/n^{\log n}$, where n is the input size under consideration.

We refer to the exact version of the $\varepsilon(n)$-Approximate Linear Search problem as the Linear Search Problem. Similarly we refer to the exact version of the $\varepsilon(n)$-Approximate Linear Decision problem as the Linear Decision problem.

3 Proof of Main Reduction (Theorem 1.3)

In this section we will prove the reduction from the rank-finding problem to the approximate version of the linear decision problem. For simplicity, throughout this section we work on the RealRAM model of computation and wherever we do not state it we assume that this is the case, so we *do not* discuss the bit complexity of the reductions. Our reduction can be modified to work on the WordRAM which we do in Section 5.

▶ **Theorem 3.1** (Restatement of Theorem 1.3). *For all constants $c > 0$, there exists a randomized Turing reduction in the RealRAM model of computation, from the rank-finding problem on $A \in \mathbb{R}^{m\times n}$ to the $(1 - 1/n^c)$-Approximate linear decision problem on $(A' \in \mathbb{R}^{n'\times n'}, \mathbf{1}^{n'})$, with dimension $n' = O(\max(m, n))$ and sparsity $\tilde{O}(nnz(A))$, where in the YES case we have the additional property that the matrices produced have full rank. The reduction runs in time $\tilde{O}(nnz(A))$, produces polylog(mn) instances of the approximate linear decision problem and works with high probability.*

We will prove this reduction in three steps:

1. Lemma 3.3: Rank-Finding Problem $\leq_T$ Full Rank problem (see Definition 3.2)
2. Lemma 3.4: Full Rank problem $\leq_T$ $(1/n^{O(1)})$-Approximate linear decision problem
3. Lemma 3.5: $1/n^{O(1)}$-Approximate linear decision problem $\leq$ $(1 - 1/n^c)$-Approximate linear decision problem

Given the lemmas, it will be straightforward to combine these reductions to get that the linear decision problem reduces to the $(1 - 1/n^c)$-Approximate linear decision problem. Let us now show each of the steps stated above. We will prove the first reduction from the rank-finding problem to the full-rank problem. This is similar to a reduction by Wiedemann [36] for finite fields. Let us formally introduce the Full rank problem.

▶ **Definition 3.2** (Full Rank Problem.). *Given a matrix $A \in \mathbb{R}^{n\times n}$, is rank$(A) = n$?*

The intuition behind the following reduction from the rank-finding problem to the full-rank problem is the following: For a matrix $A \in \mathbb{R}^{m\times n}$ with $m > n$ and rank$(A) = k < n$, adding t "random" columns will give a full column rank matrix if and only if $t \geq n - k$. Hence we can binary search for the first value of t which gives us a full column rank matrix, yielding k.

▶ **Lemma 3.3.** *There exists a randomized Turing reduction which works w.h.p. from the Rank-Finding problem on $A \in \mathbb{R}^{m\times n}$ to the Full rank problem, that runs in time $\tilde{O}(nnz(A))$ and produces $O(\log m)$ instances of the Full rank problem, such that all instances of the matrices produced have dimension $O(\max(m, n)) \times O(\max(m, n))$ and sparsity $\tilde{O}(nnz(A))$.*

We will now prove that the full rank problem reduces to the $(1/n^{O(1)})$-Approximate linear decision problem. The idea behind the proof is that for a full-rank square matrix $A \in \mathbb{R}^{n\times n}$ every vector $b \in \mathbb{R}^n$ belongs in the column space of A and hence the linear system (A, b) is satisfiable. On the other hand if $\text{rank}(A) < n$ we expect a random vector b to be outside the column space of A and hence we expect the linear system (A, b) to be unsatisfiable. We show a strengthened version of the previous statement by proving that w.h.p. for a random Gaussian vector b, it holds that for all x, $||Ax - b|| \geq \varepsilon(n)||b||$ for some $\varepsilon(n) = 1/n^{O(1)}$. We also show that we can rescale the rows of the linear system (A, b) to get the system $(A', \mathbf{1}^n)$ while maintaining the property that $||A'x - \mathbf{1}^n|| \geq \varepsilon(n)||\mathbf{1}^n||$. We perform the rescaling to obtain $b = \mathbf{1}^n$ as that will simplify our later reductions.

▶ **Lemma 3.4.** *Consider a matrix $M \in \mathbb{R}^{n\times n}$. There exists a randomized Turing reduction from the problem of checking whether M has full rank to the Linear Decision $(1/n^{O(1)})$-Approximation problem. The reduction runs in time $\tilde{O}(nnz(M))$, produces polylog(n) instances of the form $(M' \in \mathbb{R}^{n\times n}, \mathbf{1}^n)$ where in the YES case M' is a full rank matrix, and works w.h.p.*

We will now show that one can amplify the error in the NO case from $1/n^{O(1)}$ to $1 - 1/n^c$ for all constants c.

The following lemma works for all $\varepsilon(n), \delta(n)$, but we only state it for $\varepsilon(n) = 1/n^{O(1)}, \delta(n) = 1/n^c$ to maintain consistency with the later sections in which we will work over WordRAM.

▶ **Lemma 3.5.** *For all constants c and $\varepsilon(n) = 1/n^{O(1)}, \delta(n) = 1/n^c$, there exists a deterministic many-one reduction from the $\varepsilon(n)$-Approximate linear search problem on the linear system $(A \in \mathbb{R}^{n\times n}, \mathbf{1}^n)$ to the $(1 - \delta(n))$-Approximate linear search problem on the linear system $(A' \in \mathbb{R}^{n\times n}, \mathbf{1}^n)$, with $nnz(A') = O(nnz(A))$. The reduction runs in time $\tilde{O}(nnz(A))$. Additionally if A is full rank then the matrix A' produced is also full rank.*

As this is a deterministic many-one reduction we also get a gap-amplifying reduction for the $\varepsilon(n)$-Approximate linear decision problem with the same parameters.

We can now combine all the lemmas above to get the proof of Theorem 3.1.

Proof of Theorem 3.1. We have proved the following sequence of reductions which preserve sparsity:

1. Lemma 3.3: Linear decision problem $\leq_T$ Full rank problem
2. Lemma 3.4: Full rank problem $\leq_T$ $1/n^{O(1)}$-Approximate linear decision problem
3. Lemma 3.5: $1/n^{O(1)}$-Approximate decision problem $\leq_T$ $(1 - 1/n^c)$-Approximate linear decision problem.

Note that in the YES case of Full-Rank problem we have a square full rank matrix, and this is propagated through Lemma 3.4 and Lemma 3.5 hence the final instance we produce has a full rank matrix in the YES case. Since each of these reductions work whp, one can compose them to get the theorem statement. ◀

4 Hardness of finding L_2-Approximate solutions on Real RAM

In this section, we elaborate on the implications of our main reduction from the previous section (Theorem 3.1). Below is the map of reductions we showed in the previous sections. We will introduce conjectures for the Rank-finding problem in this section and given the reductions, the conjectures will imply conditional hardness of the approximate linear search problem.

All the results in this section are for the RealRAM but can be obtained over the WordRAM too. In Section 5, we prove the conditional hardness of the approximate linear search problem over general matrices in the WordRAM model.

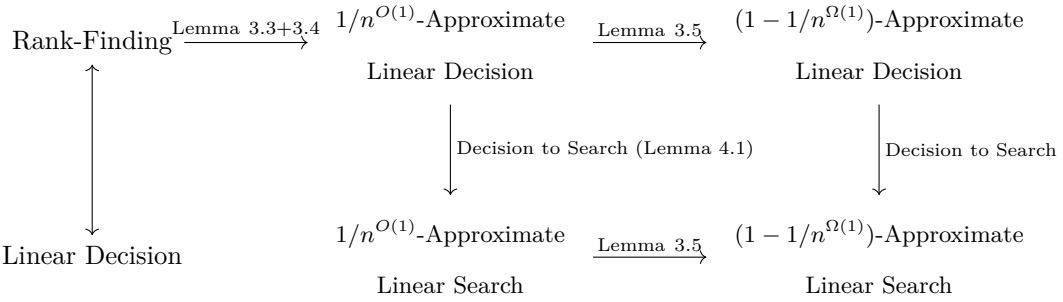


Figure 1 Reductions on RealRAM preserving sparsity (up to polylog(n) factors) and dimension (up to constant factors).

Note that all the lemmas pointed out here (except for the decision to search reduction) were shown in the previous section. The decision to search reduction is straightforward to carry out in the RealRAM and we formally prove it in Lemma 4.1.

In the sections below we discuss the hardness of the approximate linear search problem over general matrices, and then the case of restricted classes of linear systems - sparse linear systems and linear systems given by positive-semidefinite matrices. Our results give tight conditional hardness for all the problems considered.

4.1 General linear systems

We base the hardness result for the approximate linear search problem under the following conjecture for the rank-finding problem:

We now prove the search to decision reduction.

▶ **Reminder of Conjecture 1.1** *Finding the rank of a matrix $A \in \mathbb{R}^{m\times n}$ with $m = O(n)$ in the RealRAM model of computation is $\tilde{\Omega}(n^{\omega})$-hard.*

▶ **Lemma 4.1.** *If there exists a $O(t(n))$ time algorithm to solve ε-Approximate linear search problem for a linear system with sparsity s then there exists a $O(t(n) + sn)$ time algorithm to solve ε-Approximate linear decision problem for linear system with sparsity s.*

Proof. We will follow the standard decision to search reductions which proceed by solving and then confirming. Suppose we are given a linear system (A, b) with $\text{nnz}(A) = s$ for which we want to solve ε-Approximate linear decision problem.

Suppose it is a YES instance i.e. there exists an exact solution, then by the assumed algorithm for ε-Approximate linear search problem we can find an x' such that $||Ax' - b||_2 \leq \varepsilon||b||_2$. This can be done in $O(t(n))$ time.

Suppose instead we were in the NO case i.e. for all x', $||Ax' - b||_2 > \varepsilon||b||_2$ i.e. there exists no x' such that $||Ax' - b||_2 \leq \varepsilon||b||_2$.

Hence checking whether $||Ax' - b||_2 \leq \varepsilon||b||_2$ is true of not for the x' returned by the assumed algorithm for ε-Approximate linear search problem will let us solve the ε-Approximate linear decision problem. We can check if $||Ax' - b||_2 \leq \varepsilon||b||_2$ in time $O(sn)$. Hence the total running time is $O(t(n) + sn)$. ◀

Combining the main reduction (Theorem 3.1) with a decision to search reduction (Lemma 4.1) we get optimal conditional hardness for the $(1-1/n^c)$-approximate linear search problem under Conjecture 1.1:

▶ **Corollary 4.2** (Corollary 1.5 restated). *Under Conjecture 1.1, for all constants $c > 0$, the $(1-1/n^c)$-approximate linear search problem is $\tilde{\Omega}(n^\omega)$ hard in the RealRAM model of computation. Moreover, this remains true even when the matrix A in the given linear system (A, b) is square and has full rank.*

This conditional hardness result is tight as the best algorithms for (exactly) solving general linear systems run in time $\tilde{O}(n^\omega)$ [9].

The next corollary is a search to search reduction between the approximate linear search problem with small gap to one with a larger gap. This is a direct consequence of Lemma 3.5. Recall that in the proof of Lemma 3.1 we used Lemma 3.5 to amplify the gap of the approximate linear *decision* problem. Lemma 3.5 is in fact more general and can amplify the gap of the approximate linear *search* problem too (as noted in the lemma statement):

▶ **Corollary 4.3** (Corollary 1.6 restated). *If for any constant $c > 0$ and a there exists an $\tilde{O}(n^a)$-time algorithm for $(1-1/n^c)$-Approximate Linear Search problem then for all constants d there exists a $\tilde{O}(n^a)$-time algorithms for $1/n^d$-Approximate Linear Search problem.*

Even though this is stated as a reduction, one could potentially use the above corollary to get better algorithms for the $1/n^{O(1)}$-approximate linear search problem.

4.2 Sparse linear systems

In this section, we give analogous results for sparse linear systems. Here we use the fact that our main reduction (Theorem 3.1) preserves the sparsity of the original matrix. Hence if we assume the hardness of the rank-finding problem over sparse matrices, we get conditional hardness for approximately solving sparse linear systems.

▶ **Reminder of Conjecture 1.7** *Finding the rank of a matrix $A \in \mathbb{R}^{m\times n}$, where $m = O(n)$ and $nnz(A) = \tilde{O}(n)$, in the RealRAM model of computation, is $\tilde{\Omega}(n^2)$-hard.*

Combining the main reduction (Theorem 3.1) with a decision to search reduction we get optimal conditional hardness for the $(1-1/n^c)$-approximate linear search problem on sparse matrices, under Conjecture 1.7:

▶ **Corollary 4.4.** *Under Conjecture 1.7, for all constants $c > 0$, the $(1-1/n^c)$-approximate linear search problem (A, b) with $nnz(A) = \tilde{O}(n)$ is $\tilde{\Omega}(n^2)$ hard, in the RealRAM model of computation. Moreover, this remains true even when the matrix A is square and has full rank.*

Note that here we crucially used the fact that the main reduction preserves the sparsity of the original matrix. This conditional hardness result is tight as the best algorithms for (exactly) solving linear equations run in time $\tilde{O}(\text{nnz}(A)\cdot n)$ [19] which is equal to $\tilde{O}(n^2)$ for sparse matrices.

Now we state the search to search reduction for the approximate linear search problem over sparse matrices which follows from Lemma 3.5. This reduction amplifies a small gap to a large gap.

▶ **Corollary 4.5.** *If for any constant $c > 0$ and a there exists an $\tilde{O}(n^a)$-time algorithm for the $(1-1/n^c)$-Approximate Linear Search problem over (A, b) then for all constants d there exists an $\tilde{O}(n^a)$-time algorithm for the $1/n^d$-Approximate Linear Search problem over (A', b) where $nnz(A') = nnz(A)$.*

4.3 Positive semidefinite linear systems

In this section, we show hardness for dense linear systems over PSD matrices. To do so, we need some additional ideas beyond our main reduction and also a conjecture for solving linear systems on matrices with intermediate sparsities. This conjecture also allows us to show optimal conditional hardness of approximately solving linear systems (A, b) for any sparsity.

▶ **Conjecture 4.6** (Rank-finding conjecture for all sparsities)**.** *Finding the rank of a matrix $A \in \mathbb{R}^{m\times n}$, where $m = O(n)$, is $\min(\tilde{\Omega}(nnz(A)\cdot n), \tilde{\Omega}(n^\omega))$ hard in the RealRAM model of computation.*

The current best known algorithms for the Rank-Finding problem ($A \in \mathbb{R}^{O(n)\times O(n)}$) runs in time $\min(\text{nnz}(A)n, n^\omega)$. The above conjecture assumes that this is optimal. We can prove the following theorem directly by combining Conjecture 4.6 and Lemma 5.5.

▶ **Corollary 4.7.** *Under Conjecture 4.6, for all constants $c > 0$, $(1 - 1/n^c)$-approximate linear search problem (A, b), is $\min(\tilde{\Omega}(nnz(A)\cdot n), \tilde{\Omega}(n^\omega))$ hard in the RealRAM model of computation. Moreover, this remains true even when the matrix A in the given linear system (A, b) has full rank.*

The next lemma reduces the approximate linear search problem for intermediate sparsities to approximate linear search problem on dense PSD matrices. It's proof exploits the fact that the algorithm of Yuster and Zwick [39] does matrix multiplication of two matrices A, A' in $O(\min(n^\omega, z^{.7}n^{1.2} + n^2))$ time where $z \geq \text{nnz}(A)$ and $z > \text{nnz}(A')$. This is faster than the best algorithm for solving linear systems (A, b) which runs in $O(\min(n^\omega, zn))$ where $z = \text{nnz}(A)$ for certain sparsities. Specifically we will use the following result from Yuster and Zwick [39]:

▶ **Theorem 4.8** (Yuster and Zwick [39], See Theorem 3.1 and discussion)**.** *Assuming $\omega > 2$ there exists constants $.1 \geq \gamma, \gamma' > 0$ such that for two matrices $A, B \in \mathbb{R}^{O(n)\times O(n)}$ which satisfy $nnz(A), nnz(B) \leq n^{\frac{\omega+1}{2}-\gamma}$ can be multiplied in time $O(n^{\omega-\gamma'})$.*

This allows to do the following reduction:

▶ **Lemma 4.9.** *Assuming $\omega > 2$, there exists constants $.1 \geq \gamma, \gamma' > 0$ and a reduction running in time $O(\max(n^{\omega-\gamma'}, n^{\omega-\gamma}))$ from $(1 - \delta(n))$-approximate linear search problem $(V \in \mathbb{R}^{n\times n}, b)$ with $nnz(V) \leq n^{\frac{\omega+1}{2}-\gamma}$ to $(1 - \delta(n))$-approximate linear search problem (V', b) such that the matrix V' is PSD.*

We now compose the above reduction with Conjecture 4.6 to get the following tight conditional hardness.

▶ **Corollary 4.10.** *Under Conjecture 4.6, for all constants $c > 0$ $(1 - 1/n^c)$-approximate linear search problem (A, b) where A is restricted to be a PSD matrix, is $\tilde{\Omega}(n^\omega)$ hard in the RealRAM model of computation. Moreover, this remains true even when the matrix A in the given linear system (A, b) has full rank.*

4.4 Well-Conditioned Linear Systems

In this section, we show conditional hardness of approximately solving well-conditioned linear systems. The condition number of a full-rank square matrix is the ratio of its maximum and minimum singular values. If the entries of a matrix are all $O(\log n)$-bits then the condition

number of this matrix is at most exponential in n (this is true even for rectangular full column-rank matrices). Therefore linear systems over matrices with polynomially-bounded condition number could be significantly easier to solve than general linear systems.

For the case of certain restricted classes of matrices such as directed Laplacians, the algorithm of Cohen et al [12] for the $\varepsilon(n)$-approximate linear search problem runs in time $\tilde{O}(\text{nnz}(A)\log(\kappa(A)/\varepsilon(n)))$ which is a near-linear time algorithm for $\kappa(A) = \text{poly}(n)$. This is a significant improvement over algorithms for directed Laplacian systems with no bound on the condition number (which run in time $\tilde{O}(n^{\omega})$).

But for general systems no such improvement is known! Conjugate gradient [19] runs in time $\tilde{O}(\text{nnz}(A))$, when $\kappa(A) = \text{poly}\log n$, whereas when $\kappa(A) = \text{poly}(n)$ this algorithm gives *no improvement* over the algorithm for general matrices.

We show that if we assume that the rank-finding problem is hard over well-conditioned matrices ($\kappa(A) = \text{poly}(n)$), then the approximate linear search problem is hard to solve over well-conditioned linear systems. The proof goes along the same lines as that for general matrices: we show in Lemmas 4.15 and 4.16 that our main reduction (Theorem 3.1) in fact preserves the condition number of our original matrix. Then as a corollary we obtain the conditional hardness for approximately solving well-conditioned linear systems. Note that we show all the results here over the RealRAM but they can be easily modified to work on the WordRAM too.

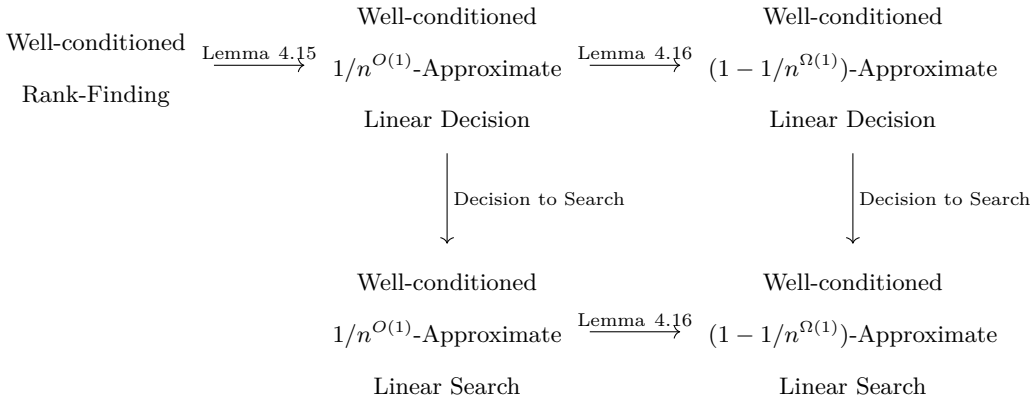


Figure 2 Reductions on the RealRAM preserving sparsity (up to polylog(n) factors), dimension (up to constant factors) and condition-number (upto poly(n) factors). One difference from the results from the previous sections is that we no longer have the equivalence for the Rank-Finding Problem and the Linear Decision problem for Well-conditioned matrices.

Before diving into the reduction, we will formally define all the problems used in the reduction-map above. As discussed in the definition of the condition-number, the condition-number is bounded only when the matrix has full column-rank, therefore in all the definitions below the matrices considered have dimension $m \times n$ with $n \leq m$.

▶ **Definition 4.11** (Well-conditioned Rank-Finding Problem)**.** *Given a matrix $A \in \mathbb{R}^{m\times n}$ where $n \leq m \leq O(n)$, with the promise that it falls into one of the following two sets of instances:*

1. *YES instances: $rank(A) = n$ and $\kappa(A) \leq poly(n)$.*
2. *NO instances: $rank(A) < n$*

decide whether A is a YES instance or a NO instance.

▶ **Definition 4.12** (Well-conditioned Full Column-Rank Problem). *Given a square matrix $A \in \mathbb{R}^{m \times n}$, where $n \leq m$, with the promise that it falls into one of the following two sets of instances:*

1. *YES instances: col-rank$(A) = n$ and $\kappa(A) \leq poly(n)$.*
2. *NO instances: col-rank$(A) < n$*

decide whether A is a YES instance or a NO instance.

Note that the Well-conditioned Rank-finding problem easily reduces to the Well-conditioned Full Column-rank problem. This is because the YES instances of the former always have full column-rank by definition. In fact, we can also reduce the well-conditioned rank-finding problem to the well-conditioned *full-rank* problem on *square* matrices, by adding random columns, since this operation preserves the condition-number [16]. For simplicity of presentation we do not perform this operation and continue to work with full column-rank and (possibly) rectangular matrices throughout.

Next we formally introduce the search and decision problems on well-conditioned linear systems:

▶ **Definition 4.13** (Well-conditioned $\varepsilon(n)$-Approximate Linear Search Problem). *For a function $\varepsilon : \mathbb{N} \to [0, 1]$, given a satisfiable linear system (A, b) with $A \in \mathbb{Z}^{m \times n}$ for $n \leq m$ and $\kappa(A) = poly(n)$ find an assignment x such that $||Ax - b|| \leq \varepsilon(n)||b||$*

▶ **Definition 4.14** (Well-Conditioned $\varepsilon(n)$-Approximate Linear Decision problem). *For a function $\varepsilon : \mathbb{N} \to [0, 1]$, given a linear system $(A \in \mathbb{Z}^{m \times n}, b \in \mathbb{Z}^n)$ for $n \leq m$, with the promise that it falls into one of the following two sets of instances:*

1. *YES instance: There exists an $x \in \mathbb{Q}^n$ such that $Ax = b$ and A is well-conditioned.*
2. *NO instances: For all $x \in \mathbb{R}^n$, $||Ax - b||_2 > \varepsilon(n)||b||_2$,*

decide whether (A, b) is a YES instance or a NO instance.

We will now show our main reduction from the Well-conditioned Rank-finding problem to the Well-conditioned $(1 - 1/n^c)$-approximate linear decision problem. As noted above the Well-conditioned Rank-finding problem reduces to the Well-conditioned Full Column-Rank problem, so we will show a reduction from the latter to the approximate linear decision problem. To do so, we will show that the proofs in Section 3 preserve the condition-number of the original matrix.

Let us start with showing that "well-conditioned" property is preserved in the reduction in Lemma 3.4.

▶ **Lemma 4.15.** *There exists a randomized Turing reduction from the Well-conditioned Full Column-Rank Problem on $M \in \mathbb{Z}^{m \times n}$ to the Well-conditioned $(1/n^{O(1)})$-Approximate Linear Decision problem. The reduction produces polylog(n) instances of the form $(M', \mathbf{1}^n)$ where $M' \in \mathbb{Z}^{m \times n}$ and $\kappa(M') = poly(n)$, runs in time $\tilde{O}(nnz(M))$, and works w.h.p.*

Next let us show that the "well-conditioned" property is preserved in the reduction in Lemma 3.5.

▶ **Lemma 4.16.** *For all constants $c, d > 0$, there exists a deterministic many-one reduction from the Well-conditioned $1/n^d$-Approximate linear search problem on the linear system $(A \in \mathbb{Z}^{m \times n}, \mathbf{1}^n)$ to the Well-conditioned $(1 - 1/n^c)$-Approximate linear search problem on the linear system $(A' \in \mathbb{Z}^{n \times n}, \mathbf{1}^n)$, with $nnz(A') = O(nnz(A))$ and $\kappa(A') = poly(n)$.*

As this is a deterministic many-one reduction we also get a gap-amplifying reduction for the $\varepsilon(n)$-Approximate linear decision problem with the same parameters.

Combining the two lemmas above we get the well-conditioned analogue of the main reduction from the Rank-finding problem to the $(1 - 1/n^c)$-approximate linear decision problem. We can now apply a decision to search reduction to get conditional hardness for approximately solving linear systems over well-conditioned matrices:

▶ **Corollary 4.17.** *For all constants c, assuming $\tilde{\Omega}(n^\omega)$ hardness of the well-conditioned rank-finding problem we get that the well-conditioned $(1 - 1/n^c)$-approximate linear search problem is $\tilde{\Omega}(n^\omega)$-hard.*

We also state the following search to search reduction which follows directly from Lemma 4.16:

▶ **Corollary 4.18.** *For all constants $a, c, d > 0$, if there exists an $\tilde{O}(n^a)$ time algorithm for well-conditioned $(1 - 1/n^c)$-approximate linear search then there exists an $\tilde{O}(n^a)$-time algorithm for the well-conditioned $(1/n^d)$-approximate linear search problem.*

5 Hardness of finding L_2-Approximate solutions on Word RAM

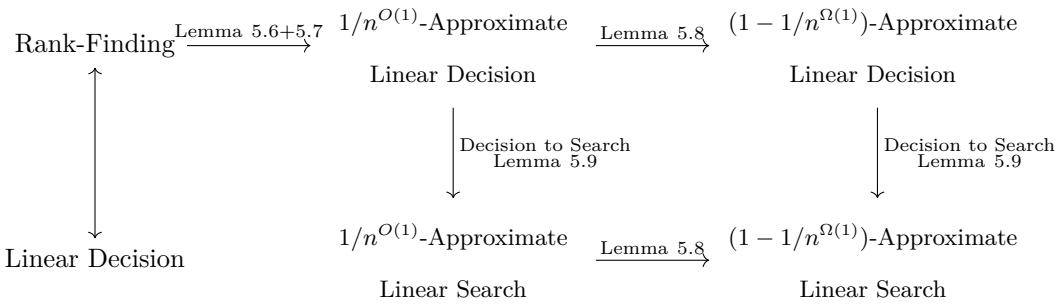


Figure 3 Reductions on WordRAM preserving sparsity (up to polylog(n) factors), dimension (up to constant factors) and bit complexity of entries (up to constant factors).

In Section 4 we gave conditional hardness for solving linear equations in RealRAM. In this section we will give hardness for WordRAM through some modifications of the reduction for RealRAM. We will assume that for a matrix/vector of dimension $m \times n$ in this section that all input entries are from $\mathbb{Z}$ and have $O(\log mn)$ bits in.

The reduction in RealRAM does not directly work for WordRAM as:

1. As we are in WordRAM we need to argue that the final problem instance has bounded bit complexity if the starting problem has bounded bit complexity. To verify this we show that this is true for all the steps of the reduction.
2. We sampled a random gaussian vector in the reduction (Lemma 3.4), this is not possible in the WordRAM. We will get around this issue (in Lemma 5.7) by sampling a random vector whose each entry is a random integer from a predefined range.
3. The decision to search reduction for RealRAM (Lemma 4.1) was nearly trivial. This is not the case for WordRAM as the solution can have large bit complexity and hence given a solution directly substituting to check if it is a good solution or not may require too much time. We give an alternative decision to search reduction in WordRAM in Lemma 5.9.

We start by defining the Linear Decision Problem over Word-RAM:

▶ **Definition 5.1** (Linear Decision Problem over Word-RAM). *Given a linear system (A, b) with $A \in \mathbb{Q}^{m\times n}$ with $\mathcal{B}(A) = O(\log mn)$, distinguish between the following two sets of instances:*

- *YES instances: There exists an x such that $Ax = b$.*
- *NO instances: For all x, $Ax \neq b$.*

We will consider the following conjecture (analogous to Conjecture 1.1) for the Word RAM:

▶ **Conjecture 5.2** (Rank-Finding Conjecture on WordRAM). *There exists no $\tilde{o}(n^\omega)$-time randomized algorithm for finding the rank of a matrix $A \in \mathbb{Z}^{m\times n}$ with $m = O(n)$ and $\mathcal{B}(A) = O(\log n)$, in the WordRAM model of computation.*

The conjecture is tight as using an easy randomized reduction [33] Rank over integers with $O(\log n)$ bit complexity can be reduced to rank over finite fields GF(poly(n)) which gives a $\tilde{O}(n^\omega)$ time randomized algorithm on WordRAM.

We will prove the following main theorem:

▶ **Corollary 5.3** (Hardness of solving linear equation on WordRAM). *For all constant $c > 0$, under Conjecture 1.1, there does not exist a $\tilde{o}(n^\omega)$ time randomized algorithm for the $(1 - 1/n^c)$-approximate linear search problem (A, b) with $\mathcal{B}(A) = O((c + 1)\log n)$ in the WordRAM model of computation. Moreover, this remains true even when the matrix A in the given linear system (A, b) has full rank.*

The conditional hardness in the above theorem is tight as there exists $O(n^\omega)$ time algorithms for exactly solving linear equations over $\mathbb{Z}$ in WordRAM [33, 29, 8].

We are also able to establish the following corollary of one of the intermediate steps in our reduction (Lemma 5.8) which reduces approximately solving linear systems with low error to approximately solving linear systems with barely non-trivial error.

▶ **Corollary 5.4.** *For all constant $c, d > 0$, if over WordRAM there exists as $\tilde{O}(n^a)$ time algorithm for $(1-1/n^c)$-Approximate Linear Search then there exists a $\tilde{O}(n^a)$-time algorithms for $(1/n^d)$-Approximate Linear Search problem.*

Analogous to Theorem 3.1 we will prove a reduction from the rank finding problem to approximate linear decision problem from which Corollary 5.3 will follow by a decision to search reduction (Lemma 5.9). Formally,

▶ **Lemma 5.5** (Reduction from exact to approximate). *For all constant c, there exists a randomized Turing reduction from the rank-finding problem on $A \in \mathbb{Z}^{m\times n}$ to the $(1 - 1/n^c)$-Approximate linear decision problem on $n' \times n'$-square matrices with bit complexity $\mathcal{B}(A) = O((c + 1)\log(n))$, with $n' = O(\max(m, n))$, where in the YES case we have the additional property that the matrices produced have full rank. The reduction runs in time $\tilde{O}(c \cdot nnz(A))$, produces polylog$(mn)$ instances of the approximate linear decision problem and works w.h.p..*

To prove the above Lemma, we will go through each of the steps in the proof of Theorem 1.3 and see that all of them work for WordRAM with small modifications. We start with the reduction the Rank-Finding Problem to the Full rank problem for WordRAM (analogous to Lemma 3.3 for RealRAM).

▶ **Lemma 5.6.** *There exists a randomized Turing reduction which works w.h.p. from the Rank-Finding problem on $A \in \mathbb{Z}^{m\times n}$ with $m = O(n)$ and $\mathcal{B}(A) = O(\log n)$, to the Full rank problem, that runs in time $\tilde{O}(nnz(A))$ and produces $O(\log m)$ instances of the Full rank problem, such that all instances of the matrices produced have dimension $O(\max(m, n)) \times O(\max(m, n))$, sparsity $\tilde{O}(nnz(A))$ and bit complexity $O(\log n)$.*

Analogous to Lemma 3.4, we now reduce the Full-Rank Problem to the $(1/n^{O(1)})$-Linear Decision Approximation problem on WordRAM.

▶ **Lemma 5.7.** *Consider a matrix $M \in \mathbb{Z}^{n\times n}$. There exists a randomized Turing reduction from the problem of checking whether M has full rank to the $(1/n^{12})$-Linear Decision Approximation problem. The reduction runs in time $\tilde{O}(nnz(M))$, produces polylog(n) instances of the form $(M', \mathbf{1}^n)$ where $M' \in \mathbb{Z}^{n\times n}, \mathcal{B}(M') = O(\mathcal{B}(M) + (\log n))$, in the YES case M' is a full rank matrix, and works w.h.p..*

Finally we reduce the $(1/n^{O(1)})$-Linear Decision Approximation problem to the $(1-1/n^c)$-Linear Decision Approximation problem over WordRAM. This is straightforward to work out from the analogous Lemma 3.5 in Section 4.

▶ **Lemma 5.8.** *For all constants c and $\varepsilon(n) = 1/n^{O(1)}, \delta(n) = 1/n^c$, there exists a deterministic many-one reduction from the $\varepsilon(n)$-Approximate linear search problem on the linear system $(A \in \mathbb{R}^{n\times n}, \mathbf{1}^n)$ to the $(1-\delta(n))$-Approximate linear search problem on the linear system $(A' \in \mathbb{R}^{n\times n}, \mathbf{1}^n)$, with $nnz(A') = O(nnz(A))$. The reduction runs in time $\tilde{O}(nnz(A))$. Additionally if A is full rank then the matrix A' produced is also full rank.*

As this is a deterministic many-one reduction we also get a gap-amplifying reduction for the $\varepsilon(n)$-Approximate linear decision problem with the same parameters.

Now we are ready to prove Lemma 5.5.

Proof of Lemma 5.5. Similar to the proof of Theorem 1.3, The proof follows by composing Lemma 5.6, Lemma 5.7 and Lemma 5.8 (for $\varepsilon(n) = 1/n^{O(1)}$ and $\delta(n) = 1/n^c$). The bit complexity of the final matrix is $O(\log n) + O(\log(n/(\varepsilon(n)\delta(n)))) = O((c+1)(\log n))$ and the running time is $\tilde{O}(c \cdot \text{nnz}(A))$ ◀

To prove Corollary 5.3 we need the following decision to search reduction:

▶ **Lemma 5.9.** *Let $\varepsilon(n) = 1/n^{O(1)}$, given an $A \in \mathbb{Z}^{m\times n}, x, b$ where $m = O(n), \mathcal{B}(A), \mathcal{B}(b) =$ polylog$(n), \mathcal{B}(x) = \tilde{O}(n)$ we can distinguish between:*
1. $||Ax - b|| \le \varepsilon(n)||b||/2$.
2. $||Ax - b|| \ge \varepsilon(n)||b||$.

w.h.p. in time $\tilde{O}(nnz(A)n)$.

Combining Lemma 5.5 and Lemma 5.9 give us Corollary 5.3:

Proof of Corollary 5.3. For all constants c, Composing Conjecture 5.2 and Lemma 5.5 gives us $\tilde{\Omega}(n^\omega)$ hardness of $(1-1/n^c)$-Approximate linear decision problem $(A \in \mathbb{Z}^{n\times n}, b)$ with bit complexity $O((c+1)\log(n))$ where in the YES case we have the additional property that A has full rank. The hardness of $(1-1/n^c)$-Approximate linear search problem with the same properties follows from the decision to search reduction from Lemma 5.9. ◀

We now prove Corollary 5.4:

Proof of Corollary 5.4. Note that the input size is $\tilde{O}(n^2)$ and hence $a \ge 2$. The corollary directly follows from noting that Lemma 5.8 applied for $\varepsilon(n) = 1/n^d$ and $\delta(n) = 1/n^c$ reduces $(1/n^d)$-Approximate Linear Search problem to $(1-1/n^d)$-Approximate Linear Search problem in time $\tilde{O}(n^2) = \tilde{O}(n^a)$. ◀

5.1 Sparse Matrices

Starting from the WordRAM version of Conjecture 1.7 and using Lemma 5.5 we can establish that there does not exist a $\tilde{o}(n^2)$ algorithm for $(1 - 1/\text{poly}(n))$-Approximate linear decision problem on sparse matrices in the WordRAM model. By the decision to search reduction in Lemma 5.9 we get that there does not exist a $\tilde{o}(n^2)$ algorithm for $(1 - 1/\text{poly}(n))$-Approximate linear decision search on sparse matrices in the WordRAM model. Note though that this hardness is trivial to obtain since there exist sparse linear systems such that every $(1 - 1/\text{poly}(n))$-approximate solution to the system requires $\Omega(n^2)$ bits to represent.

On the algorithmic side no improvement over the dense case algorithmic runtime of $O(n^\omega)$ was known until the recent result of Peng and Vempala [30] who gave an asymptotically faster algorithm for the $1/\text{poly}(n)$-approximate linear search problem.

References

1 Amir Abboud, Aviad Rubinstein, and R. Ryan Williams. Distributed PCP theorems for hardness of approximation in P. In Chris Umans, editor, *58th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2017, Berkeley, CA, USA, October 15-17, 2017*, pages 25–36. IEEE Computer Society, 2017. doi:10.1109/FOCS.2017.12.

2 Josh Alman and Ryan Williams. Probabilistic polynomials and hamming nearest neighbors. In Venkatesan Guruswami, editor, *IEEE 56th Annual Symposium on Foundations of Computer Science, FOCS 2015, Berkeley, CA, USA, 17-20 October, 2015*, pages 136–150. IEEE Computer Society, 2015. doi:10.1109/FOCS.2015.18.

3 Josh Alman and Virginia Vassilevska Williams. A refined laser method and faster matrix multiplication. *CoRR*, abs/2010.05846, 2020. arXiv:2010.05846.

4 Alexandr Andoni, Robert Krauthgamer, and Yosef Pogrow. On solving linear systems in sublinear time. In *10th Innovations in Theoretical Computer Science Conference, ITCS 2019, January 10-12, 2019, San Diego, California, USA*, pages 3:1–3:19, 2019.

5 Sanjeev Arora, Carsten Lund, Rajeev Motwani, Madhu Sudan, and Mario Szegedy. Proof verification and the hardness of approximation problems. *J. ACM*, 45(3):501–555, 1998. doi:10.1145/278298.278306.

6 Arturs Backurs and Piotr Indyk. Edit distance cannot be computed in strongly subquadratic time (unless SETH is false). *SIAM J. Comput.*, 47(3):1087–1097, 2018. doi:10.1137/15M1053128.

7 Walter Baur and Volker Strassen. The complexity of partial derivatives. *Theor. Comput. Sci.*, 22:317–330, 1983. doi:10.1016/0304-3975(83)90110-X.

8 Stavros Birmpilis, George Labahn, and Arne Storjohann. Deterministic reduction of integer nonsingular linear system solving to matrix multiplication. In *Proceedings of the 2019 on International Symposium on Symbolic and Algebraic Computation, ISSAC 2019, Beijing, China, July 15-18, 2019*, pages 58–65, 2019.

9 James R Bunch and John E Hopcroft. Triangular factorization and inversion by fast matrix multiplication. *Mathematics of Computation*, 28(125):231–236, 1974.

10 Lijie Chen, Shafi Goldwasser, Kaifeng Lyu, Guy N. Rothblum, and Aviad Rubinstein. Fine-grained complexity meets IP = PSPACE. In Timothy M. Chan, editor, *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2019, San Diego, California, USA, January 6-9, 2019*, pages 1–20. SIAM, 2019. doi:10.1137/1.9781611975482.1.

11 Ho Yee Cheung, Tsz Chiu Kwok, and Lap Chi Lau. Fast matrix rank algorithms and applications. *J. ACM*, 60(5):31:1–31:25, 2013. doi:10.1145/2528404.

12 Michael B. Cohen, Jonathan A. Kelner, Rasmus Kyng, John Peebles, Richard Peng, Anup B. Rao, and Aaron Sidford. Solving directed laplacian systems in nearly-linear time through sparse LU factorizations. In *59th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2018, Paris, France, October 7-9, 2018*, pages 898–909, 2018.

13 Michael B. Cohen, Jonathan A. Kelner, John Peebles, Richard Peng, Anup B. Rao, Aaron Sidford, and Adrian Vladu. Almost-linear-time algorithms for markov chains and new spectral primitives for directed graphs. In *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2017, Montreal, QC, Canada, June 19-23, 2017*, pages 410–419, 2017.

14 Don Coppersmith and Shmuel Winograd. Matrix multiplication via arithmetic progressions. *Journal of symbolic computation*, 9(3):251–280, 1990.

15 Wayne Eberly, Mark Giesbrecht, Pascal Giorgi, Arne Storjohann, and Gilles Villard. Faster inversion and other black box matrix computations using efficient block projections. In *Symbolic and Algebraic Computation, International Symposium, ISSAC 2007, Waterloo, Ontario, Canada, July 28 - August 1, 2007, Proceedings*, pages 143–150, 2007.

16 Alan Edelman. Eigenvalues and condition numbers of random matrices. *SIAM journal on matrix analysis and applications*, 9(4):543–560, 1988.

17 François Le Gall. Powers of tensors and fast matrix multiplication. In *International Symposium on Symbolic and Algebraic Computation, ISSAC '14, Kobe, Japan, July 23-25, 2014*, pages 296–303, 2014.

18 Johan Håstad. Some optimal inapproximability results. *J. ACM*, 48(4):798–859, 2001. doi:10.1145/502090.502098.

19 Magnus Rudolph Hestenes and Eduard Stiefel. *Methods of conjugate gradients for solving linear systems*, volume 49. DC: NBS, 1952.

20 Oscar H. Ibarra, Shlomo Moran, and Roger Hui. A generalization of the fast LUP matrix decomposition algorithm and applications. *J. Algorithms*, 3(1):45–56, 1982. doi:10.1016/0196-6774(82)90007-4.

21 Erich Kaltofen and B. David Saunders. On wiedemann's method of solving sparse linear systems. In Harold F. Mattson, Teo Mora, and T. R. N. Rao, editors, *Applied Algebra, Algebraic Algorithms and Error-Correcting Codes, 9th International Symposium, AAECC-9, New Orleans, LA, USA, October 7-11, 1991, Proceedings*, volume 539 of *Lecture Notes in Computer Science*, pages 29–38. Springer, 1991. doi:10.1007/3-540-54522-0_93.

22 Karthik C. S., Bundit Laekhanukit, and Pasin Manurangsi. On the parameterized complexity of approximating dominating set. In Ilias Diakonikolas, David Kempe, and Monika Henzinger, editors, *Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2018, Los Angeles, CA, USA, June 25-29, 2018*, pages 1283–1296. ACM, 2018. doi:10.1145/3188745.3188896.

23 Rasmus Kyng, Yin Tat Lee, Richard Peng, Sushant Sachdeva, and Daniel A. Spielman. Sparsified cholesky and multigrid solvers for connection laplacians. In *Proceedings of the 48th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2016, Cambridge, MA, USA, June 18-21, 2016*, pages 842–850, 2016.

24 Rasmus Kyng, Di Wang, and Peng Zhang. Packing lps are hard to solve accurately, assuming linear equations are hard. In Shuchi Chawla, editor, *Proceedings of the 2020 ACM-SIAM Symposium on Discrete Algorithms, SODA 2020, Salt Lake City, UT, USA, January 5-8, 2020*, pages 279–296. SIAM, 2020. doi:10.1137/1.9781611975994.17.

25 Rasmus Kyng and Peng Zhang. Hardness results for structured linear systems. In *58th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2017, Berkeley, CA, USA, October 15-17, 2017*, pages 684–695, 2017.

26 László Lovász. On determinants, matchings, and random algorithms. In *Fundamentals of Computation Theory, FCT 1979, Proceedings of the Conference on Algebraic, Arthmetic, and Categorial Methods in Computation Theory, Berlin/Wendisch-Rietz, Germany, September 17-21, 1979*, pages 565–574, 1979.

27 Marcin Mucha and Piotr Sankowski. Maximum matchings in planar graphs via gaussian elimination. *Algorithmica*, 45(1):3–20, 2006. doi:10.1007/s00453-005-1187-5.

28 Cameron Musco, Praneeth Netrapalli, Aaron Sidford, Shashanka Ubaru, and David P. Woodruff. Spectrum approximation beyond fast matrix multiplication: Algorithms and hardness. In Anna R. Karlin, editor, *9th Innovations in Theoretical Computer Science Conference, ITCS 2018, January 11-14, 2018, Cambridge, MA, USA*, volume 94 of *LIPIcs*, pages 8:1–8:21. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2018. doi:10.4230/LIPIcs.ITCS.2018.8.

29 Colton Pauderis and Arne Storjohann. Deterministic unimodularity certification. In Joris van der Hoeven and Mark van Hoeij, editors, *International Symposium on Symbolic and Algebraic Computation, ISSAC'12, Grenoble, France - July 22 - 25, 2012*, pages 281–288. ACM, 2012. doi:10.1145/2442829.2442870.

30 Richard Peng and Santosh S. Vempala. Solving sparse linear systems faster than matrix multiplication. In Dániel Marx, editor, *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms, SODA 2021, Virtual Conference, January 10 - 13, 2021*, pages 504–521. SIAM, 2021. doi:10.1137/1.9781611976465.31.

31 Liam Roditty and Virginia Vassilevska Williams. Fast approximation algorithms for the diameter and radius of sparse graphs. In Dan Boneh, Tim Roughgarden, and Joan Feigenbaum, editors, *Symposium on Theory of Computing Conference, STOC'13, Palo Alto, CA, USA, June 1-4, 2013*, pages 515–524. ACM, 2013. doi:10.1145/2488608.2488673.

32 Daniel A. Spielman and Shang-Hua Teng. Nearly linear time algorithms for preconditioning and solving symmetric, diagonally dominant linear systems. *SIAM J. Matrix Analysis Applications*, 35(3):835–885, 2014. doi:10.1137/090771430.

33 Arne Storjohann. The shifted number system for fast linear algebra on integer matrices. *J. Complex.*, 21(4):609–650, 2005. doi:10.1016/j.jco.2005.04.002.

34 Andrew James Stothers. On the complexity of matrix multiplication, 2010.

35 Volker Strassen. Gaussian elimination is not optimal. *Numerische mathematik*, 13(4):354–356, 1969.

36 Douglas H. Wiedemann. Solving sparse linear equations over finite fields. *IEEE Trans. Inf. Theory*, 32(1):54–62, 1986. doi:10.1109/TIT.1986.1057137.

37 Virginia Vassilevska Williams. Multiplying matrices in o (n2. 373) time, 2014.

38 Virginia Vassilevska Williams. On some fine-grained questions in algorithms and complexity. In *Proceedings of the ICM 2018*, pages 3447–3487. World Scientific, 2019.

39 Raphael Yuster and Uri Zwick. Fast sparse matrix multiplication. *ACM Trans. Algorithms*, 1(1):2–13, 2005. doi:10.1145/1077464.1077466.

Revisiting Priority k-Center: Fairness and Outliers

Tanvi Bajpai
University of Illinois, Urbana-Champaign, Urbana, IL, USA

Deeparnab Chakrabarty
Dartmouth College, Hanover, NH, USA

Chandra Chekuri
University of Illinois, Urbana-Champaign, Urbana, IL, USA

Maryam Negahbani
Dartmouth College, Hanover, NH, USA

Abstract

In the *Priority k-Center* problem, the input consists of a metric space (X, d), an integer k and for each point $v \in X$ a priority radius $r(v)$. The goal is to choose k-centers $S \subseteq X$ to *minimize* $\max_{v \in X} \frac{1}{r(v)} d(v, S)$. If all $r(v)$'s were uniform, one obtains the classical k-center problem. Plesník [32] introduced this problem and gave a 2-approximation algorithm matching the best possible algorithm for vanilla k-center. We show how the Priority k-Center problem is related to two different notions of *fair* clustering [23, 28]. Motivated by these developments we revisit the problem and, in our main technical contribution, develop a framework that yields constant factor approximation algorithms for Priority k-Center with *outliers*. Our framework extends to generalizations of Priority k-Center to matroid and knapsack constraints, and as a corollary, also yields algorithms with fairness guarantees in the lottery model of Harris et al.

2012 ACM Subject Classification Theory of computation → Facility location and clustering

Keywords and phrases Fairness, Clustering, Approximation, Outliers

Digital Object Identifier 10.4230/LIPIcs.ICALP.2021.21

Category Track A: Algorithms, Complexity and Games

Related Version *Full Version*: `https://arxiv.org/abs/2103.03337` [2]

Funding *Tanvi Bajpai*: Supported in part by NSF grant CCF-1910149.
Deeparnab Chakrabarty: Supported by NSF grants CCF-1813053 and CCF-2041920.
Chandra Chekuri: Supported by NSF grants CCF-1910149 and CCF-1907939.

1 Introduction

Clustering is a basic task in a variety of areas, and clustering problems are ubiquitous in practice, and are well-studied in algorithms and discrete optimization. Recently *fairness* has become an important concern as automated data analysis and decision making have become increasingly prevalent in society. This has motivated several problems in fair clustering and associated algorithmic challenges. In this paper, we show that two different fairness views are inherently connected with a previously studied clustering problem called the *Priority k-Center* problem.

The input to Priority k-Center is a metric space (X, d) and a priority radius $r(v)$ for each $v \in X$. The objective is to choose k-centers $S \subseteq X$ such that $\max_{v \in X} \frac{d(v,S)}{r(v)}$ is minimized. If one imagines clients located at each point in X, and $r(v)$ is the "speed" of a client at point v, then the objective is to open k-centers so that every client can reach an open center as quickly as possible. When all the $r(v)$'s are the same, then one obtains the classic

k-center problem [26]. Plesník [32] introduced this problem and showed how to generalize Hochbaum and Shmoys' [26] 2-approximation algorithm for the k-center problem, to obtain a 2-approximation for Priority k-Center. This approximation ratio is tight since $(2-\varepsilon)$-factor approximation is ruled out even for the classic k-center problem under the assumption that $P \neq NP$ [19, 25].

Connections to Fair Clustering. Our motivation to revisit Priority k-Center came from two recent papers that considered fair variants in clustering, without explicitly realizing the connection to Priority k-Center. One of them is the paper of Jung, Kannan and Lutz [28] who defined a version of fair clustering as follows. Given (X, d) representing clients/people in a geographic area, and an integer k, for each $v \in X$ let $r_\ell(v)$ denote the smallest radius r such that there are at least ℓ points of X inside a ball of radius r around v. They suggested a notion of fair k-clustering as one in which each point $v \in X$ should be served by a center not farther than $r(v) = r_{n/k}(v)$ since the average size of a cluster in a k-clustering is n/k. [28] describe an algorithm that finds k centers such that each point v is served by a center at most distance $\leq 2r_{n/k}(v)$ away from v. Once the radii are fixed for the points, then one obtains an instance of Priority k-Center, and the result essentially[1] follows from the algorithm in [32]; indeed, the algorithm in [28] is the same.

Another notion of fairness related to the Priority k-Center is the *lottery model* introduced by Harris et al. [22]. In this model, every client $v \in X$ has a "probability demand" $p(v)$ and a "distance demand" $r(v)$. The objective is to find a *distribution* $\mathcal{S}$ over k-center locations such that for every client $v \in X$, $\Pr_{S\sim\mathcal{S}}[d(v, S) \leq r(v)] \geq p(v)$. One needs to either prove such a solution is not possible, or provide a distribution where the distance to S can be relaxed to $\alpha r(v)$. Using a by now almost standard reduction via the ellipsoid method [6, 1], this boils down to the *outlier* version of Priority k-Center, where some points in X are allowed to be discarded. The outlier version of Priority k-Center had not been explicitly studied before.

Our Contributions. Motivated by these connections to fairness, we study the natural generalizations of Priority k-Center that have been studied for the classical k-center problem. The main generalization is the *outlier* version of Priority k-Center: the algorithm is allowed to discard a certain number of points when evaluating the quality of the centers chosen. First, the outlier version arises in the lottery model of fairness. Second, in many situations it is useful and important to discard outliers to obtain a better solution. Finally, it is also interesting from a technical point of view. We also consider the situation when the constraint on where centers can be opened is more general than the cardinality constraint. In particular, we study the matroid priority center problem where the set of centers must be an independent set of a given matroid, and the knapsack priority center problem where the total weight of centers opened is at most a certain amount. Our main contribution is an algorithmic framework to study the outlier problems in all these variations. Our results also imply interesting generalizations for fair clustering.

1.1 Statement of Results

We briefly describe some variants of Priority k-Center. In the supplier version, the metric space is partitioned into facilities F and clients C, and goal is to select k facilities $S \subseteq F$ to minimize $\max_{v\in C} d(v, S)/r(v)$. In the Priority Matroid Supplier problem, the subset of

[1] One needs to observe that Plesník's analysis [32] can be made with respect to a natural LP which has a feasible solution with $r(v) := r_{n/k}(v)$.

facilities need to be an independent set of matroid on F. In the Priority Knapsack Supplier problem, the subset of facilities must have weight at most a certain amount. All these generalizations have a 3-approximation [26, 14] in the vanilla version where all $r(v)$'s are the same. Our first observation is that these extend to Priority k-Center in a simple fashion. This result also implicitly relates the approximation ratio to the integrality gap of the natural LP relaxation. This allows us to rederive and extend the algorithmic results in [28] we give details in Section 6.

▶ **Result 1.** *There is a* 3*-approximation for Priority* k*-Supplier and Priority Matroid Supplier and Priority Knapsack Supplier.*

Our second, and the main technical contribution, is a general framework to handle *outliers*. Given an instance of Priority k-Center and an integer $m \leq n$, the outlier version that we refer to as PkCO, is to find k centers S and a set C' of at least an m points from C such that $\max_{v \in C'} \frac{1}{r(v)} d(v, S)$ is minimized. While the k-Center with outliers admits a clever, yet relatively simple, greedy 3-approximation due to Charikar et al. [10], a similar approach seems difficult to adapt for Priority k-Center. Instead, we take a more general and powerful LP-based approach from [7, 8] to develop a framework to handle PkCO, and also the outlier version of Matroid Center (PMCO), where the opened centers must be an independent set, and Knapsack Center (PKnapCO), where the total weight of the open centers must fit in a budget. We obtain the following results.

▶ **Result 2.** *There is a* 9*-approximation for P*k*CO and PMCO and a* 14*-approximation for PKnapCO. Moreover the approximation ratio for P*k*CO and PMCO are based on a natural LP relaxation.*

At this point we remark that a result in Harris et al. [22] (Theorem 2.8 in the arXiv version) also indirectly gives a 9-approximation for PkCO. We believe that our framework is more general and can handle PMCO and PKnapCO easily. The [22] paper do not consider these versions, and indeed for the PKnapCO problem their framework cannot give a constant factor approximation for they (in essence) use a weak LP relaxation.

Furthermore, our framework yields *better* approximation factors when either the number of distinct priorities are small, or they are in different scales. In practice, one indeed expects this to be the case. In particular, when there are only two distinct types of radii, then we get a 3-approximation which is tight; it is not too hard to show that it is NP-hard to obtain a better than 3-approximation for PkCO with two types[2] of priorities. We get improved factors (5 and 7) when the number of radii are three and four as well. On the other hand, if all the different priorities are powers of b (for some parameter $b > 1$), then we get a $\frac{3b-1}{b-1}$-approximation. Thus, if all the priorities are in vastly different scales ($b \to \infty$), then our approximation factor approaches 3.

▶ **Result 3.** *Suppose there are only two distinct priority radii among the clients. Then there is a* 3*-approximation for P*k*CO, PMCO and PKnapCO. With* t *distinct types of priorities, the approximation factor for P*k*CO and PMCO is* $2t - 1$*. If all distinct types are powers of* b*, the approximation factor for P*k*CO and PMCO becomes* $(3b - 1)/(b - 1)$*.*

It is possible that the PkCO problem has a 3-approximation in general, and even the natural LP-relaxation may suffice; we have not been able to obtain a worse than 3 integrality gap example. As we explain in Section 1.2 below, many approaches to the k-center type problems

[2] Interestingly, when there is a single priority, the vanilla k-center with outliers has a 2-approximation [7] showing a gap between the two problems.

begin with a Hochbaum-Shmoys [26] style partition of the points X to representatives. We could show examples where such an *approach* has a gap worse than 3, though not showing an integrality gap instance. Resolving the integrality gap of the natural LP-relaxation and/or obtaiting improved approximation ratios are interesting open questions highlighted by our work.

1.2 Technical Discussion

Almost all clustering algorithms for the k-center objective proceeds via a partitioning subroutine due to Hochbaum and Shmoys [26] (HS, henceforth). This procedure returns a partition Π of X along with a representative for each part such that all vertices of a part "piggy-back" on the representative. More precisely, if the representative is assigned to a center $f \in X$, then so are all other vertices in that part. To ensure a good algorithm in vanilla k-center, it suffices to ensure the radius of each part is small.

For the Priority k-Center objective, one needs to be more careful : to use the above idea, one needs to make sure that if vertex v is piggybacking on vertex u, then $r(v)$ better be more than $r(u)$. Indeed, this can be ensured by running the HS procedure in a particular order, namely by allowing vertices with smaller $r(v)$ to form the parts first. This precisely gives Plesník's algorithm [32]. In fact, this idea easily gives a 3-approximation for the matroid and supplier versions as well.

Outliers are challenging in the setting of Priority k-Center. We start with the approach of Chakrabarty et al. [7] for k-center. First, they construct an LP where $\mathsf{cov}(v)$ denotes the fractional coverage (amount to which one is *not* an outlier) of any point, and then write a natural LP for it. They show that if the HS algorithm is run according to the $\mathsf{cov}(v)$ order (higher coverage vertices first), then the resulting partition can be used to obtain a 2-approximation for the k-center with outliers problem.

When one moves to the priority k-center with outliers, one sees the obvious trouble: what if the $r(v)$ order and the $\mathsf{cov}(v)$ order are at loggerheads? Our approach out of this is a simple bucketing idea. We first write a natural LP with fractional coverages $\mathsf{cov}(v)$ for every point. Then, we partition vertices into classes: all vertices v with $r(v)$ between 2^i and 2^{i+1} are in the same class. We then use the HS partitioning algorithm in the decreasing $\mathsf{cov}(v)$ order separately on each class. The issue now is to handle the interaction across classes. To handle this, we define a directed acyclic graph across these various partitions where representative u has an edge to representative v iff $d(u, v)$ is small ($\leq r(u) + r(v)$). It is a DAG because we point edges from higher $r(u)$ to the lower $r(v)$. Our main observation is that if we can peel out k paths with "large value" (each representative's value is how many points piggyback on it), then we can get a 9-approximation for the priority k-center with outlier problem. We can show that a *fractional* solution of large value does exist using the fact that the DAG was constructed in a greedy fashion. Also, since the graph is a DAG, this LP is an integral min-cost max-flow LP. The factor 9 arises out of a geometric series and bucketing. Indeed, when the radii are exact powers of 2, we get a 5-approximation, and when there are only two type of radii, we get a 3 approximation which is tight.

The above framework can handle the outlier versions for the matroid and knapsack version. For the matroid version, the flow problem is no longer a min-cost max-flow problem, but rather it reduces to a *submodular flow* problem which is solvable in polynomial time. Modulo this, the above framework gives a 9-approximation. For the knapsack version, there are two issues. One is that the flow problem involves non-uniform numbers and is no longer integral and solving the underlying optimization problem is likely to be NP-hard (we did not attempt a formal proof). Nevertheless, our framework has sufficient flexibility that by

increasing the approximation factor from 9 to 14, the DAG can in fact be made into a rooted forest. In this rooted forest, we can employ dynamic programming to solve the problem of finding the desired paths. The second issue is that a fractional LP solution of the natural LP does not suffice when using the DP based algorithm on the forest; indeed the natural LP has an unbounded gap. Here we need to use the round-or-cut framework from [8]; either the DP on the rooted forest succeeds or we find a violated inequality for the large implicit LP that we use.

1.3 Other Related Works

There is a huge literature on clustering, and instead of summarizing the landscape, we mention a few works relevant to our paper. Gørtz and Wirth [20] study the priorty k-center problem in the asymmetric metric case, and prove that it is NP-hard to obtain any non-trivial approximation. A related problem to priority k-clustering is the *non-uniform* k-center problem by Chakrabarty et al. [7] where instead of clients having radii bounds, the objective is to figure out centers of balls for different types of radii. Another related problem [21] is the local k-median problem where clients need to connect to facilities within a certain radius, but the objective is the sum instead of the max.

Fairness in clustering has also seen a lot of works recently. Apart from the two notions of fairness described above, which can be thought of as "individual fairness" guarantees, Chierichetti et al. [16] introduce the "group fairness" notion where points have color classes, and each cluster needs to contain similar proportion of colors as in the universe. Their results were generalized by a series of follow ups [33, 5, 4]. A similar concept for outliers led to the study of *fair colorful k-center*. In this problem, the objective is to find k centers which covers at least a prescribed number of points from each color class. This was introduced by Bandapadhyay et al. [3], and recently true approximation algorithms were concurrently obtained by Jia et al. [27] and Anegg et al. [1].

Another notion of fairness is introduced by Chen et al. [15] in which a solution is called fair if there is no facility and a group of at least n/k clients, such that opening that facility lowers the cost of all members of the group. They give a $(1+\sqrt{2})$-approximation for L_1, L_2, and L_∞ norm distances for the setting where facilities can be places anywhere in the real space. Recently Micha and Shah [31] showed that a modification of the same approach can give a close to 2-approximation for L_2 case and proved $(1+\sqrt{2})$ factor is tight for L_1 and L_∞.

Coming back to the model of Jung et al. [28], the local notion of neighborhood radius is also present in the metric embedding works of [9, 11] and were recently used by Mahabadi and Vakilian [30] to extend the results in [28] to other objectives such as k-median and k-means. We leave the outlier versions of these problems as an open direction of study.

2 Preliminaries

We provide some formal definitions and describe a clustering routine from [26].

▶ **Definition 1** (Priority $k-$Center). *The input is a metric space (X,d) and radius function $r: X \to \mathbb{R}^+$, and integer k. The goal is to find $S \subseteq X$ of size at most k to minimize α such that for all $v \in X$, $d(v,S) \leq \alpha \cdot r(v)$*

▶ **Definition 2** (Priority $\mathscr{F}$-supplier). *(Generalization from [8]). The input is a metric space (X,d) where $X = F \cup C$, C is the set of points, and F the set of facilities. We are also given a radius function $r: C \to \mathbb{R}^+$. The goal is to find $S \subseteq F$ to minimize α such that for all*

$v \in C$, $d(v, S) \leq \alpha \cdot r(v)$. The constraint on F is that it must be selected from a down-ward closed family $\mathscr{F}$. Different families lead to different problems. We get the priority k-supplier problem if $\mathscr{F} = \{F : |F| \leq k\}$. We get the priority matroid supplier problem when $(F, \mathscr{F})$ is a matroid. We get the priority knapsack supplier problem when there is a weight function $w : F \to \mathbb{R}_{\geq 0}$ and $\mathscr{F} = \{F : w(F) \leq B\}$ for some budget B.

For the remainder of this manuscript, we focus on the *feasibility* version of the problem. More precisely, given an instance of the problem, we either want to show there is no solution with $\alpha = 1$, or find a solution with $\alpha \leq \rho$. If we succeed, then via binary search we get a ρ-approximation.

Plesník [32] obtained a 2-approximation for Priority k-Center. Algorithm 1 is a slight generalization of his algorithm; in addition to the radius function and the metric, we take as input a function $\phi : X \to R_{\geq 0}$ which encodes an ordering over the points (we can think of the points as being ordered from largest to smallest ϕ values). The algorithm is a similar procedure to that of Hochbaum and Shmoys from [26], but while [26] picks points arbitrarily, points get picked in the order mandated by ϕ.

▶ **Fact 1.** *The following is true for the output of HS: (a) $\forall u, v \in S, d(u, v) > r_u + r_v$, (b) The set $\{D(u) : u \in S\}$ partitions X, (c) $\forall u \in S, \forall v \in D(u), \phi(u) \geq \phi(v)$, and (d) $\forall u \in S, \forall v \in D(u), d(u, v) \leq r_u + r_v$.*

Algorithm 1 HS.

Input: Metric (X, d), radius function $r : X \to \mathbb{R}_{>0}$, and ordering $\phi : X \to \mathbb{R}_{\geq 0}$

$U \leftarrow X$ ▷ The set of uncovered points
$S \leftarrow \emptyset$ ▷ The set of "representatives"
while $U \neq \emptyset$ **do**
  $u \leftarrow \arg\max_{v \in U} \phi(v)$ ▷ The first point in U in non-increasing ϕ order
  $S \leftarrow S \cup u$
  $D(u) \leftarrow \{v \in U : d(u, v) \leq r_u + r_v\}$ ▷ Note: $D(u)$ includes u itself
  $U \leftarrow U \backslash D(u)$
end while

Output: S, $\{D(u) : u \in S\}$

▶ **Theorem 3** ([32]). *There is a 2-approximation for Priority k-Center.*

Proof. (For completeness and later use.) We claim that S, the output of Algorithm 1 for $\phi := 1/r$, is a 2-approximate solution; this follows from the observations in Fact 1. For any $v \in X$ there is some $u \in S$ for which $v \in D(u)$. By our choice of ϕ, $r_u \leq r_v$. Since $d(u, v) \leq r_u + r_v$, we have $d(u, v) \leq 2r_v$. To see why $|S| \leq k$, recall that for any $u, v \in S$, by Fact 1, $d(u, v) > r_u + r_v$ so no two points in S can be covered by the same center. Thus any feasible solution needs at least $|S|$ many points to cover all of S. ◀

In fact, the algorithm almost immediately gives a 3-approximation for Priority $\mathscr{F}$-Supplier for many families via the framework in [8].

One needs to check if given any partition Π of F, whether the following *partition feasibility* problem is solvable: does there exist $A \in \mathscr{F}$ such that $|A \cap P| = 1$ for all $P \in \Pi$? We ask this for the partition returned by Algorithm 1, that is, $\Pi = \{\{f \in F : d(f, u) \leq r_u\} : u \in S\}$. If no such A exists, then the instance is infeasible since the centers S of the parts cannot be covered. If such an A exists, then by construction every $v \in X$ in part $D(u)$ satisfies

$d(v, A) \leq d(u, v) + d(u, A) \leq 2r_u + r_v \leq 3r_v$ since $r_u \leq r_v$. It is easy to see for the supplier, knapsack, and matroid center versions, the partition feasibility problem is solvable in polynomial time. This leads to the following theorem.

▶ **Theorem 4.** *There is a 3-approximation for Priority k-Supplier, Priority Knapsack Center, and the Priority Matroid Center problem.*

3 Priority k-Center with Outliers

In this section we describe our framework for handling priorities and outliers and give a 9-approximation algorithm for the following problem.

▶ **Definition 5** (Priority $k-$Center with Outliers (PkCO))**.** *The input is a metric space (X, d), a radius function $r : X \to \mathbb{R}_{>0}$, and parameters $k, m \in \mathbb{N}$. The goal is to find $S \subseteq X$ of size at most k to minimize α such that for at least m points $v \in X$, $d(v, S) \leq \alpha \cdot r(v)$.*

▶ **Theorem 6.** *There is a 9-approximation for PkCO.*

The following is the natural LP relaxation for the feasibility version of PkCO. For each point $v \in X$, there is a variable $0 \leq x_v \leq 1$ that denotes the (fractional) amount by which v is opened as a center. We use $\mathsf{cov}(v)$ to indicate the amount by which v is *covered* by itself or other open facilities. To be precise, $\mathsf{cov}(v)$ is the sum of x_u over all $u \in X$ at distance at most r_v from v. Note that $\mathsf{cov}(v)$ is an auxiliary variable. We want to ensure that at least m units of coverage are assigned using at most k centers (hence the first two constraints).

$$\sum_{v \in X} \mathsf{cov}(v) \geq m \qquad \text{(P}k\text{CO LP)}$$

$$\sum_{v \in X} x_v \leq k$$

$$\mathsf{cov}(v) := \sum_{\substack{u \in X: \\ d(u,v) \leq r_v}} x_u \leq 1 \qquad \forall v \in X$$

$$0 \leq x_v \leq 1 \qquad \forall v \in X.$$

Next, we define another problem called Weighted k-Path Packing (WkPP) on a DAG. Our approach is to do an LP-aware reduction from PkCO to WkPP. To be precise, we use a fractional solution of the PkCO LP to reduce to a WkPP instance $\mathcal{J}$. We show that a good integral solution for $\mathcal{J}$ translates to a 9-approximate solution for the PkCO instance. We prove that $\mathcal{J}$ has a good integral solution by constructing a feasible fractional solution for an LP relaxation of WkPP; this LP relaxation is integral. Henceforth, $\mathcal{P}(G)$ denotes the set of all the paths in G where each path is an ordered subset of the edges in G.

▶ **Definition 7** (Weighted k-Path Packing (WkPP))**.** *The input is $\mathcal{J} = (G = (V, E), \lambda, k)$ where G is a DAG, $\lambda : V \to \{0, 1, \ldots, n\}$ for some integer n. The goal is to find a set of k* vertex *disjoint paths $P \subseteq \mathcal{P}(G)$ that maximizes:*

$$\mathit{val}(P) := \sum_{p \in P} \sum_{v \in p} \lambda(v).$$

Even though this problem is NP-hard on general graphs[3], it can be easily solved if G is a DAG by reducing to Min-Cost Max-Flow (MCMF). To build the corresponding flow network,

[3] $k = 1$ and unit λ is the longest path problem which is known to be NP-hard [18].

we augment G to a new DAG $G' = (V', E')$ with source and sink nodes s, t. $V' = V \cup \{s, t\}$. Each node $v \in V$ has unit capacity and cost equal to $-\lambda(v)$. s and t have zero cost with capacities ∞ and k respectively. As for the arcs, E' includes the entirety of E, plus arcs (s, v) and (v, t) for all $v \in V$. All the arcs have unit capacity and zero cost. One can now write the MCMF LP for WkPP which is known to be integral. We use $\delta^+(v)$ and $\delta^-(v)$ to denote the set of outgoing and incoming edges of a vertex v respectively. The LP has a variable y_e for each arc $e \in E'$ to denote the amount of (fractional) flow passing through it. Similarly, the amount of flow entering a vertex is denoted by $\mathsf{flow}(v) := \sum_{e \in \delta^-(v)} y_e$. The objective is to minimize the cost of the flow which is equivalent to maximizing the negation of the costs.

$$\max \sum_{v \in V} \mathsf{flow}(v)\lambda(v) \qquad \text{(W}k\text{PP LP)}$$

$$\mathsf{flow}(v) := \sum_{e \in \delta^-(v)} y_e = \sum_{e \in \delta^+(v)} y_e \qquad \forall v \in V$$

$$\mathsf{flow}(t) \leq k$$

$$\mathsf{flow}(v) \leq 1 \quad \forall v \in V, \qquad 0 \leq y_e \leq 1 \quad \forall e \in E'.$$

▷ Claim 8. WkPP is equivalent to solving MCMF on G'.

Proof. Observe that any solution P for the WkPP instance translates to a valid flow of cost $-\mathsf{val}(P)$ for the flow problem. For any path $p \in P$ with start vertex u and sink vertex v, send one unit of flow from s to u, through p to v and then to t. Since the paths in P are vertex disjoint and there are at most k of them, the edge and vertex capacity constraints in the network are satisfied.

Now we argue that any solution to the MCMF instance with cost $-m$ translates to a solution P for the original WkPP instance with $\mathsf{val}(P) = m$. To see this, note that the MCMF solution consists of at most k many s, t paths that are vertex disjoint with respect to V. This is because of our choice of vertex capacities. Let P be those paths modulo vertices s and t. For a $v \in V$, $-\lambda(v)$ is counted towards the MCMF cost iff v has a flow passing through it which means v is included in some path in P. Thus $\mathsf{val}(P) = m$. ◁

3.1 Reduction to WkPP

Using a fractional solution of the PkCO LP we construct a WkPP instance. In particular, we use the cov assignment generated by the LP solution. Without loss of generality, by scaling the distances, we assume that the smallest neighborhood radius is 1. Let $t := \lceil \log_2 r_{max} \rceil$, where r_{max} is the largest value of r (after scaling). We use $[t]$ to denote $\{1, 2, \ldots, t\}$. Partition X according to each point's radius into $C_1, \ldots, C_t$, where $C_i := \{v \in X : 2^{i-1} \leq r_v < 2^i\}$ for $i \in [t]$. Note that some sets may be empty if no radius falls within its range.

Algorithm 2 shows the PkCO to WkPP reduction. The algorithm constructs a DAG called contact DAG (see Definition 9) as a part of the WkPP instance definition. We first run Algorithm 1 on each C_i to produce a set of representatives R_i and their respective clusters $\{D(u) : u \in R_i\}$. The λ function is constructed using the $D(v)$'s. Each R_i defines a *row* of the contact DAG starting with R_t at the top. Arcs in the contact DAG exist only between points in different rows, and only when they share a point in X that can cover them both within their desired radii. We always have arcs pointing downwards, that is, from points in R_i to points in R_j where $i > j$. See Figure 1 for an example on how a contact DAG looks like.

Algorithm 2 Reduction to WkPP.

Input: PkCO instance $\mathcal{I} = ((X, d), r, k, m)$ and assignment $\{\mathsf{cov}(v) \in \mathbb{R}_{\geq 0} : v \in X\}$
1: $R_i, \{D(u) : u \in R_i\} \leftarrow \mathsf{HS}((C_i, d), r, \mathsf{cov})$ for all $i \in [t]$
2: Construct contact DAG $G = (V, E)$ per Definition 9
3: $\lambda(v) \leftarrow |D(v)|$ for all $v \in V$
Output: WkPP instance $\mathcal{J} = (G = (V, E), \lambda, k)$

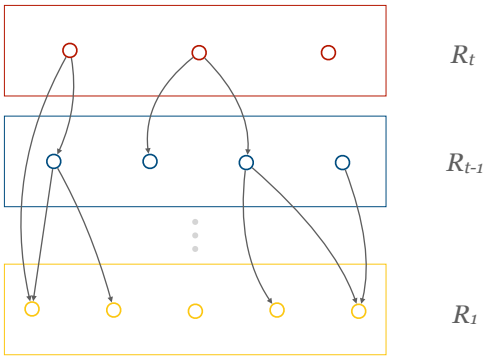


Figure 1 A contact DAG.

▶ **Definition 9** (contact DAG). *Let $R_i \subseteq C_i$, $i \in [t]$ be the set of representatives acquired after running HS on C_i according to Line 1 of Algorithm 2. contact DAG $G = (V, E)$ is a DAG on vertex set $V = \bigcup_i R_i$ where the arcs are constructed by the following rule:*

For $u \in R_i$ and $v \in R_j$ where $i > j$, $(u, v) \in E$
$\iff \exists f \in X : d(u, f) \leq r_u$ and $d(v, f) \leq r_v$.

Our first observation is that the WkPP instance has a good fractional solution and since the LP is integral, it also has a good integral solution.

▶ **Lemma 10.** *There is a valid solution to WkPP LP of value $\geq m$ for the WkPP instance $\mathcal{J}$. Since WkPP LP is integral, this implies $\mathcal{J}$ has an integral solution of value $\geq m$.*

The proof of this lemma can be found in [2]. Theorem 6 now follows from the following lemma.

▶ **Lemma 11.** *Any solution with value at least m for the WkPP instance $\mathcal{J}$ given by Algorithm 2 translates to a 9-approximation for the PkCO instance $\mathcal{I}$.*

Proof. We begin with a few observations. Per definition of contact DAG we have the following property. Note that the converse is not necessarily true.

▶ **Fact 2.** *If $u \in R_i$, $v \in R_j$, and (u, v) is an arc in contact DAG, $d(u, v) \leq r_u + r_v$.*

▶ **Fact 3.** *$\{D(v), v \in V\}$ as constructed in Algorithm 2 partitions X.*

Proof. $\{C_i\}_{i \in [t]}$ partitions X and HS further partitions each C_i according to Fact 1. ◀

▷ Claim 12. For any $u \in R_i$, $v \in R_j$ reachable from u in a contact DAG, $d(u, v) < 3 \cdot 2^i$.

Proof. Observe that by definition of contact DAG, $i > j$. A path from u to v may contain a vertex from any level of the DAG between i and j. In the worst case, the path has a vertex w_k from every level R_k for $j < k < i$:

$$\begin{aligned} d(u,v) &\le d(u,w_{i-1}) + d(w_{i-1},w_{i-2}) + \ldots + d(w_{j+1},v) \\ &\le (r_u + r_{w_{i-1}}) + (r_{w_{i-1}} + r_{w_{i-2}}) + \ldots + (r_{w_{j+1}} + r_v) && \text{(by Fact 2)} \\ &= r_u + 2\sum_{k=j+1}^{i-1} r_{w_k} + r_v < r_u + 2\sum_{k=1}^{i-1} 2^k \\ &= r_u + 2\cdot(2^i - 2) < 3\cdot 2^i. && (r_u < 2^i) \end{aligned}$$ ◁

Now we are armed with all the facts we need to prove Lemma 11. We are assuming the constructed WkPP instance has a solution of value at least m, which means there exists a set of k disjoint paths $P \subseteq \mathcal{P}(G)$ in the contact DAG such that $\mathsf{val}(P) \ge m$. For any path $p \in P$, let $\mathsf{sink}(p)$ denote the last node in this path (i.e. $\mathsf{sink}(p) = \arg\min_{u \in p} r_u$). Our final solution would be $S := \{\mathsf{sink}(p) : p \in P\}$. We argue that this S is a 9-approximate solution for the initial PkCO instance. Since P has at most k many paths, $|S| \le k$.

Now we show any $w \in D(u)$ where $u \in p \in P$, can be covered by $v = \mathsf{sink}(p)$ with dilation at most 9. Assume $u \in R_i$ for some $i \in [t]$.

$$\begin{aligned} d(w,v) &\le d(w,u) + d(u,v) < r_w + r_u + 3\cdot 2^i && \text{(by Fact 1 and above claim)} \\ &< r_w + 4\cdot 2^i \le 9 r_w. && (r_u < 2^i \text{ and } 2^{i-1} \le r_w) \end{aligned}$$

The last piece is to argue at least m points will be covered by S. The set of points that are covered by S within 9 times their radius is precisely the set $D_{\mathsf{total}} := \bigcup_{p\in P}\bigcup_{u\in p} D(v)$. So we need to show $|D_{\mathsf{total}}| \ge m$. By Fact 3 we have:

$$|D_{\mathsf{total}}| = |\bigcup_{p\in P}\bigcup_{v\in p} D(v)| = \sum_{p\in P}\sum_{v\in p} |D(v)| = \mathsf{val}(P),$$

where the last equality is by the definition of $\lambda(v), v \in V$ (in Line 3 of Algorithm 2) and definition of $\mathsf{val}(P)$. By assumption $\mathsf{val}(P) \ge m$ thus D_{total} contains at least m points. ◀

In the special case where there are 2 types of radii we can slightly modify our approach to get a 3-approximation algorithm. This result is tight. To see this consider PkCO instances where clients having priority radii in $\{0, 1\}$ with n_0 of the former type and n_1 of the latter, and the number of outliers allowed is $n_0 - k$. Clients with priority radii 0 either need to have a facility opened at that same point, or need to be an outlier. Since only $n_0 - k$ outliers and k centers are allowed, all the outliers and centers are on these n_0 points. Thus, the n_0 points act as facilities in the k-supplier problem which is hard to approximate with a factor better than 3. This shows a gap with the vanilla k-center with outliers has a 2-approximation [7].

In general, our framework yields improved approximation factors when the number of distinct priorities are less than 5 (see Theorem 13). In the special case when all radii are powers of 2, our algorithm is actually a 5-approximation. This factor improves if the radii are powers of some $b > 2$ and approaches 3 as b goes to infinity (see Theorem 14).

▶ **Theorem 13.** *There is a $(2t-1)$-approximation for PkCO instances where there are only t types of radii.*

Proof. Given PkCO instance $\mathcal{I}$ obtain fractional solution x by solving the PkCO LP. Partition X according to each point's radius into $C_1, \ldots, C_t$, where C_i is points of radius type i for $i \in [t]$. Run Algorithm 2 with input cov corresponding to x and take resulting WkPP instance

$\mathcal{J}$. Assuming the WkPP instance has a solution with value at least m, we can show how to obtain a $(2t-1)$-approximate solution as follows. Let P be the WkPP solution. Take any $p \in P$. If p is a single vertex, simply add it to solution S. Otherwise, instead of adding $v' = \mathsf{sink}(p)$ to S, if v is the vertex before v' in p, add a point $f \in X$ that covers both the endpoints v and v' (f exists by Definition 9).

Take any $w \in D(u)$ where $u \in p$ and assume $u \in R_i$ for some $i \in [t]$. Similar to the proof of Claim 12 one can show $d(u,v) < 2(i-2)r_u$ by bounding the radius of any vertex in between them by r_u and noting that v is in level 2 or higher (remember p ends at v' and (v,v') is an edge). Since $d(w,f) \le d(w,u) + d(u,v) + d(v,f)$ and $d(w,u) \le r_w + r_u$ (Fact 1), plus $d(v,f) < r_v \le r_u$ we have $d(w,f) < r_w + 2(i-1)r_u$. But $r_w = r_u$ by definition of C_i so w is covered by f with dilation at most $2i-1 \le 2t-1$. The part to argue at least m points will be covered by S, is done similar to the proof of Lemma 11.

The remainder of this proof, i.e. showing that $\mathcal{J}$ does indeed have a solution of value at least m that can be determined in polynomial time using an MCMF algorithm, is identical to the proof of Theorem 6. ◀

▶ **Theorem 14.** *There is a $((3b-1)/(b-1))$-approximation for PkCO instances where the radii are powers of $b \ge 2$.*

Proof. Given PkCO instance $\mathcal{I}$ obtain fractional solution x by solving the PkCO LP. Partition X according to each point's radius into $C_1, \ldots, C_t$, where $t := \lceil \log_b r_{max} \rceil$ and $C_i := \{v \in X : r_v = b^{i-1}\}$ for $i \in [t]$. Run Algorithm 2 with input cov corresponding to x and take resulting WkPP instance $\mathcal{J}$. Assume the WkPP instance has a solution P with value at least m. For any $p \in P$ add $v = \mathsf{sink}(p)$ to solution S. Consider arbitrary $w \in D(u)$ where $u \in p \in P$ and assume $u \in R_i$ for some $i \in [t]$. Similar to the proof of Claim 12 one can show $d(u,v) < ((b+1)/(b-1)) \times b^{i-1}$. By Fact 1 $d(w,u) \le r_w + r_u = 2b^{i-1}$. Thus any w is covered by dilation $(3b-1)/(b-1)$ as $d(w,v) \le d(w,u) + d(u,v) < 2b^{i-1} + ((b+1)/(b-1)) \times b^{i-1} = (3b-1)/(b-1)r_w$. To argue at least m points will be covered by S, see the proof of Lemma 11. Showing that $\mathcal{J}$ does indeed have a solution of value at least m that can be determined in polynomial time using an MCMF algorithm, is identical to the proof of Theorem 6 as well. ◀

4 Priority Matroid-Center with Outliers

In this section, we show how to generalize the results from the previous section for the case of Priority Matroid-Center with Outliers (PMCO).

▶ **Definition 15** (Priority Matroid-Center with Outliers (PMCO))**.** *The input is a metric space (X,d), parameter $m \in \mathbb{N}$, radius function $r : X \to \mathbb{R}_{>0}$, and $\mathscr{F} \subseteq 2^X$ a family of independent sets of a matroid. The goal is to find $S \in \mathscr{F}$ to minimize α such that for at least m points $v \in X$, $d(v,S) \le \alpha \cdot r(v)$.*

▶ **Theorem 16.** *There is a 9-approximation for PMCO.*

As in the previous section, we assume $\alpha = 1$ and consider the feasibility version of the problem. For any $S \subseteq V$, let $\mathsf{rank}_{\mathscr{F}}(S)$ be the rank of S in the given matroid. The natural LP relaxation for this problem is very similar to that of PkCO LP except that we replace the cardinality constraints with *rank constraints* $x(S) \le \mathsf{rank}_{\mathscr{F}}(S)$ for all $S \subseteq V$. This is because for any $S \in \mathscr{F}$, $|S| = \mathsf{rank}_{\mathscr{F}}(S)$.

$$\sum_{v \in X} \mathsf{cov}(v) \geq m \qquad \text{(PMCO LP)}$$

$$\sum_{v \in S} x_v \leq \mathsf{rank}_{\mathscr{F}}(S) \qquad \forall S \subseteq V$$

$$\mathsf{cov}(v) := \sum_{\substack{u \in X: \\ d(u,v) \leq r_v}} x_u \leq 1 \qquad \forall v \in X$$

$$0 \leq x_v \leq 1 \qquad \forall v \in X.$$

Similar to WkPP, we have the path packing version of PMCO defined below. Recall from the last section, that after reducing from PkCO to WkPP we returned a set of k vertices in DAG G as our final solution. Now that we have matroid constraints, we must instead return a set S of vertices such that $S \in \mathscr{F}$. Doing so is not as straightforward, since our reduction does not guarantee that such a subset of vertices actually exists and covers enough points in their corresponding vertex disjoint paths. Instead, we show there is an $S \in \mathscr{F}$ such that each member of this S is *close* to some vertex of G. These close points in G will correspond to a set of vertex disjoint paths that will cover enough points.

▶ **Definition 17** (Weighted $\mathscr{F}$-Path Packing (WMatPP))**.** *The input is $G = (V, E)$ and λ same as in WkPP, plus a finite set X, $\mathcal{Y} = \{Y_v \subseteq X : v \in V\}$, and $\mathscr{F} \subseteq 2^X$ a family of independent sets of a matroid. The goal is to find a set of disjoint paths $P \in \mathcal{P}(G)$ with maximum $\mathsf{val}(P)$ for which there exists $S \in \mathscr{F}$ such that $\forall p \subseteq P$, $S \cap Y_{\mathsf{sink}(p)} \neq \emptyset$.*

Observe that the reduction procedure in Algorithm 2 and all of our subsequent observations in Section 3.1 do not rely on how we define a feasible set of centers. Hence, the main obstacle in proving Theorem 16 lies in our reduction to MCMF. Luckily, the result of [13] helps us address this by giving LP integrality results similar to MCMF using the following formulation on directed *polymatroidal flows* [17, 24, 29]: For a network $G' = (V', E')$, for all $v \in V'$, we are given polymatroids[4] ρ_v^- and ρ_v^+ on $\delta^-(v)$ and $\delta^+(v)$ respectively. For every arc $e \in E'$ there is a variable $0 \leq y_e \leq 1$. The capacity constraints for each $v \in V'$ are defined as:

$$\sum_{e \in U} y_e \leq \rho_v^-(U) \qquad \forall U \subseteq \delta^-(v)$$

$$\sum_{e \in U} y_e \leq \rho_v^+(U) \qquad \forall U \subseteq \delta^+(v).$$

We augment the DAG G given in WMatPP to construct a polymatroidal flow network G'. In this new network, $V' = V \cup X \cup \{s, t\}$ where each node $v \in V$ has cost $-\lambda(v)$. **Note:** Even though a vertex $v \in V$ might correspond to a point in X, in V' we make a distinction between the two copies. E' includes all of E, plus arcs (s, v) for all $v \in V$. Finally, instead of adding arcs (v, t), we add arcs (v, f) and (f, t) for all $f \in Y_v$.

The polymatroids for this instance are constructed as follows: for any $v \in V \cup X$, $\rho_v^-(U) = 1$ for all non-empty $U \subseteq \delta^-(v)$ and ρ_v^+ is defined similarly on $\delta^+(v)$. For s, we only have outgoing edges where $\rho_s^+(U) = |U|$ for all $U \subseteq \delta^+(s)$. Finally, we enforce the matroid constraints of $\mathscr{F}$ on t. For any $U \subseteq \delta^-(t)$, let $T \subseteq X$ be the set of starting nodes in U. That is, $U = \{(f, t) : f \in T\}$. Set $\rho_t^-(U) = \mathsf{rank}_{\mathscr{F}}(T)$. Since $\delta^-(t) \subseteq X$, these capacity constraints on t are equivalent to the following set of constraints:

$$\sum_{f \in T} y_{(f,t)} \leq \mathsf{rank}_{\mathscr{F}}(T) \qquad \forall T \subseteq X : \{(f, t) : f \in T\} \subseteq \delta^-(t).$$

[4] Monotone integer-valued submodular functions.

Now, we prove a claim analogous to that of Claim 8.

▷ Claim 18. WMatPP is equivalent to solving the polymatroidal flow on network G'.

Proof. Any solution P for the WMatPP instance translates to a valid flow of cost $-\mathsf{val}(P)$ for the flow problem. Let $S \in \mathscr{F}$ be the independent set that intersects $Y_{\mathsf{sink}(p)}$ for all $p \in P$. For any path $p \in P$ with start vertex u and sink vertex v, take arbitrary $f \in S \cap Y_v$. Send one unit of flow from s to u, through p to v and then to f and t. All the polymatroidal constraints in WMatPP LP are satisfied.

Now we argue that any solution to the flow instance with cost $-m$ translates to a solution P for WMatPP with $\mathsf{val}(P) = m$. To see this, note that the flow solution consists of s,t paths that are vertex disjoint with respect to $V \cup X$. This is due to our choice of $V \cup X$ polymatroids. Each path passes through one $v \in V$, then immediately to $f \in Y_v$ and then ends in t. By polymatroidal constraints on t, the subset of X that has a flow going through it will be an independent set of $\mathscr{F}$.

Let P be the described paths induced on V. For a $v \in V$, $-\lambda(v)$ is counted towards the MCMF cost iff v has a flow passing through it. This means v is included in some path in P. Thus $\mathsf{val}(P) = m$. ◁

The polymatroidal LP for this particular construction is as follows (recall $\mathsf{flow}(v) := \sum_{e \in \delta^-(v)} y_e$):

$$\max \sum_{v \in V} \mathsf{flow}(v)\lambda(v) \qquad \text{(WMatPP LP)}$$

$$\mathsf{flow}(v) := \sum_{e \in \delta^-(v)} y_e = \sum_{e \in \delta^+(v)} y_e \qquad \forall v \in V \cup X$$

$$\sum_{f \in T} y_{(f,t)} \leq \mathsf{rank}_{\mathscr{F}}(T) \qquad \forall T \subseteq X : \{(f,t) : f \in T\} \subseteq \delta^-(t)$$

$$\mathsf{flow}(v) \leq 1 \qquad \forall v \in V \cup X$$

$$0 \leq y_e \leq 1 \qquad \forall e \in E'$$

By [13], WMatPP LP is integral and there are polynomial time algorithms to solve it.

As for reducing PMCO to WMatPP, most of the notation and results can be recycled from Section 3.1. Specially, the reduction itself (Algorithm 3) is just Algorithm 2 with Line 4 added. **Note:** By definition of an arc in contact DAG, for two nodes $u, v \in V$, (u,v) is an arc iff Y_v intersects Y_u.

Algorithm 3 Reduction to WMatPP.

Input: PMCO instance $\mathcal{I} = ((X,d), r, \mathscr{F}, m)$ and assignment $\{\mathsf{cov}(v) \in \mathbb{R}_{\geq 0} : v \in X\}$

1: $R_i, \{D(u) : u \in R_i\} \leftarrow \mathsf{HS}((C_i, d), r, \mathsf{cov})$ for all $i \in [t]$
2: Construct contact DAG $G = (V, E)$ per Definition 9
3: $\lambda(v) \leftarrow |D(v)|$ for all $v \in V$
4: $Y_v \leftarrow \{u \in X : d(u,v) \leq r_v\}$ for all $v \in V$

Output: WMatPP instance $(G = (V,E), \lambda, X, \mathcal{Y}, \mathscr{F})$

Before we start to prove our 9-approximation result for PMCO, we need to slightly modify Claim 12 to account for the fact that a vertex covered by v (the sink of some path) has to travel slightly farther than v to reach an $f \in Y_v$. Fortunately, the proof of Claim 12 has a slight slack that allows us to derive the same distance guarantees even with this extra step.

▷ Claim 19. For any $u \in R_i$ and $v \in R_j$ reachable from u in contact DAG G, and any $f \in Y_v$, $d(u, f) < 3 \cdot 2^i$.

Proof. By definition of G it must be the case that $i > j$. Also for all $f \in Y_v$, $d(f, v) \leq r_v$. If v is reachable from u, a path between u and v may contain a vertex w_k from every level R_k for $j < k < i$:

$$\begin{aligned} d(u,f) &\leq d(u,v) + d(v,f) \leq d(u,v) + r_v \\ &\leq (r_u + r_{w_{i-1}}) + (r_{w_{i-1}} + r_{w_{i-2}}) + \ldots + (r_{w_{j+1}} + r_v) + r_v && \text{(by Fact 2)} \\ &\leq r_u + 2\sum_{k=1}^{i-1} 2^k && \text{(by definition of } C_k\text{)} \\ &= r_u + 2 \cdot (2^i - 2) < 3 \cdot 2^i. && (u \in C_i,\ r_u < 2^i) \end{aligned}$$

◁

Since the previous claim has the same guarantee as Claim 12, Lemma 11 easily translates to the following:

▶ **Lemma 20.** *Any solution with value at least m for the output of Algorithm 3 translates to a 9-approximation for the input $\mathcal{I}$.*

Proof. Let $P \in \mathcal{P}(G)$ be the promised WMatPP solution. Let $S \in \mathscr{F}$ be the independent set that intersects $Y_{\mathsf{sink}(p)}$ for all $p \in P$. By Claim 19, S covers all the vertices $v \in V$ that are included in P by dilation 3. Proof of Lemma 11 shows that for any such $v \in V$ covered by P and any $w \in D(v)$, $d(w, S) \leq 9r_w$. This holds for at least m points. ◀

We can now prove our 9-approximation result for PMCO.

Proof of Theorem 16. The algorithm is very similar to that of Theorem 6: Given PMCO instance $\mathcal{I} = ((X, d), r, \mathscr{F}, m)$, solve the PMCO LP and use the solution in the procedure of Algorithm 3 to reduce to WMatPP instance $\mathcal{J} = (G = (V, E), \lambda, X, \mathcal{Y}, \mathscr{F})$. Let $P \in \mathcal{P}(G)$ be the solution to this instance and $S \in \mathscr{F}$ be the independent set that intersects $Y_{\mathsf{sink}(p)}$ for all $p \in P$. If $\mathsf{val}(P) \geq m$, S is a 9-approximate solution for $\mathcal{I}$ via Lemma 20. So we prove such solution P exists by constructing a feasible (possibly fractional) WMatPP LP solution.

Take the contact DAG of Algorithm 3 $G = (V, E)$ and recall that each $v \in V$ is also a point in X. For any $f \in X$ let $A_f := \{v \in V : d(f, v) \leq r_v\}$ be the set of points $v \in V$ for which x_f contributes to $\mathsf{cov}(v)$. By definition of an edge in contact DAG, for any $u, v \in A_f$, we have $(u, v) \in E$. Define p_f to be the s, t path that passes through A_f in the order of decreasing neighborhood radii. Formally, let $(u_1, \ldots, u_l)$ be A_f sorted in decreasing order of neighborhood radii. Then, $p_f = ((s, u_1), (u_1, u_2), \ldots (u_l, f), (f, t))$. Similar to the proof of Theorem 6 we define $H_e := \{f \in X : e \in p_f\}$ and set y as follows:

$$y_e := \sum_{f \in H_e} x_f.$$

Now, we argue that y is a feasible solution for WMatPP LP with objective value at least m. The flow is conserved for each vertex $v \in V \cup X$ since for any $f \in X$, we add the same amount x_f to y_e of all $e \in p_f$. Observe that $\mathsf{flow}(v) = \mathsf{cov}(v)$ thus the constraint $\mathsf{cov}(v) \leq 1$ in PkCO LP implies $\mathsf{flow}(v) \leq 1$. To see why the rank constraints are satisfied, the key observation is that any $e \in \delta^-(t)$ must be of the form (f, t) for some $f \in X$, and by our construction $y_e = x_f$. So according to constraint $\sum_{f \in T} x_f \leq \mathsf{rank}_{\mathscr{F}}(T)$ in PMCO LP we have $\sum_{f \in T} y_{(f,t)} \leq \mathsf{rank}_{\mathscr{F}}(T)$. Lastly, the WMatPP LP objective for this solution is at least m. The proof is identical to what we had for Theorem 6. ◀

5 Priority Knapsack-Center with Outliers

In this section, we show how to generalize the results from the previous section for the case of Priority Knapsack-Center with Outliers (PKnapCO).

▶ **Definition 21** (Priority Knapsack-Center with Outliers (PKnapCO)). *The input is a metric space (X, d), a radius function $r : X \to \mathbb{R}_{>0}$, a weight function $\mathsf{w} : X \to \mathbb{R}_{\geq 0}$, parameters $B > 0$ and $m \in \mathbb{N}$. The goal is to find $S \subseteq X$ with $\mathsf{w}(S) \leq B$ to minimize α such that for at least m points $v \in X$, $d(v, S) \leq \alpha \cdot r(v)$.*

▶ **Theorem 22.** *There is a 14-approximation for PKnapCO.*

As in PkCO and PMCO, we reduce to the following path packing problem.

▶ **Definition 23** (Weighted Knapsack-Path Packing (WNapPP)). *The input is $G = (V, E)$ and λ same as in WkPP, plus X a finite set, $\mathsf{w} : X \to \mathbb{R}_{\geq 0}$, $\mathcal{Y} = \{Y_v \subseteq X : v \in V\}$, and parameter $B > 0$. The goal is to find a set of disjoint paths $P \in \mathcal{P}(G)$ with maximum $\mathsf{val}(M)$ for which there exists $S \subseteq X$ with $w(S) \leq B$ such that $\forall p \in P$, $S \cap Y_{\mathsf{sink}(p)} \neq \emptyset$.*

There are two main issues in generalizing our techniques from Section 3 and Section 4. First, the WNapPP problem seems hard on a general DAG. To circumvent this, we make two changes to the LP-aware PKnapCO to WNapPP reduction (given in Algorithm 4). First, we modify Algorithm 1 so that a representative captures points at *larger* distances. To be precise, for a representative u, Algorithm 1 is modified to: $D(u) \leftarrow \{v \in U : d(u, v) \leq r_u + 2r_v\}$. Second, the partition induced by C_i's in Algorithm 2 is done via powers of 4 instead of 2. This is what bumps our approximation factor from 9 to 14. However it helps, as the resulting contact DAG is in fact a directed out-forest. It is not too hard to solve WNapPP when G is a directed-out forest using dynamic programming.

Algorithm 4 Reduction to WNapPP.

Input: PKnapCO instance $\mathcal{I} = ((X, d), r, \mathsf{w}, B, m)$ and assignment $\{\mathsf{cov}(v) \in \mathbb{R}_{\geq 0} : v \in X\}$

1: $R_i, \{D(u) : u \in R_i\} \leftarrow \mathsf{ModHS}((C_i, d), r, \mathsf{cov})$ for all $i \in [t]$
2: Construct contact forest $G = (V, E)$ per Definition 24
3: $\lambda(v) \leftarrow |D(v)|$ for all $v \in V$
4: $Y_v \leftarrow \{u \in X : d(u, v) \leq r_v\}$ for all $v \in V$

Output: WNapPP instance $(G = (V, E), \lambda, X, \mathcal{Y}, \mathsf{w}, B)$

▶ **Definition 24** (contact forest). *Let $R_i \subseteq C_i$, $i \in [t]$ be the set of representatives acquired after running ModHS procedure on C_i according to Line 2 of Algorithm 4. contact forest $G = (V, E)$ is a directed forest on vertex set $V = \bigcup_i R_i$ where the arcs are constructed by the as follows: For $u \in R_i$ and $v \in R_j$ where $i > j$, add the arc $(u, v) \in E$ if there exists $f \in X$ such that $d(u, f) \leq r_u$ and $d(v, f) \leq 2r_v$. Next, remove all the forward edges*[5].

The second issue is more difficult to handle. In PkCO and PMCO, we used the fact that WkPP and WMatPP LPs are integral to show that the instances constructed by the reduction have large value. This is not true any more as the WNapPP LP is not integral even when G is a forest. Indeed, the natural LP relaxation for PKnapCO has unbounded integrality gap even without priorities [14].

[5] In a DAG, edge (u, v) is a forward edge if there is a path of length two or more in the graph that connects u to v.

We circumvent this by using the round and cut framework of [8]. Instead of using the PKnapCO LP, we would use cov in the convex hull of the integral solutions (call it $\mathscr{P}_{\mathsf{cov}}$). Of course, we do not know the integral solutions and there may indeed be exponentially such solutions. So, we have to employ the ellipsoid algorithm. In each iteration of ellipsoid, we get some cov that may or may not be in $\mathscr{P}_{\mathsf{cov}}$. In any case, if we manage to get a good path packing solution using this cov, we get an approximate PKnapCO solution and we are done. Otherwise, we are able to give ellipsoid a linear constraint that should be satisfied by any point in $\mathscr{P}_{\mathsf{cov}}$ but is violated by the current cov. Ultimately, either we find an approximate solution for PKnapCO along the way, or ellipsoid prompts that $\mathscr{P}_{\mathsf{cov}}$ is empty, indicating that the problem is infeasible.

From here on, let $\mathscr{F}$ be the set of all possible centers that fit in the budget. That is, $\mathscr{F} := \{S \subseteq X : \mathsf{w}(S) \leq B\}$. The following is the convex hull of the integral solutions for PKnapCO.

$$\mathscr{P}_{\mathsf{cov}} = \{(\mathsf{cov}(v) : v \in X) : \sum_{v \in X} \mathsf{cov}(v) \geq m \qquad (\mathscr{P}_{\mathsf{cov}}.1)$$

$$\forall v \in X, \quad \mathsf{cov}(v) := \sum_{\substack{S \in \mathscr{F}: \\ d(v,S) \leq r_v}} z_S \qquad (\mathscr{P}_{\mathsf{cov}}.2)$$

$$\sum_{S \in \mathscr{F}} z_S = 1 \qquad (\mathscr{P}_{\mathsf{cov}}.3)$$

$$\forall S \in \mathscr{F}, \quad z_S \geq 0\} \qquad (\mathscr{P}_{\mathsf{cov}}.4)$$

We show if $\mathsf{cov}(v) \in \mathscr{P}_{\mathsf{cov}}$, then indeed the WNapPP instance obtained is "valuable", that is, has value $\geq m$. More importantly, we show that if the instance is *not* valuable, then we can find a hyperplane separating cov from $\mathscr{P}_{\mathsf{cov}}$. One can now use the ellipsoid method to get the 14-approximation: given cov, we either get a valuable WNapPP instance leading to a 14 approximation, or we find a separating hyperplane which can be fed to the ellipsoid method to obtain a new cov vector. The details are omitted in this version due to space restrictions and can be found in [2].

6 Connections to Fair Clustering

In this section, we show how our results imply results in the two fairness notions as defined by [28] and [22].

6.1 "A Center in your Neighborhood" notion of [28]

Jung et al. [28] argue that fairness in clustering should take into account population densities and geography. For every $v \in X$, they define a *neighborhood radius* $\mathrm{NR}(v)$ to be the distance to its $(\lceil n/k \rceil - 1)$th nearest neighbor. A solution is fair, they argue, if every v is served within their $\mathrm{NR}(v)$. They also observe that this may not always be possible, and therefore they wish to find a placement $S \subseteq X$ minimizing $\max_v \frac{d(v,S)}{\mathrm{NR}(v)}$. As an optimization problem, the problem is precisely an instantiation of Priority k-Center. Thus, one can easily obtain a 2-approximation once $r(v) = \mathrm{NR}(v)$ is fixed.

[28] in fact show that it is always possible to find S such that $d(v, S) \leq 2\mathrm{NR}(v)$. They do so by looking at the centers obtained from running their algorithm which is the same as that of Plesník. Note that a 2-approximation to the instance of Priority k-Center defined by

$r(v) = \mathrm{NR}(v)$ does not necessarily imply this additional property. Here we show why it is not a coincidence by considering the natural LP relaxation for Priority k-Center. Given an instance of Priority k-Center one can obtain a lower bound on the optimum value by finding the smallest α such that the following LP is feasible.

$$\mathrm{PkCFeasLP}(\alpha) \;:=\; \{(y_u \geq 0 \;:\; u \in X) : \quad \sum_{u \in X} y_u \leq k; \quad \forall v \in X : \sum_{u : d(u,v) \leq \alpha r(v)} y_u \geq 1\} \quad (1)$$

▷ Claim 25. Suppose $\mathrm{PkCFeasLP}(\alpha)$ has a feasible solution, then Algorithm 1 run with $\phi(v) = \frac{1}{r(v)}$ finds at most k centers that cover each point v within distance $2\alpha r(v)$.

Proof. The proof is similar to that of Theorem 3. Without loss of generality we can assume $\alpha = 1$, otherwise we can scale all the radii by $1/\alpha$. We need to argue $|S| \leq k$. For any $u \in S$, we have $\sum_{v \in D(u)} y_v \geq 1$ since $B(u, r(u)) \subseteq D(u)$. Since $D(u)$'s are disjoint and $\sum_{u \in X} y_u \leq k$, the claim follows. ◁

The preceding discussion and the claim show the utility of viewing the clustering problem of [28] as a special case of Priority k-Center. One can then bring to bear all the positive algorithmic results on Priority k-Center (such as Theorem 4) to fine-tune the fair clustering model. Below we list a few high-level speculative ideas on how the Priority k-Center view can help.

- The LP relaxation could be useful in obtaining better empirical solutions. For example, it has been shown that for k-center, the LP relaxation is integral under notions of stability [12].
- The model of [28] allows $\mathrm{NR}(v)$ to be very large for points v which may not be near many points. However, one may want to put an upper bound M on the radius that is independent of $\mathrm{NR}(v)$. The same algorithm works to give a 2-approximation but one may no longer have the property that all points are covered within twice $\mathrm{NR}(v)$.
- In many scenarios it makes sense to work with the supplier version since centers cannot necessarily be placed at all locations in X. Second, there could be several additional constraints on the set of centers that can be chosen. Theorem 4 shows that more general constraints than cardinality can be handled.
- Related to the first point above, far away points in less dense regions (outliers) can be harmed by setting $\mathrm{NR}(v)$ to be a large number. Alternatively, one can skew the choice of centers if one tries to set a small radius for these points. In this situation it is useful to have algorithms that can handle outliers such that one can find a good solution for vast majority of points and help the outliers via other techniques.

6.2 The Lottery Model of Harris et al. [22]

Harris et al. [22] define a lottery model of fairness where every client $v \in X$ has a "distance demand" $r(v)$ and a "probability demand" $p(v)$. They deem a lottery or a distribution over feasible solutions fair if every client is connected to a facility within their distance demand with probability at least the probability demand. The computational question is to figure out if this is (approximately) feasible. We show a connection to the outlier version of the priority k-center problem, and then generalize their results. We start with a definition.

▶ **Definition 26** (Lottery Priority $\mathscr{F}$-Center (LP$\mathscr{F}$C)). *The input is a metric space (X, d) where each point v has a distance demand $r(v) > 0$ and probability demand $\mathsf{prob}(v)$. The input also (implicitly) specifies a family $\mathscr{F} \subseteq 2^X$ of allowed locations where centers can be opened. A distribution over $\mathscr{F}$ is α-approximate if*

$$\forall v \in X : \quad \Pr_{S \sim \mathscr{F}}[d(v, S) \leq \alpha \cdot r(v)] \geq \mathsf{prob}(v).$$

An α-approximation algorithm in the lottery model either asserts the instance infeasible in that an 1-approximate distribution doesn't exist, or returns an α-approximate distribution.

Harris et al. [22] show that for the case when $\mathscr{F}$ is simply $\{S : |S| \leq k\}$, there is a 9-approximate distribution. Using our results described before, and the by now standard framework using the ellipsoid method (as in [6, 1]), we can get the following results.

▶ **Theorem 27.** *There is a 9-approximation for LP$\mathscr{F}$C where $\mathscr{F}$ is the independent set of a matroid.*

▶ **Theorem 28.** *There is a 14-approximation for LP$\mathscr{F}$C on points X where $\mathscr{F} = \{S \subseteq X : \mathsf{w}(S) \leq B\}$ for a poly-bounded weight function $\mathsf{w} : X \to \mathbb{R}_{\geq 0}$ and parameter $B > 0$.*

We first describe the reduction. For this, we need to define the Fractional Priority $\mathscr{F}$-Center where each point comes with a (possibly fractional) weight μ_v and given $m \geq 0$, the goal is to find a set $S \in \mathscr{F}$ that covers a total weight of more than m with minimum dilation of neighborhood radii.

▶ **Definition 29** (Fractional Priority $\mathscr{F}$-Center (FP$\mathscr{F}$C)). *The input is a metric space (X, d) where each point v has a radius $r_v > 0$ and a weight $\mu_v \geq 0$. Given parameter $m \geq 0$ and a family of subsets of points $\mathscr{F} \subseteq 2^X$, the goal is to find $S \in \mathscr{F}$ to minimize α such that $\mu(\{v \in X : d(v, S) \leq \alpha \cdot r_v\}) > m$.*

An instance of FP$\mathscr{F}$C is specified by the tuple $((X, d), r, \mu, \mathscr{F}, m)$. The following theorem states the reduction from LP$\mathscr{F}$C to FP$\mathscr{F}$C using the ellipsoid method. The proof of this theorem can be found in [2].

▶ **Theorem 30.** *Given LP$\mathscr{F}$C instance $\mathcal{I}$ and a black-box α-approximate algorithm $\mathcal{A}$ for FP$\mathscr{F}$C that runs in time $T(\mathcal{A})$, one can get an α-approximate solution for $\mathcal{I}$ in time poly$(|\mathcal{I}|)T(\mathcal{A})$.*

Now we discuss how our results generalize to solve FP$\mathscr{F}$C for matroid and knapsack constraints.

Proof of Theorem 27. According to Theorem 30 we only need to prove that we can find a 9-approximate solution for any given FP$\mathscr{F}$C instance $\mathcal{I} = ((X, d), r, \mu, \mathscr{F}, m)$. First, observe that the LP for $\mathcal{I}$ is the same as PMCO LP with a minor modification: The constraint $\sum_{v \in X} \mathsf{cov}(v) \geq m$ is changed to $\sum_{v \in X} \mu_v \mathsf{cov}(v) > m$. Solve the LP for $\mathcal{I}$ and use the obtained cov to run the reduction in Algorithm 3; but with a change in Line 3): instead of setting $\lambda(v) \leftarrow |D(v)|$ for all $v \in V$, we will have $\lambda(v) \leftarrow \mu(D(v))$. This results in a WMatPP instance $\mathcal{J}$ with fractional λ. The procedure in [13] can handle fractional λ's so we can still compute the solution for $\mathcal{J}$ in polynomial time. If this solution has value less than or equal to m, we know that $\mathcal{I}$ is infeasible. Otherwise, Lemma 20 tells us that this solution for $\mathcal{J}$ translates to a 9-approximation for $\mathcal{I}$ and we are done. ◀

Proof of Theorem 28. We follow a procedure similar to the proof of Theorem 27. Per Theorem 30 we only need to prove there is a 14-approximation for the FP$\mathscr{F}$C instance where $\mathscr{F}$ is a set of feasible knapsack solutions with poly-bounded weights $\mathsf{w} : X \to \mathbb{R}_{\geq 0}$ and budget $B > 0$. Change the constraint $\mathscr{P}_{\mathsf{cov}}.1$ in $\mathscr{P}_{\mathsf{cov}}$ to $\sum_{v \in X} \mu_v \mathsf{cov}(v) > m$ and modify Line 3 of Algorithm 4 to $\lambda(v) \leftarrow \mu(D(v))$ then follow the round-or-cut procedure in the proof of Theorem 22. The only challenge here is to prove the WNapPP problem can be solved in polynomial time. The dynamic program (which can be found in [2]) depends on the assumption that λ's are poly-bounded. But here, our λ's are real numbers so instead, we assume that our weights $\mathsf{w} : X \to \mathbb{R}_{\geq 0}$ are poly-bounded so we can still solve the problem via dynamic programming. ◀

References

1. Georg Anegg, Haris Angelidakis, Adam Kurpisz, and Rico Zenklusen. A technique for obtaining true approximations for k-center with covering constraints. In *Proceedings, MPS Conference on Integer Programming and Combinatorial Optimization (IPCO)*, pages 52–65, 2020.
2. Tanvi Bajpai, Deeparnab Chakrabarty, Chandra Chekuri, and Maryam Negahbani. Revisiting priority k-center: Fairness and outliers, 2021. `arXiv:2103.03337`.
3. Sayan Bandyapadhyay, Tanmay Inamdar, Shreyas Pai, and Kasturi R. Varadarajan. A constant approximation for colorful k-center. In *Proceedings, European Symposium on Algorithms (ESA)*, pages 12:1–12:14, 2019.
4. Suman Kalyan Bera, Deeparnab Chakrabarty, Nicolas Flores, and Maryam Negahbani. Fair algorithms for clustering. In *Adv. in Neural Information Processing Systems (NeurIPS)*, pages 4955–4966, 2019.
5. Ioana Oriana Bercea, Martin Groß, Samir Khuller, Aounon Kumar, Clemens Rösner, Daniel R. Schmidt, and Melanie Schmidt. On the cost of essentially fair clusterings. In *Proceedings, International Workshop on Approximation Algorithms for Combinatorial Optimization Problems (APPROX)*, pages 18:1–18:22, 2019.
6. Robert D. Carr and Santosh S. Vempala. Randomized metarounding. *Random Struct. Algorithms*, 20(3):343–352, 2002. Preliminary version in STOC 2000.
7. Deeparnab Chakrabarty, Prachi Goyal, and Ravishankar Krishnaswamy. The non-uniform k-center problem. *ACM Transactions on Algorithms (TALG)*, 16(4):1–19, 2020. Preliminary version in ICALP, 2016.
8. Deeparnab Chakrabarty and Maryam Negahbani. Generalized center problems with outliers. *ACM Transactions on Algorithms (TALG)*, 15(3):1–14, 2019. Preliminary version in ICALP 2018.
9. T-H Hubert Chan, Michael Dinitz, and Anupam Gupta. Spanners with slack. In *Proceedings, European Symposium on Algorithms (ESA)*, pages 196–207, 2006.
10. Moses Charikar, Samir Khuller, David M. Mount, and Giri Narasimhan. Algorithms for facility location problems with outliers. In *Proceedings, ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 642–651, 2001.
11. Moses Charikar, Konstantin Makarychev, and Yury Makarychev. Local global tradeoffs in metric embeddings. *SIAM Journal on Computing (SICOMP)*, 39(6):2487–2512, 2010.
12. Chandra Chekuri and Shalmoli Gupta. Perturbation resilient clustering for k-center and related problems via LP relaxations. In *Proceedings, International Workshop on Approximation Algorithms for Combinatorial Optimization Problems (APPROX)*, pages 9:1–9:16, 2018.
13. Chandra Chekuri, Sreeram Kannan, Adnan Raja, and Pramod Viswanath. Multicommodity flows and cuts in polymatroidal networks. *SIAM Journal on Computing*, 44(4):912–943, 2015. Preliminary version in ITCS 2012.
14. Danny Z. Chen, Jian Li, Hongyu Liang, and Haitao Wang. Matroid and knapsack center problems. *Algorithmica*, 75(1):27–52, 2016.

15 Xingyu Chen, Brandon Fain, Liang Lyu, and Kamesh Munagala. Proportionally fair clustering. In *Proceedings, International Conference on Machine Leanring (ICML)*, volume 97, pages 1032–1041, 2019.

16 Flavio Chierichetti, Ravi Kumar, Silvio Lattanzi, and Sergei Vassilvitskii. Fair clustering through fairlets. In *Adv. in Neural Information Processing Systems (NeurIPS)*, pages 5029–5037, 2017.

17 Jack Edmonds and Rick Giles. A min-max relation for submodular functions on graphs. *Annals of Discrete Mathematics*, 1:185–204, 1977.

18 Michael R. Garey and David S. Johnson. *Computers and Intractability; A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., 1979.

19 Teofilo F. Gonzalez. Clustering to Minimize the Maximum Intercluster Distance. *Theoretical Computer Science*, 38:293–306, 1985.

20 Inge Li Gørtz and Anthony Wirth. Asymmetry in k-center variants. *Theoretical Computer Science*, 361(2-3):188–199, 2006. Preliminary version in APPROX 2003.

21 Anupam Gupta, Guru Guruganesh, and Melanie Schmidt. Approximation algorithms for aversion k-clustering via local k-median. In *Proceedings, International Colloquium on Automata, Languages and Programming (ICALP)*, 2016.

22 David G. Harris, Shi Li, Thomas Pensyl, Aravind Srinivasan, and Khoa Trinh. Approximation algorithms for stochastic clustering. *Journal of Machine Learning Research*, 20(153):1–33, 2019. Preliminary version in NeurIPS 2018.

23 David G Harris, Thomas Pensyl, Aravind Srinivasan, and Khoa Trinh. A lottery model for center-type problems with outliers. *ACM Transactions on Algorithms (TALG)*, 2019. Preliminary version in APPROX 2017.

24 Refael Hassin. Minimum cost flow with set-constraints. *Networks*, 12(1):1–21, 1982.

25 Dorit S. Hochbaum and David B. Shmoys. A best possible heuristic for the k-center problem. *Math. Oper. Res.*, 1985.

26 Dorit S. Hochbaum and David B. Shmoys. A unified approach to approximation algorithms for bottleneck problems. *J. ACM*, 33(3):533–550, 1986.

27 Xinrui Jia, Kshiteej Sheth, and Ola Svensson. Fair colorful k-center clustering. In *Proceedings, MPS Conference on Integer Programming and Combinatorial Optimization (IPCO)*, pages 209–222, 2020.

28 Christopher Jung, Sampath Kannan, and Neil Lutz. Service in your neighborhood: Fairness in center location. In *Proceedings, Foundations of Responsible Computing, FORC 2020*, volume 156, pages 5:1–5:15, 2020.

29 Eugene L Lawler and Charles U Martel. Computing maximal "polymatroidal" network flows. *Mathematics of Operations Research*, 7(3):334–347, 1982.

30 Sepideh Mahabadi and Ali Vakilian. Individual fairness for k-clustering. In *International Conference on Machine Learning*, pages 6586–6596. PMLR, 2020.

31 Evi Micha and Nisarg Shah. Proportionally Fair Clustering Revisited. *Proceedings, International Colloquium on Automata, Languages and Programming (ICALP)*, 2020.

32 Ján Plesník. A heuristic for the p-center problems in graphs. *Discrete Applied Mathematics*, 17(3):263–268, 1987.

33 Clemens Rösner and Melanie Schmidt. Privacy preserving clustering with constraints. In *Proceedings, International Colloquium on Automata, Languages and Programming (ICALP)*, volume 107, pages 96:1–96:14, 2018.

The Submodular Santa Claus Problem in the Restricted Assignment Case

Etienne Bamas
EPFL, Lausanne, Switzerland

Paritosh Garg
EPFL, Lausanne, Switzerland

Lars Rohwedder
EPFL, Lausanne, Switzerland

Abstract

The submodular Santa Claus problem was introduced in a seminal work by Goemans, Harvey, Iwata, and Mirrokni (SODA'09) as an application of their structural result. In the mentioned problem n unsplittable resources have to be assigned to m players, each with a monotone submodular utility function f_i. The goal is to maximize $\min_i f_i(S_i)$ where $S_1, \dots, S_m$ is a partition of the resources. The result by Goemans et al. implies a polynomial time $O(n^{1/2+\varepsilon})$-approximation algorithm.

Since then progress on this problem was limited to the linear case, that is, all f_i are linear functions. In particular, a line of research has shown that there is a polynomial time constant approximation algorithm for linear valuation functions in the restricted assignment case. This is the special case where each player is given a set of desired resources Γ_i and the individual valuation functions are defined as $f_i(S) = f(S \cap \Gamma_i)$ for a global linear function f. This can also be interpreted as maximizing $\min_i f(S_i)$ with additional assignment restrictions, i.e., resources can only be assigned to certain players.

In this paper we make comparable progress for the submodular variant: If f is a monotone submodular function, we can in polynomial time compute an $O(\log\log(n))$-approximate solution.

2012 ACM Subject Classification Theory of computation → Scheduling algorithms

Keywords and phrases Scheduling, submodularity, approximation algorithm, hypergraph matching

Digital Object Identifier 10.4230/LIPIcs.ICALP.2021.22

Category Track A: Algorithms, Complexity and Games

Related Version *Full Version*: `https://arxiv.org/abs/2011.06939`

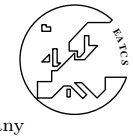
Funding This research was supported in part by the Swiss National Science Foundation project 200021-184656 "Randomness in Problem Instances and Randomized Algorithms."

Acknowledgements The authors wish to thank Ola Svensson for helpful discussions on the problem.

1 Introduction

In the Santa Claus problem (sometimes referred to as Max-Min Fair Allocation) we are given a set of n players P and a set of m indivisible resources R. In its full generality, each player $i \in P$ has a utility function $f_i : 2^R \mapsto \mathbb{R}_{\geq 0}$, where $f_i(S)$ measures the happiness of player i if he is assigned the resource set S. The goal is to find a partition of the resources that maximizes the happiness of the least happy player. Formally, we want to find a partition $\{S_i\}_{i \in P}$ of the resources that maximizes $\min_{i \in P} f_i(S_i)$.

With such an objective function one seeks to find the fairest solution as opposed to for example the best average happiness. Most of the recent literature on this problem focuses on cases where f_i is a linear function for all players i. If we assume all valuation functions are linear, the best approximation algorithm known for this problem, designed by Chakrabarty, Chuzhoy, and Khanna [4], has an approximation rate of n^ϵ and runs in time $n^{O(1/\epsilon)}$ for $\epsilon \in \Omega(\log\log(n)/\log(n))$. On the negative side, it is only known that computing



a $(2-\delta)$-approximation is NP-hard [13]. Apart from this there has been significant attention on the so-called *restricted assignment case*. Here the utility functions are defined by one linear function f and a set of resources Γ_i for each player i. Intuitively, player i is interested in the resources Γ_i, whereas the other resources are worthless for him. The individual utility functions are then implicitly defined by $f_i(S) = f(S \cap \Gamma_i)$. In a seminal work, Bansal and Srividenko [3] provide a $O(\log\log(m)/\log\log\log(m))$-approximation algorithm for this case. This was improved by Feige [8] to an $O(1)$-approximation. Further progress on the constant or the running time was made since then, see e.g. [1, 7, 6, 5, 10, 2, 15].

Let us now move to the non-linear case. Indeed, the problem becomes hopelessly difficult without any restrictions on the utility functions. Consider the following reduction from set packing. There are sets of resources $\{S_1, \ldots, S_k\}$ and all utility functions are equal and defined by $f_i(S) = 1$ if $S_j \subseteq S$ for some j and $f_i(S) = 0$ otherwise. Deciding whether there are m disjoint sets in $S_1, \ldots, S_k$ (a classical NP-hard problem) is equivalent to deciding whether the optimum of the Santa Claus problem is non-zero. In particular, obtaining any bounded approximation ratio for Santa Claus in this case is NP-hard.

Two naturally arising properties of utility functions are monotonicity and submodularity, see for example the related submodular welfare problem [12, 16] where the goal is to maximize $\sum_i f_i(S_i)$. A function f is monotone, if $f(S) \leq f(T)$ for all $S \subseteq T$. It is submodular, if $f(S \cup \{a\}) - f(S) \geq f(T \cup \{a\}) - f(T)$ for all $S \subseteq T$ and $a \notin T$. The latter is also known as the *diminishing returns* property in economics. A standard assumption on monotone submodular functions (used throughout this work) is that the value on the empty set is zero, i.e., $f(\emptyset) = 0$. Goemans, Harvey, Iwata, and Mirrokni [9] first considered the Santa Claus problem with monotone submodular utility functions as an application of their fundamental result on submodular functions. Together with the algorithm of [4] it implies an $O(n^{1/2+\epsilon})$-approximation in time $n^{O(1/\epsilon)}$. In the case that the valuation functions are all equal, that is, $f_i(S) = f(S)$ for a monotone submodular function f, Krause, Rajagopal, Gupta, and Guestrin gave a constant approximation [11]. We also refer to their work for an application of this problem in sensor placement.

In this paper we investigate the restricted assignment case with a monotone submodular utility function. That is, all utility functions are defined by $f_i(S) = f(S \cap \Gamma_i)$, where f is a monotone submodular function and Γ_i is a subset of resources for each players i. Before our work, the state-of-the-art for this problem was the $O(n^{1/2+\epsilon})$-approximation algorithm mentioned above, since none of the previous results for the restricted assignment case with a linear utility function apply when the utility function becomes monotone submodular.

1.1 Overview of results and techniques

Our main result is an approximation algorithm for the submodular Santa Claus problem in the restricted assignment case.

▶ **Theorem 1.** *There is a randomized polynomial time $O(\log\log(n))$-approximation algorithm for the restricted assignment case with a monotone submodular utility function.*

Our way to this result is organised as follows. In Section 2, we first reduce our problem to a hypergraph matching problem (see next paragraph for a formal definition). We then solve this problem using Lovasz Local Lemma (LLL) in Section 3. In [3] the authors also reduce to a hypergraph matching problem which they then solve using LLL, although both parts are substantially simpler. The higher generality of our utility functions is reflected in the more general hypergraph matching problem. Namely, our problem is precisely the weighted variant of the (unweighted) problem in [3]. We will elaborate later in this section why the previous techniques do not easily extend to the weighted variant.

The hypergraph matching problem. After the reduction in Section 2 we arrive at the following problem. There is a hypergraph $\mathcal{H} = (P \cup R, \mathcal{C})$ with hyperedges $\mathcal{C}$ over the vertices P and R. We write $m = |P|$ and $n = |R|$. We will refer to hyperedges as configurations, the vertices in P as players and R as resources[1]. Moreover, a hypergraph is said to be regular if all vertices in P and R have the same degree, that is, they are contained in the same number of configurations. The hypergraph may contain multiple copies of the same configuration. Each configuration $C \in \mathcal{C}$ contains exactly one vertex in P, that is, $|C \cap P| = 1$. Additionally, for each configuration $C \in \mathcal{C}$ the resources $j \in C$ have weights $w_{j,C} \geq 0$. We emphasize that the same resource j can be given different weights in two different configurations, that is, we may have $w_{j,C} \neq w_{j,C'}$ for two different configurations C, C'.

We require to select for each player $i \in P$ one configuration C that contains i. For each configuration C that was selected we require to assign a subset of the resources in C which has a total weight of at least $(1/\alpha) \cdot \sum_{j \in C} w_{j,C}$ to the player in C. A resource can only be assigned to one player. We call such a solution an α-relaxed perfect matching. One seeks to minimize α.

We show that every regular hypergraph has an α-relaxed perfect matching for some $\alpha = O(\log \log(n))$ assuming that $w_{j,C} \leq (1/\alpha) \cdot \sum_{j' \in C} w_{j',C}$ for all j, C, that is, all weights are small compared to the total weight of the configuration. Moreover, we can find such a matching in randomized polynomial time. In the reduction we use this result to round a certain LP relaxation and α essentially translates to the approximation rate. This result generalizes that of Bansal and Srividenko on hypergraph matching in the following way. They proved the same result for unit weights and uniform hyperedges, that is, $w_{j,C} = 1$ for all j, C and all hyperedges have the same number of resources[2]. In the next paragraph we briefly go over the techniques to prove our result for the hypergraph matching problem.

Our techniques. Already the extension from uniform to non-uniform hypergraphs (assuming unit weights) is highly non-trivial and captures the core difficulty of our result. Indeed, we show with a (perhaps surprising) reduction, that we can reduce our weighted hypergraph matching problem to the unweighted (but non-uniform) version by introducing some bounded dependencies between the choices of the different players. For sake of brevity we therefore focus in this section on the unweighted non-uniform variant, that is, we need to assign to each player a configuration C and at least $|C|/\alpha$ resources in C. We show that for any regular hypergraph there exists such a matching for $\alpha = O(\log \log(n))$ assuming that all configurations contain at least α resources and we can find it in randomized polynomial time. Without the assumption of uniformity the problem becomes significantly more challenging. To see this, we lay out the techniques of Bansal and Srividenko that allowed them to solve the problem in the uniform case. We note that for $\alpha = O(\log(n))$ the statement is easy to prove: We select for each player i one of the configurations containing i uniformly at random. Then by standard concentration bounds each resource is contained in at most $O(\log(n))$ of the selected configurations with high probability. This implies that there is a fractional assignment of resources to configurations such that each of the selected configurations C receives $\lfloor |C|/O(\log(n)) \rfloor$ of the resources in C. By integrality of the bipartite matching polytope, there is also an integral assignment with this property.

To improve to $\alpha = O(\log \log(n))$ in the uniform case, Bansal and Srividenko proceed as follows. Let k be the size of each configuration. First they reduce the degree of each player and resource to $O(\log(n))$ using the argument above, but taking $O(\log(n))$ configurations for

[1] We note that these do not have to be the same players and resources as in the Santa Claus problem we reduced from, but n and m do not increase.

[2] In fact they get a slightly better ratio of $\alpha = O(\log \log(m)/ \log \log \log(m))$.

each player. Then they sample uniformly at random $O(n \log(n)/k)$ resources and drop all others. This is sensible, because they manage to prove the (perhaps surprising) fact that an α-relaxed perfect matching with respect to the smaller set of resources is still an $O(\alpha)$-relaxed perfect matching with respect to all resources with high probability (when assigning the dropped resources to the selected configurations appropriately). Indeed, the smaller instance is easier to solve: With high probability all configurations have size $O(\log(n))$ and this greatly reduces the dependencies between the bad events of the random experiment above (the event that a resource is contained in too many selected configurations). This allows them to apply Lovász Local Lemma (LLL) in order to show that with positive probability the experiment succeeds for $\alpha = O(\log \log(n))$.

It is not obvious how to extend this approach to non-uniform hypergraphs: Sampling a fixed fraction of the resources will either make the small configurations empty – which makes it impossible to retain guarantees for the original instance – or it leaves the big configurations big – which fails to reduce the dependencies enough to apply LLL. Hence it requires new sophisticated ideas for non-uniform hypergraphs, which we describe next.

Suppose we are able to find a set $\mathcal{K} \subseteq \mathcal{C}$ of configurations (one for each player) such that for each $K \in \mathcal{K}$ the sum of intersections $|K \cap K'|$ with smaller configurations $K' \in \mathcal{K}$ is very small, say at most $|K|/2$. Then it is easy to derive a 2-relaxed perfect matching: We iterate over all $K \in \mathcal{K}$ from large to small and reassign all resources to K (possibly stealing them from the configuration that previously had them). In this process every configuration gets stolen at most $|K|/2$ of its resources, in particular, it keeps the other half. However, it is non-trivial to obtain a property like the one mentioned above. If we take a random configuration for each player, the dependencies of the intersections are too complex. To avoid this we invoke an advanced variant of the sampling approach where we construct not only one set of resources, but a hierarchy of resource sets $R_0 \supseteq \cdots \supseteq R_d$ by repeatedly dropping a fraction of resources from the previous set. We then formulate bad events based on the intersections of a configuration C with smaller configurations C', but we write it only considering a resource set R_k of convenient granularity (chosen based on the size of C'). In this way we formulate a number of bad events using various sets R_k. This succeeds in reducing the dependencies enough to apply LLL. Unfortunately, even with this new way of defining bad events, the guarantee that for each $K \in \mathcal{K}$ the sum of intersections $|K \cap K'|$ with smaller configurations $K' \in \mathcal{K}$ is at most $|K|/2$ is still too much to ask. We can only prove some weaker property which makes it more difficult to reconstruct a good solution from it. The reconstruction still starts from the biggest configurations and iterates to finish by including the smallest configurations but it requires a delicate induction where at each step, both the resource set expands and some new small configurations that were not considered before come into play.

Additional implications of non-uniform hypergraph matchings to the Santa Claus problem. We believe this hypergraph matching problem is interesting in its own right. Our last contribution is to show that finding good matchings in unweighted hypergraphs with fewer assumptions than ours would have important applications for the Santa Claus problem with linear utility functions. We recall that here, each player i has its own utility function f_i that can be any linear function. In this case, the best approximation algorithm is due to Chakrabarty, Chuzhoy, and Khanna [4] who gave a $O(n^\epsilon)$-approximation running in time $O(n^{1/\epsilon})$. In particular, no sub-polynomial approximation running in polynomial time is known. Consider as before $\mathcal{H} = (P \cup R, \mathcal{C})$ a non-uniform hypergraph with unit weights ($w_{j,C} = 1$ for all j, C such that $j \in C$). Finding the smallest α (or an approximation of it) such that there exists an α-relaxed perfect matching in $\mathcal{H}$ is already a very non-trivial question to solve in polynomial time.

We show, via a reduction, that a c-approximation for this problem would yield a $O((c\log^*(n))^2)$-approximation for the Santa Claus problem with arbitrary linear utility functions. In particular, any sub-polynomial approximation for this problem would significantly improve the state-of-the-art[3]. Details of this last result can be found in the full version of the paper.

A remark on local search techniques. We focus here on an extension of the LLL technique of Bansal and Srividenko. However, another technique proved itself very successful for the Santa Claus problem in the restricted assignment case with a linear utility function. This is a local search technique discovered by Asadpour, Feige, and Saberi [2] who used it to give a non-constructive proof that the integrality gap of the configuration LP of Bansal and Srividenko is at most 4. One may wonder if this technique could also be extended to the submodular case as we did with LLL. Unfortunately, this seems problematic as the local search arguments heavily rely on amortizing different volumes of configurations (i.e., the sum of their resources' weights or the number of resources in the unweighted case). Amortizing the volumes of configurations works well, if each configuration has the same volume, which is the case for the problem derived from linear valuation functions, but not the one derived from submodular functions. If the volumes differ then the amortization arguments break and the authors of this paper believe this is a fundamental problem for this approach.

2 Reduction to hypergraph matching problem

In this section we give a reduction of the restricted submodular Santa Claus problem to the hypergraph matching problem. As a starting point we solve the configuration LP, a linear programming relaxation of our problem. The LP is constructed using a parameter T which denotes the value of its solution. The goal is to find the maximal T such that the LP is feasible. In the LP we have a variable $x_{i,C}$ for every player $i \in P$ and every configuration $C \in \mathcal{C}(i,T)$. The configurations $\mathcal{C}(i,T)$ are defined as the sets of resources $C \subseteq \Gamma_i$ such that $f(C) \geq T$. We require every player $i \in P$ to have at least one configuration and every resource $j \in R$ to be contained in at most one configuration.

$$\sum_{C \in \mathcal{C}(i,T)} x_{i,C} \geq 1 \quad \text{for all } i \in P$$

$$\sum_{i \in P} \sum_{C \in \mathcal{C}(i,T): j \in C} x_{i,C} \leq 1 \quad \text{for all } j \in R$$

$$x_{i,C} \geq 0 \quad \text{for all } i \in P, C \in \mathcal{C}(i,T)$$

Since this linear program has exponentially many variables, we cannot directly solve it in polynomial time. We will give a polynomial time constant approximation for it via its dual. This is similar to the linear variant in [3], but requires some more work. In their case they can reduce the problem to one where the separation problem of the dual can be solved in polynomial time. In our case even the separation problem can only be approximated. Nevertheless, this is sufficient to approximate the linear program in polynomial time.

▶ **Theorem 2.** *The configuration LP of the restricted submodular Santa Claus problem can be approximated within a factor of* $(1-1/e)/2$ *in polynomial time.*

[3] We mention that our result on relaxed matchings in Section 3 does not imply an $O(\log\log(n))$-approximation for this problem since we make additional assumptions on the regularity of the hypergraph or the size of hyperedges.

We defer the proof of this theorem to the full version of the paper. Given a solution x^* of the configuration LP we want to arrive at the hypergraph matching problem from the introduction such that an α-relaxed perfect matching of that problem corresponds to an $O(\alpha)$-approximate solution of the restricted submodular Santa Claus problem. Let T^* denote the value of the solution x^*. We will define a resource $j \in R$ as *fat* if $f(\{j\}) \geq T^*/(100\alpha)$.

Resources that are not fat are called *thin*. We call a configuration $C \in \mathcal{C}(i,T)$ thin, if it contains only thin resources and denote by $\mathcal{C}_t(i,T) \subseteq \mathcal{C}(i,T)$ the set of thin configurations. Intuitively in order to obtain an $O(\alpha)$-approximate solution, it suffices to give each player i either one fat resource $j \in \Gamma_i$ or a thin configuration $C \in \mathcal{C}_t(i, T^*/O(\alpha))$. For our next step towards the hypergraph problem we use a technique borrowed from Bansal and Srividenko [3]. This technique allows us to simplify the structure of the problem significantly using the solution of the configuration LP. Namely, one can find a partition of the players into clusters such that we only need to cover one player from each cluster with thin resources. All other players can then be covered by fat resources. Informally speaking, the following lemma is proved by sampling configurations randomly according to a distribution derived in a non-trivial way from the configuration LP.

▶ **Lemma 3.** *Let $\ell \geq 12\log(n)$. Given a solution of value T^* for the configuration LP in randomized polynomial time we can find a partition of the players into clusters $K_1 \cup \cdots \cup K_k \cup Q = P$ and multisets of configurations $\mathcal{C}_h \subseteq \bigcup_{i \in K_h} \mathcal{C}_t(i, T^*/5)$, $h = 1, \ldots, k$, such that*

1. *$|\mathcal{C}_h| = \ell$ for all $h = 1, \ldots, k$ and*
2. *Each small resource appears in at most ℓ configurations of $\bigcup_h \mathcal{C}_h$.*
3. *given any $i_1 \in K_1, i_2 \in K_2, \ldots, i_k \in K_k$ there is a matching of fat resources to players $P \setminus \{i_1, \ldots, i_k\}$ such that each of these players i gets a unique fat resource $j \in \Gamma_i$.*

The role of the players Q in the lemma above is that each one of them gets a fat resource for certain. The proof follows closely that in [3]. For completeness we include it in the full version of the paper. We are now ready to define the hypergraph matching instance. The vertices of our hypergraph are the clusters $K_1, \ldots, K_k$ and the thin resources. Let $\mathcal{C}_1, \ldots, \mathcal{C}_k$ be the multisets of configurations as in Lemma 3. For each K_h and $C \in \mathcal{C}_h$ there is a hyperedge containing K_h and all resources in C. Let $\{j_1, \ldots, j_m\} = C$ ordered arbitrarily, but consistently. Then we define the weights as normalized marginal gains of resources if they are taken in this order, that is,

$$w_{j_i,C} = \frac{5}{T^*} f(\{j_i\} \mid \{j_1, \ldots, j_{i-1}\}) = \frac{5}{T^*}(f(\{j_1, \ldots, j_{i-1}, j_i\}) - f(\{j_1, \ldots, j_{i-1}\})).$$

This implies that $\sum_{j \in C} w_{j,C} \geq 5f(C)/T^* \geq 1$ for each $C \in \mathcal{C}_h$, $h = 1, \ldots, k$.

▶ **Lemma 4.** *Given an α-relaxed perfect matching to the instance as described by the reduction, one can find in polynomial time an $O(\alpha)$-approximation to the instance of restricted submodular Santa Claus.*

Proof. The α-relaxed perfect matching implies that each cluster K_h gets some small resources C' where $C' \subseteq C$ for some $C \in \mathcal{C}_h$ and $\sum_{j \in C'} w_{j,C} \geq 1/\alpha$. By submodularity we have that $f(C') \geq T^*/(5\alpha)$. Therefore we can satisfy one player in each cluster using thin resources and by Lemma 3 all others using fat resources. ◀

The proof above is the most critical place in the paper where we make use of the submodularity of the valuation function f. We note that since all resources considered are thin resources we have, by submodularity of f, the assumption that

$$w_{j,C} \leq \frac{5}{T^*} f(\{j\}) \leq \frac{5}{T^*} \frac{T^*}{100\alpha} \leq \frac{5}{100\alpha} \sum_{j \in C} w_{j,C}$$

for all j, C such that $j \in C$. This means that the weights are all small enough, as promised in introduction. From now on, we will assume that $\sum_{j \in C} w_{j,C} = 1$ for all configurations C. This is without loss of generality, since we can just rescale the weights inside each configuration. This does not hurt the property that all weights are small enough.

2.1 Reduction to unweighted hypergraph matching

Before proceeding to the solution of this hypergraph matching problem, we first give a reduction to an unweighted variant of the problem. We will then solve this unweighted variant in the next section. First, we note that we can assume that all the weights $w_{j,C}$ are powers of 2 by standard rounding arguments. This only loses a constant factor in the approximation rate. Second, we can assume that inside each configuration C, each resource has a weight that is at least a $1/(2n)$. Formally, we can assume that $\min_{j \in C} w_{j,C} \geq 1/(2n)$ for all $C \in \mathcal{C}$. If this is not the case for some $C \in \mathcal{C}$, simply delete from C all the resources that have a weight less than $1/(2n)$. By doing this, the total weight of C is only decreased by a factor $1/2$ since it looses in total at most a weight of $n \cdot (1/2n) = 1/2$. (Recall that we rescaled the weights so that $\sum_{j \in C} w_{j,C} = 1$).

Hence after these two operations, an α-relaxed perfect matching in the new hypergraph is still an $O(\alpha)$-relaxed perfect matching in the original hypergraph. From there we reduce to an unweighted variant of the matching problem. Note that each configuration contains resources of at most $\log(n)$ different possible weights (powers of 2 from $1/(2n)$ to $1/\alpha$). We create the following new unweighted hypergraph $\mathcal{H}' = (P' \cup R, \mathcal{C}')$. The resource set R remains unchanged. For each player $i \in P$, we create $\log(n)$ players, which later correspond each to a distinct weight. We will say that the players obtained from duplicating the original player form a *group*. For every configuration C containing player i in the hypergraph $\mathcal{H}$, we add a set $\mathcal{S}_C = \{C_1, \ldots, C_s, \ldots, C_{\log(n)}\}$ of configurations in $\mathcal{H}'$. C_s contains player i_s and all resources that are given a weight $2^{-(s+1)}$ in C. In this new hypergraph, the resources are not weighted. Note that if the hypergraph $\mathcal{H}$ is regular then $\mathcal{H}'$ is regular as well.

Additionally, for a group of player and a set of $\log(n)$ configurations (one for each player in the group), we say that this set of configurations is *consistent* if all the configurations selected are obtained from the same configuration in the original hypergraph $\mathcal{H}$ (i.e. the selected configurations all belong to $\mathcal{S}_C$ for some C in $\mathcal{H}$).

Formally, we focus of the following problem. Given the regular hypergraph $\mathcal{H}'$, we want to select, for each group of $\log(n)$ players, a consistent set of configurations $C_1, \ldots, C_s, \ldots, C_{\log(n)}$ and assign to each player i_s a subset of the resources in the corresponding configuration C_s so that i_s is assigned at least $\lfloor |C_s|/\alpha \rfloor$ resources. No resource can be assigned to more than one player. We refer to this assignment as a consistent α-relaxed perfect matching. Note that in the case where $|C_s|$ is small (e.g. of constant size) we are not required to assign any resource to player i_s.

▶ **Lemma 5.** *A consistent α-relaxed matching in $\mathcal{H}'$ induces a $O(\alpha)$-relaxed matching in $\mathcal{H}$.*

Due to space constraint, the proof of this lemma is moved to the full version of the paper.

3 Matchings in regular hypergraphs

In this section we solve the hypergraph matching problem we arrived to in the previous section. For convenience, we give a self contained definition of the problem before formulating and proving our result.

Input. We are given $\mathcal{H} = (P \cup R, \mathcal{C})$ a hypergraph with hyperedges $\mathcal{C}$ over the vertices P (players) and R (resources) with $m = |P|$ and $n = |R|$. As in previous sections, we will refer to hyperedges as configurations. Each configuration $C \in \mathcal{C}$ contains exactly one vertex in P, that is, $|C \cap P| = 1$. The set of players is partitioned into groups of size at most $\log(n)$, we will use A to denote a group. These groups are disjoint and contain all players. Finally there exists an integer ℓ such that for each group A there are ℓ consistent sets of configurations. A consistent set of configurations for a group A is a set of $|A|$ configurations such that all players in the group appear in exactly one of these configurations. We will denote by $\mathcal{S}_A$ such a set and for a player $i \in A$, we will denote by $\mathcal{S}_A^{(i)}$ the unique configuration in $\mathcal{S}_A$ containing i. Finally, no resource appears in more than ℓ configurations. We say that the hypergraph is regular (although some resources may appear in less than ℓ configurations).

Output. We wish to select a matching that covers all players in P. More precisely, for each group A we want to select a consistent set of configurations (denoted by $\{\mathcal{S}_A^{(i)}\}_{i \in A}$). Then for each player $i \in A$, we wish to assign a subset of the resources in $\mathcal{S}_A^{(i)}$ to the player i such that:

1. No resource is assigned to more than one player in total.
2. For any group A and any player $i \in A$, player i is assigned at least $\lfloor |\mathcal{S}_A^{(i)}|/\alpha \rfloor$ resources from $\mathcal{S}_A^{(i)}$.

We call this a consistent α-relaxed perfect matching. Our goal in this section will be to prove the following theorem.

▶ **Theorem 6.** *Let $\mathcal{H} = (P \cup R, \mathcal{C})$ be a regular (non-uniform) hypergraph where the set of players is partitioned into groups of size at most $\log(n)$. Then we can, in randomized polynomial time, compute a consistent α-relaxed perfect matching for $\alpha = O(\log\log(n))$.*

We note that Theorem 6 together with the reduction from the previous section will prove our main result (Theorem 1) stated in introduction.

3.1 Overview and notations

To prove Theorem 6, we introduce the following notations. Let $\ell \in \mathbb{N}$ be the regularity parameter as described in the problem input (i.e. each group has ℓ consistent sets and each resource appears in no more than ℓ configurations). As we proved in Lemma 3 we can assume with standard sampling arguments that $\ell = 300.000 \log^3(n)$ at a constant loss. If this is not the case because we might want to solve the hypergraph matching problem by itself (i.e. not obtained by the reduction in Section 2), the proof of Lemma 3 can be repeated in a very similar way here.

For a configuration C, its size will be defined as $|C \cap R|$ (i.e. its cardinality over the resource set). For each player i, we denote by $\mathcal{C}_i$ the set of configurations that contain i. We now group the configurations in $\mathcal{C}_i$ by size: We denote by $\mathcal{C}_i^{(0)}$ the configurations of size in $[0, \ell^4)$ and for $k \geq 1$ we write $\mathcal{C}_i^{(k)}$ for the configurations of size in $[\ell^{k+3}, \ell^{k+4})$. Moreover, define $\mathcal{C}^{(k)} = \bigcup_i \mathcal{C}_i^{(k)}$ and $\mathcal{C}^{(\geq k)} = \bigcup_{h \geq k} \mathcal{C}^{(h)}$. Let d be the smallest number such that $\mathcal{C}^{(\geq d)}$ is empty. Note that $d \leq \log(n)/\log(\ell)$. Now consider the following random process.

▶ **Random Experiment 7.** *We construct a nested sequence of resource sets $R = R_0 \supseteq R_1 \supseteq \ldots \supseteq R_d$ as follows. Each R_k is obtained from R_{k-1} by deleting every resource in R_{k-1} independently with probability $(\ell - 1)/\ell$.*

In expectation only a $1/\ell$ fraction of resources in R_{k-1} survives in R_k. Also notice that for $C \in \mathcal{C}^{(k)}$ we have that $\mathbb{E}[|R_k \cap C|] = \text{poly}(\ell)$.

The proof of Theorem 6 is organized as follows. In Section 3.2, we give some properties of the resource sets constructed by Random Experiment 7 that hold with high probability. Then in Section 3.3, we show that we can find a single consistent set of configurations for each group of players such that for each configuration selected, its intersection with smaller selected configurations is bounded if we restrict the resource set to an appropriate R_k. Restricting the resource set is important to bound the dependencies of bad events in order to apply Lovasz Local Lemma. Finally in Section 3.4, we demonstrate how these configurations allow us to reconstruct a consistent α-relaxed perfect matching for an appropriate assignment of resources to configurations.

3.2 Properties of resource sets

In this subsection, we give a precise statement of the key properties that we need from Random Experiment 7. The first two lemmas have a straight-forward proof. The last one is a generalization of an argument used by Bansal and Srividenko [3]. Since the proof is more technical and tedious, we also defer it to the full version of the paper along with the proof of the first two statements.

We start with the first property which bounds the size of the configurations when restricted to some R_k. This property is useful to reduce the dependencies while applying LLL later.

▶ **Lemma 8.** *Consider Random Experiment 7 with $\ell \geq 300.000 \log^3(n)$. For any $k \geq 0$ and any $C \in \mathcal{C}^{(\geq k)}$ we have*

$$\frac{1}{2}\ell^{-k}|C| \leq |R_k \cap C| \leq \frac{3}{2}\ell^{-k}|C|$$

with probability at least $1 - 1/n^{10}$.

The next property expresses that for any configuration the sum of intersections with configurations of a particular size does not deviate much from its expectation. In particular, for any configuration C, the sum of it's intersections with other configurations is at most $|C|\ell$ as each resource is in at most ℓ configurations. By the lemma stated below, we recover this up to a multiplicative constant factor when we consider the appropriately weighted sum of the intersection of C with other configurations C' of smaller sizes where each configuration $C' \in \mathcal{C}^{(k)}$ is restricted to the resource set R_k.

▶ **Lemma 9.** *Consider Random Experiment 7 with $\ell \geq 300.000 \log^3(n)$. For any $k \geq 0$ and any $C \in \mathcal{C}^{(\geq k)}$ we have*

$$\sum_{C' \in \mathcal{C}^{(k)}} |C' \cap C \cap R_k| \leq \frac{10}{\ell^k}\left(|C| + \sum_{C' \in \mathcal{C}^{(k)}} |C' \cap C|\right)$$

with probability at least $1 - 1/n^{10}$.

We now define the notion of *good* solutions which is helpful in stating our last property. Let $\mathcal{F}$ be a set of configurations, $\alpha : \mathcal{F} \to \mathbb{N}$, $\gamma \in \mathbb{N}$, and $R' \subseteq R$. We say that an assignment of R' to $\mathcal{F}$ is (α, γ)-good if every configuration $C \in \mathcal{F}$ receives at least $\alpha(C)$ resources of $C \cap R'$ and if no resource in R' is assigned more than γ times in total.

Below we obtain that given a (α, γ)-good solution with respect to resource set R_{k+1}, one can construct an almost $(\ell \cdot \alpha, \gamma)$-good solution with respect to the bigger resource set R_k. Informally, starting from a good solution with respect to the final resource set and iteratively applying this lemma would give us a good solution with respect to our complete set of resources.

▶ **Lemma 10.** *Consider Random Experiment 7 with $\ell \geq 300.000 \log^3(n)$. Fix $k \geq 0$. Conditioned on the event that the bounds in Lemma 8 hold for k, then with probability at least $1 - 1/n^{10}$ the following holds for all $\mathcal{F} \subseteq \mathcal{C}^{(\geq k+1)}$, $\alpha : \mathcal{F} \to \mathbb{N}$, and $\gamma \in \mathbb{N}$ such that $\ell^3/1000 \leq \alpha(C) \leq n$ for all $C \in \mathcal{F}$ and $\gamma \in \{1, \ldots, \ell\}$: If there is a (α, γ)-good assignment of R_{k+1} to $\mathcal{F}$, then there is a (α', γ)-good assignment of R_k to $\mathcal{F}$ where*

$$\alpha'(C) \geq \ell \left(1 - \frac{1}{\log(n)}\right) \alpha(C)$$

for all $C \in \mathcal{F}$. Moreover, this assignment can be found in polynomial time.

Given the lemmata above, by a simple union bound one gets that all the properties of resource sets hold.

3.3 Selection of configurations

In this subsection, we give a random process that selects one consistent set of configurations for each group of players such that the intersection of the selected configurations with smaller configurations is bounded when considered on appropriate sets R_k. We will denote $\mathcal{S}_A$ the selected consistent set for group A and for ease of notation we will denote $K_i = \mathcal{S}_A^{(i)}$ the selected configuration for player $i \in A$. For any integer k, we write $\mathcal{K}_i^{(k)} = \{K_i\}$ if $K_i \in \mathcal{C}_i^{(k)}$ and $\mathcal{K}_i^{(k)} = \emptyset$ otherwise. As for the configuration set, we will also denote $\mathcal{K}^{(k)} = \bigcup_i \mathcal{K}_i^{(k)}$ and $\mathcal{K} = \bigcup_k \mathcal{K}^{(k)}$. The following lemma describes what are the properties we want to have while selecting the configurations. For better clarity we also recall what the properties of the sets $R_0, \ldots, R_d$ that we need are. These hold with high probability by the lemmata of the previous section.

▶ **Lemma 11.** *Let $R = R_0 \supseteq \ldots \supseteq R_d$ be sets of fewer and fewer resources. Assume that for each k and $C \in \mathcal{C}_i^{(k)}$ we have*

$$1/2 \cdot \ell^{k-h} \leq |C \cap R_h| \leq 3/2 \cdot \ell^{-h}|C| < 3/2 \cdot \ell^{k-h+4}$$

for all $h = 0, \ldots, k$. Then there exists a selection of one consistent set $\mathcal{S}_A$ for each group A such for all $k = 0, \ldots, d$, $C \in \mathcal{C}^{(k)}$ and $j = 0, \ldots, k$ then we have

$$\sum_{j \leq h \leq k} \sum_{K \in \mathcal{K}^{(h)}} \ell^h |K \cap C \cap R_h| \leq \frac{1}{\ell} \sum_{j \leq h \leq k} \sum_{C' \in \mathcal{C}^{(h)}} \ell^h |C' \cap C \cap R_h| + 1000 \frac{d+\ell}{\ell} \log(\ell)|C|.$$

Moreover, this selection of consistent sets can be found in polynomial time.

Before we prove this lemma, we give an intuition of the statement. Consider the sets $R_1, \ldots, R_d$ constructed as in Random Experiment 7. Then for $C' \in \mathcal{C}^{(h)}$ we have $\mathbb{E}[\ell^h |C' \cap C \cap R_h|] = |C' \cap C|$. Hence

$$\sum_{h \leq k} \sum_{K \in \mathcal{K}^{(h)}} |K \cap C| = \mathbb{E}[\sum_{h \leq k} \sum_{K \in \mathcal{K}^{(h)}} \ell^h |K \cap C \cap R_h|].$$

Similarly for the right-hand side we have

$$\mathbb{E}[\frac{1}{\ell} \sum_{j \leq h \leq k} \sum_{C' \in \mathcal{C}^{(h)}} \ell^h |C' \cap C \cap R_h| + O(\frac{d+\ell}{\ell} \log(\ell)|C|)]$$
$$= \frac{1}{\ell} \underbrace{\sum_{j \leq h \leq k} \sum_{C' \in \mathcal{C}^{(h)}} |C' \cap C|}_{\leq \ell |C|} + O\left(\frac{d+\ell}{\ell} \log(\ell)|C|\right) = O\left(\frac{d+\ell}{\ell} \log(\ell)|C|\right).$$

Hence, the lemma says that each resource in C is roughly covered $O((d+\ell)/\ell \cdot \log(\ell))$ times by smaller configurations.

We now proceed to the proof of Lemma 11.

Proof. We perform the following random experiment and show with LLL that there is a positive probability of success.

▶ **Random Experiment 12.** *For each group A, select one consistent set $\mathcal{S}_A$ uniformly at random. Then for each player $i \in A$ set $K_i = \mathcal{S}_A^{(i)}$.*

Given this experiment we can define the following random variables. For all $h = 0, \dots, d$ and $i \in P$ we define

$$X_{i,C}^{(h)} = \sum_{K \in \mathcal{K}_i^{(h)}} |K \cap C \cap R_h| \leq \min\{3/2 \cdot \ell^4, |C \cap R_h|\}.$$

Let $X_C^{(h)} = \sum_{i=1}^m X_{i,C}^{(h)}$. Then

$$\mathbb{E}[X_C^{(h)}] \leq \frac{1}{\ell} \sum_{C' \in \mathcal{C}^{(h)}} |C' \cap C \cap R_h| \leq |C \cap R_h|.$$

We are now ready to define the bad events on which we will apply the Lovasz Local Lemma. As we will show later, if none of them occur, Lemma 11 will hold. For each k, $C \in \mathcal{C}^{(k)}$, and $h \leq k$ let $B_C^{(h)}$ be the event that

$$X_C^{(h)} \geq \begin{cases} \mathbb{E}[X_C^{(h)}] + 63|C \cap R_h| \log(\ell) & \text{if } k-5 \leq h \leq k, \\ \mathbb{E}[X_C^{(h)}] + 135|C \cap R_h| \log(\ell) \cdot \ell^{-1} & \text{if } h \leq k-6. \end{cases}$$

The intuitive reason as to why we define these two different bad events can be summarized as follows. In the case $h \leq k-6$, we are counting how many times C is intersected by configurations that are much smaller than C. Hence the size of this intersection can be written as a sum of independent random variables of value at most $O(\ell^4)$ which is much smaller than the total size of the configuration $|C \cap R_h|$. Since the random variables are in a much smaller range, Chernoff bounds give much better concentration guarantees and we can afford a very small deviation from the expectation. In the other case, we do not have this property hence we need a bigger deviation to maintain a sufficiently low probability of failure. However, this does not hurt the statement of Lemma 11 since we sum this bigger deviation only a constant number of times. One key idea to be able to apply Lovasz Local Lemma here is also to consider intersection of C with smaller configurations but restricted to a set R_h of convenient granularity. One can notice that $|C' \cap R_h| = \text{poly}(\ell)$ if $C' \in \mathcal{C}^{(h)}$ (by the assumption made in Lemma 11). This allows to reduce significantly the dependencies between bad events which is crucial to make any use of LLL here.

With this in mind, we claim that the probability of each bad event happening is small.

▷ Claim 13. For each k, $C \in \mathcal{C}^{(k)}$, and $h \leq k$ we have

$$\mathbb{P}[B_C^{(h)}] \leq \exp\left(-2\frac{|C \cap R_h|}{\ell^9} - 18\log(\ell)\right).$$

Proof. Consider first the case that $h \geq k-5$. By a Chernoff bound (see full version for the precise formulation) with

$$\delta = 63\frac{|C \cap R_h| \log(\ell)}{\mathbb{E}[X_C^{(h)}]} \geq 1$$

we get

$$\mathbb{P}[B_C^{(h)}] \leq \exp\left(-\frac{\delta\mathbb{E}[X_C^{(h)}]}{3|C\cap R_h|}\right) \leq \exp(-21\log(\ell))) \leq \exp\Bigg(-2\underbrace{\frac{|C\cap R_h|}{\ell^9}}_{\leq 3/2} - 18\log(\ell)\Bigg).$$

Now consider $h \leq k-6$. We apply again a Chernoff bound with

$$\delta = 135\frac{|C\cap R_h|\log(\ell)}{\ell\mathbb{E}[X_C^{(h)}]} \geq \frac{1}{\ell}.$$

This implies

$$\begin{aligned}\mathbb{P}[B_C^{(h)}] \leq \exp\left(-\frac{\min\{\delta,\delta^2\}\mathbb{E}[X_C^{(h)}]}{3\cdot 3/2\cdot \ell^4}\right) &\leq \exp\left(-30\frac{|C\cap R_h|\log(\ell)}{\ell^6}\right)\\ &\leq \exp\left(-2\frac{|C\cap R_h|}{\ell^9} - 18\log(\ell)\right).\end{aligned}$$ ◁

We can now state Lovasz Local Lemma and use it in our setting.

▶ **Proposition 14** (Lovasz Local Lemma (LLL)). *Let $B_1,\dots,B_t$ be bad events, and let $G=(\{B_1,\dots,B_t\},E)$ be a dependency graph for them, in which for every i, event B_i is mutually independent of all events B_j for which $(B_i,B_j)\notin E$. Let x_i for $1\leq i\leq t$ be such that $0<x(B_i)<1$ and $\mathbb{P}[B_i]\leq x(B_i)\prod_{(B_i,B_j)\in E}(1-x(B_j))$. Then with positive probability no event B_i holds.*

Let $k\in\{0,\dots,d\}$, $C\in\mathcal{C}^{(k)}$ and $h\leq k$. For event $B_C^{(h)}$ we set

$$x(B_C^{(h)}) = \exp(-|C\cap R_h|/\ell^9 - 18\log(\ell)).$$

We now analyze the dependencies of $B_C^{(h)}$. The event depends only on random variables $\mathcal{S}_A$ for groups A that contain at least one player i that has a configuration in $\mathcal{C}_i^{(h)}$ which overlaps with $C\cap R_h$. The number of such configurations (in particular, of such groups) is at most $\ell|C\cap R_h|$ since the hypergraph is regular.

In each of these groups, we count at most $\log(n)$ players, each having ℓ configurations hence in total at most $\ell\cdot\log(n)$ configurations.

Each configuration $C'\in\mathcal{C}^{(h')}$ can only influence those events $B_{C''}^{(h')}$ where $C'\cap C''\cap R_{h'}\neq\emptyset$. Since $|C'\cap R_{h'}|\leq 3/2\cdot\ell^4$ and since each resource appears in at most ℓ configurations, we see that each configuration can influence at most $3/2\cdot\ell^5$ events.

Putting everything together, we see that the bad event $B_C^{(h)}$ is independent of all but at most

$$(\ell|C\cap R_h|)\cdot(\ell\cdot\log(n))\cdot(3/2\cdot\ell^5) = 3/2\cdot\ell^7\cdot\log(n)|C\cap R_h| \leq |C\cap R_h|\ell^8$$

other bad events.

We can now verify the condition for Proposition 14 by calculating

$$\begin{aligned}x(B_C^{(h)}) &\prod_{(B_C^{(h)},B_{C'}^{(h')})\in E}(1-x(B_{C'}^{(h')}))\\ &\geq \exp(-|C\cap R_h|/\ell^9 - 18\log(\ell))\cdot(1-\ell^{-18})^{|C\cap R_h|\ell^8}\\ &\geq \exp(-|C\cap R_h|/\ell^9 - 18\log(\ell))\cdot\exp(-|C\cap R_h|/\ell^9)\\ &\geq \exp(-2|C\cap R_h|/\ell^9 - 18\log(\ell)) \geq \mathbb{P}[B_C^{(h)}].\end{aligned}$$

By LLL we have that with positive probability none of the bad events happen. Let $k \in \{0, \dots, d\}$ and $C \in \mathcal{C}^{(k)}$. Then for $k - 5 \leq h \leq k$ we have

$$\ell^h X_C^{(h)} \leq \ell^h \mathbb{E}[X_C^{(h)}] + 63\ell^h |C \cap R_h| \log(\ell) \leq \ell^h \mathbb{E}[X_C^{(h)}] + 95|C| \log(\ell).$$

Moreover, for $h \leq k - 6$ it holds that

$$\ell^h X_C^{(h)} \leq \ell^h \mathbb{E}[X_C^{(h)}] + 135\ell^{h-1} |C \cap R_h| \log(\ell) \leq \ell^h \mathbb{E}[X_C^{(h)}] + 203|C| \log(\ell) \cdot \ell^{-1}.$$

We conclude that, for any $0 \leq j \leq k$,

$$\begin{aligned}\sum_{j \leq h \leq k} \sum_{K \in \mathcal{K}^{(h)}} \ell^h |K \cap C \cap R_h| &\leq \sum_{j \leq h \leq k} \ell^h \mathbb{E}[X_C^{(h)}] + 1000 \frac{(k-j+1)+\ell}{\ell} |C| \log(\ell) \\ &\leq \frac{1}{\ell} \sum_{j \leq h \leq k} \ell^h \sum_{C' \in \mathcal{C}^{(h)}} |C' \cap C \cap R_h| + 1000 \frac{d+\ell}{\ell} |C| \log(\ell).\end{aligned}$$

This proves Lemma 11. ◀

▶ Remark 15. Since there are at most $\text{poly}(n, m, \ell)$ bad events and each bad event B has $\frac{x(B)}{1-x(B)} \leq 1/2$ (because $x(B) \leq \ell^{-18}$), the constructive variant of LLL by Moser and Tardos [14] can be applied to find a selection of configurations such that no bad events occur in randomized polynomial time.

3.4 Assignment of resources to configurations

In this subsection, we show how all the previously established properties allow us to find, in polynomial time, a good assignment of resources to the configurations $\mathcal{K}$ chosen as in the previous subsection. We will denote as in the previous subsection $\mathcal{K}_i^{(k)} = \{K_i\}$ if $K_i \in \mathcal{C}_i^{(k)}$ and $\mathcal{K}_i^{(k)} = \emptyset$ otherwise. We also define $\mathcal{K}^{(k)} = \bigcup_i \mathcal{K}_i^{(k)}$ and $\mathcal{K}^{(\geq k)} = \bigcup_{h \geq k} \mathcal{K}^{(k)}$. Finally we define the parameter

$$\gamma = 100.000 \frac{d+\ell}{\ell} \log(\ell),$$

which will define how many times each resource can be assigned to configurations in an intermediate solution. Note that $d \leq \log(n)/\log(\ell)$. By our choice of $\ell = 300.000 \log^3(n)$, we have that $\gamma \leq 310.000 \log\log(n)$. Lemma 11 implies the following bound.

▷ Claim 16. For any $k \geq 0$, any $0 \leq j \leq k$, and any $C \in \mathcal{K}^{(k)}$

$$\sum_{j \leq h \leq k} \sum_{K \in \mathcal{K}^{(h)}} \ell^h |K \cap C \cap R_h| \leq 2000 \frac{d+\ell}{\ell} \log(\ell) |C|$$

Proof. By Lemma 11 we have that

$$\sum_{j \leq h \leq k} \sum_{K \in \mathcal{K}^{(h)}} \ell^h |K \cap C \cap R_h| \leq \frac{1}{\ell} \sum_{j \leq h \leq k} \sum_{C' \in \mathcal{C}^{(h)}} \ell^h |C' \cap C \cap R_h| + 1000 \frac{d+\ell}{\ell} \log(\ell) |C|.$$

Furthermore, by Lemma 9, we get

$$\sum_{C' \in \mathcal{C}^{(h)}} \ell^h |C' \cap C \cap R_h| \leq \ell^h \frac{10}{\ell^h} \left(|C| + \sum_{C' \in \mathcal{C}^{(h)}} |C' \cap C| \right).$$

Finally note that each resource appears in at most ℓ configurations, hence

$$\sum_{j\le h\le k}\sum_{C'\in\mathcal{C}^{(h)}}|C'\cap C|\le \ell|C|.$$

Putting everything together we conclude

$$\begin{aligned}\sum_{j\le h\le k}\sum_{K\in\mathcal{K}^{(h)}}\ell^h|K\cap C\cap R_h| &\le \frac{1}{\ell}\sum_{j\le h\le k}\sum_{C'\in\mathcal{C}^{(h)}}\ell^h|C'\cap C\cap R_h| + 1000\frac{d+\ell}{\ell}\log(\ell)|C|\\ &\le \frac{1}{\ell}\sum_{j\le h\le k}10\left(|C|+\sum_{C'\in\mathcal{C}^{(h)}}|C'\cap C|\right)+1000\frac{d+\ell}{\ell}\log(\ell)|C|\\ &\le \frac{k-j}{\ell}10|C|+10|C|+1000\frac{d+\ell}{\ell}\log(\ell)|C|\\ &\le 20|C|+1000\frac{d+\ell}{\ell}\log(\ell)|C|\\ &\le 2000\frac{d+\ell}{\ell}\log(\ell)|C|.\end{aligned}$$ ◁

We can now proceed to the main technical part of this section which is the following lemma proved by induction.

▶ **Lemma 17.** *For any $j\ge 0$, there exists an assignment of resources of R_j to configurations in $\mathcal{K}^{(\ge j)}$ such that no resource is taken more than γ times and each configuration $C\in\mathcal{K}^{(k)}$ $(k\ge j)$ receives at least*

$$\left(1-\frac{1}{\log(n)}\right)^{2(k-j)}\ell^{k-j}|C\cap R_k| - \frac{3}{\gamma}\sum_{j\le h\le k}\sum_{K\in\mathcal{K}^{(h)}}\ell^{h-j}|K\cap C\cap R_h|$$

resources from R_k.

Before going through the proof, we give here the intuition of why this is what we want to prove. Note that the term $\ell^{k-j}|C\cap R_k|$ is roughly equal to $\ell^{-j}|C|$ by the properties of the resource sets (precisely Lemma 8). The second term

$$\sum_{j\le h\le k}\sum_{K\in\mathcal{K}^{(h)}}\ell^{h-j}|K\cap C\cap R_h|$$

can be shown to be

$$O\left(\ell^{-j}\frac{d+\ell}{\ell}\log(\ell)|C|\right)=O(\ell^{-j}\log\log(n)|C|)$$

by Claim 16. Hence by choosing γ to be $\Theta(\log\log(n))$ we get that the bound in Lemma 17 will be $\Theta(\ell^{-j}|C|)$. At the end of the induction, we have $j=0$ which indeed implies that we have an assignment in which configurations receive

$$\Theta(\ell^{-0}|C|)=\Theta(|C|)$$

resources and such that each resource is assigned to at most $O(\log\log(n))$ configurations. With this in mind, we give the formal proof of Lemma 17.

Proof. We start from the biggest configurations and then iteratively reconstruct a good solution for smaller and smaller configurations. Recall d is the smallest integer such that $\mathcal{K}^{(\ge d)}$ is empty. Our base case for these configurations in $\mathcal{K}^{(\ge d)}$ is vacuously satisfied.

Now assume that we have a solution at level j, i.e. an assignment of resources to configurations in $\mathcal{K}^{(\geq j)}$ such that no resource is taken more than γ times and each configuration $C \in \mathcal{K}^{(k)}$ such that $k \geq j$ receives at least

$$\left(1 - \frac{1}{\log(n)}\right)^{2(k-j)} \ell^{k-j}|C \cap R_k| - \frac{3}{\gamma} \sum_{j \leq h \leq k} \sum_{K \in \mathcal{K}^{(h)}} \ell^{h-j}|K \cap C \cap R_h|$$

resources from R_j. We show that this implies a solution at level $j-1$ in the following way. First by Lemma 10, this implies an assignment of resources of R_{j-1} to configurations in $\mathcal{K}^{(\geq j)}$ such that each $C \in \mathcal{K}^{(k)}$ receives at least

$$\begin{aligned}
&\left(1 - \frac{1}{\log(n)}\right) \ell \left(\ell^{k-j} \left(1 - \frac{1}{\log(n)}\right)^{2(k-j)} |C \cap R_k| - \frac{3}{\gamma} \sum_{j \leq h \leq k} \sum_{K \in \mathcal{K}^{(h)}} \ell^{h-j}|K \cap C \cap R_h| \right) \\
&= \left(1 - \frac{1}{\log(n)}\right)^{2(k-(j-1))-1} \ell^{k-(j-1)}|C \cap R_k| - \frac{3}{\gamma} \left(1 - \frac{1}{\log(n)}\right) \sum_{j \leq h \leq k} \sum_{K \in \mathcal{K}^{(h)}} \ell^{h-(j-1)}|K \cap C \cap R_h| \\
&\geq \left(1 - \frac{1}{\log(n)}\right)^{2(k-(j-1))-1} \ell^{k-(j-1)}|C \cap R_k| - \frac{3}{\gamma} \sum_{j \leq h \leq k} \sum_{K \in \mathcal{K}^{(h)}} \ell^{h-(j-1)}|K \cap C \cap R_h|
\end{aligned}$$

resources and no resource of R_{j-1} is taken more than γ times. Note that we can apply Lemma 10 since we have by Claim 16 and Lemma 8

$$\begin{aligned}
&\left(1 - \frac{1}{\log(n)}\right)^{2(k-j)} \ell^{k-j}|C \cap R_k| - \frac{3}{\gamma} \sum_{j \leq h \leq k} \sum_{K \in \mathcal{K}^{(h)}} \ell^{h-j}|K \cap C \cap R_h| \\
&\geq \frac{\ell^{k-j}}{e^2}|C \cap R_k| - \frac{3}{\gamma} 2000 \ell^{-j} \frac{d+\ell}{\ell} \log(\ell)|C| \\
&\geq \ell^{-j}|C| \left(\frac{1}{2e^2} - \frac{6000}{\gamma} \frac{d+\ell}{\ell} \log(\ell) \right) \\
&\geq \frac{\ell^{-j}|C|}{3e^2} > \frac{\ell^3}{1000}
\end{aligned}$$

Now consider configurations in $\mathcal{K}^{(j-1)}$ and proceed for them as follows. Give to each $C \in \mathcal{K}^{(j-1)}$ all the resources in $C \cap R_{j-1}$ except all the resources that appear in more than γ configurations in $\mathcal{K}^{(j-1)}$. Since each deleted resource is counted at least γ times in the sum $\sum_{K \in \mathcal{K}^{(j-1)}} |K \cap C \cap R_{j-1}|$, we have that each configuration C in $\mathcal{K}^{(j-1)}$ receives at least

$$|C \cap R_{j-1}| - \frac{1}{\gamma} \sum_{K \in \mathcal{K}^{(j-1)}} |K \cap C \cap R_{j-1}|$$

resources and no resource is taken more than γ times by configurations in $\mathcal{K}^{(j-1)}$. Notice that now every resource is taken no more than γ times by configurations in $\mathcal{K}^{(\geq j)}$ and no more than γ times by configurations in $\mathcal{K}^{(j-1)}$ which in total can sum up to 2γ times.

Therefore to finish the proof consider an resource $i \in R_{j-1}$. This resource is taken b_i times by configurations in $\mathcal{K}^{(\geq j)}$ and a_i times by configurations in $\mathcal{K}^{(j-1)}$. If $a_i + b_i \leq \gamma$, nothing needs to be done. Otherwise, denote by O the set of problematic resources (i.e. resources i such that $a_i + b_i > \gamma$). For every $i \in O$, select uniformly at random $a_i + b_i - \gamma$ configurations in $\mathcal{K}^{(\geq j)}$ that currently contain resource i and delete the resource from these configurations. When this happens, each configuration in $C \in \mathcal{K}^{(\geq j)}$ that contains i has a probability of $(a_i + b_i - \gamma)/b_i$ to be selected to loose this resource. Hence the expected number of resources that C looses with such a process is

$$\mu = \sum_{i \in O \cap C} \frac{a_i + b_i - \gamma}{b_i}$$

It is not difficult to prove the following claim.

▷ Claim 18. For any $C \in \mathcal{K}^{(\geq j)}$,

$$\frac{1}{\gamma^2} \sum_{K \in \mathcal{K}^{(j-1)}} |K \cap C \cap R_{j-1} \cap O| \leq \mu \leq \frac{2}{\gamma} \sum_{K \in \mathcal{K}^{(j-1)}} |K \cap C \cap R_{j-1} \cap O|$$

Proof. Note that we can write

$$\mu = \sum_{i \in O \cap C} \frac{a_i + b_i - \gamma}{b_i} \leq \max_{i \in O \cap C} \left\{ \frac{a_i + b_i - \gamma}{a_i b_i} \right\} \sum_{K \in \mathcal{K}^{(j-1)}} |K \cap C \cap R_{j-1} \cap O|.$$

The reason for this is that each resource i accounts for an expected loss of $(a_i + b_i - \gamma)/b_i$ while it is counted a_i times in the sum

$$\sum_{K \in \mathcal{K}^{(j-1)}} |K \cap C \cap R_{j-1} \cap O|.$$

Similarly,

$$\mu = \sum_{i \in O \cap C} \frac{a_i + b_i - \gamma}{b_i} \geq \min_{i \in O \cap C} \left\{ \frac{a_i + b_i - \gamma}{a_i b_i} \right\} \sum_{K \in \mathcal{K}^{(j-1)}} |K \cap C \cap R_{j-1} \cap O|.$$

Note that by assumption we have that $a_i + b_i > \gamma$. This implies that either a_i or b_i is greater than $\gamma/2$. Assume w.l.o.g. that $a_i \geq \gamma/2$. Since by assumption $a_i \leq \gamma$ we have that

$$\frac{a_i + b_i - \gamma}{a_i b_i} \leq \frac{b_i}{a_i b_i} = \frac{1}{a_i} \leq \frac{2}{\gamma}.$$

In the same manner, since $a_i + b_i > \gamma$ and that $a_i, b_i \leq \gamma$, we can write

$$\frac{a_i + b_i - \gamma}{a_i b_i} \geq \frac{1}{a_i b_i} \geq \frac{1}{\gamma^2}.$$

We therefore get the following bounds

$$\frac{1}{\gamma^2} \sum_{K \in \mathcal{K}^{(j-1)}} |K \cap C \cap R_{j-1} \cap O| \leq \mu \leq \frac{2}{\gamma} \sum_{K \in \mathcal{K}^{(j-1)}} |K \cap C \cap R_{j-1} \cap O|,$$

which is what we wanted to prove. ◁

Assume then that $\mu \leq \frac{|C \cap R_k|}{10^{12} \log^3(n)}$. Note that C cannot loose more than $\sum_{K \in \mathcal{K}^{(j-1)}} |K \cap C \cap R_{j-1} \cap O|$ resources in any case. Therefore, by assumption on μ, and since

$$\mu \geq \frac{1}{\gamma^2} \sum_{K \in \mathcal{K}^{(j-1)}} |K \cap C \cap R_{j-1} \cap O| \ ,$$

we have that

$$\sum_{K \in \mathcal{K}^{(j-1)}} |K \cap C \cap R_{j-1} \cap O| \leq \frac{\gamma^2}{10^{12} \log^3(n)} |C \cap R_k| \leq \frac{10^{11} \log^2 \log(n)}{10^{12} \log^3(n)} |C \cap R_k| \leq \frac{1}{\log(n)} |C \cap R_k| \ .$$

Therefore C looses at most $|C \cap R_k| / \log(n)$ resources. Otherwise we have that

$$\mu > \frac{|C \cap R_k|}{10^{12} \log^2(n)} \geq \frac{\ell^3}{10^{12} \log^3(n)} \geq 200 \log(n)$$

by Lemma 8. Hence noting X the number of deleted resources in C we have that

$$\mathbb{P}\left(X \geq \frac{3}{2}\mu\right) \leq \exp\left(-\frac{\mu}{12}\right) \leq \frac{1}{n^{10}}.$$

With high probability no configuration looses more than

$$\frac{3}{2}\mu \leq \frac{3}{\gamma} \sum_{K \in \mathcal{K}^{(j-1)}} |K \cap C \cap R_{j-1} \cap O| \leq \frac{3}{\gamma} \sum_{K \in \mathcal{K}^{(j-1)}} |K \cap C \cap R_{j-1}|$$

resources. Hence each configuration $C \in \mathcal{K}^{(\geq j)}$ ends with at least

$$\begin{aligned}
&\left(1 - \frac{1}{\log(n)}\right)^{2(k-(j-1))-1} \ell^{k-(j-1)}|C \cap R_k| - \frac{3}{\gamma} \sum_{j \leq h \leq k} \sum_{K \in \mathcal{K}^{(h)}} \ell^{h-(j-1)}|K \cap C \cap R_h| \\
&- \frac{1}{\log(n)}\left(1 - \frac{1}{\log(n)}\right)^{2(k-(j-1))-1} \ell^{k-(j-1)}|C \cap R_k| - \frac{3}{\gamma} \sum_{K \in \mathcal{K}^{(j-1)}} |K \cap C \cap R_{j-1}| \\
&\geq \left(1 - \frac{1}{\log(n)}\right)^{2(k-(j-1))} \ell^{k-(j-1)}|C \cap R_k| - \frac{3}{\gamma} \sum_{j-1 \leq h \leq k} \sum_{K \in \mathcal{K}^{(h)}} \ell^{h-(j-1)}|K \cap C \cap R_h|
\end{aligned}$$

resources which concludes the proof of Lemma 17. ◀

Given Lemma 17 and the intuition below it, it is straightforward to prove the following corollary which will complete the proof of Theorem 6.

▶ **Corollary 19.** *There exists an assignment of resources R to $\mathcal{K}$ such that each configuration $C \in \mathcal{K}$ receives at least $\lfloor |C|/(100\gamma) \rfloor$ resources. Moreover, this assignment can be found in polynomial time.*

Proof. Lemma 17 with $k = 0$ and Claim 16 together imply that we can assign at least

$$\frac{|C|}{2e^2} - \frac{6000}{100.000}|C| \geq \frac{|C|}{100}$$

resources to every $C \in \mathcal{K}$ such that no resource in R is assigned more than γ times. In particular, we can fractionally assign at least $|C|/(100\gamma)$ resources to each $C \in \mathcal{K}$ such that no resource is assigned more than once. By integrality of the bipartite matching polytope, the corollary follows. ◀

References

1. Chidambaram Annamalai, Christos Kalaitzis, and Ola Svensson. Combinatorial algorithm for restricted max-min fair allocation. *ACM Trans. Algorithms*, 13(3):37:1–37:28, 2017. doi:10.1145/3070694.
2. Arash Asadpour, Uriel Feige, and Amin Saberi. Santa claus meets hypergraph matchings. *ACM Trans. Algorithms*, 8(3), 2012. doi:10.1145/2229163.2229168.
3. Nikhil Bansal and Maxim Sviridenko. The santa claus problem. In *Proceedings of the Thirty-Eighth Annual ACM Symposium on Theory of Computing*, STOC '06, page 31–40, New York, NY, USA, 2006. Association for Computing Machinery. doi:10.1145/1132516.1132522.
4. Deeparnab Chakrabarty, Julia Chuzhoy, and Sanjeev Khanna. On allocating goods to maximize fairness. In *50th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2009, October 25-27, 2009, Atlanta, Georgia, USA*, pages 107–116. IEEE Computer Society, 2009. doi:10.1109/FOCS.2009.51.

5 Siu-Wing Cheng and Yuchen Mao. Restricted max-min fair allocation. In Ioannis Chatzigiannakis, Christos Kaklamanis, Dániel Marx, and Donald Sannella, editors, *45th International Colloquium on Automata, Languages, and Programming, ICALP 2018, July 9-13, 2018, Prague, Czech Republic*, volume 107 of *LIPIcs*, pages 37:1–37:13. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2018. doi:10.4230/LIPIcs.ICALP.2018.37.

6 Siu-Wing Cheng and Yuchen Mao. Restricted max-min allocation: Approximation and integrality gap. In *46th International Colloquium on Automata, Languages, and Programming, ICALP 2019, July 9-12, 2019, Patras, Greece*, pages 38:1–38:13, 2019. doi:10.4230/LIPIcs.ICALP.2019.38.

7 Sami Davies, Thomas Rothvoss, and Yihao Zhang. A tale of santa claus, hypergraphs and matroids. In *Proceedings of the 2020 ACM-SIAM Symposium on Discrete Algorithms, SODA 2020, Salt Lake City, UT, USA, January 5-8, 2020*, pages 2748–2757, 2020. doi:10.1137/1.9781611975994.167.

8 Uriel Feige. On allocations that maximize fairness. In *Proceedings of the Nineteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '08, page 287–293, USA, 2008. Society for Industrial and Applied Mathematics.

9 Michel X Goemans, Nicholas JA Harvey, Satoru Iwata, and Vahab Mirrokni. Approximating submodular functions everywhere. In *Proceedings of the twentieth annual ACM-SIAM symposium on Discrete algorithms*, pages 535–544. SIAM, 2009.

10 Klaus Jansen and Lars Rohwedder. A note on the integrality gap of the configuration lp for restricted santa claus. *Information Processing Letters*, 164:106025, 2020. doi:10.1016/j.ipl.2020.106025.

11 Andreas Krause, Ram Rajagopal, Anupam Gupta, and Carlos Guestrin. Simultaneous placement and scheduling of sensors. In *Proceedings of the 8th International Conference on Information Processing in Sensor Networks, IPSN 2009, April 13-16, 2009, San Francisco, California, USA*, pages 181–192. IEEE Computer Society, 2009. URL: http://ieeexplore.ieee.org/document/5211932/.

12 Benny Lehmann, Daniel Lehmann, and Noam Nisan. Combinatorial auctions with decreasing marginal utilities. *Games Econ. Behav.*, 55(2):270–296, 2006. doi:10.1016/j.geb.2005.02.006.

13 J. K. Lenstra, D. B. Shmoys, and E. Tardos. Approximation algorithms for scheduling unrelated parallel machines. In *28th Annual Symposium on Foundations of Computer Science (sfcs 1987)*, pages 217–224, 1987. doi:10.1109/SFCS.1987.8.

14 Robin A Moser and Gábor Tardos. A constructive proof of the general lovász local lemma. *Journal of the ACM (JACM)*, 57(2):1–15, 2010.

15 Lukas Polacek and Ola Svensson. Quasi-polynomial local search for restricted max-min fair allocation. In Artur Czumaj, Kurt Mehlhorn, Andrew Pitts, and Roger Wattenhofer, editors, *Automata, Languages, and Programming*, pages 726–737, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg.

16 Jan Vondrák. Optimal approximation for the submodular welfare problem in the value oracle model. In Cynthia Dwork, editor, *Proceedings of the 40th Annual ACM Symposium on Theory of Computing, Victoria, British Columbia, Canada, May 17-20, 2008*, pages 67–74. ACM, 2008. doi:10.1145/1374376.1374389.

On Coresets for Fair Clustering in Metric and Euclidean Spaces and Their Applications

Sayan Bandyapadhyay
Department of Informatics, University of Bergen, Norway

Fedor V. Fomin
Department of Informatics, University of Bergen, Norway

Kirill Simonov
Department of Informatics, University of Bergen, Norway

Abstract

Fair clustering is a variant of constrained clustering where the goal is to partition a set of colored points. The fraction of points of each color in every cluster should be more or less equal to the fraction of points of this color in the dataset. This variant was recently introduced by Chierichetti et al. [NeurIPS 2017] and became widely popular. This paper proposes a new construction of coresets for fair k-means and k-median clustering for Euclidean and general metrics based on random sampling. For the Euclidean space $\mathbb{R}^d$, we provide the first coresets whose size does not depend exponentially on the dimension d. The question of whether such constructions exist was asked by Schmidt, Schwiegelshohn, and Sohler [WAOA 2019] and Huang, Jiang, and Vishnoi [NeurIPS 2019]. For general metric, our construction provides the first coreset for fair k-means and k-median.

New coresets appear to be a handy tool for designing better approximation and streaming algorithms for fair and other constrained clustering variants. In particular, we obtain

- the first fixed-parameter tractable (FPT) PTAS for fair k-means and k-median clustering in $\mathbb{R}^d$. The near-linear time of our PTAS improves over the previous scheme of Böhm, Fazzone, Leonardi, and Schwiegelshohn [ArXiv 2020] with running time $n^{\mathrm{poly}(k/\epsilon)}$;
- FPT "true" constant-approximation for metric fair clustering. All previous algorithms for fair k-means and k-median in general metric are bicriteria and violate the fairness constraints;
- FPT 3-approximation for lower-bounded k-median improving the best-known 3.736 factor of Bera, Chakrabarty, and Negahbani [ArXiv 2019];
- the first FPT constant-approximations for metric chromatic clustering and ℓ-Diversity clustering;
- near linear-time (in n) PTAS for capacitated and lower-bounded clustering improving over PTAS of Bhattacharya, Jaiswal, and Kumar [TOCS 2018] with super-quadratic running time;
- a streaming $(1+\epsilon)$-approximation for fair k-means and k-median of space complexity polynomial in k, d, ϵ and $\log n$ (the previous algorithms have exponential space complexity on either d or k).

2012 ACM Subject Classification Theory of computation → Facility location and clustering

Keywords and phrases fair clustering, coresets, approximation algorithms

Digital Object Identifier 10.4230/LIPIcs.ICALP.2021.23

Category Track A: Algorithms, Complexity and Games

Related Version *Full Version*: `https://arxiv.org/abs/2007.10137`

Funding This work is supported by the Research Council of Norway via the project "MULTIVAL".

Acknowledgements The authors are thankful to Vincent Cohen-Addad for sharing the full version of [19].

1 Introduction

Given a set of n data points in a metric space and an integer k, *clustering* is the task of partitioning the points into k groups or clusters so that the points in each cluster are similar. In this paper, we consider clustering problems with *fairness* constraints. While there are many competing notions of fairness in the literature, here we consider clustering with fairness constraints or fair clustering as introduced by Chierichetti et al. [16] in their seminal work. The notion became widely popular within a short period triggering a large body of new work [38, 7, 9, 28, 4, 11, 15, 1, 32]. The idea of fair clustering is to enforce additional (fairness) constraints to remove the inherent bias or discrimination from vanilla (unconstrained) clustering. For example, suppose we have a sensitive feature (e.g, race or gender). We want to find a clustering where the fraction of points from a traditionally underrepresented group in every cluster is more or less equal to the fraction of points from this group in the dataset. Indeed, the work of Chierichetti et al. [16] shows that clustering computed by classical vanilla algorithms can lead to widely varied ratios for a particular group, especially when the number of clusters is large enough.

In this work, we consider the fair clustering model independently formulated by Bercea et al. [9] and Bera et al. [7]. In this model, we are given ℓ groups $P_1, \ldots, P_\ell$ of points in a metric space and balance parameters $\alpha_i, \beta_i \in [0, 1]$ for each group $1 \leq i \leq \ell$. A clustering is *fair* if the fraction of points from group i in every cluster is at least β_i and at most α_i. Additionally, in [7], the groups are allowed to overlap, i.e, a point can belong to multiple protected classes. We refer to the fair clustering problem with overlapping groups as (α, β)-*fair clustering*. We note that this is the most general version of fair clustering considered in the literature, and this is the notion of fairness we adapt in this paper. Both [9] and [7] obtain polynomial time $O(1)$-approximation for this problem that violates the fairness constraints by at most small additive factors.

We denote by Γ the maximum number of distinct collections of groups to which a point belongs. If all the groups are disjoint, then $\Gamma = \ell$. Note that if every point belongs to at most Λ groups, then Γ is at most ℓ^Λ. As noted in [7, 28], while Λ can very well be more than 1, it is usually a constant in most of the applications. Thus, in this case, $\Gamma = \ell^{O(1)}$, which is expected to be much smaller compared to n, the total number of points in the union of the groups.

Several works related to fair clustering were devoted to scalability [28, 38, 11, 4]. Along this line, in a beautiful work, Schmidt et al. [38] defined coresets for fair clustering. Note that a coreset for a center-based vanilla clustering problem is roughly a summary of the data that for every set C of k centers approximately (within $(1 \pm \epsilon)$ factor) preserves the optimal clustering cost. Over the years, researchers have paid increasing attention to the design of coreset construction algorithms to optimize the coreset size. Indeed, finding smaller coresets continues to be an active research area in the context of vanilla k-median and k-means clustering. For general metric spaces, the best-known upper bound on coreset size is $O((k \log n)/\epsilon^2)$ [21] and the lower bound is known to be $\Omega((k \log n)/\epsilon)$ [5]. For Euclidean spaces of dimension d, it is possible to construct coresets, based on random sampling, of size $(k/\epsilon)^{O(1)}$ [22, 39, 29, 13], which in particular does not depend on n and d.

In the vanilla version of a clustering problem, given the cluster centers, clusters are formed by assigning each point to its nearest center. In contrast, in a constrained version, such an assignment might not lead to a clustering that satisfies the constraints. Hence, for fair clustering, we need a stronger definition of coreset. Accordingly, Schmidt et al. [38] initiated the study of fair coresets. Schmidt et al. [38] and subsequently Huang et al. [28]

designed deterministic algorithms in $\mathbb{R}^d$ that construct fair coresets whose sizes exponentially depend on d. To remove this exponential dependency on d, Schmidt et al. [38] proposed an interesting open question whether it is possible to use random sampling for construction of fair coresets. Huang et al. [28] also suggested the same open question. Besides, Huang et al. asked whether it is possible to achieve a similar size bound as in the vanilla setting.

1.1 Our Results and Contributions

We study fair clustering under the k-median and k-means objectives. Our first main result is the following theorem.

▶ **Theorem 1.** *There is an $O(n(k+\ell))$ time randomized algorithm that w.p. at least $1-1/n$, computes a coreset of size $O(\Gamma(k\log n)^2/\epsilon^3)$ for (α,β)-fair k-median and $O(\Gamma(k\log n)^7/\epsilon^5)$ for (α,β)-fair k-means, where Γ is the number of distinct collections of groups to which a point may belong. If the groups are disjoint, the algorithm runs in $O(nk)$ time. Moreover, in $\mathbb{R}^d$, the coreset sizes are $O\left(\frac{\Gamma}{\epsilon^3}\cdot k^2\log n(\log n + d\log(1/\epsilon))\right)$ for (α,β)-fair k-median[1] and $O\left(\frac{\Gamma}{\epsilon^5}\cdot k^7(\log n)^6(\log n + d\log(1/\epsilon))\right)$ for (α,β)-fair k-means.*

Theorem 1 provides the first coreset construction for fair clustering problem in general metric spaces. Note that if the number of groups is just 1, we obtain coresets of size $O_\epsilon(\text{poly}(k\log n))$, which is somewhat comparable to the best-known bound of $O_\epsilon(k\log n)$ [21] in the vanilla case. We also note that this is the first sampling based coreset construction scheme for fair clustering, and in $\mathbb{R}^d$, the first coreset construction scheme where the size of the coreset does not depend exponentially on the dimension d (see Table 1). In fact, the dependency on d is only linear. Specifically, our result improves the bound (for k-median) in [28] by a factor of $\Theta\left(\frac{\epsilon^{-d+3}}{\log n(\log n+d)}\right)$ (see Table 1). Thus, if d is sufficiently large, our coreset size is much smaller compared to theirs. In fact, the dependency of the previous coreset size on n can be super-polylogarithmic (or super-polynomial) when d is large, whereas ours is only polylogarithmic in the worst case. In the light of the above discussion, our result solves the open question proposed in [38] and partly solves the open question proposed in [28].

Table 1 Previous and current coreset results in $\mathbb{R}^d$.

	k-median		k-means	
	size	construction time	size	construction time
[38]			$O(\Gamma k\epsilon^{-d-2}\log n)$	$O(k\epsilon^{-d-2}n\log n)$
[28]	$O(\Gamma k^2\epsilon^{-d})$	$O(k\epsilon^{-d+1}n)$	$O(\Gamma k^3\epsilon^{-d-1})$	$O(k\epsilon^{-d+1}n)$
This	$O(\frac{\Gamma}{\epsilon^3}\cdot k^2\log n(\log n+ d\log(1/\epsilon)))$	$O(nd(k+\ell))$	$O(\frac{\Gamma}{\epsilon^5}\cdot k^7(\log n)^6(\log n+ d\log(1/\epsilon)))$	$O(nd(k+\ell))$

Actually, our coreset construction scheme, similar to [38, 28], is much more general in the following sense. The coreset can preserve not only the cost of optimal fair clustering, but also the cost of any optimal clustering with group-cardinality constraints. In particular, for

[1] The Euclidean version is not a special case of the general metrics case, as here the set of potential centers is infinite.

any clustering problem with constraints that can be expressed in terms of the number of elements that go from each group to each cluster (formally defined in Section 2), we obtain a small size coreset. This gives rise to new coresets for a wide range of clustering problems including lower-bounded clustering [40, 2, 8].

We further exploit the new coreset construction to design clustering algorithms in various settings. In general metrics, we obtain the first fixed-parameter tractable (FPT) constant-factor approximation for (α, β)-fair clustering with parameters k and Γ. That is, the running time of our algorithm is exponential only in the values k and Γ while polynomial in the size of the input. All previous constant-approximation algorithms were bicriteria and violated the fairness constraints by some additive factors. Hence, the study of FPT approximation is well-motivated. Our approximation factors are reasonably small and improve the best-known approximation factors of the existing bicriteria algorithms (see Table 2). Moreover, our coreset leads to improved constant FPT approximations for many other clustering problems. For example, we obtain an improved ≈ 3-approximation algorithm for lower-bounded k-median [40, 2, 8] that is FPT parameterized by k. Previously, the best-known factor for FPT approximation for this problem was 3.736 [8].

Based on our coreset, we also obtain the first FPT $(1+\epsilon)$-approximation for (α, β)-fair clustering in $\mathbb{R}^d$ with parameters k and Γ. For constant Γ, the running time of our algorithm is near-linear and up to $\log n$, matches the time of the algorithm of Kumar, Sabharwal, and Sen for vanilla clustering [34]. A comparison with the running time of the previous $(1+\epsilon)$-approximation algorithms can be found in Table 3. We also obtain FPT $(1+\epsilon)$-approximations with parameter k for the Euclidean version of several other problems including capacitated clustering [19, 17] and lower-bounded clustering. We note that these are the first $(1+\epsilon)$-approximations for these problems with near-linear dependency on n. For Euclidean capacitated clustering, quadratic time FPT algorithms follow due to [20, 10] (see Table 4). Also, the $(1+\epsilon)$-approximation for Euclidean capacitated clustering in [19] and [17] have running time $(k\epsilon^{-1})^{k\epsilon^{-O(1)}} n^{O(1)}$ and at least $n^{\epsilon^{-O(1)}}$ (see Table 4).

Table 2 Approximation results for (α, β)-fair clustering in general metrics. "multi" denotes if the algorithm can handle overlapping groups. In "approx." columns, the first (resp. second) value in a tuple is the approximation factor (resp. violation). [7] does not explicitly compute the $O(1)$ factor, but it is $> 3+\epsilon$ (resp. $> 9+\epsilon$) for k-median (resp. k-means), where ϵ is a sufficiently large constant.

	multi	k-median		k-means	
		approx.	time	approx.	time
[9]		$(4.675, 1)$	$\text{poly}(n)$	$(62.856, 1)$	$\text{poly}(n)$
[7]	✓	$(O(1), 4\Lambda + 3)$	$\text{poly}(n)$	$(O(1), 4\Lambda + 3)$	$\text{poly}(n)$
This		≈ 3	$(k\ell)^{O(k\ell)} n \log n$	≈ 9	$(k\ell)^{O(k\ell)} n \log n$
This	✓	≈ 3	$(k\Gamma)^{O(k\Gamma)} n \log n$	≈ 9	$(k\Gamma)^{O(k\Gamma)} n \log n$

Our coreset also leads to small space $(1+\epsilon)$-approximation in streaming setting for (α, β)-fair clustering in $\mathbb{R}^d$ when the groups are disjoint. We show how to maintain an $O(d^2 \ell \cdot \text{poly}(k \log n)/\epsilon^4)$ size coreset in each step. One can apply our $(1+\epsilon)$-approximation algorithm on the coreset to compute a near-optimal clustering. In the previous streaming algorithms [38], the space complexity depended exponentially on either d or k.

Our coreset construction scheme is based on the algorithm for the vanilla case due to Chen [14]. Chen showed that a small number of points can be chosen randomly from a metric space (in a clever way) that form a coreset for vanilla clustering. Our construction first divides the input points into a number of subsets, and for each subset, applies Chen's algorithm to compute a coreset for that subset. Our final coreset is the union of all these computed coresets. Surprisingly, our algorithm for fair clustering is as simple as that. Our main technical contribution is the analysis of this algorithm, which shows that the returned weighted subset of the points is a coreset for fair clustering with high probability.

Table 3 Running time of the $(1+\epsilon)$-approximations for fair clustering in $\mathbb{R}^d$.

	running time	version
[38]	$n^{O(k/\epsilon)}$	2-color, $(1,k)$-fair clustering
[11]	$n^{\text{poly}(k/\epsilon)}$	ℓ-color, $(1,k)$-fair clustering
This	$2^{\tilde{O}(k/\epsilon^{O(1)})}(k\Gamma)^{O(k\Gamma)}nd\log n$	(α,β)-fair clustering

Although our algorithm follows the framework of Chen's, the proof that the algorithm correctly computes a fair coreset is much more complicated. Our analysis is strongly inspired by the analysis of Cohen-Addad and Li [19] of Chen's algorithm in the context of capacitated clustering. However, there are two major difficulties of applying the analysis technique of [19] directly to our problem. Firstly, in our case input points belong to groups which can overlap, and secondly, fairness constraints are much more general compared to capacity constraints. The novelty of our work lies in overcoming the two above-mentioned hurdles. To overcome the first hurdle, we divide the input points into equivalence classes based on their group membership and using a probabilistic argument prove that it is possible to consider these classes separately. To overcome the second hurdle, we prove that fairness constraints can be treated as a collection of independent constraints each of which is similar to a capacity constraint. All these ideas together help us generalize the approach of [19] to fair clustering.

Table 4 Running time of the $(1+\epsilon)$-approximations for capacitated clustering in $\mathbb{R}^d$.

	running time
[20]	$2^{\text{poly}(k/\epsilon)}n^2(\log n)^{k+2}d$
[10]	$2^{\tilde{O}(k/\epsilon^{O(1)})}\cdot n^2(\log n)^2 d$
[19]	$(k\epsilon^{-1})^{k\epsilon^{-O(1)}}n^{O(1)}$
[17]	$n^{\epsilon^{-O(1)}}(d=2)$ $n^{(\log n/\epsilon)^{O(d)}}(d\geq 3)$
This	$2^{\tilde{O}(k/\epsilon^{O(1)})}nd^{O(1)}+nk^2\epsilon^{-O(1)}\log n$

Apart from the coreset construction, the novelty of our work lies in the design of an algorithm for computing the minimum cost fair assignment to given centers, based on mixed-integer linear programming. This is the heart of all our approximation algorithms.

1.2 Related Work

Schmidt et al. [38] defined the concept of fair coresets and gave coreset of size $O(\ell k \epsilon^{-d-2} \log n)$ for the disjoint group case of Euclidean (α, β)-fair k-means. They also gave an $n^{O(k/\epsilon)}$ time $(1+\epsilon)$-approximation for the two-color version of the problem. Using the framework in [25], Huang et al. [28] improved the coreset size bound of [38] by a factor of $\Theta\left(\frac{\log n}{\epsilon k^2}\right)$ and gave the first coreset for Euclidean (α, β)-fair k-median of size $O(\Gamma k^2 \epsilon^{-d})$. Böhm et al. [11] obtained near-linear time constant-approximation in a restricted setting. They also obtained an $n^{\text{poly}(k/\epsilon)}$ time $(1+\epsilon)$-approximation for the Euclidean version in the same setting.

Chierichetti et al. [16] gave a polynomial time $\Theta(t)$-approximation for a special version of (α, β)-fair k-median with two groups, where t is a balance parameter. Bera et al. [7] obtained polynomial time $O(1)$-approximation for (α, β)-fair clustering that violates the fairness constraints by at most an additive factor of $4\Lambda + 3$. For the disjoint group case, their violation factor is only 3. Independently, Bercea et al. [9] obtained algorithms with the same approximation guarantees as in [7] for the disjoint version, but with at most 1 additive factor violation. For other related works on fair clustering, see [4, 16, 37, 9, 7, 1, 15, 32, 33]. For related works on coresets, see [27, 26, 23, 35, 21, 22, 39, 6, 12, 29].

2 Preliminaries

In all the clustering problems we study in this paper, we are given a set P of points in a metric space $(\mathcal{X}, d(\cdot,\cdot))$, that we have to cluster. We are also given a set F of cluster centers in the same metric space. We note that P and F are not-necessarily disjoint, and in fact, P may be equal to F. We assume that the distance function $d(\cdot,\cdot)$ is provided by an oracle that for any given $x, y \in \mathcal{X}$ in constant time returns $d(x,y)$. In the Euclidean version of a clustering problem, $P \subseteq \mathbb{R}^d$, $F = \mathbb{R}^d$ and $d(\cdot,\cdot)$ is the Euclidean metric.[2] In the metric version, we assume that F is finite. Thus, strictly speaking, the Euclidean version is not a special case of the metric version. In the metric version, we denote $|P \cup F|$ by n and in the Euclidean version, $|P|$ by n. For any set S and a point p, $d(p,S) := \min_{q \in S} d(p,q)$. Also, for any integer $t \geq 1$, we denote the set $\{1, 2, \ldots, t\}$ by $[t]$.

In the k-median problem, given an additional parameter k, the goal is to select a set of at most k centers $C \subset F$ such that the quantity $\sum_{p \in P} d(p,C)$ is minimized. k-means is identical to k-median, except here we would like to minimize $\sum_{p \in P} (d(p,C))^2$.

Next, we define our notion of fair clustering following the definition in [7].

▶ **Definition 2** (Definition 1, [7]). *In the fair version of a clustering problem (k-median or k-means), one is additionally given ℓ many (not necessarily disjoint)* groups *of P, namely P_1, P_2, …, P_ℓ. One is also given two* fairness vectors *$\alpha, \beta \in [0,1]^\ell$, $\alpha = (\alpha_1, \ldots, \alpha_\ell)$, $\beta = (\beta_1, \ldots, \beta_\ell)$. The objective is to select a set of at most k centers $C \subset F$ and an assignment $\varphi : P \to C$ such that φ satisfies the following* fairness constraints*:*

$$|\{x \in P_i : \varphi(x) = c\}| \leq \alpha_i \cdot |\{x \in P : \varphi(x) = c\}|, \quad \forall c \in C, \forall i \in [\ell],$$
$$|\{x \in P_i : \varphi(x) = c\}| \geq \beta_i \cdot |\{x \in P : \varphi(x) = c\}|, \quad \forall c \in C, \forall i \in [\ell],$$

and $\text{cost}(\varphi)$ *is minimized among all such assignments.*

In the (α, β)-Fair k-median problem, $\text{cost}(\varphi) := \sum_{x \in P} d(x, \varphi(x))$, and in the (α, β)-Fair k-means problem, $\text{cost}(\varphi) := \sum_{x \in P} d(x, \varphi(x))^2$. To refer to these two problems together, we will use the term (α, β)-Fair Clustering. We call φ that satisfies the fairness constraints *a*

[2] Due to the lack of better notations, we denote the dimension by d and distance function by $d(\cdot,\cdot)$.

fair assignment. We denote the minimum cost of a fair assignment of a set of points P to a set of k centers C by faircost(P, C), and faircost(P) denotes the minimum of faircost(P, C') over all possible sets of k centers C'.

Next, we state our notion of coresets. We follow the definitions in [38, 28]. For a clustering problem with k centers and ℓ groups $P_1, \ldots, P_\ell$, a *coloring constraint* is a $k \times \ell$ matrix M having non-negative integer entries. The entry of M corresponding to row i and column j is denoted by M_{ij}. Next, we have the following observation, which was also noted in [38, 28].

▶ **Proposition 3.** *Given a set C of k centers, the assignment restriction required for (α, β)-Fair Clustering can be expressed as a collection of coloring constraints.*

In our definition, a coreset is required to preserve the optimal clustering cost w.r.t. all coloring constraints, and hence it also preserves the optimal fair clustering cost. Next, we formally define the cost of a clustering w.r.t. a set of centers and a coloring constraint.

First, consider the k-median objective. Suppose we are given a weight function $w : P \to \mathbb{R}_{\geq 0}$ (non-negative reals). Let $W \subseteq P \times \mathbb{R}$ be the set of pairs $\{(p, w(p)) \mid p \in P \text{ and } w(p) > 0\}$. For a set of centers $C = \{c_1, \ldots, c_k\}$ and a coloring constraint M, wcost(W, M, C) is the minimum value $\sum_{p \in P, c_i \in C} \psi(p, c_i) \cdot d(p, c_i)$ over all assignments $\psi : P \times C \to \mathbb{R}_{\geq 0}$ such that

1. For each $p \in P$, $\sum_{c_i \in C} \psi(p, c_i) = w(p)$.
2. For each $c_i \in C$ and group $1 \leq j \leq \ell$, $\sum_{p \in P_j} \psi(p, c_i) = M_{ij}$.

For k-means, wcost(W, M, C) is defined in the same way except it is the minimum value $\sum_{p \in P, c_i \in C} \psi(p, c_i) \cdot d(p, c_i)^2$. If there is no such assignment ψ, wcost$(W, M, C) = \infty$. When $w(p) = 1$ for all $p \in P$, we simply denote W by P and wcost(W, M, C) by cost(P, M, C). Now we define a coreset. We call it universal coreset, as it is required to preserve optimal clustering cost w.r.t. all coloring constraints.

▶ **Definition 4** (Universal coreset)**.** *For a given unweighted point set P and a clustering objective, a universal coreset is a set of weighted points $W \subseteq P \times \mathbb{R}$ such that for every set of centers C of size k and any coloring constraint M,*

$$(1 - \epsilon) \cdot cost(P, M, C) \leq wcost(W, M, C) \leq (1 + \epsilon) \cdot cost(P, M, C).$$

3 Our Techniques

Here, we describe the techniques and key ideas used to obtain the new results of the paper. The detailed version of our results and formal proofs appear in the attached full version. For simplicity, we limit our discussion to k-median clustering. We start with the coreset results.

3.1 Universal Coreset Construction

Our coreset construction algorithms are based on random sampling and we will prove that our algorithms produce universal coresets with high probability (w.h.p.). At a first glance, it is not easy to see how to sample points in the overlapping group case, as the decision has an effect on multiple groups. For simplicity, first we discuss the disjoint group case.

The Disjoint Group Case

Our coreset construction algorithm is built upon the coreset construction algorithm for vanilla clustering due to Chen [14]. In our case, we have points from ℓ disjoint color classes. So, we apply Chen's algorithm for each color class independently. Note that Chen's algorithm

was used to show that for any given set of centers C, the constructed coreset approximately preserves the optimal clustering cost. However, we would like to show that for any given set of centers C, the constructed coreset approximately preserves the optimal clustering cost corresponding to any given constraint M. At this stage, it is not clear why Chen's algorithm should work in such a generic setting. Our main technical contribution is to show that sampling based approaches like Chen's algorithm can be used even for such a stronger notion of universal coreset. We will try to give some intuition after describing our algorithm. Our algorithm is as follows.

Given the set of points P, first we apply the algorithm of Indyk [30] for computing a vanilla k-median clustering of P. This is a bicriteria approximation algorithm that uses $O(k)$ centers and runs in $O(nk)$ time. Let C^* be the set of computed centers, ν be the constant approximation factor and Π be the cost of the clustering. Also, let $\mu = \Pi/(\nu n)$ be a lower bound on the average cost of the points in any optimal k-median clustering. Note that for any point p, $d(p, C^*) \leq \Pi = \nu n \cdot \mu$.

For each center $c_i^* \in C^*$, let $P_i^* \subseteq P$ be the corresponding cluster of points assigned to c_i^*. We consider the ball $B_{i,j}$ centered at c_i^* and having radius $2^j\mu$ for $0 \leq j \leq N$, where $N = \lceil \log(\nu n) \rceil$. We note that any point at a distance $2^N\mu \geq \nu n \cdot \mu$ from c_i^* is in $B_{i,N}$, and thus all the points in P_i^* are also in $B_{i,N}$. Let $B'_{i,0} = B_{i,0}$ and $B'_{i,j} = B_{i,j} \setminus B_{i,j-1}$ for $1 \leq j \leq N$. We refer to each such $B'_{i,j}$ as a ring for $1 \leq i \leq k, 0 \leq j \leq N$. For each $0 \leq j \leq N$ and color $1 \leq t \leq \ell$, let $P'_{i,j,t}$ be the set of points in $B'_{i,j}$ of color t, and $P_{i,j} = \cup_{t=1}^{\ell} P'_{i,j,t}$. Let $s = \Theta(k \log n/\epsilon^3)$ for a sufficiently large constant hidden in $\Theta(.)$.

For each center $c_i^* \in C^*$, we perform the following steps.

Random Sampling. For each color $1 \leq t \leq \ell$ and ring index $0 \leq j \leq N$, do the following. If $|P'_{i,j,t}| \leq s$, add all the points of $P'_{i,j,t}$ to $W_{i,j}$ and set the weight of each such point to 1. Otherwise, select s points from $P'_{i,j,t}$ independently and randomly (without replacement) and add them to $W_{i,j}$. Set the weight of each such point to $|P'_{i,j,t}|/s$.

The set $W = \cup_{i,j} W_{i,j}$ is the desired universal coreset. As the number of rings is $O(k \log n)$, the size of W is $O(\ell(k \log n)^2/\epsilon^3)$. From [14], it follows that for each color, the coreset points can be computed in time linear in the number of points of that color times $O(k)$. Thus, our coreset construction algorithm runs in $O(nk)$ time. Next, we show that W is indeed a universal coreset w.h.p.

Note that we need to show that for any set of centers C, the optimal clustering cost is approximately preserved w.r.t. all possible combinations of cluster sizes as defined by the constraint matrices. In Chen's analysis, it was sufficient to argue that for any set of centers C, the optimal clustering cost needs to be preserved. This seems much easier compared to our case. (Obviously, the details are much more complicated even in the vanilla case.) For example, in the vanilla case, let $p \in P$ be a point that is assigned to a center $c \in C$ in an optimal clustering. Note that c must be a closest center to p. For simplicity, suppose p has a unique closest center. Now, if p is chosen in the coreset, then the total weight of p must also be assigned to c in any optimal assignment w.r.t. C. Thus, the assignment function for original and coreset points remains same in the vanilla case. This fact is in the heart of their analysis. Note that this is not necessarily true in our case. We cannot just use the nearest neighbor assignment scheme, as in our case cluster sizes are predefined through M. Indeed, in our case we might very well need to assign the weight of a coreset point to multiple centers to satisfy M. In general, this is the main hurdle one faces while analyzing a sampling based approach for fair coreset construction.

For analyzing our algorithm, we follow an approach similar to the one by Cohen-Addad and Li in [19]. They considered the capacitated clustering problem, where for each center c a capacity value U_c is given, and if the center c is chosen, at most U_c points can be assigned to c. They analyzed Chen's algorithm and showed that for any center C, the coreset approximately preserves the optimal capacitated clustering cost. One crucial idea they use in their proof is representation of assignments through network flow. Suppose we are given a fixed set of centers and weighted input points, and we would like to compute a minimum cost assignment of the points to the centers such that the capacities are not violated. This problem can be modeled as a minimum cost network flow problem.

The first hurdle to adapt the approach in [19] is that it is not possible to represent the assignment problem for fair clustering as a simple flow computation problem. Thus it is not clear how to directly use their approach for fair clustering. Nonetheless, we show that for a fixed constraint M, the assignment problem can be modeled in the desired way. Thus, we can get high probability bound w.r.t. a fixed constraint M. However, to obtain a coreset for fair clustering we need to show this w.r.t. all such constraints (and this leads us towards a universal coreset). The number of such constraints can be as large as $n^{\Omega(k\ell)}$. Hence, to obtain the h.p. bound over all M, we need to show that for a fixed M the error probability is at most $1/n^{\Omega(k\ell)}$. However, it is not clear how to show such a strong bound ($1/n^{\Omega(k)}$ bound can be shown). Nevertheless, we show that it is not necessary to consider all those choices of the constraints together – one can focus on a single color and the constraints w.r.t. that color only. Indeed, this is the reason that we apply Chen's algorithm to different color classes independently. Unfortunately, we pay a heavy toll for this: the coreset size is proportional to ℓ, unlike the vanilla coreset size, and it is not clear how to avoid this dependency. Anyway, this solves our problem, as now we have only $n^{\Omega(k)}$ constraints. It follows that, for a fixed color and a fixed constraint matrix, one can apply an approach similar to the one in [19] (the details are slightly different). This allows us to adapt the ideas from [19] and [14] to construct coresets for much more general clustering problems. Next, we describe the details. We will prove the following lemma.

▶ **Lemma 5.** *For any fixed set C of k centers and for all $k \times \ell$ matrices M, w.p. at least $1 - 1/n^{k+2}$, $|\mathrm{cost}(P, M, C) - \mathrm{wcost}(W, M, C)| \leq \sum_{(i,j)} \epsilon |P_{i,j}| \cdot 2^j \mu$.*

Now, consider all the rings $B'_{i,j}$ with $j = 0$. Then,

$$\sum_{(i,j):j=0} \epsilon |P_{i,j}| \cdot 2^j \mu \leq \epsilon n \cdot \mu \leq \epsilon \cdot \mathrm{OPT}_v \leq \epsilon \cdot \mathrm{cost}(P, M, C).$$

Here, OPT_v is the optimal cost of vanilla k-median clustering. The last inequality follows, as the optimal cost of vanilla clustering is at most the cost of any constrained clustering. On the other hand, for any ring $B'_{i,j}$ with $j \geq 1$ and any point p in the ring, $d(p, c^*_i) \geq 2^{j-1}\mu$. Thus,

$$\sum_{(i,j):j\geq 1} \epsilon |P_{i,j}| \cdot 2^j \mu \leq \epsilon \sum_{p \in P} 2 \cdot d(p, c^*_{i_p}) \leq 2\epsilon \cdot \mathrm{OPT}_v \leq 2\epsilon \cdot \mathrm{cost}(P, M, C),$$

where for a point $p \in P$ by i_p we denote the index of a center such that p belongs to $B'_{i_p,j}$ for some j.

Taking union bound over all C and scaling down ϵ by 3 factor, we get the desired result.

▶ **Lemma 6.** *For every set C of k centers and every $k \times \ell$ matrices M, w.p. at least $1 - 1/n$, $|\mathrm{cost}(P, M, C) - \mathrm{wcost}(W, M, C)| \leq \epsilon \cdot \mathrm{cost}(P, M, C)$.*

3.2 Proof of Lemma 5

Let P_τ be the points in P of color τ. Also, let W_τ be the chosen samples of color τ. For $1 \leq t \leq \ell - 1$, let $W^t = (\cup_{\tau=1}^{t} W_\tau) \cup (\cup_{\tau=t+1}^{\ell} P_\tau)$. Also, let $W^\ell = \cup_{\tau=1}^{\ell} W_\tau$ be the coreset points of all colors. Recall that for any ring $B'_{i,j}$, $P'_{i,j,\tau}$ is the points of color τ in the ring. Also, $P_{i,j} = \cup_{\tau=1}^{\ell} P'_{i,j,\tau}$.

Note that in the above, W^t contains the sampled points for color 1 to t and original points of color $t+1$ to ℓ. We will prove the following lemma that gives a bound when the coreset contains sampled points of a fixed color t and original points of the other colors.

▶ **Lemma 7.** *Consider any color $1 \leq t \leq \ell$. For any fixed set C of k centers and for all $k \times \ell$ matrices M, w.p. at least $1 - 1/n^{k+4}$, $|\text{cost}(P, M, C) - \text{wcost}(W_t \cup (P \setminus P_t), M, C)| \leq \sum_{(i,j)} \epsilon |P'_{i,j,t}| \cdot 2^j \mu$.*

Note that for a particular color class, if we select all original points in the coreset, there is no error corresponding to those coreset points. This is true, as one can use the corresponding optimal assignment for these points. Assuming that the above lemma holds, now we prove Lemma 5. Consider the coreset W^1. From the above lemma, we readily obtain the following.

▶ **Corollary 8.** *For any fixed set C of k centers and for all $k \times \ell$ matrices M, w.p. at least $1 - 1/n^{k+4}$, $|\text{cost}(P, M, C) - \text{wcost}(W^1, M, C)| \leq \sum_{(i,j)} \epsilon |P'_{i,j,1}| \cdot 2^j \mu$.*

Now, in W^1 consider replacing the points of P_2 by the samples in W_2. We obtain the coreset W^2. Note that the samples in W_1 and W_2 are chosen independent of each other. Thus, by taking union bound over colors 1 and 2, from Lemma 7 we obtain, for all M, w.p. $\geq 1 - 2/n^{k+4}$, $|\text{cost}(P, M, C) - \text{wcost}(W^2, M, C)| \leq \sum_{(i,j)} \epsilon (|P'_{i,j,1}| + |P'_{i,j,2}|) \cdot 2^j \mu$. Similarly, by taking union bound over all $\ell \leq n$ colors and noting that $W^\ell = W$, Lemma 5 follows.

3.3 Proof of Lemma 7

Recall that P_t is the set of points of color t, and W_t is the coreset points of color t. C is the given set of centers. For any matrix M, let M^t be the t^{th} column of M. We have the following observation that implies that it is sufficient to consider the points only in P_t to give the error bound.

▶ **Observation 9.** *Suppose w.p. at least $1 - 1/n^{k+4}$, for all column matrices M', $|\text{cost}(P_t, M', C) - \text{wcost}(W_t, M', C)| \leq \sum_{(i,j)} \epsilon |P'_{i,j,t}| \cdot 2^j \mu$. Then, with the same probability, for all $k \times \ell$ matrices M, $|\text{cost}(P, M, C) - \text{wcost}(W_t \cup (P \setminus P_t), M, C)| \leq \sum_{(i,j)} \epsilon |P'_{i,j,t}| \cdot 2^j \mu$.*

Proof. Consider any $k \times \ell$ matrix M. Then,

$$\text{cost}(P, M, C) = \sum_{\tau=1}^{\ell} \text{cost}(P_\tau, M^\tau, C), \text{ and}$$

$$\text{wcost}(W_t \cup (P \setminus P_t), M, C) = \text{wcost}(W_t, M^t, C) + \sum_{\tau \in [\ell] \setminus \{t\}} \text{cost}(P_\tau, M^\tau, C)$$

It follows that,

$$|\text{cost}(P, M, C) - \text{wcost}(W_t \cup (P \setminus P_t), M, C)| = |\text{cost}(P_t, M^t, C) - \text{wcost}(W_t, M^t, C)|$$

Now, by our assumption, it follows that the probability of the event: for all M, $|\text{cost}(P_t, M^t, C) - \text{wcost}(W_t, M^t, C)|$ exceeds $\sum_{(i,j)} \epsilon |P'_{i,j,t}| \cdot 2^j \mu$ is at most $1/n^{k+4}$. Hence, the observation follows. ◀

By the above observation, it is sufficient to prove that w.p. at least $1 - 1/n^{k+4}$, for all column matrices M, $|\text{cost}(P_t, M, C) - \text{wcost}(W_t, M, C)| \leq \sum_{(i,j)} \epsilon |P'_{i,j,t}| \cdot 2^j \mu$. The proof of this claim is similar to the analysis in [19] and appears in Section 4 of the full version.

The Overlapping Group Case

We are given ℓ groups of points $P_1, \ldots, P_\ell$ such that a point can potentially belong to multiple groups. Note that the algorithm in the disjoint case does not work here. This is because we sample points from each group separately and independently, and thus it is not clear how to assign the weight of a point that belongs to multiple groups. One might think of the following trivial modification of this algorithm. Assign each point to a single group to which it belongs. Based on this assignment, now we have disjoint groups, and we can apply our previous algorithm. But, the new algorithm can have a very large error bound. For example, suppose a point p belongs to two groups i and j, and it is assigned to group i. Also, suppose p was not chosen in the sampling process. Note that the weight of p is represented by some other chosen point p', which was also assigned to group i. However, now we have lost the information that this weight of p was also contributing towards fairness of group j. Thus, the constructed coreset might not preserve any optimal fair clustering with a small error. In the overlapping case, it is not clear how to obtain a coreset whose size depends linearly in ℓ – we design a new coreset construction algorithm where the size depends linearly on Γ. Recall that Γ is the maximum number of distinct collections of groups to which a point belongs.

The main idea of our algorithm is to divide the points into equivalence classes based on their group membership and sample points from each equivalence class. Let $P = \cup_{i=1}^{\ell} P_i$. For each point $p \in P$, let $J_p \subseteq [\ell]$ be the set of indexes of the groups to which p belongs. Let I be the distinct collection of these sets $\{J_p \mid p \in P\}$ and $|I| = \Gamma$. In particular, let $I_1, \ldots, I_\Gamma$ be the distinct sets in I. Now, we partition the points in P based on these sets. For $1 \leq i \leq \Gamma$, let $P^i = \{p \in P \mid I_i = J_p\}$. Thus, $\{P^i \mid 1 \leq i \leq \Gamma\}$ defines equivalence classes for P such that two points $p, p' \in P$ belong to the same equivalence class if they are in exactly the same set of groups. Now we apply our algorithm in the disjoint case on the disjoint sets of points $P^1, \ldots, P^\Gamma$. Let W be the constructed coreset.

Note that here we have Γ disjoint classes, and thus the coreset size is $O(\Gamma(k \log n)^2/\epsilon^3)$. As our coreset size is at least Γ, we assume that $\Gamma < n$. Note that the equivalence classes can be computed in $O(n\ell)$ time, and thus the algorithm runs in time $O(n\ell) + O(nk) = O(n(k+\ell))$. The proof that W is indeed a universal coreset w.h.p. follows the same line as of the disjoint-group case. Again, the idea here is to reduce the analysis to the one class case. However, this is not as straightforward as in the disjoint case. Note that although the classes $P^1, \ldots, P^\Gamma$ are disjoint, two classes can contain points from the same group. Moreover, the constraints are defined w.r.t. the groups, not w.r.t. the classes. Thus, two classes need to interact to satisfy the constraints. Nevertheless, we use the independence of the samples from different classes and exploit the structure of a special class of matrices to complete the proof, which appears in Section 5 of the full version.

The algorithm in the Euclidean case is the same as for general metrics, except we set s to $\Theta(k \log(nb)/\epsilon^3)$, where $b = \Theta(k \log(n/\epsilon)/\epsilon^d)$. Here the main challenge is that it is not possible to take union bound over all possible sets of k centers. Nevertheless, we show that for every set $C \subseteq \mathbb{R}^d$ of k centers and constraint M, the optimal cost is preserved approximately w.h.p. Our main contribution in this part is to devise a discretization technique for obtaining a finite set of centers, so that if instead we draw centers from this set, the cost of any clustering is preserved approximately. The details appear in Section 6 of the full version.

3.4 Approximation Algorithms Based on Universal Coresets

All the algorithms that we design follow one general strategy: first, compute a universal coreset, then, enumerate a small family of sets of possible k centers, such that at least one of them is guaranteed to provide a good approximation, and finally pick the best set of centers by finding the optimal assignment from the coreset to each of the center sets. Throughout this section, we limit our discussion to fair clustering. One major difficulty in the case of fair clustering is solving the assignment problem, for which we devise a novel FPT algorithm.

Solving the Assignment Problem

The fair assignment problem is the following: given an instance of (α, β)-Fair Clustering and a set of k centers C, compute a minimum-cost fair assignment to the centers of C. For fair clustering, the assignment problem is one of the features that makes it harder than other constrained clustering problems. While often the optimal assignment can be found with the help of network flow, e.g. for capacitated clustering, there was no previously known algorithms to compute an optimal or approximate fair assignment without violating the constraints. Moreover, it was observed by Bera et al. [7] that the fair assignment problem is NP-hard, so there is no hope to have a polynomial time assignment algorithm.

We design a fair assignment algorithm with running time $(k\Gamma)^{O(k\Gamma)}n^{O(1)}$. The general idea is to reduce to a linear programming instance. The unknown optimal assignment can be naturally expressed in terms of linear inequalities by introducing a variable f_{ij} for the i-th point and the j-th center, denoting which fraction of the point is assigned to each center, and constraints $f_{ij} \geq 0$ for all i, j, and $\sum_{j=1}^{k} f_{ij} = 1$. Clearly this generalizes a discrete assignment, which corresponds to exactly one of $\{f_{ij}\}_{j=1}^{k}$ being equal to 1, for each $i \in [n]$. Moreover, the fairness of the assignment can also be expressed as linear constraints on $\{f_{ij}\}$.

The main obstacle is that in general the optimal solution to this linear program is not integral, and the integrality gap could be arbitrarily large. Thus, an optimal fractional solution does not yield the desired assignment, and this is not surprising since the fair assignment problem is NP-hard. One possible solution could be restricting the variables to be integral, solving an integer linear program (ILP) instead. But the number of variables is too large for an FPT algorithm. Instead, we introduce the integral variables $\{g_{tj}\}$ denoting how many points from the t-th point equivalence class gets to the j-th center, while leaving the $\{f_{ij}\}$ variables to be fractional. Thus, we obtain an instance of mixed-integer linear programming (MILP) with $k\Gamma$ integer variables and nk fractional variables. By using the celebrated result of Lenstra [36] with subsequent improvements by Kannan [31], and Frank and Tardos [24], we obtain an optimal solution to the MILP instance in time $(k\Gamma)^{O(k\Gamma)}n^{O(1)}$.

Now we explain that after constraining the $\{g_{tj}\}$ variables to be integral, we can assume that all the other variables $\{f_{ij}\}$ are integral too, thus we actually obtain an optimal discrete assignment of the same cost. Consider a particular point equivalence class P^t, and the integral values $\{g_{tj}\}_{j=1}^{k}$ from the optimal solution to the MILP. When these values are fixed, the problem boils down to finding an assignment from P^t to C such that exactly g_{tj} points are assigned to the j-th center. This problem can be solved by a minimum-cost maximum flow in the network where each point has supply one, the j-th center has demand of g_{tj}, and the costs are the distances between the respective points. Moreover, the values $\{f_{ij}\}$ from the MILP correspond exactly to the flow values on the respective edges. Since there is an optimal integral flow in this network, this flow is also an optimal integral solution for $\{f_{ij}\}$.

The downside of the above algorithm is that the time complexity is roughly n^5, and we cannot use it directly to obtain a near-linear time algorithm. So, we also show how to obtain a $(1+\epsilon)$-approximate fair assignment in near-linear time with the help of the coreset. For

this, we compute a universal coreset, and then compute the optimal fair assignment from the coreset to the centers C. However, this does not yet give us a fair assignment of the original points to the centers. To construct this assignment, we take the values $\{g_{tj}\}$ computed by the assignment algorithm on the coreset, and then, for each point equivalence class P^t, we solve the simple assignment problem from P^t to C that assigns exactly g_{tj} points to the j-th center. As mentioned above, this can be done by a network flow algorithm. Since the network is bipartite and one of the parts is small (k), this problem can be solved in near-linear time by the specialized flow algorithm in [3]. By the property of the universal coreset, the resulting assignment achieves a $(1+\epsilon)$-approximation (see Section 8 of the full version).

$(1+\epsilon)$-Approximation in $\mathbb{R}^d$. Besides our coreset construction and assignment algorithm, the key ingredient to obtain a $(1+\epsilon)$-approximation is the generic clustering algorithm of Bhattacharya et al. [10]. Their algorithm outputs a list of $2^{\tilde{O}(k/\epsilon^{O(1)})}$ candidate sets of k centers, such that for any clustering of the points there exists a set of centers C in this list that is slightly worse than the optimal set of centers for this clustering. Together with our assignment algorithm this provides a $(1+\epsilon)$-approximation algorithm with the running time of $2^{\tilde{O}(k/\epsilon^{O(1)})}(k\Gamma)^{O(k\Gamma)}nd\log n$. We describe this algorithm in Section 9 of the full version.

$(3+\epsilon)$-Approximation in General Metric. With the help of our universal coreset, the strategy to obtain $(3+\epsilon)$-approximation for (α,β)-Fair k-median is essentially same as that used in [18, 19]: from each of the clusters in an optimal solution on the coreset we guess the closest point to the center, called a *leader* of that cluster. We also guess a suitably discretized distance from each leader to the center of the corresponding cluster. Finally, selecting any center that has roughly the guessed distance to the leader provides us with a $(3+\epsilon)$-approximation. In this way we obtain a list of $|W|^k(\log n/\epsilon)^{O(k)}$ candidate sets of k centers. Afterwards, our algorithm proceeds similarly to the Euclidean case above, resulting in the running time of $(k\Gamma)^{O(k\Gamma)}/\epsilon^{O(k)}\cdot n\log n$ (see Section 10 of the full version).

References

1. Sara Ahmadian, Alessandro Epasto, Ravi Kumar, and Mohammad Mahdian. Clustering without over-representation. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 267–275, 2019.
2. Sara Ahmadian and Chaitanya Swamy. Improved approximation guarantees for lower-bounded facility location. In *International Workshop on Approximation and Online Algorithms*, pages 257–271. Springer, 2012.
3. Ravindra K. Ahuja, James B. Orlin, Clifford Stein, and Robert E. Tarjan. Improved algorithms for bipartite network flow, 1994.
4. Arturs Backurs, Piotr Indyk, Krzysztof Onak, Baruch Schieber, Ali Vakilian, and Tal Wagner. Scalable fair clustering. In *International Conference on Machine Learning*, pages 405–413, 2019.
5. Daniel Baker, Vladimir Braverman, Lingxiao Huang, Shaofeng H-C Jiang, Robert Krauthgamer, and Xuan Wu. Coresets for clustering in graphs of bounded treewidth. *arXiv preprint*, 2019. `arXiv:1907.04733`.
6. Luca Becchetti, Marc Bury, Vincent Cohen-Addad, Fabrizio Grandoni, and Chris Schwiegelshohn. Oblivious dimension reduction for k-means: beyond subspaces and the johnson-lindenstrauss lemma. In *Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing*, pages 1039–1050, 2019.
7. Suman Bera, Deeparnab Chakrabarty, Nicolas Flores, and Maryam Negahbani. Fair algorithms for clustering. In *Advances in Neural Information Processing Systems*, pages 4954–4965, 2019.

8 Suman K. Bera, Deeparnab Chakrabarty, and Maryam Negahbani. Fair algorithms for clustering. *CoRR*, abs/1901.02393, 2019. arXiv:1901.02393.

9 Ioana O Bercea, Martin Groß, Samir Khuller, Aounon Kumar, Clemens Rösner, Daniel R Schmidt, and Melanie Schmidt. On the cost of essentially fair clusterings. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques (APPROX/RANDOM 2019)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2019.

10 Anup Bhattacharya, Ragesh Jaiswal, and Amit Kumar. Faster Algorithms for the Constrained k-means Problem. *Theory of Computing Systems*, 62(1):93–115, 2018. doi:10.1007/s00224-017-9820-7.

11 Matteo Böhm, Adriano Fazzone, Stefano Leonardi, and Chris Schwiegelshohn. Fair clustering with multiple colors. *arXiv preprint*, 2020. arXiv:2002.07892.

12 Vladimir Braverman, Dan Feldman, and Harry Lang. New frameworks for offline and streaming coreset constructions. *arXiv preprint*, 2016. arXiv:1612.00889.

13 Vladimir Braverman, Shaofeng H-C Jiang, Robert Krauthgamer, and Xuan Wu. Coresets for clustering in excluded-minor graphs and beyond. In *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 2679–2696. SIAM, 2021.

14 Ke Chen. On coresets for k-median and k-means clustering in metric and euclidean spaces and their applications. *SIAM Journal on Computing*, 39(3):923–947, 2009.

15 Xingyu Chen, Brandon Fain, Liang Lyu, and Kamesh Munagala. Proportionally fair clustering. In *International Conference on Machine Learning*, pages 1032–1041, 2019.

16 Flavio Chierichetti, Ravi Kumar, Silvio Lattanzi, and Sergei Vassilvitskii. Fair clustering through fairlets. In *Advances in Neural Information Processing Systems*, pages 5029–5037, 2017.

17 Vincent Cohen-Addad. Approximation schemes for capacitated clustering in doubling metrics. In Shuchi Chawla, editor, *Proceedings of the 2020 ACM-SIAM Symposium on Discrete Algorithms, SODA 2020, Salt Lake City, UT, USA, January 5-8, 2020*, pages 2241–2259. SIAM, 2020.

18 Vincent Cohen-Addad, Anupam Gupta, Amit Kumar, Euiwoong Lee, and Jason Li. Tight FPT approximations for k-median and k-means. In Christel Baier, Ioannis Chatzigiannakis, Paola Flocchini, and Stefano Leonardi, editors, *46th International Colloquium on Automata, Languages, and Programming, ICALP 2019, July 9-12, 2019, Patras, Greece*, volume 132 of *LIPIcs*, pages 42:1–42:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019.

19 Vincent Cohen-Addad and Jason Li. On the fixed-parameter tractability of capacitated clustering. In Christel Baier, Ioannis Chatzigiannakis, Paola Flocchini, and Stefano Leonardi, editors, *46th International Colloquium on Automata, Languages, and Programming, ICALP 2019, July 9-12, 2019, Patras, Greece*, volume 132 of *LIPIcs*, pages 41:1–41:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019.

20 Hu Ding and Jinhui Xu. A unified framework for clustering constrained data without locality property. *Algorithmica*, 82(4):808–852, 2020.

21 Dan Feldman and Michael Langberg. A unified framework for approximating and clustering data. In *Proceedings of the forty-third annual ACM symposium on Theory of computing*, pages 569–578, 2011.

22 Dan Feldman, Melanie Schmidt, and Christian Sohler. Turning big data into tiny data: Constant-size coresets for k-means, pca, and projective clustering. *SIAM Journal on Computing*, 49(3):601–657, 2020.

23 Gereon Frahling and Christian Sohler. Coresets in dynamic geometric data streams. In *Proceedings of the thirty-seventh annual ACM symposium on Theory of computing*, pages 209–217, 2005.

24 András Frank and Éva Tardos. An application of simultaneous diophantine approximation in combinatorial optimization. *Combinatorica*, 7(1):49–65, 1987. doi:10.1007/bf02579200.

25 Sariel Har-Peled and Akash Kushal. Smaller coresets for k-median and k-means clustering. *Discrete & Computational Geometry*, 37(1):3–19, 2007.

26 Sariel Har-Peled and Akash Kushal. Smaller coresets for k-median and k-means clustering. *Discret. Comput. Geom.*, 37(1):3–19, 2007.

27 Sariel Har-Peled and Soham Mazumdar. On coresets for k-means and k-median clustering. In László Babai, editor, *Proceedings of the 36th Annual ACM Symposium on Theory of Computing, Chicago, IL, USA, June 13-16, 2004*, pages 291–300. ACM, 2004.

28 Lingxiao Huang, Shaofeng Jiang, and Nisheeth Vishnoi. Coresets for clustering with fairness constraints. In *Advances in Neural Information Processing Systems*, pages 7589–7600, 2019.

29 Lingxiao Huang and Nisheeth K Vishnoi. Coresets for clustering in euclidean spaces: Importance sampling is nearly optimal. *arXiv preprint*, 2020. `arXiv:2004.06263`.

30 Piotr Indyk. Sublinear time algorithms for metric space problems. In *Proceedings of the thirty-first annual ACM symposium on Theory of computing*, pages 428–434, 1999.

31 Ravi Kannan. Minkowski's convex body theorem and integer programming. *Mathematics of Operations Research*, 12(3):415–440, 1987. `doi:10.1287/moor.12.3.415`.

32 Matthäus Kleindessner, Pranjal Awasthi, and Jamie Morgenstern. Fair k-center clustering for data summarization. In *36th International Conference on Machine Learning, ICML 2019*, pages 5984–6003. International Machine Learning Society (IMLS), 2019.

33 Matthäus Kleindessner, Samira Samadi, Pranjal Awasthi, and Jamie Morgenstern. Guarantees for spectral clustering with fairness constraints. *arXiv preprint*, 2019. `arXiv:1901.08668`.

34 Amit Kumar, Yogish Sabharwal, and Sandeep Sen. Linear-time approximation schemes for clustering problems in any dimensions. *J. ACM*, 57(2):5:1–5:32, 2010.

35 Michael Langberg and Leonard J Schulman. Universal ε-approximators for integrals. In *Proceedings of the twenty-first annual ACM-SIAM symposium on Discrete Algorithms*, pages 598–607. SIAM, 2010.

36 H. W. Lenstra. Integer programming with a fixed number of variables. *Mathematics of Operations Research*, 8(4):538–548, 1983. `doi:10.1287/moor.8.4.538`.

37 Clemens Rösner and Melanie Schmidt. Privacy preserving clustering with constraints. In *45th International Colloquium on Automata, Languages, and Programming (ICALP 2018)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2018.

38 Melanie Schmidt, Chris Schwiegelshohn, and Christian Sohler. Fair coresets and streaming algorithms for fair k-means. In *International Workshop on Approximation and Online Algorithms*, pages 232–251. Springer, 2019.

39 Christian Sohler and David P Woodruff. Strong coresets for k-median and subspace approximation: Goodbye dimension. In *2018 IEEE 59th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 802–813. IEEE, 2018.

40 Zoya Svitkina. Lower-bounded facility location. *ACM Transactions on Algorithms (TALG)*, 6(4):1–16, 2010.

Strong Approximate Consensus Halving and the Borsuk-Ulam Theorem

Eleni Batziou
Technical University of Munich, Germany

Kristoffer Arnsfelt Hansen
Aarhus University, Denmark

Kasper Høgh
Aarhus University, Denmark

Abstract

In the consensus halving problem we are given n agents with valuations over the interval $[0, 1]$. The goal is to divide the interval into at most $n + 1$ pieces (by placing at most n cuts), which may be combined to give a partition of $[0, 1]$ into two sets valued equally by all agents. The existence of a solution may be established by the Borsuk-Ulam theorem. We consider the task of computing an approximation of an exact solution of the consensus halving problem, where the valuations are given by distribution functions computed by algebraic circuits. Here approximation refers to computing a point that is ε-close to an exact solution, also called *strong* approximation. We show that this task is polynomial time equivalent to computing an approximation to an exact solution of the Borsuk-Ulam search problem defined by a continuous function that is computed by an algebraic circuit.

The Borsuk-Ulam search problem is the defining problem of the complexity class BU. We introduce a new complexity class BBU to also capture an alternative formulation of the Borsuk-Ulam theorem from a computational point of view. We investigate their relationship and prove several structural results for these classes as well as for the complexity class FIXP.

2012 ACM Subject Classification Theory of computation → Complexity classes; Theory of computation → Problems, reductions and completeness

Keywords and phrases Consensus halving, Computational Complexity, Borsuk-Ulam

Digital Object Identifier 10.4230/LIPIcs.ICALP.2021.24

Category Track A: Algorithms, Complexity and Games

Related Version *Full Version*: `https://arxiv.org/abs/2103.04452`

Funding *Kristoffer Arnsfelt Hansen and Kasper Høgh*: Supported by the Independent Research Fund Denmark under grant no. 9040-00433B.
Eleni Batziou: Supported by the German Research Foundation (DFG) within the Research Training Group AdONE (GRK 2201).

1 Introduction

Many computational problems, e.g. linear and semidefinite programming, are most naturally expressed using real numbers. When the model of computation is discrete, these problems must be recast as discrete problems. In the case of linear programming this causes no problems. Namely, when the input is given as rational numbers and an optimal solution exists, a rational valued optimal solution exists and may be computed in polynomial time. For semidefinite programming however, it may be the case that all optimal solutions are irrational. For dealing with such cases we may instead consider the *weak optimization* problem as defined by Grötschel, Lovász and Schrijver [18]: Given $\varepsilon > 0$, the task is to compute a rational-valued vector x that is ε-close to the set of feasible solutions and has objective value ε-close to optimal. Assuming we are also given, as an additional input, a strictly feasible

solution and a bound on the magnitude of the coordinates of an optimal solution, the weak optimization problem may be solved in polynomial time using the ellipsoid algorithm [18]. Let us note however that without additional assumptions, even the complexity of the basic *existence problem* of semidefinite feasibility is unknown. In fact, the problem is likely to be computationally very hard [23]. More precisely, it is hard for the problem PosSLP, which is the fundamental problem of deciding whether an integer given by a division free arithmetic circuit is positive [2].

In this paper we consider real valued search problems, where existence of a solution is guaranteed by topological existence theorems such as the Brouwer fixed point theorem and the Borsuk-Ulam theorem. This means that the search problems are *total*, thereby fundamentally differentiating them from general search problems where, as described above, even the existence problem may be computationally hard. We are mainly interested in the *approximation* problem: given $\varepsilon > 0$, the task is to compute a rational-valued vector x that is ε-close to the set of solutions.

Recall that the Brouwer fixed point theorem states that every continuous function $f : B^n \to B^n$, where B^n is the unit n-ball, has a fixed point, i.e. there is $x \in B^n$ such that $f(x) = x$ [6]. The Borsuk-Ulam theorem states that every continuous function $f \colon S^n \to \mathbb{R}^n$, where S^n is the unit n-sphere in $\mathbb{R}^{n+1}$, maps a pair of antipodal points of S^n to the same point in $\mathbb{R}^n$, i.e. there is $x \in S^n$ such that $f(x) = f(-x)$ [5]. The Brouwer fixed point theorem is of course not restricted to apply to the domain B^n, but applies to any domain that is homeomorphic to B^n. Similarly the Borsuk-Ulam theorem applies to any domain homeomorphic to S^n by an antipode-preserving homeomorphism. It is well-known that the Borsuk-Ulam theorem generalizes the Brouwer fixed point theorem, in the sense that the Brouwer fixed point theorem is easy to prove using the Borsuk-Ulam theorem [22, 24].

The Brouwer fixed point theorem and the Borsuk-Ulam theorem naturally define corresponding real valued search problems, and thereby also corresponding approximation problems. In addition, the statements of the theorems naturally lead to another notion of approximation. For the case of the Brouwer fixed point theorem we may look for an *almost* fixed point, i.e. $x \in B^n$ such that $f(x)$ is ε-close to x, and for the case of the Borsuk-Ulam theorem we look for a pair of antipodal points that *almost* map to the same point, i.e. $x \in S^n$ such that $f(x)$ and $f(-x)$ are ε-close. Following [12], we shall refer to this notion of approximation as *weak* approximation and to make the distinction clear we refer to the former (and general) notion of approximation as *strong* approximation. In the setting of weak approximation in relation to the Borsuk-Ulam theorem we assume that f has domain B^n.

In their seminal work, Etessami and Yannakakis [12] introduced the complexity class FIXP to capture the computational complexity of the real-valued search problems associated with the Brouwer fixed point theorem, and proved that the problem of finding a Nash equilibrium in a given 3-player game in strategic form is FIXP-complete. In order to have a notion of completeness, the class FIXP is defined to be closed under reductions. The type of reductions chosen by Etessami and Yannakakis, SL-reductions, consists of mapping between sets of solutions by a composition of a *projection* reduction followed by individual affine transformations applied to each coordinate.

Etessami and Yannakakis considered different ways to cast real valued search problems as discrete search problems. In addition to the approximation problem, these are the *partial computation* problem where the task is to compute a solution to a given number of bits of precision and *decision* problems, where the task is to evaluate a sign condition of the set of solutions given the promise that either all solutions satisfy the condition or none of them do. Of these we shall only consider the approximation problem. The class FIXP_a denotes the class of discrete search problems corresponding to strong approximation of Brouwer

fixed points and is defined to be closed under polynomial time reductions. Etessami and Yannakakis also prove that the problem PosSLP reduces to the problem of approximating a Nash equilibrium, thereby showing that FIXP_a likely contains search problems that are computationally very hard.

While the notion of SL-reductions is very restricted, it is sufficient for proving completeness of the problem of finding a Nash equilibrium. Likewise, SL-reductions are sufficient for showing that FIXP is robust with respect to the choice of domain for the Brouwer function.

Another important reason for using SL-reductions is that they immediately imply polynomial time reductions between the corresponding decision and approximation problems (the partial computation problem is more fragile and requires additional assumptions, cf. [12]). As we are mainly interested in the approximation problem more expressive notions of reducibility can be considered, while maintaining the property that reducibility implies polynomial time reducibility between the corresponding approximation problems. A sufficient condition for this is that the mapping of solutions is *polynomially continuous* and polynomial time computable.

1.1 The Borsuk-Ulam Theorem

Deligkas, Fearnley, Melissourgos, and Spirakis [11] recently introduced the complexity class BU to capture, in an analogy to FIXP, the computational complexity of the real-valued search problems associated with the Borsuk-Ulam theorem.

The Borsuk-Ulam theorem has a number of equivalent statements that are also easy to derive from each other. A function f defined on the unit sphere S^n is *odd* if $f(x) = -f(-x)$ for all $x \in S^n$. Note that the boundary ∂B^n of the unit n-ball B^n is identical to S^{n-1}. We thus say that a function f defined on B^n is odd on ∂B^n if f is odd when restricted to S^{n-1}. We present the simple proof of the known fact that the different formulations can be derived from each other, for the purpose of discussing equivalence from a computational point of view.

▶ **Theorem 1** (Borsuk-Ulam). *The following statements hold:*
(1) *If $f\colon S^n \to \mathbb{R}^n$ is continuous there exists $x \in S^n$ such that $f(x) = f(-x)$.*
(2) *If $g\colon S^n \to \mathbb{R}^n$ is continuous and odd there exists $x \in S^n$ such that $g(x) = 0$.*
(3) *If $h\colon B^n \to \mathbb{R}^n$ is continuous and odd on ∂B^n there exists $x \in B^n$ such that $h(x) = 0$.*

Proof of equivalence. Given f we may define $g(x) = f(x) - f(-x)$. Clearly g is odd and we have $g(x) = 0$ if and only if $f(x) = f(-x)$, which shows that (2) implies (1). Conversely, given g we simply let $f = g$. If $f(x) = f(-x)$, then since g is odd we have $f(x) = g(x) = -g(-x) = -f(-x) = -f(x)$ and hence $g(x) = f(x) = 0$, which therefore shows that (1) implies (2).

We may view S^n as two hemispheres, each homeomorphic to B^n, which are glued together along their equators. Let $\pi\colon S^n \to B^n$ be the orthogonal projection defined by $\pi(x_1, \ldots, x_{n+1}) = (x_1, \ldots, x_n)$. Then given h we may define $g(x) = h(\pi(x))$ for $x_{n+1} \geq 0$ and $g(x) = -h(-\pi(x))$ for $x_{n+1} \leq 0$. The assumption that h is odd on ∂B^n makes g a well-defined continuous odd function. We have $g(x) = 0$ if and only if $h(x) = 0$, which shows that (2) implies (3). Conversely, given g we define h by $h(x) = g\left(x, (1 - \|x\|_2^2)^{\frac{1}{2}}\right)$. Then h is continuous and odd on ∂B^n, since $x \in \partial B^n$ if and only if $\|x\|_2^2 = 1$. Clearly if $h(x) = 0$ we may let $y = (x, (1 - \|x\|_2^2)^{\frac{1}{2}})$ and have $g(y) = 0$. On the other hand, when $g(y) = 0$ we may define $x = (y_1, \ldots, y_n)$ if $y_{n+1} \geq 0$ and $x = (-y_1, \ldots, -y_n)$ if $y_{n+1} < 0$, and we have $h(x) = 0$. Together this shows that (3) implies (2). ◀

The class BU defined in [11] corresponds to the first formulation of the above theorem. We may clearly consider the second formulation equivalent to the first also from a computational point of view. In particular, when translating between formulations, the set of solutions is unchanged. Note that this set of solutions has the property that all solutions come in pairs: when x is a solution then $-x$ is a solution as well. For the third formulation of the theorem this property only holds for solutions on the boundary ∂B^n.

In contrast, while the mapping of solutions of the third formulation to the second (and first) formulation given above is continuous this is not the case in the other direction. More precisely, consider $y \in S^n$ such that $g(y) = 0$. For a solution strictly contained in the upper hemisphere, the orthogonal projection to the first n coordinates produces $x \in B^n$ such that $h(x) = 0$. For a solution y strictly contained in the lower hemisphere, the projection is instead applied to the antipodal solution $-y$.

To clarify this issue from a computational point of view we introduce a new class BBU of real valued search problems corresponding to the third formulation of Theorem 1, and it will follow from definitions that BU $\subseteq$ BBU. In the context of strong approximation however, the corresponding classes of discrete search problems BU_a and BBU_a will be shown to coincide. The idea is that given an approximation to $y \in S^n$, where $g(y) = 0$, that is sufficiently close to the equator of S^n, there is no harm in *incorrectly* deciding to which hemisphere y belongs, since solutions $x \in \partial B^n$ for which $h(x) = 0$ also come in pairs.

For the class BU, the notion of SL-reductions is clearly too restrictive to allow a reasonable comparison to FIXP. Closing the class BU by SL-reductions, the solutions would still come in pairs, thereby imposing strong conditions on the set of solutions. On the other hand the reductions should also not be *too* strong. In particular it would be desirable that FIXP would still be closed under the chosen notion of reductions. This issue is not discussed in [11]. We shall therefore propose a suitable notion of reductions for both BU and BBU.

1.2 Consensus Halving

The Consensus halving problem is a classical problem of *fair division* [21]. We are given a set of n bounded and continuous measures $\mu_1, \ldots, \mu_n$ defined on the interval $A = [0, 1]$. The goal is to partition the interval A into at most $n + 1$ intervals, i.e. by placing at most n *cuts*, such that unions of these intervals form another partition $A = A^+ \cup A^-$ of A satisfying $\mu_i(A^+) = \mu_i(A^-)$ for every i. We may think of the intervals being assigned a *label* from the set $\{+, -\}$, and A^+ is precisely the union of the intervals labeled by $+$. Such a partition is also known as a consensus halving. Using the Borsuk-Ulam theorem, Simmons and Su [21] proved that a consensus halving using at most n cuts always exists. Simmons and Su represent a division of A as a point x on the unit n-sphere S_1^n with respect to the ℓ_1-norm. The point x is viewed as representing a division into *precisely* $n + 1$ intervals, where some intervals are possibly empty. More precisely, the i-th interval has length $|x_i|$, and intervals of length 0 may simply be discarded. The intervals of positive length are then labeled according to $\operatorname{sgn}(x_i)$. Note that for any x, the antipode $-x$ represents the division where the sets A^+ and A^- are exchanged. This naturally leads to a formulation using the Borsuk-Ulam theorem [21]. Namely we may consider the function $F\colon S_1^n \to \mathbb{R}^n$ given by $F(x)_i = \mu_i(A^+)$, and note that any $x \in S_1^n$ for which $F(x) = F(-x)$ represent a consensus halving.

We are interesting in the simple setting of additive measures, where we have corresponding density functions $f_1, \ldots, f_n$ such that $\mu_i(B) = \int_B f_i(x)\,dx$. To cast the consensus halving problem as a real valued search problem we follow [11] and assume that the measures $\mu_1, \ldots, \mu_n$ are given by the distribution functions $F_1, \ldots, F_n$ defined by $F_i(x) = \int_0^x f_i(t)\,dt$. An instance of the consensus halving problem is then given as a list of algebraic circuits computing these distribution functions.

1.3 Strong versus Weak Approximation

The difference between weak and strong approximation was studied in detail in the general context of the Brouwer fixed point theorem by Etessami and Yannakakis. A central example is the problem of finding a Nash equilibrium (NE). An important notion of approximation of a NE is the notion of an ε-NE. Computing an ε-NE of a given strategic form game Γ is polynomial time equivalent to computing a weak ε'-approximation to a fixed point the Nash's Brouwer function F_Γ associated to Γ [12, Proposition 2.3]. In turn, computing a weak ε'-approximation to a fixed point of F_Γ polynomial time reduces to computing a strong ε''-approximation to a fixed point of F_Γ [12, Proposition 2.2], since the function F_Γ is polynomially continuous and polynomial time computable. In general however an ε-NE might be far from any actual NE, unless ε is inverse doubly exponentially small as a function of the size of the game [12, Corollary 3.8].

For the problem of consensus halving we can illustrate the difference between weak and strong approximation by a simple example. We shall refer to a weak ε-approximation of a consensus halving as simply an ε-consensus halving. Consider a single agent whose measure μ on the interval $[0,1]$ is given by the following density

$$f(x) = \begin{cases} (1+\varepsilon)/\varepsilon & \text{if } 0 \leq x < \varepsilon/2 \\ 0 & \text{if } \varepsilon/2 \leq x < 1-\varepsilon/2 \\ (1-\varepsilon)/\varepsilon & \text{if } 1-\varepsilon/2 \leq x \leq 1 \end{cases}$$

We have $\mu([0,1]) = 1$ and since μ is a step function, the corresponding distribution function F is piecewise linear. The unique consensus halving is obtained by placing a cut at the point $\varepsilon/2 - \varepsilon^2/(2+2\varepsilon)$. Placing a cut at any point $t \in [\varepsilon/2 - \varepsilon^2/(1+\varepsilon), 1-\varepsilon/2]$ results in an ε-consensus halving, i.e. such that $\left|\mu([0,t]) - \mu([t,1])\right| \leq \varepsilon$. Thus an ε-consensus halving might be very far from an actual consensus halving. Note also that placing a cut at any point $t \in [0, 3\varepsilon/2 - \varepsilon^2/(2+2\varepsilon)]$ is a strong ε-approximation, which illustrates that a strong approximation is not necessarily a weak approximation. On the other hand, a strong $(\varepsilon^2/2)$-approximation is also an ε-consensus halving.

The Brouwer fixed point theorem and the Borsuk-Ulam theorem can both be proved starting from combinatorial analogues of the two theorems, namely from Sperner's lemma and Tucker's lemma, respectively. The proofs of these two lemmas are constructive, but using them to derive the Brouwer fixed point theorem and the Borsuk-Ulam theorem involve a nonconstructive limit argument. Let us note in passing that while Sperner's lemma, like the Borsuk-Ulam theorem, has several different formulations, it is usually formulated as the combinatorial analogue of the third formulation of Theorem 1.

Sperner's and Tucker's lemma give rise to total NP search problems. These turn out to be complete for the complexity classes PPAD and PPA introduced in the seminal work by Papadimitriou [19]. Papadimitriou proved PPAD-completeness of the problem given by Sperner's lemma as well as membership in PPA of the problem given by Tucker's lemma, while PPA-completeness of the latter problem was proved recently by Aisenberg, Bonet, and Buss [1]. These results also imply that the classes PPAD and PPA correspond to the problems of computing weak approximations to Brouwer fixed points and to Borsuk-Ulam points.

The computational complexity of the problems of computing an ε-NE and of computing an ε-consensus halving was settled in breakthroughs of two lines of research. Computing an ε-NE was shown to be PPAD-complete by Daskalakis, Goldberg and Papadimitriou [9] and Cheng and Deng [8]. Computing an ε-consensus halving was shown to be PPA-complete by Filos-Ratsikas and Goldberg [14, 15].

1.4 Our Results

Our main result is that the problem of strong approximation of consensus halving is equivalent to strong approximation of the Borsuk-Ulam theorem.

▶ **Theorem 2.** *The strong approximation problem for* CH *is* BU_a*-complete.*

As described, we view the consensus halving problem as the real valued search problem with its domain being either the unit sphere or the unit ball with respect to the ℓ_1-norm. The theorem is proved by reduction from the real valued search problem associated with the Borsuk-Ulam theorem on the domain being the unit ball with respect to the ℓ_∞-norm, i.e. from a defining problem of the class BBU.

It is of general interest to study the relationship between search problems given by the Borsuk-Ulam theorem on different domains from a computational point of view. The reduction establishing the proof of Theorem 2 gives additional motivation for this. The domains we consider are unit spheres S_p^n and unit balls B_p^n with respect to the ℓ_p-norm for $p \geq 1$ or $p = \infty$. It is of course straightforward to construct homeomorphisms between unit spheres or unit balls with respect to different norms, and these could be used to define reductions between the different problems. We would however like that the mapping of solutions is simple, and in particular we would like to avoid divisions and root operations. We prove that one may in fact reduce between domains using SL-reductions.

Deligkas et al. gave a reduction from the FIXP-complete problem of finding a Nash equilibrium to CH. Combined with membership of CH in BU, this gives the inclusion $\mathrm{FIXP} \subseteq \mathrm{BU}$. We observe that a proof due to Volovikov [24] of the Brouwer fixed point theorem from the Borsuk-Ulam theorem may be adapted to give a simple proof of the inclusion $\mathrm{FIXP} \subseteq \mathrm{BU}$.

For the class FIXP we prove two interesting structural properties that do not appear to have been observed earlier. While FIXP is defined using SL-reductions, we show that FIXP is closed under polynomial time reductions where the mapping of solutions is expressed by *general* algebraic circuits. This in particular supports that one may reasonably define the classes BU and BBU using less restrictive notions of reductions than SL-reductions. We propose to have the mapping of solutions be computed by algebraic circuits involving the operations of addition, multiplication by scalars, as well as maximization. This means that the mapping of solutions is a piecewise linear function, and we refer to these as PL-reductions. The second structural result for FIXP is a characterization of the class by very simple Brouwer functions. These are defined on the unit-hypercube domain $[0,1]^n$ and each coordinate function is simply one of the operations $\{+, -, *, \max, \min\}$, modified to have the output truncated to the interval $[0,1]$.

For the classes BU and BBU we prove that they are also closed under reductions where the mapping of solutions is computed by general algebraic circuits, but with the additional requirement that this function must be odd.

For the class FIXP, an interesting consequence of the proof that finding a Nash equilibrium is complete, is that the class may be characterized by Brouwer functions computed by algebraic circuits without the division operation. The proof also shows that the class FIXP is unchanged even when allowing root operations as basic operations. We prove by a simple transformation that the classes BU and BBU may be characterized using algebraic circuits without the division operation. Furthermore, as a consequence of Theorem 2 the class of strong approximation problems $\mathrm{BU}_a = \mathrm{BBU}_a$ is unchanged even when allowing root operations as basic operations.

1.5 Comparison to previous work

As a precursor to the proof of PPA-completeness of computing an ε-consensus halving, Filos-Ratsikas, Frederiksen, Goldberg and Zhang [13] proved the problem to be PPAD-hard. Deligkas et al. [11] uses ideas from this proof together with additional new ideas to obtain their proof of FIXP-hardness for computing an exact consensus halving.

While PPAD $\subseteq$ PPA, the PPAD-hardness result of [13] is not implied by the recent proofs of PPA-completeness. In particular, the work [13] proves PPAD-hardness even for *constant* ε, while the work of [15] only proves PPA-hardness for ε being inverse polynomially small. In the same way, while FIXP $\subseteq$ BU, FIXP-hardness of computing an exact consensus halving is not implied by our reduction, since Theorem 2 establishes BU_a-hardness rather than BU-hardness. Recently a considerably simpler proof of PPA-hardness for computing an ε-consensus halving was given by Filos-Ratsikas, Hollender, Sotiraki and Zampetakis [16], and our reduction is inspired by this work.

All reductions described above are similar in the sense that one or more evaluations of a circuit are expressed in the consensus halving instance. The full interval A is partitioned into subintervals, cuts within these subintervals encode values in various ways, and agents implement the gates of the circuit by placing cuts. A main difference between the reductions establishing PPAD-hardness and FIXP-hardness to those establishing PPA-hardness is that in the former reductions, all cuts are constrained to be placed in distinct subintervals. The reason this is possible is that the objective is to find a fixed point of the circuit, which means that inputs and outputs may be identified.

In the setting of PPA and BBU the objective is to find a "zero" of the circuit. More precisely, for the setting of PPA the objective is to find two adjacent points of a given Tucker labeling that receive complementary labels, i.e. labels of different sign but same absolute value. For the setting of BBU the objective is to find an actual zero point of the circuit. All of the reductions establishing PPA-hardness of computing an ε-consensus halving have the property that cuts encoding the input of the circuit are *free* cuts, meaning that they can in principle be placed anywhere, and as a result also interfere with the evaluations of the circuit. This is also the case for our reduction, and this invariably limits its applicability to the approximation problem.

In the reduction of [16], the interval A is structured into different regions, a coordinate-encoding region, a constant-creation region, several circuit-simulation regions, and finally a feedback region. Our reduction also has a coordinate-encoding region and several circuit simulation regions, but the functions performed by the constant-creation region and feedback regions in [16] are in our reduction integrated into the individual circuit simulation regions and done differently.

A novelty of the reduction of [16] compared to previous reductions is in how values are encoded by cuts in subintervals. In previous reductions, values are encoded by what we will call *position encoding*. In that encoding it is required that there is exactly one cut in the subinterval, and the value encoded is determined by the distance between the cut position and the left endpoint of the interval. In [16] values are encoded by what we will call *label encoding*. Here there is no requirement on the number of cuts in the subinterval, and the value encoded is simply the difference between the Lebesgue measures of the subsets of the interval receiving label $+$ and label $-$. We shall employ a hybrid approach where the coordinate-encoding region uses label encoding while the circuit-simulation regions uses position encoding. The first step performed in a circuit-simulation region is thus to copy the input from the coordinate-encoding region. Switching to position encoding allows us in particular to implement a multiplication gate, similarly to [11]. Here the multiplication xy is

computed via the identity $xy = ((x+y)^2 - x^2 - y^2)/2$. In [11] where values range over $[0,1]$, the squaring operation may be implemented directly by agents. In our case values range over the interval $[-1,1]$, and the squaring operation is decomposed further, having agents compute it separately over the intervals $[-1,0]$ and $[0,1]$.

In analogy to [16] we have feedback agents that ensures that the circuit evaluates to 0 on the encoded input. The criteria that the agents check is however different, and for our purposes it is crucial that we have the same sign pattern in the position encoding of the output of the circuit as the copy of the input made by the circuit-simulation region. The actual detection of an output of 0 is performed by using approximations of the Dirac delta function. For computing the distribution functions of the feedback agents, we make use of the fact that these are computed by algebraic circuits, which enable us to make a strong approximation of the Dirac delta function via repeated squaring.

1.6 Organization of Paper

We introduce required terminology in Section 2. We refer to the full version of this paper for our structural results for the classes FIXP, BU, and BBU as well as the simple proof of the inclusion FIXP $\subseteq$ BU. The proof of our main result, Theorem 2, is given in Section 3.

2 Preliminaries

2.1 Algebraic Circuits

Central to our work are algebraic circuits computing real valued functions. Let B be a finite set of real valued functions, for example $B = \{+, -, *, \div, \max, \min\}$. An *algebraic circuit* C with n inputs and m outputs over the *basis* B is given by an acyclic graph $G = (V, A)$ as follows. The *size* of C is equal to the number of nodes of G, which are also referred to as *gates*. The *depth* of C is equal to the length of the longest path of G. Every node of indegree 0 is either an *input gate* labeled by a variable from the set $\{x_1, \ldots, x_n\}$ or a *constant gate* labeled by a real valued constant. Every other node is labeled by an element of B called the *gate function*. If a node v is labeled by a gate function $g\colon A \to \mathbb{R}$ with $A \subset \mathbb{R}^k$ we require that g has exactly k ingoing arcs with a linear order specifying the order of arguments to g. The output of C is specified by an ordered list of m (not necessarily distinct) nodes of G. The computation of C on a given input $x \in \mathbb{R}^n$ is defined in the natural way. Computation may fail in case a gate of C labeled by a function $g\colon A \to \mathbb{R}$ receives an input outside A, and in this case the output of C is undefined. Otherwise we say that the output is well defined and denote its value by $C(x)$. If $D \subseteq \mathbb{R}^n$ we say that C computes a function $f\colon D \to \mathbb{R}^m$ if $C(x)$ is well defined for all $x \in D$.

We shall in this paper just consider algebraic circuits where the basis consists only of continuous functions. This means in particular that any algebraic circuits computes a continous function as well. We shall also only consider consider constant gates labeled with rational numbers. In this case we are also interested in the *bitsize* of the encoding of the constants, which is the maximum bitsize of the numerator or denominator.

By using multiplication with the constant -1, the functions $-$ and min may be simulated using $+$ and max, respectively. In this way we may convert a circuit over the full basis $\{+, -, *, \div, \max, \min\}$ into an equivalent $\{+, *, \div, \max\}$-circuit. We shall also consider circuits where use of the multiplication operator $*$ is *restricted* to having one of the arguments being a constant gate. We denote this by the symbol $*\zeta$ and use it in particular for defining $\{+, *\zeta, \max\}$-circuits.

2.2 Search problems

A general search problem Π is defined by specifying to each input instance I a *search space* (or *domain*) D_I and a set $\mathrm{Sol}(I) \subseteq D_I$ of *solutions*. We distinguish between *discrete* and *real-valued* search problems. For discrete search problems we assume that $D_I \subseteq \{0,1\}^{d_I}$ for an integer d_I depending on I. Analogously, for real-valued search problems we assume that $D_I \subseteq \mathbb{R}^{d_I}$ for an integer d_I depending on I. One could likewise distinguish between search problems with discrete input and real-valued input. We are however mostly interested in problems where the input is discrete. That is, we assume that instances I are encoded as strings over a given finite alphabet Σ (e.g. $\Sigma = \{0,1\}$).

Many natural search problems are however defined with a continous search space. Not all of these may adequately be recast as discrete search problems, but are more naturally viewed as real-valued search problems. One approach for studying such problems would be to switch to the Blum-Shub-Smale model of computation [4]. A BSS machine resembles a Turing machine, but operates with real numbers instead of symbols from a finite alphabet. In particular is the input real-valued, and input instances are therefore encoded as real-valued vectors. All basic arithmetic operations and comparisons are unit-cost operations. One may then define real-valued analogues of Turing machine based classes. In particular, Blum, Shub and Smale defined and studied the real-valued analogues $\mathrm{P}_{\mathbb{R}}$ and $\mathrm{NP}_{\mathbb{R}}$ of P and NP. A BSS machine may in general make use of real-valued *machine constants*. If a BSS machine only uses rational valued machine constants we shall call it *constant-free*.

For the classes $\mathrm{P}_{\mathbb{R}}$ and $\mathrm{NP}_{\mathbb{R}}$, if we simply restrict the input to be discrete and consider only constant-free BSS machines this results in complexity classes, denoted by $\mathrm{BP}(\mathrm{P}^0_{\mathbb{R}})$ and $\mathrm{BP}(\mathrm{NP}^0_{\mathbb{R}})$, that may directly be compared to Turing machine based complexity classes. Indeed, it was proved by Allender, Bürgisser, Kjeldgaard-Pedersen and Miltersen [2, Proposition 1.1] that $\mathrm{BP}(\mathrm{P}^0_{\mathbb{R}}) = \mathrm{P}^{\mathrm{PosSLP}}$, where PosSLP is the problem of deciding whether an integer given by a division free arithmetic circuit (i.e. a $\{+,-,*\}$-circuit using just the constant 1) is positive. While the precise complexity of PosSLP is not known, Allender et al. proved that it is contained in the *counting hierarchy* CH (not to be confused with the consensus halving problem whose abbreviation coincides). The class $\mathrm{BP}(\mathrm{NP}^0_{\mathbb{R}})$ is equal to the class $\exists\mathbb{R}$ that was defined by Schaefer and Štefankovič [20] to capture the complexity of the existential theory of the reals ETR. It is known that $\mathrm{NP} \subseteq \exists\mathbb{R} \subseteq \mathrm{PSPACE}$, where the latter inclusion follows from the decision procedure for ETR due to Canny [7].

We define the class of $\exists\mathbb{R}$ search problems as the following subclass of all real valued search problems. Instaces I are encoded as string over a given finite alphabet Σ and we assume there is a polynomial time algorithm that given I computes d_I, where $D_I \subseteq \mathbb{R}^{d_I}$. We next assume that there are polynomial time constant free *BSS machines* that given I and $x \in \mathbb{R}^{d_I}$ checks whether $x \in D_I$, and given I and $x \in D_I$ checks whether $x \in \mathrm{Sol}(I)$. The corresponding language in $\exists\mathbb{R}$ is then $L = \{I \mid \mathrm{Sol}(I) \neq \emptyset\}$.

2.3 Solving real-valued search problems

Let Π be a $\exists\mathbb{R}$ search problem. In analogy with the case of NP search problems, one could consider the task of solving Π to be that of giving as output some member y of $\mathrm{Sol}(I)$ in case $\mathrm{Sol}(I) \neq \emptyset$. In general each member of $\mathrm{Sol}(I)$ may be irrational valued which precludes a Turing machine to compute a solution explicitly. This is in general also the case for a BSS machine, even when allowing machine constants. Regardless, we shall restrict our attention to Turing machines below.

On the other hand, when $\mathrm{Sol}(I) \neq \emptyset$ a solution is guaranteed to exist with coordinates being algebraic numbers, since a member of $\mathrm{Sol}(I)$ may be defined by an existential first-order formula over the reals with only rational-valued coefficients. This means that one could instead compute an indirect description of the coordinates of a solution, for instance by describing isolated roots of univariate polynomials. If such a description could be computed in polynomial time in $|I|$ we could consider that to be a polynomial time solution of Π.

Etessami and Yannakakis [12] suggest several other computational problems one may alternatively consider in place of solving a search problems Π explicitly or exactly. Our main interest is in the problem of *approximation*. We shall assume for simplicity that $D_I \subseteq [-1,1]^{d_I}$. Together with an instance I of Π we are now given as an auxiliary input a rational number $\varepsilon > 0$, and the task is to compute $x \in \mathbb{Q}^{d_I}$ such that there exist $x^* \in \mathrm{Sol}(I)$ with $\|x^* - x\|_\infty \leq \varepsilon$. We shall turn this into a *discrete* search problem by encoding the coordinates of x as binary strings. More precisely, to Π we shall associate a discrete search problem Π_a where instances are of the form (I,k), where I is an instance of Π and k is a positive integer. We define $\varepsilon = 2^{-k}$ and let the domain of (I,k) be $D_{I,k} = \{0,1\}^{d_I(k+3)}$, thereby allowing the specification of a point $x \in D_I$ with coordinates of the form $x_i = a_i 2^{-k+1}$, where $a_i \in \{-2^{k+1}, \ldots, 2^{k+1}\}$. The solution set $\mathrm{Sol}(I,k)$ is defined from $\mathrm{Sol}(I)$ by approximating each coordinate. That is, we define $\mathrm{Sol}(I,k) = \{x \in D_{I,k} \mid \exists x^* \in \mathrm{Sol}(I) : \|x^* - x\|_\infty \leq \varepsilon\}$. Note that if we had defined $\mathrm{Sol}(I,k)$ by instead truncating the coordinates of solutions $x^* \in \mathrm{Sol}(I)$ to k bits of precision, we would have obtained the possibly harder problem of *partial computation* which was also considered by Etessami and Yannakakis [12].

We say that Π can be approximated in polynomial time if the approximation problem Π_a can be solved in time polynomial in $|I|$ and k.

2.4 Reductions between search problems

Let Π and Γ be search problems. A *many-one reduction* from Π to Γ consists of a pair of functions (f,g). The function f is called the instance mapping and the function g the solution mapping. The instance mapping f maps any instance I of Π to an instance $f(I)$ of Γ and for any solution $y \in \mathrm{Sol}(f(I))$ of Γ the solution mapping g maps the pair (I,y) to a solution $x = g(I,y) \in \mathrm{Sol}(I)$ of Π. It is required that $\mathrm{Sol}(f(I)) \neq \emptyset$ whenever $\mathrm{Sol}(I) \neq \emptyset$. We will only consider many-one reductions, and will refer to these simply as *reductions*.

If Π_1 and Π_2 are discrete search problems a reduction (f,g) between Π_1 and Π_2 is a *polynomial time reduction* if both functions f and g are computable in polynomial time. If Π_1 and Π_2 are real-valued search problems it is less obvious which notion of reduction is most appropriate and we shall consider several different types of reductions. For all these we assume that f is computable in polynomial time. The reduction (f,g) is a *real polynomial time reduction* if g is computable in polynomial time by a constant free BSS machine. We shall generally consider this notion of reduction too powerful. In particular the definition does not guarantee that the function g is a continuous function in its second argument y. For this reason we instead consider reductions defined by algebraic circuits over a given basis B of real-valued basis functions.

We say that the reduction (f,g) is a *polynomial time B-circuit reduction* if there is a function computable in polynomial time thats maps an instance I to a B-circuit C_I in such a way that C_I computes a function $C_I \colon D_{f(I)} \to D_I$ where $g(I,y) = C_I(y)$ for all $y \in \mathrm{Sol}(f(I))$. Note in particular that the size of C_I and the bitsize of all constant gates are bounded by a polynomial in $|I|$. If in addition there exists a constant h such that the depth of C_I is bounded by h for all I we say that the reduction (f,g) is a *polynomial time constant depth B-circuit reduction*. Etessami and Yannakakis [12] defined the even weaker notion where the function

f is a *separable linear* transformation. The reduction (f, g) is an SL-reduction if there is a function $\pi\colon \{1,\dots,d_I\} \to \{1,\dots,d_{f(I)}\}$ and rational constants a_i, b_i, for $i = 1,\dots,d_I$, all computable in polynomial time from I, such that for all $y \in \mathrm{Sol}(f(I))$ it holds that $x_i = a_i y_{\pi(i)} + b_i$, where $x = g(I, y)$. Thus an SL-reduction is simply a *projection reduction* together with an individual affine transformation of each coordinate of the solution.

Functions computed by algebraic circuits over the basis $\{+, *\zeta, \max\}$ are piecewise linear. We shall thus call polynomial time $\{+, *\zeta, \max\}$-circuit reductions for polynomial time piecewise linear reductions, or simply PL-reductions.

It is easy to see that all notions of reductions defined above are transitive, i.e. if Π reduces to Γ and Γ reduces to Λ, then Π reduces to Λ as well.

A desirable property of PL-reductions is that the solution mapping g is *polynomially continuous*. By this we mean that for all rational $\varepsilon > 0$ there is a rational $\delta > 0$ such that for all points x and y of the domain, $\|x - y\|_\infty \leq \delta$ implies $\left\|g(x) - g(y)\right\|_\infty \leq \varepsilon$, and the bitsize of δ is bounded by a polynomial in the bitsize of ε and of $|I|$.

2.5 Total real-valued search problems

Like in the case of TFNP where interesting classes of total NP search problems may be defined in terms of existence theorems for finite structures [19, 17], we may define classes of total real valued $\exists\mathbb{R}$ search problems based on existence theorems concerning domains $D_I \subseteq \mathbb{R}^n$. Typical examples of such domains D_I are spheres and balls. Suppose p is either a real number $p \geq 1$ or $p = \infty$. By S_p^n and B_p^n we denote the unit n-sphere and unit n-ball with respect to the ℓ_p-norm defined as $S_p^n = \{x \in \mathbb{R}^{n+1} \mid \|x\|_p = 1\}$ and $B_p^n = \{x \in \mathbb{R}^n \mid \|x\|_p \leq 1\}$, respectively. If p is not specified, we simply assume $p = 2$.

2.5.1 The Brouwer fixed point theorem and FIXP

We recall here the definition of the class FIXP by Etessami and Yannakis [12]. The class FIXP is defined by starting with $\exists\mathbb{R}$ search problems given by the Brouwer fixed point theorem, and afterwards closing the class with respect to SL-reductions. We shall refer to these defining problems as *basic* FIXP problems.

▶ **Definition 3.** *An $\exists\mathbb{R}$ search problem Π is a* basic FIXP *problem if every instance I describes a nonempty compact convex domain D_I and a continuous function $F_I\colon D_I \to D_I$, computed by an algebraic circuit C_I, and these descriptions must be computable in polynomial time. The solution set is* $\mathrm{Sol}(I) = \{x \in D_I \mid F_I(x) = x\}$.

The Brouwer fixed point theorem guarantees that every basic FIXP problem is a total $\exists\mathbb{R}$ search problem. To define the class FIXP, Etessami and Yannakis restrict attention to a concrete class of basic FIXP problems.

▶ **Definition 4.** *The class* FIXP *consists of all* total $\exists\mathbb{R}$ *search problems that are SL-reducible to a basic* FIXP *problem for which each domain D_I is a convex polytope described by a set of linear inequalities with rational coefficients and the function F_I is defined by a $\{+, -, *, \div, \max, \min\}$-circuit C_I.*

The class FIXP_a is the class of strong approximation problems corresponding to FIXP. More precisely, FIXP_a consist of all discrete search problems polynomial time reducible to the problem Π_a for $\Pi \in \mathrm{FIXP}$.

The definition of FIXP is quite robust with respect to the choice of domain and set of basis functions allowed by circuits in the basic FIXP problems. Etessami and Yannakis proved that basic FIXP problems defined by $\{+, -, *, \div, \max, \min, \sqrt[k]{\ }\}$-circuits are still in

the class FIXP. Likewise, basic FIXP-problems where D_I is a ball with rational-valued center and diameter, or more generally an ellipsoid given by a rational center-point and a positive-definite matrix with rational entries, are still in the class FIXP [12, Lemma 4.1]. The same argument allows for using as domain the ball B_p^d with respect to the ℓ_p norm for any rational $p \geq 1$ or $p = \infty$, with the coordinates possibly transformed by individual affine functions.

2.5.2 The Borsuk-Ulam theorem, BU, and BBU

A new class BU of total $\exists\mathbb{R}$ search problems based on the Borsuk Ulam theorem was recently introduced by Deligkas et al. [11]. The definition of BU is meant to capture the Borsuk-Ulam theorem as stated in formulation (1) of Theorem 1. Following the definition of FIXP by Etessami and Yannakakis, Deligkas et al. first consider a set of basic search problems and then close the class under reductions. For defining BU, Deligkas et al. restrict their attention to spheres with respect to the ℓ_1-norm as domains and functions computed by $\{+, -, *, \max, \min\}$-circuits. Compared to the definition of FIXP, division gates are thus excluded. Having thus fixed the set of basic BU search problems what remains in order to define BU is to settle on a notion of reductions. In their journal paper, Deligkas et al. [11] suggest using reductions computable by general algebraic circuits including non-continuous comparison gates, whereas in the preceeding conference paper [10] they did not precisely define a choice of reductions.

We propose defininining BU using a different notion of reduction below. We additionally define a class BBU based on the Borsuk-Ulam theorem corresponding to formulation (3) of Theorem 1. We start by defining basic BU and basic BBU problems. We shall restrict our attention to domains being the unit n-sphere and unit n-ball, but with regards to any ℓ_p-norm for $p \geq 1$ or $p = \infty$.

▶ **Definition 5.**

1. *An $\exists\mathbb{R}$ search problem Π is a basic ℓ_p-*BU *problem if for every instance I we have $D_I = S_p^{d_I}$ and I describes a continuous function $F_I \colon D_I \to \mathbb{R}^{d_I - 1}$, computed by an algebraic circuit C_I whose description is computable in polynomial time. The solution set is* $\mathrm{Sol}(I) = \{x \in D_I \mid F_I(x) = F_I(-x)\}$.
2. *An $\exists\mathbb{R}$ search problem Π is a basic ℓ_p-*BBU *problem if for every instance I we have $D_I = B_p^{d_I}$ and I describes a continuous function $F_I \colon D_I \to \mathbb{R}^{d_I}$, which is odd on the boundary $\partial B_p^{d_I}$. The function F_I must be computed by an algebraic circuit C_I whose description is computable in polynomial time. The solution set is* $\mathrm{Sol}(I) = \{x \in D_I \mid F_I(x) = 0\}$.

The condition that the function F_I is odd on $\partial B_p^{d_I}$ for basic ℓ_p-BBU problems is a semantic condition. However, typically the function F_I would be defined from a basic ℓ_p-BU problem by a transformation done in a similar way as in the proof of Theorem 1, and thereby F_I would satisfy the condition automatically.

To define the classes BU and BBU, we restrict our attention to domains with respect to the ℓ_∞-norm.

▶ **Definition 6.** *The class* BU *(respectively,* BBU*) consists of all* total $\exists\mathbb{R}$ *search problems that are PL-reducible to a basic ℓ_∞-*BU *problem (respectively, basic ℓ_∞-*BBU *problem) for which the function F_I is defined by a $\{+, -, *, \div, \max, \min\}$-circuit C_I.*

While the definition of BU in [11] was using as domain the unit sphere with respect to the ℓ_1-norm and not allowing for division gates, we show in the full version of this paper that these changes do not change the class. We propose choosing PL-reductions for closing the

class under reductions. PL-reductions are sufficient for obtaining all of our results and they are polynomially continuous. Another reason for this choice is that if we restrict the circuits defining the classes FIXP and BU to also be piecewise linear, i.e. be $\{+, *\zeta, \max\}$-circuits, we obtain the classes LinearFIXP and LinearBU, that when closed under polynomial-time reductions are equal to PPAD and PPA, respectively [12, 11].

2.6 Consensus Halving

We give here a formal definition of consensus halving with additive measures as real valued search problems.

▶ **Definition 7.** *The problem* CH *is defined as follows. An instance I consists of a list of $\{+, -, *, \div, \max, \min\}$-circuits $C_1, \ldots, C_n$ computing distribution functions $F_1, \ldots, F_n$ defined on the interval $A = [0, 1]$. The domain is $D_I = S_1^n$ and* $\mathrm{Sol}(I)$ *constists of all x for which*

$$\sum_{j:x_j>0} F_i(t_j) - F_i(t_{j-1}) = \sum_{j:x_j<0} F_i(t_j) - F(t_{j-1}) \ , \tag{1}$$

where $t_0 = 0$ and $t_j = \sum_{k\leq j}|x_k|$, for $j = 1, \ldots, n+1$.

Given $\{+, -, *, \div, \max, \min\}$-circuits computing the distribution functions F_i, the function F computing the left-hand-side of equation (1) may now clearly be computed by $\{+, -, *, \div, \max, \min\}$-circuits as well. The result of Deligkas et al. that CH is contained in BU follows.

The existence proof of a consensus halving by Simmons and Su as well the formulation as an $\exists\mathbb{R}$ search problem by Deligkas et al. match the Borsuk-Ulam theorem as stated in formulation (1) of Theorem 1.

3 Consensus Halving

In this section we present the proof of our main result Theorem 2. This result enables an additional structural result about the class of strong approximation problems $\mathrm{BU}_a = \mathrm{BBU}_a$, showing that the class is unchanged even when allowing root operations as basic operations. We refer to the full version of this paper for details of this.

Suppose we are given a basic $\ell_\infty - \mathrm{BBU}_a$ problem Π_a with circuits over the basis $\{+, -, *, \max, \min\}$. Let (I, k) denote an instance of Π_a and put $\varepsilon = 2^{-k}$. We may in polynomial time compute a circuit C defining a function $F\colon B_\infty^n \to \mathbb{R}^n$ that is odd on the boundary S_∞^{n-1} such that $\mathrm{Sol}(I) = \{x \in B_\infty^n \mid F(x) = 0\}$. We now provide a reduction from Π_a to a CH_a-problem. In the reduction we will make use of the "almost implies near" paradigm as expressed in the following lemma. The simple proof is given in the full version of this paper.

▶ **Lemma 8.** *Let $F\colon B_\infty^n \to \mathbb{R}^n$ be a continuous map. For any $\varepsilon > 0$ there is a $\delta > 0$ such that if $\left\|F(x)\right\|_\infty \leq \delta$ then there is an $x^* \in B_\infty^n$ such that $\|x - x^*\|_\infty \leq \varepsilon$ and $F(x^*) = 0$.*

The lemma says that for any $\varepsilon > 0$, if $\left\|F(x)\right\|_\infty$ is sufficiently close to being zero, then x is ε-close to a real zero of F. When F is computed by an algebraic circuit of polynomial size, it follows by using tools from real algebraic complexity [3] that there exists some fixed polynomial q with integer coefficients such that the above lemma holds true for some $\delta \geq \varepsilon^{2^{q(|I|)}}$. We refer to the full version of this paper for details. The lemma then holds

true for $\delta = \varepsilon^{2^{q(|I|)}}$, and we may construct this number using a circuit of polynomial size by repeatedly squaring the number ε exactly $q(|I|)$ times. This number will be used by the feedback agents in our CH_a instance in order to ensure that any solution gives a solution to the $\ell_\infty - \mathrm{BBU}_a$ instance.

3.1 Overview of the Reduction

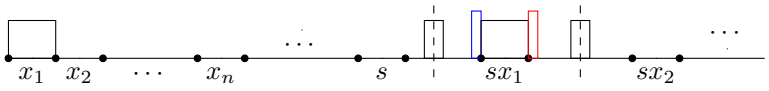
Overview. As in previous works, we describe a consensus halving instance on an interval $A = [0, M]$, where M is bounded by a polynomial in $|I|$, rather than the interval $[0, 1]$. This instance may then be translated to an instance on the interval $[0, 1]$ by simple scaling. Like [16], in the leftmost end of the instance we place the *Coordinate-Encoding* region consisting of n intervals. In a solution these intervals will encode a value $x \in [-1, 1]^n$. A *circuit simulator* C will simulate the circuit of F on this value x. The circuit simulators will consist of a number of agents each implementing one gate of the circuit. However, such a circuit simulator may fail in simulating F properly, so we will use a polynomial number of circuit simulators $C_1, \ldots, C_{p(n)}$. Each of these circuit simulators will output n values $[C_j(x)]_1, \ldots, [C_j(x)]_n$ into intervals $I_{1j}, \ldots, I_{nj}$ immediately after the simulation. Finally, we introduce the so-called *feedback agents* $f_1, \ldots, f_n$. The agent f_i will have some very thin *Dirac blocks* centered in each of the intervals I_{ij} where $j \in [p(n)]$. These agents will ensure that if z is an exact solution to the CH instance, then the encoded value x satisfies that $\left\|F(x)\right\|_\infty$ is sufficiently small that we may conclude that x is ε-close to a zero x^* of F.

Label Encoding. For a unit interval I we let $I^\pm$ denote the subsets of I assigned the corresponding label. We define the *label encoding* of I to be a value in $[-1, 1]$ given by the formula $v_l(I) := \mu(I^+) - \mu(I^-)$. This makes sense as $I^\pm$ is measurable, because it is the union of a finite number of intervals.

Coordinate-Encoding Region. The interval $[0, n]$ is called the *Coordinate-Encoding* region. For every $i \in [n]$, the subinterval $[i-1, i]$ of the Coordinate-Encoding region encodes a value $x_i := v_l([i-1, i])$ via the label encoding.

Position Encoding. For an an interval I which contains only a single cut, thus dividing I into two subintervals $I = I_a \cup I_b$, where I_a is the subinterval at the left of the cut and I_b the subinterval at the right of the cut, we define the *position encoding* of I to be the value $v_p(I) := \mu(I_a) - \mu(I_b)$. We note that $v_p(I) = v_l(I)$ if the labeling sequence is $+/-$, and $v_p(I) = -v_l(I)$ in the case the labeling sequence is $-/+$.

From Label to Position. Before a circuit simulator there is a *sign detection* interval I_s which detects the labeling sequence. Unless it contains a stray cut, this interval will encode a sign $s = \pm 1$ (to be precise 1 if the label is $+$ and -1 is the label is $-$). By placing agents that flip the label as indicated in the figure below, we may now obtain position encodings of the values $sx_1, \ldots, sx_n$. These values will be read-in as inputs to the subsequent circuit simulator.



Circuit Simulators. As mentioned above, each circuit simulator C_j will read-in the values $s_j x_1, \ldots, s_j x_n$ and simulate the circuit computing F on this input. They then output their values into n intervals immediately after the simulation.

Feedback Agents. By the discussion after the statement of Lemma 8 we may by repeated squaring construct a circuit of polynomial size in $|I|$ computing a tiny number $\delta > 0$ such that if $\left\|F(x)\right\|_\infty \leq \delta$ then x is $(\varepsilon/2)$-close to a zero of F. Now fix $i \in [n]$ and let c_{ij} denote the centre of the feedback interval I_{ij} that outputs the value $[C_j(s_j \cdot x)]_i$. We then define the ith feedback agent to have constant density $1/\delta$ in the intervals $[c_{ij} - \delta/2, c_{ij} + \delta/2]$.

The reason for having the feedback agents have these very narrow Dirac blocks is that if $F_i(x) > \delta$ for some i, then in any of the "uncorrupted" circuits (i.e. circuits outputting the correct values) all the density of the ith agent will contribute to the same label. Moreover, we will show using the boundary condition of F that the contribution is to the same label in all the uncorrupted circuit simulators. This will contradict that the feedback agents should value I^+ and I^- equally. That is, the feedback agents ensure that $\left\|F(x)\right\|_\infty \leq \delta$ if x is the value encoded by an exact solution to the consensus halving instance we construct.

Stray Cuts. Any of the agents implementing one of the gates in a circuit simulator will force a cut to be placed in an interval in that same circuit simulator. The only agents whose cuts we have no control over are the n feedback agents. The expectation is that these agents should make cuts in the Coordinate-Encoding region that flip the label. If they do not do this we will call it a *stray cut*. If a circuit simulator contains a stray cut, we will say nothing about its value.

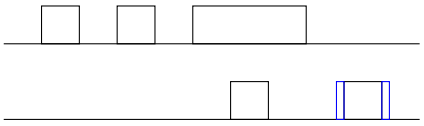
▶ **Observation 9.** *If it is not the case that every unit interval encoding a coordinate x_i in the Coordinate-Encoding region contains a cut that flips the label, then the encoded point $x \in B^n_\infty$ will lie on the boundary S^n_∞. With this in mind we may ensure that $x \in S^n_\infty$ or $s_1 = s_2 = \cdots = s_{p(n)} = \pm 1$ where the sign is the same as the label of the first interval. This can be done by, if necessary, placing one single-block agent after the Coordinate-Encoding region and each of the circuit simulators (if placing such an agent is necessary depends on, respectively, the number of variables n and the size of the circuits).*

3.2 Construction of Gates

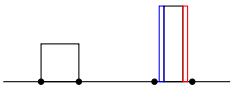
In this section we describe how to construct Consensus-Halving agents implementing the required gates $\{+, -, *, \max, \min\}$ for building the circuit simulators. By placing single-block agents between intervals, we may assume that the labeling sequence is the same in all the intervals of a circuit simulator. First, we show that we may transform the given circuit such that all gates only take values in the interval $[-1, 1]$ on input from B^n_∞.

Transforming the Circuit. By propagating every gate to the top of the circuit we may assume that the circuit is layered. Let C' denote the resulting circuit. By repeated squaring we may maintain a gate with value $1/2^{2^d}$ in the dth layer. Suppose $g = \alpha(g_1, g_2)$ is a gate with inputs g_1, g_2 in layer d. We modify the gates as follows: if $\alpha \in \{+, -, \max, \min\}$ then we multiply g_i by $1/2^{2^d}$ before applying α; if $\alpha = *$, then we multiply the input by 1 before applying α. Finally, we transform C' into the circuit C'' as follows: on input x, the circuit C'' multiplies the input by $1/2$ and then evaluates C' on input $x/2$. Inductively, one may show that if g is a gate in layer d in the circuit C', then the corresponding gate in in the circuit C'' has value $g/2^{2^d}$. As all the gates are among $\{+, -, *, \max, \min\}$, this ensures that all the gates in C'' take values in $[-1, 1]$.

Addition Gate [G_+]. We may construct an addition gate using two agents. The first agent has two unit input intervals that we assume contain one cut each. This then forces a cut in the long output interval that has length 3. The second agent then truncates this value (a cut is forced by the adjacent narrow rectangles).



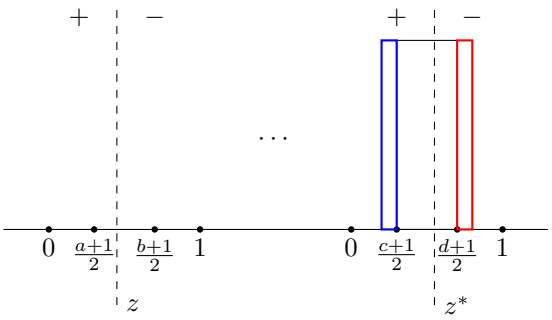
Constant Gate [G_ζ]. Let $\zeta \in [-1,1] \cap \mathbb{Q}$ be a rational constant. The agent will have a block of unit height in the sign interval and a block of width $\zeta/2$ and height $2/\zeta$ centered in another interval.



Before proceeding with the remaining gates, we construct a general function gate, an agent that implements any decreasing function.

Function Gate [G_h]. Let $-1 \leq a < b \leq 1$ and $-1 \leq c < d \leq 1$ be rational numbers and consider a continuously differentiable map $h\colon [a,b] \to [c,d]$ satisfying $h(a) = d$ and $h(c) = c$. Let $\overline{h}$ denote the extension of h that is constant on $[-1,a]$ and $[b,1]$. We now construct an agent with input interval I and output interval O computing this map, that is the agent should force a cut in the output interval such that $\overline{h}(v_p(I)) = v_p(O)$.

The agent that we construct has a block of height $2/(d-c)$ in the sub-interval $[(c+1)/2, (d+1)/2]$ of the output interval and density $f(z) := -2h'(2z-1)/(d-c)$ in the sub-interval $((a+1)/2, (b+1)/2)$ of the input interval. We note that f is positive in this interval as h is assumed to be a decreasing map, so it makes sense for the agent to have density f. One may verify that the agent values the input interval and output interval equally. We further add two narrow rectangles adjacent to the output interval. These will ensure that if the cut in the input interval is placed at $z \leq (a+1)/2$ such that $v_p(I) \leq a$, then the cut in the output interval must be placed at $z^* = (d+1)/2$, meaning that $v_p(O) = d$. Similarly, if $v_p(I) \geq b$ then $v_p(O) = c$.



Suppose cuts are placed in z in the input interval and in z^* in the output interval. As the agent must value the parts with positive and negative label equally, we get the equality

$$1 = \int_{(a+1)/2}^{z} \frac{-2h'(2t-1)}{d-c}\,dt + \left(z^* - \tfrac{c+1}{2}\right)\tfrac{2}{d-c}$$

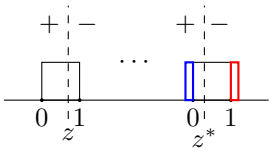
From this we obtain that

$$d - c = -\int_a^{2z-1} h'(u)\,du + 2z^* - c - 1 = -h(2z-1) + d + 2z^* - c - 1$$

where we use that $h(a) = d$ by assumption. We conclude that $h(2z-1) = 2z^* - 1$, that is we obtain the equality $h(v_p(I)) = v_p(O)$.

Using this general function gate, we may now build up the remaining gates required by the circuit.

Multiplication By -1 Gate [$G_{-(\cdot)}$]. In order to realise this gate, we consider the function $h\colon [-1,1] \to [-1,1]$ given by $x \mapsto -x$. The agent's density function in the input interval is then given by $f(z) = 1$.



Subtraction Gate [G_-]. We may build this using the gates $G_{-(\cdot)}$ and G_+.

Multiplication by $\zeta \in [-1,1]$ [$G_{\cdot\zeta}$]. If $\zeta < 0$ we mahy construct $G_{\cdot\zeta}$ as a function gate using the function $h\colon [-1,1] \to [\zeta, -\zeta]$. If $\zeta > 0$ we construct using $-\zeta$ and a minus gate, i.e. $G_{\cdot\zeta} = -G_{\cdot(-\zeta)}$.

Maximum Gate [$G_{\max}$]. First we show how to construct a gate computing the absolute value of the input. We may construct gates G_1, G_2 such that $G_1(x) = -\max(x, 0)$ and $G_2(x) = \max(-x, 0)$ as function gates by using the functions $h_1\colon [0,1] \to [-1,0]$ given by $x \mapsto -x$ and $h_2\colon [-1,0] \to [0,1]$ given by $x \mapsto -x$. Now, we may constrcuct the absolute value gate as $G_{|\cdot|} = -G_1 + G_2$. We may now construct $G_{\max}$ by using the formula $\max(x,y) = (x + y + |x - y|)/2$.

Minimum Gate [$G_{\min}$]. We may build this using $\min(x,y) = x + y - \max(x,y)$.

Multiplication Gate [G_*]. We start off by constructing a gate squaring the input. First we construct G_1 and G_2 as function gates with respect to $h_1\colon [-1,0] \to [0,1]$ given by $x \mapsto x^2$ and $h_2\colon [0,1] \to [-1,0]$ given by $x \mapsto -x^2$. Then we may construct the squaring gate as $G_{(\cdot)^2} = G_1 - G_2$. Now we may use the previously constructed gates to make a multiplication gate via the identity $xy = ((x+y)^2 - x^2 - y^2)/2$.

3.3 Describing valuation functions as circuits

In the description above, we described the valuations of the agents by providing formulas for their densities. However, an instance of CH actually consists of a list of algebraic circuits computing the distribution functions of the agents. In order to construct gates, it is sufficient for agents to have densities that are piece-wise polynomial. Therefore, consider an agent with polynomial densities f_i in the intervals $[a_i, b_i)$ for $i = 1, \ldots, s$, and let F_i denote the indefinite integral of f_i. We note that F_i is a polynomial so it may be computed by an algebraic circuit. Now we claim that the distribution function of this agent may be computed by an algebraic circuit via the formula $F(x) = \sum_{i=1}^{s} [F_i(\max(a_i, \min(x, b_i))) - F_i(a_i)]$.

This is the case, because the summands will be equal to $F_i(a_i) - F_i(a_i) = 0$ if $x < a_i$, to $F_i(x) - F_i(a)$ if $a_i \leq x \leq b_i$ and to $F_i(b) - F_i(a)$ if $x > b_i$, meaning that this formula does indeed calculate the valuation of the agent in the interval $[0, x]$.

3.4 Reduction and Correctness

Recall that we are given an instance (F, ε) of the BBU_a problem and that we have to construct an instance of the CH_a problem. The reduction now outputs an instance of the CH_a problem where the consensus halving instance is constructed as above with $p(n) = 2n + 1$ circuit simulators and the approximation parameter is given by $\varepsilon' = \varepsilon/(4n)$. Let z denote a solution to this CH_a instance. By definition, there exists an exact solution z^* to the consensus-halving problem such that $\|z - z^*\|_\infty \leq \varepsilon'$.

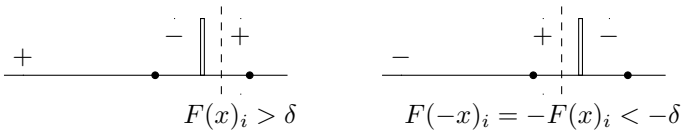
Let x and x^* denote the values encoded by respectively z and z^* in the Coordinate-Encoding region. Suppose, generally, we are given an interval I with a number of cut points $t_1, \ldots, t_s$. Moving a cut point by a distance $\leq \varepsilon'$ we create a new interval I'. This changes the label encoding by at most $2\varepsilon'$, that is $|v_l(I) - v_l(I')| \leq 2\varepsilon'$. Succesively, if we move all the cuts by a distance $\leq \varepsilon'$, then we get an interval I^* such that $|v_l(I) - v_l(I^*)| \leq 2s\varepsilon'$. As $\|z - z^*\|_\infty \leq \varepsilon'$ and any of the subintervals in the Coordinate-encoding region can contain at most n cuts, we conclude that $\|x - x^*\|_\infty \leq 2n\varepsilon' = 2n(\varepsilon/(4n)) = \varepsilon/2$. In order to show that x is ε-close to a zero of F, it now suffices by the triangle inequality to show that x^* is $(\varepsilon/2)$-close to a zero of F. This will follow from the two following lemmas.

▶ **Lemma 10.** *If there are no stray cuts in the exact solution z^*, then the associated value x^* encoded in the Coordinate-encoding region satisfies $F(x^*) = 0$.*

Proof. We recall that if the solution z^* contain no stray cuts, then the signs of all the circuit simulators are equal $s_1 = \cdots = s_{2n+1} = s$ where $s = \pm 1$. Furthermore, all the circuit simulators will output the same values $F_1(sx^*), \ldots, F_n(sx^*)$ into the feedback intervals. Thus, there can be no cancellation, so in order for the feedback agents to value the positive and negative part equally it must be the case that $F(sx^*) = 0$. ◀

▶ **Lemma 11.** *If there is a stray cut in the exact solution z^*, then the associated value x^* encoded in the Encoding-region satisfies the inequality $\left\|F(x^*)\right\|_\infty \leq \delta$.*

Proof. Suppose toward contradiction that $|F(x^*)_i| > \delta$ for some i. Without loss of generality we assume that $F(x^*)_i > \delta$. As there is a stray cut, the Coordinate-Encoding region can contain at most $n - 1$ cuts. Thus, at least one of the coordinates x_i^* must be ± 1 showing that $x^* \in S^{n-1}$. From this and the boundary condition we conclude that $F(x^*) = -F(-x^*)$. Furthermore, there is at most n stray cuts, so at most n circuit simulators can become corrupted. This means that $n + 1$ circuit simulators work correctly. Now suppose that the circuit simulator C_j is uncorrupted. If the label is $s_j = +1$, then C_j will output $F(x^*)$ into the feedback region and the labeling sequence will be $+/-$; if the label is $s_j = -1$ then C_j will output $F(-x^*) = -F(x^*)$ into the feedback region and the labeling sequence will be $-/+$. This is indicated below:



From this we conclude that the $n+1$ uncorrupted circuit simulators altogether contribute $(n+1)\delta$ to the part with negative label. However, the n corrupted circuit simulators can contribute at most $n\delta$ to the part with positive label. This implies that f_i cannot value the negative and positive part equally. This contradicts the assumption that z^* is an exact consensus-halving. We conclude that $\left\|F(x^*)\right\|_\infty \leq \delta$. ◀

By the two lemmas above, it follows that the value x^* encoded by the exact consensus-halving z^* satisfies the inequality $\left\|F(x^*)\right\|_\infty \leq \delta$. By choice of δ, this implies that there exists some x^{**} such that $\|x^* - x^{**}\|_\infty \leq \varepsilon/2$ and $F(x^{**}) = 0$. From the discussion before the two lemmas, it follows that x is ε-close to a zero of F and is thus a solution to the BBU_a instance (F, ε).

Mapping back a Solution. What remains is to show that we may recover a solution x to the BBU_a instance from the solution z to the CH_a instance. Recall that in a solution $z = (z_1, \ldots, z_N)$ to the consensus-halving problem $|z_i|$ and $\mathrm{sgn}(z_i)$ represents the length and label of the ith interval. For $i \leq n$ and $j \leq n+1$ we let $t_j = \sum_{k=1}^{j-1} |z_k|$ and define

$$x_{ij}^+ = \max(0, \min(t_{j-1} + z_j, i) - \max(t_{j-1}, i-1)) \quad \text{and}$$
$$x_{ij}^- = \max(0, \min(t_{j-1} - z_j, i) - \max(t_{j-1}, i-1))$$

These numbers may be computed efficiently by a circuit over the basis $\{+, -, \max, \min\}$. We notice that if $z_j > 0$ then $x_{ij}^- = 0$ (and if $z_j < 0$ then $x_{ij}^+ = 0$). Furthermore, by checking a couple of cases, one finds that if $z_j > 0$ (respectively $z_j < 0$) then x_{ij}^+ (respectively x_{ij}^-) is the length of the jth interval that is contained in $[i-1, i]$. As the coordinate-encoding region can contain at most n cuts (corresponding to at most $n+1$ intervals), we deduce from the above that the values encoded can be computed as $x_i = \sum_{j=1}^{n+1} x_{ij}^+ - x_{ij}^-$, for every $i \leq n$. If there is a stray cut then both x and $-x$ are valid solutions by the boundary condition of F. If there is no stray cut, then $s_1 = s_2 = \cdots = s_{p(n)} = s = \mathrm{sgn}(z_1)$ by Observation 9 and in this case we may recover a solution as sx.

References

1. James Aisenberg, Maria Luisa Bonet, and Sam Buss. 2-d tucker is PPA complete. *J. Comput. Syst. Sci.*, 108:92–103, 2020. doi:10.1016/j.jcss.2019.09.002.
2. Eric Allender, Peter Bürgisser, Johan Kjeldgaard-Pedersen, and Peter Bro Miltersen. On the complexity of numerical analysis. *SIAM J. Comput.*, 38(5):1987–2006, 2009. doi:10.1137/070697926.
3. S. Basu, R. Pollack, and M. Roy. *Algorithms in Real Algebraic Geometry*. Springer, second edition, 2008.
4. Lenore Blum, M. Schub, and Steve Smale. On a theory of computation and complexity over the real numbers: NP-completeness, recursive functions and universal machines. *Bull. Amer. Math. Soc.*, 21:1–46, July 1989. doi:10.1090/S0273-0979-1989-15750-9.
5. Karol Borsuk. Drei sätze über die n-dimensionale euklidische sphäre. *Fundamenta Mathematicae*, 20(1):177–190, 1933. doi:10.4064/fm-20-1-177-190.
6. L. E. J. Brouwer. Über abbildung von mannigfaltigkeiten. *Mathematische Annalen*, 71:97–115, 1911. doi:10.1007/BF01456931.
7. John Canny. Some algebraic and geometric computations in PSPACE. In *STOC*, pages 460–467. ACM, January 1988. doi:10.1145/62212.62257.
8. Xi Chen and Xiaotie Deng. Settling the complexity of two-player Nash equilibrium. In *FOCS 2006*, pages 261–272. IEEE Computer Society Press, 2006.

9 Constantinos Daskalakis, Paul W. Goldberg, and Christos H. Papadimitriou. The complexity of computing a Nash equilibrium. *SIAM Journal on Computing*, 39(1):195–259, 2009.

10 Argyrios Deligkas, John Fearnley, Themistoklis Melissourgos, and Paul G. Spirakis. Computing exact solutions of consensus halving and the Borsuk-Ulam theorem. In *ICALP*, volume 132 of *LIPIcs*, pages 138:1–138:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019. doi:10.4230/LIPIcs.ICALP.2019.138.

11 Argyrios Deligkas, John Fearnley, Themistoklis Melissourgos, and Paul G. Spirakis. Computing exact solutions of consensus halving and the Borsuk-Ulam theorem. *Journal of Computer and System Sciences*, 117:75–98, 2021. doi:10.1016/j.jcss.2020.10.006.

12 Kousha Etessami and Mihalis Yannakakis. On the complexity of Nash equilibria and other fixed points. *SIAM J. Comput.*, 39(6):2531–2597, 2010.

13 Aris Filos-Ratsikas, Søren Kristoffer Stiil Frederiksen, Paul W. Goldberg, and Jie Zhang. Hardness results for consensus-halving. In *MFCS*, volume 117 of *LIPIcs*, pages 24:1–24:16. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2018. doi:10.4230/LIPIcs.MFCS.2018.24.

14 Aris Filos-Ratsikas and Paul W. Goldberg. Consensus halving is PPA-complete. In *STOC*, pages 51–64. ACM, 2018. doi:10.1145/3188745.3188880.

15 Aris Filos-Ratsikas and Paul W. Goldberg. The complexity of splitting necklaces and bisecting ham sandwiches. In *STOC*, pages 638–649. ACM, 2019. doi:10.1145/3313276.3316334.

16 Aris Filos-Ratsikas, Alexandros Hollender, Katerina Sotiraki, and Manolis Zampetakis. Consensus-halving: Does it ever get easier? In *EC*, pages 381–399. ACM, 2020. doi:10.1145/3391403.3399527.

17 Paul W. Goldberg and Christos H. Papadimitriou. Towards a unified complexity theory of total functions. *Journal of Computer and System Sciences*, 94:167–192, 2018. doi:10.1016/j.jcss.2017.12.003.

18 Martin Grötschel, László Lovász, and Alexander Schrijver. *Geometric Algorithms and Combinatorial Optimization*, volume 2 of *Algorithms and Combinatorics*. Springer, 1988.

19 Christos H. Papadimitriou. On the complexity of the parity argument and other inefficient proofs of existence. *J. Comput. Syst. Sci*, 48(3):498–532, 1994.

20 Marcus Schaefer and Daniel Štefankovič. Fixed points, Nash equilibria, and the existential theory of the reals. *Theory Comput Syst*, 60:172–193, November 2017. doi:10.1007/s00224-015-9662-0.

21 Forest W. Simmons and Francis Edward Su. Consensus-halving via theorems of Borsuk-Ulam and Tucker. *Math. Soc. Sci.*, 45(1):15–25, 2003. doi:10.1016/S0165-4896(02)00087-2.

22 Francis Edward Su. Borsuk-Ulam implies Brouwer: A direct construction. *The American Mathematical Monthly*, 104(9):855–859, 1997. doi:10.2307/2975293.

23 Sergey P. Tarasov and Mikhail N. Vyalyi. Semidefinite programming and arithmetic circuit evaluation. *Discrete Applied Mathematics*, 156(11):2070–2078, 2008. doi:10.1016/j.dam.2007.04.023.

24 Alexey Yu. Volovikov. Borsuk-Ulam implies Brouwer: A direct construction revisited. *Am. Math. Mon.*, 115(6):553–556, 2008.

How to Send a Real Number Using a Single Bit (And Some Shared Randomness)

Ran Ben Basat
University College London, UK

Michael Mitzenmacher
Harvard University, Cambridge, MA, USA

Shay Vargaftik
VMware Research, Herzliya, Israel

Abstract

We consider the fundamental problem of communicating an estimate of a real number $x \in [0, 1]$ using a single bit. A sender that knows x chooses a value $X \in \{0, 1\}$ to transmit. In turn, a receiver estimates x based on the value of X. The goal is to minimize the cost, defined as the worst-case (over the choice of x) expected squared error.

We first overview common biased and unbiased estimation approaches and prove their optimality when no shared randomness is allowed. We then show how a small amount of shared randomness, which can be as low as a single bit, reduces the cost in both cases. Specifically, we derive lower bounds on the cost attainable by any algorithm with unrestricted use of shared randomness and propose optimal and near-optimal solutions that use a small number of shared random bits. Finally, we discuss open problems and future directions.

2012 ACM Subject Classification Theory of computation → Rounding techniques; Theory of computation → Stochastic approximation

Keywords and phrases Randomized Algorithms, Approximation Algorithms, Shared Randomness, Distributed Protocols, Estimation, Subtractive Dithering

Digital Object Identifier 10.4230/LIPIcs.ICALP.2021.25

Category Track A: Algorithms, Complexity and Games

Related Version *Full Version*: `https://arxiv.org/abs/2010.02331`

Funding MM was supported in part by NSF grants CCF-1563710, CCF-1535795 and DMS-2023528. MM and RBB were supported in part by a gift to the Center for Research on Computation and Society at Harvard University.

Acknowledgements We thank the anonymous reviewers, Moshe Gabel, and Gal Mendelson for their helpful feedback and comments.

1 Introduction

We consider the fundamental problem of communicating an estimate of a real number $x \in [0, 1]$ using a single bit. A sender, that we call *Buffy*, knows x, and chooses a value $X \in \{0, 1\}$ to transmit. In turn, a receiver, that we call *Angel*, estimates x based on the value of X.

This problem naturally appears in distributed computations where multiple machines perform parallel tasks and transmit their results/state to an aggregator. If the bandwidth to the aggregator is limited, the machines must compress the data before sending it. Bandwidth optimization is fundamental in many domains, including network measurements [3, 11] and telemetry [5], load balancing [16, 22], and satellite communication [25]. We are especially motivated by recent work addressing the communication bottleneck in distributed and federated machine learning [13]. There, *clients* compute a local gradient and send it to a

parameter server that computes the global gradient and updates the model [15]. For the typical large-scale federated learning problems over edge devices (e.g., mobile phones), the devices may only be able to communicate a small number of bits per gradient coordinate. In fact, solutions such as 1-Bit SGD [20] and signSGD [6], have recently been studied as appealing low-communication solutions that use a single bit per coordinate. Another common communication-efficient solution is TernGrad [23] that quantizes each coordinate to $\{-1, 0, 1\}$ instead of $\{-1, 1\}$ as commonly done by the 1-bit algorithms.

It is often desirable that each estimate be an *unbiased* random variable with a mean equal to corresponding x. For example, this provides that the estimates' average is an unbiased estimate of the average value. Alternatively, there are cases where it is beneficial to allow *biased* estimates if it reduces the error for that setting (e.g., see EF-signSGD [14]).

In this work, we consider several variations of the above problem. For algorithms that provide unbiased estimates for every value x, we use the *worst-case* (over all values of x) variance as the cost function to be minimized. For biased estimates, we consider the worst-case *expected squared error* as the cost, as it coincides with variance for unbiased algorithms. That is, the worst-case is over the value of x, and the expectation of the cost is over the random choices used by the algorithm. Note that any lower bound for biased algorithms with these costs also applies to unbiased algorithms, and an upper bound for unbiased algorithms also applies to biased algorithms. We are interested in both lower and upper bounds in our work.

Beyond unbiased and biased variations, we also consider settings where Buffy and Angel have access to *shared randomness*. Shared randomness (often also referred to as public or common randomness) has been intensively studied in the field of communication complexity (e.g., see [17]). In our context, such shared randomness can arise naturally by having Buffy and Angel share a common seed for a pseudo-random number generator, for example. Here, we model the shared randomness as "perfectly random," leaving issues related to pseudo-randomness aside. Nevertheless, we consider solutions using limited amounts of shared randomness, including the case of just one bit of shared randomness. Such solutions may be easier and cheaper to implement, including with pseudo-random generators.

We remark that there are known approaches to this problem. These include (deterministic) rounding, randomized rounding (also called stochastic quantization), and subtractive dithering [18]. For a detailed survey of such techniques, we refer the reader to [9]. We discuss these methods and compare our results with them in context throughout the paper.

Our contribution. In this paper, we study how to minimize the cost (i.e., the worst-case variance or worst-case expected squared error) for various settings. First, we consider the setting where there is no shared randomness. In this setting, we show that randomized rounding is the optimal unbiased algorithm and that deterministic rounding is optimal when biased estimations are allowed. While these algorithms are widely used in practice, the optimality proofs under these cost models have not appeared elsewhere to the best of our knowledge.

Next, we explore how to reduce the cost if Buffy and Angel have access to shared randomness. We prove upper and lower bounds on the attainable variance for unbiased algorithms and expected squared error for biased ones. For our upper bounds, we assume that Buffy and Angel have access to ℓ shared random bits, for some $\ell \in \mathbb{N}$. We also consider the *limiting algorithms* where ℓ is not restricted. Our work addresses several extensions for cases where unbounded private randomness is allowed and when it is not. Finally, we consider the special case where x is known to be in $\{0, 1/2, 1\}$, a setting that is of high interest, for example, for the sign-based federated learning algorithms (e.g., [6, 14]) and particularly for TernGrad [23] that uses 3-level quantization. We provide an improved algorithm and a matching lower bound for this setting, thus proving its optimality. A summary of our results appears in Table 1.

Table 1 A summary of our results.

Scenario	Unbiased (Variance)	Biased (Exp. Squared Error)
No shared randomness	$1/4 = 0.25$ (randomized rounding) Optimal (Section 3.1)	$1/16 = 0.0625$ (deterministic rounding) Optimal (Section 3.2)
ℓ-bit shared randomness Unbounded private randomness	$\ell = 1 : \frac{1}{8} = 0.125$ $\ell = 8 : \frac{1}{12} + \frac{1}{393216} \approx 0.08334$ In general: $1/6 \cdot (1/2 + 4^{-\ell})$ (Section 5)	Open
ℓ-bit shared randomness No private randomness	Impossible (Section 6)	$\ell = 1 : \frac{1}{20} = 0.05$ $\ell = 8 : \approx 0.04599$ (Section 6.2.4)
Lower Bounds for $x \in [0, 1]$ Unbounded shared randomness	$1/16 = 0.0625$ (Section 4.2)	≈ 0.0459 (Section 4.1.2)
$x \in \{0, 1/2, 1\}$ 1-bit shared randomness No private randomness	$1/16 = 0.0625$ (Section 5)	$3/4 - 1/\sqrt{2} \approx 0.04289$ (Section 6.1)
Lower Bounds for $x \in \{0, 1/2, 1\}$ Unbounded shared randomness	$1/16 = 0.0625$ (Section 4.2)	$3/4 - 1/\sqrt{2} \approx 0.04289$ (Section 4)

2 Preliminaries

We start with some notation. We use $[n]$ to denote $\{0, 1, \ldots, n-1\}$, and $\Delta(S)$ to denote all possible probability distributions over the set S. (An element of $\Delta(S)$ will be expressed as a density function when S is uncountable, e.g., if $S = [0, 1]$.) We also use, for a binary predicate B, $\mathbb{1}_B$ as an indicator such that $\mathbb{1}_B = 1$ if B is true and 0 otherwise. Lastly, $\phi = (1 + \sqrt{5})/2$ denotes the Golden Ratio, which naturally comes up in some of our results.

Problem statement. Given a real number $x \in [0, 1]$, Buffy compresses it to a single bit value $X \in \{0, 1\}$ that is sent to Angel, who derives an estimate $\widehat{x}$. We also consider the special case where x is known to be in $\{0, 1/2, 1\}$. Our objective is to minimize the *cost* that is defined as the *worst-case expected squared error*, i.e., $\max_{x \in [0,1]} \mathbb{E}[(\widehat{x} - x)^2]$. Note that the worst-case is taken over the value of x and the expectation is over the randomness of the algorithm. In the *unbiased* setting, we additionally require $\mathbb{E}[\widehat{x}] = x$, in which case the cost becomes $\mathrm{Var}[\widehat{x}]$, i.e., the estimation variance. In some cases, we allow the parties to use ℓ bits of *shared randomness*. That is, we assume that they have access to a random value $h \in [2^\ell]$, known to both Buffy and Angel. When applicable, we use $r \in [0, 1]$ to denote the private randomness of Buffy.

3 Algorithms without Shared Randomness

We recap the performance of two standard algorithms – randomized and deterministic rounding. Interestingly, we show that when no shared randomness is allowed, randomized rounding is an optimal unbiased algorithm, and deterministic rounding is an optimal biased algorithm.

3.1 Randomized Rounding

In randomized rounding, Buffy uses private randomness to generate $X \sim \mathrm{Bernoulli}(x)$ which is sent using a single bit. In turn, Angel estimates $\widehat{x} = X$. Clearly, we have that $\mathbb{E}[\widehat{x}] = \mathbb{E}[X] = x$, and thus the algorithm is unbiased. The variance of the algorithm is $\mathrm{Var}[\widehat{x}] = \mathrm{Var}[X] = x(1 - x)$, and thus the worst-case is reached at $x = 1/2$, which gives a cost of $1/4$. The following theorem, whose proof is deferred to full version [4],

shows that randomized rounding is optimal, in the sense that no unbiased algorithm without shared randomness can have a worst-case variance lower than 1/4. Intuitively, requiring the estimate to be unbiased forces the algorithm to send 1 with a probability that is linear in x, maximizing its cost for $x = 1/2$. The proof also establishes the intuitive idea that it is not possible to benefit from randomness used solely by Angel.

▶ **Theorem 1.** *Any unbiased algorithm without shared randomness must have a worst-case variance of at least* 1/4.

3.2 Deterministic Rounding

With deterministic rounding, Buffy sends $X = 1$ when $x \geq 1/2$. Angel then estimates $\widehat{x} = X/2 + 1/4$. Deterministic rounding has an (absolute) error of at most 1/4, which is achieved for $x \in \{0, 1/2, 1\}$. Therefore, its cost is 1/16. The next theorem, whose proof appears in Supplement 9.1, shows that deterministic rounding is optimal, as no algorithm that does not use shared randomness can have a lower cost (even with unrestricted private randomness). We show that any such algorithm must have an expected squared error of at least 1/16 on at least one of $\{0, 1/2, 1\}$.

▶ **Theorem 2.** *Any algorithm without shared randomness must have a worst-case expected squared error of at least* 1/16.

4 Lower Bounds

We next explore lower bounds for algorithms with shared randomness. We use Yao's minimax principle [24] to prove a lower bound on the cost of any biased shared randomness protocol. Then, we show a stronger lower bound for unbiased algorithms using a different approach.

4.1 Lower Bound for Biased Algorithms

We place Yao's general formulation in the context of our specific problem.

▶ **Theorem 3** ([24])**.** *Consider our estimation problem over the inputs $x \in [0, 1]$, and let $\mathcal{A}$ be the set of all possible* deterministic *algorithms. For a (deterministic) algorithm $a \in \mathcal{A}$ and input $x \in [0, 1]$, let the function $c(a, x) = (a(x) - x)^2$ be its squared error.*
Then for any randomized *algorithm A and input distribution $q \in \Delta([0, 1])$ such that $X \sim q$:*

$$\max_{x\in[0,1]} \mathbb{E}\left[c(A, x)\right] \geq \min_{a\in\mathcal{A}} \mathbb{E}\left[c(a, X)\right] \ .$$

That is, the expected *squared error (over the choice of x from distribution q) of the* best *deterministic algorithm (for q) lower bounds the* expected *squared error of any randomized (potentially biased) algorithm A for the* worst-case *x (i.e., its cost). Further, the inequality holds as an equality for the optimal distribution q and algorithm A, i.e.,*

$$\min_{A} \max_{x\in[0,1]} \mathbb{E}\left[c(A, x)\right] = \max_{q} \min_{a\in\mathcal{A}} \mathbb{E}\left[c(a, X)\right] \ .$$

We proceed by selecting distributions q to lower bound $\min_{a\in\mathcal{A}} \mathbb{E}\left[c(a, X)\right]$. Notice that a deterministic algorithm can be defined using two values $v_0, v_1 \in [0, 1]$, such that if $|x-v_0| \leq |x-v_1|$ then Buffy sends 0 and Angel estimates x as v_0. Similarly, if $|x-v_0| > |x-v_1|$ then Buffy sends 1 and Angel estimates x as v_1.[1] In general, the above framework asserts that the cost, for the worst-case input, of any randomized algorithm is

[1] Other deterministic algorithms, e.g., that send 0 despite having $|x - v_0| > |x - v_1|$, can trivially be improved by an algorithm with the above form.

$$\max_{q\in\Delta([0,1])}\min_{v_0,v_1\in[0,1]}\int_0^1 \min\left\{(x-v_0)^2,(x-v_1)^2\right\}q(x)dx\ . \tag{1}$$

Our framework lower bounds the cost for any (biased or unbiased) algorithm that may use any amount of (shared or private) randomness. We now consider distributions q to lower bound the cost and later discuss the limitations of this approach.

4.1.1 The $\{0, 1/2, 1\}$ case

First, consider a discrete probability distribution q over $\{0, 1/2, 1\}$, and assume without loss of generality that $q(0) \le q(1)$. Any deterministic algorithm cannot estimate all values exactly, and it must map at least two of the points to a single value, thus allowing us to lower bound its cost. In Supplement 9.2, we prove the following.

▶ **Lemma 4.** *Any deterministic algorithm must incur a cost of at least* $\frac{q(0)\cdot q(1/2)}{4(q(0)+q(1/2))}$.

For $q(0) = q(1) = (2-\sqrt{2})/2$ and $q(1/2) = \sqrt{2}-1$, this lemma yields a lower bound of $3/4 - 1/\sqrt{2} \approx 0.04289$. In Section 6.1, we show that this is *an optimal* lower bound when x is known to be in $\{0, 1/2, 1\}$, by giving an algorithm with a matching cost.

4.1.2 The $[0, 1]$ case

In the general case, where x can take on any value in $[0, 1]$, we can get a tighter bound by looking at mixed distributions. Specifically, for parameters $a, w \in [0, 1/2]$, we consider the distribution where:

$$q(x) = \begin{cases} 0 & \text{with probability } w \\ 1 & \text{with probability } w \\ \text{uniform on } [a, 1-a] & \text{otherwise} \end{cases}\ .$$

Directly analyzing the optimal deterministic algorithm for this distribution proves complex. Instead, we first *hypothesize* that there exists an optimal deterministic algorithm for which either (1) $v_1 = 1 - v_0$ or (2) $v_1 = 1$. We emphasize that the lower bound holds even if the hypothesis is false. We then analyze what values of a, w maximize the cost of the best deterministic algorithm with the above form. Finally, we verify that the lower bound for the resulting distribution (with the specific a, w values) holds *for all deterministic algorithms.*

For case (1), we can express the cost as $2\cdot\left(wv_0^2 + \frac{1/2-w}{1/2-a}\int_a^{1/2}(x-v_0)^2dx\right)$. Similarly, for case (2), we get a cost of $wv_0^2 + \frac{1-2w}{1-2a}\cdot\left(\int_a^{\min\{1-a,(v_0+1)/2\}}(x-v_0)^2dx + \int_{(v_0+1)/2}^{1-a}(x-1)^2dx\right)$. Therefore, the cost of the optimal algorithm from the above family is given as:

$$\min\left\{2\cdot\left(wv_0^2 + \frac{1/2-w}{1/2-a}\int_a^{1/2}(x-v_0)^2dx\right),\right.$$
$$\left. wv_0^2 + \frac{1-2w}{1-2a}\cdot\left(\int_a^{\min\{1-a,(v_0+1)/2\}}(x-v_0)^2dx + \int_{(v_0+1)/2}^{1-a}(x-1)^2dx\right)\right\}\ .$$

This cost is maximized for $a = \frac{-2w^2+w-2\sqrt{w(1-w)}+1}{4w^2-6w+2}$, where the value of w satisfies

$$32w^3 - 56w^2 + \sqrt{w(1-w)}\cdot(8w^4 - 24w^3 + 38w^2 - 8w - 7) + 24w = 0.$$

The resulting bound is slightly larger than 0.0459. Next, we verify that for these a, w values, no deterministic algorithm can achieve a lower cost. Specifically, instead of using $q(x)$ as described above, we generate a finite discrete distribution. For a parameter $n \in \mathbb{N}$, we define:

$$q_n(x) = \begin{cases} \frac{\lfloor n \cdot w \rfloor}{n} & \text{if } x \in \{0,1\} \\ \frac{1}{n} & \text{if } x \in \left\{ a + \frac{1-2a}{2(n-2\lfloor n \cdot w \rfloor))} + i \cdot \frac{1-2a}{n-2\lfloor n \cdot w \rfloor} \mid i \in [n - 2\lfloor n \cdot w \rfloor] \right\} \\ 0 & \text{otherwise} \end{cases} .$$

Note that $\lim_{n\to\infty} q_n(x) = q(x)$. We then find the optimal deterministic solution for this distribution by using a deterministic k-means clustering algorithm (for $k = 2$), that is guaranteed to converge, e.g., using [10]. The code that we used to obtain this result is available at [2]. The optimal deterministic algorithm for $q_n(x)$ tends to have either $v_1 = 1 - v_0$ or $v_1 = 1$ as hypothesized. Finally, for $n = 10^6$ we get a cost higher than 0.0459 which we use as a lower bound.

We do not believe that this bound is tight. Nonetheless, as we show in Section 6.2.4, our bound is within 0.2% of the optimum.

4.2 Lower Bound for Unbiased Algorithms - Beyond MiniMax

We now consider lower bounds for unbiased algorithms. Utilizing Yao's lemma does not appear to provide means to obtain sharper bounds when requiring the algorithm to be unbiased. We consider the case where $x \in \{0, 1/2, 1\}$ and directly prove that any unbiased algorithm must have a worst-case variance of at least $1/16$. This lower bound then also holds for $x \in [0, 1]$, although an improved bound of $\pi^2/64 - 1/12 \approx 0.07$ for this case, based on an average-case analysis, is presented in [7].

Assume that we have $h \in [0, 1]$. Buffy sends $X(x, h)$ to Angel, which determines an estimate $\widehat{x}(X(x, h), h)$. For $x', x'' \in \{0, 1/2, 1\}$, let $p_{x',x''} = \Pr[X(x', h) = X(x'', h)]$ denote the probability (with respect to h) that the same bit is sent for x', x''. Since we send a single bit, we have that $p_{0,1/2} + p_{1/2,1} + p_{0,1} \geq 1$. For all $x', x'' \in \{0, 1/2, 1\}$, we define $H_{x',x''} = \{h \in [0, 1] : X(x', h) = X(x'', h)\}$ to be the set of shared-randomness values that would lead Buffy to send the same bit for both x' and x''. Next, denote by $G_{x',x''} = \mathbb{E}[\widehat{x} | h \in H_{x',x''}, x \in \{x', x''\}]$ the expected estimate value, conditioned on the shared randomness being in $H_{x',x''}$. We have that:

$$\begin{aligned} \mathrm{Var}[\widehat{x}|x = 0] &\geq p_{0,1/2} \cdot (G_{0,1/2} - 0)^2 + p_{0,1} \cdot (G_{0,1} - 0)^2 + p_{1/2,1} \cdot (\mathbb{E}[\widehat{x}|h \in H_{1/2,1}, x = 0] - 0)^2 \\ \mathrm{Var}[\widehat{x}|x = 1/2] &\geq p_{0,1/2} \cdot (G_{0,1/2} - 1/2)^2 + p_{0,1} \cdot (\mathbb{E}[\widehat{x}|h \in H_{0,1}, x = 1/2] - 1/2)^2 \\ &\qquad + p_{1/2,1} \cdot (G_{1/2,1} - 1/2)^2 \\ \mathrm{Var}[\widehat{x}|x = 1] &\geq p_{0,1/2} \cdot (\mathbb{E}[\widehat{x}|h \in H_{0,1/2}, x = 1] - 1)^2 + p_{0,1} \cdot (G_{0,1} - 1)^2 + p_{1/2,1} \cdot (G_{1/2,1} - 1)^2. \end{aligned}$$

To proceed, we require the algorithm to be unbiased:

$$\begin{aligned} &G_{0,1/2}p_{0,1/2} + G_{0,1}p_{0,1} + \mathbb{E}[\widehat{x}|h \in H_{1/2,1}, x = 0]p_{1/2,1} = 0 \\ &G_{0,1/2}p_{0,1/2} + \mathbb{E}[\widehat{x}|h \in H_{0,1}, x = 1/2]p_{0,1} + G_{1/2,1}p_{1/2,1} = 1/2 \\ &\mathbb{E}[\widehat{x}|h \in H_{0,1/2}, x = 1]p_{0,1/2} + G_{0,1}p_{0,1} + G_{1/2,1}p_{1/2,1} = 1. \end{aligned}$$

This allows us to express the expectations $\{E[\widehat{x}|h \in H_{x',x''}, x = x'''] | x', x'', x''' \in \{0, 1/2, 1\}\}$ using $p_{x',x''}, G_{x',x''}$ and obtain a set of three inequalities with six variables. Our full analysis, given in the full version [4], proceeds with a case analysis based on the value of $p_{0,1}$, the probability that the sender would send the same bit for $0, 1$. We show that there exists an optimal algorithm in which $p_{0,1/2} + p_{1/2,1} + p_{0,1} = 1$, $p_{0,1/2} = p_{1/2,1}$, and $G_{1/2,1} = 1 - G_{0,1/2}$. This reduces the number of variables to three, allowing us to optimize the expression and show a lower bound of $1/16$ on any unbiased algorithm.

5 Algorithms with Unbounded Private Randomness

Here, we consider the case where the shared randomness is limited to ℓ bits, i.e., $h \in [2^\ell]$, but Buffy may use unbounded private randomness $r \sim U[0,1]$ (that is independent of h).

We present the following algorithm: Buffy sends X to Angel, where

$$X \triangleq \begin{cases} 1 & \text{if } x \geq (r+h)2^{-\ell} \\ 0 & \text{otherwise} \end{cases}.$$

Angel then estimates $\widehat{x} = X + (h - 0.5(2^\ell - 1)) \cdot 2^{-\ell}$.

We first show that our protocol is unbiased. It holds that $\mathbb{E}[h] = 0.5(2^\ell - 1)$ and $(r+h) \sim U[0, 2^\ell]$ (i.e., $(r+h)2^{-\ell} \sim U[0,1]$), and thus $\mathbb{E}[\widehat{x}] = \mathbb{E}[X] = x$.
We state theorem, whose proof appears in the full version [4], that bounds the variance:

▶ **Theorem 5.** *$Var[\widehat{x}] \leq 1/12 \cdot (1 - 4^{-\ell}) + 1/4 \cdot 4^{-\ell} = 1/6 \cdot (1/2 + 4^{-\ell})$.*

In Supplement 9.3, we describe a simple generalization of this algorithm, together with a lower bound, for sending $k > 1$ bits. We now explain the connection to subtractive dithering and explore the applicability of the algorithm for the $x \in \{0, 1/2, 1\}$ special case.

Connection to subtractive dithering. First invented for improving the visibility of quantized pictures [18], subtractive dithering aims to alleviate potential distortions that originate from quantization. Subtractive dithering was later extended for other domains such as speech [8], distributed deep learning [1], and federated learning [21].

In our setting, subtractive dithering corresponds to using shared randomness to add *noise* ς to x before applying a deterministic quantization and subtracting ς from the estimation. Specifically, let $\mathcal{Q} : [0,1] \to \{0,1\}$ be a two-level deterministic quantizer such that $\mathcal{Q}(g) = 1$ if $g \geq 1/2$ and 0 otherwise. Then, in subtractive dithering Buffy sends $X = \mathcal{Q}(x + \varsigma)$ and Angel estimates $\widehat{x} = X - \varsigma$.

There are several noise classes that ς can be drawn from, as classified in [19], that yield $\widehat{x} \sim U[x - 1/2, x + 1/2]$. For example, ς can be distributed uniformly on $[-1/2, 1/2]$.

Consider our algorithm of this section without restricting the number of random bits (i.e., $\ell \to \infty$, and rescale so $h \in U[0,1]$). This would yield the following algorithm:

$$X \triangleq \begin{cases} 1 & \text{if } x \geq h \\ 0 & \text{otherwise} \end{cases}$$

and $\widehat{x} = X + h - 0.5$. Similarly to subtractive dithering, we get that $\widehat{x} \sim U[x - 1/2, x + 1/2]$, as we prove in the full version [4] for completeness. To see that the two algorithms are equivalent (for $\varsigma \sim U[-1/2, 1/2]$), denote $h' = 1/2 - h$ (i.e., $h' \sim U[-1/2, 1/2]$). Then $X = 1$ if $x + h' \geq 1/2$ and $\widehat{x} = X - h'$.

Therefore, we conclude that our algorithm provides a spectrum between randomized rounding ($\ell = 0$) and a form of subtractive dithering ($\ell \to \infty$). In practice, this means that a small number of shared random bits yields a variance that is close to that of subtractive dithering ($\text{Var}[\widehat{x}] = 1/12$). For example, with a single shared random byte (i.e., $\ell = 8$), our algorithm has a worst-case variance that is within 0.02% of 1/12.

The $x \in \{0, 1/2, 1\}$ case. Notice that if x is known to be in $\{0, 1/2, 1\}$, then our ($\ell = 1$) algorithm gives $\text{Var}[\widehat{x}] = 1/16$, as evident from Theorem 5. Further, in this case, we do not require the private randomness as we can rewrite Buffy's algorithm as:

$$X \triangleq \begin{cases} 0 & \text{if } x = 0 \\ 1-h & \text{if } x = 1/2 \\ 1 & \text{if } x = 1 \end{cases}$$

while Angel estimates $\widehat{x} = X + (h - 0.5)/2$. This algorithm considerably improves over randomized rounding (which is optimal when no shared randomness is allowed, as shown in the full version [4]), that has a variance of $1/4$ for $x = 1/2$; i.e., a single shared random bit reduces the worst-case variance by a factor of 4. Further, it also improves over subtractive dithering, reducing the variance by a $4/3$ factor. Finally, this result is optimal according to the Section 4.2 lower bound, even if unbounded shared randomness is allowed.

6 Algorithms without Private Randomness

In some cases, generating random bits may be expensive, e.g., when running on power-constrained devices. This is particularly acute when the device operates in an energy harvesting mode [26]. Past works have even considered how to "recycle" random bits (e.g., [12]). Therefore, it is important to study how to design algorithms that use just a few random bits. To address this need, we consider scenarios where Buffy and Angel have access to a shared ℓ-bit random value h, but no private randomness.

One thing to notice is that Angel can produce at most $2^{\ell+1}$ different values since Angel is deterministic after obtaining the $\ell + 1$ bits of h and X. In particular, this means there is no unbiased protocol for general $x \in [0, 1]$. Therefore, we focus on biased algorithms and study how shared randomness allows improving over deterministic rounding (which is optimal without shared randomness, as we show in Section 3.2).

We start by proposing an optimal algorithm for the case where x is known to be in $\{0, 1/2, 1\}$. Then, we present adaptations of the subtractive dithering estimation method for the biased $x \in [0, 1]$ setting. These improve over both (unbiased) subtractive dithering and deterministic rounding. To the best of our knowledge, these adaptations are novel. Next, we show how Buffy can further reduce the cost while, among other changes, using a small number of shared random bits. We conclude by giving design principles for numerically approximating the optimal algorithm and give realizations for small number of shared random bits.

6.1 The $x \in \{0, 1/2, 1\}$ Case

We now consider the scenario where x is guaranteed to be in $\{0, 1/2, 1\}$ using a single shared randomness bit $h \in \{0, 1\}$. For some $\alpha \in [0, 1]$, Buffy sends

$$X = \begin{cases} 1 & \text{if } x = 1 \vee (x = 1/2 \wedge h = 0) \\ 0 & \text{otherwise} \end{cases}$$

while Angel estimates $\widehat{x} = \alpha \cdot h + (1 - \alpha) \cdot X$.

For example, this means that if $x = 0$, the squared error is 0 if $h = 0$ and α^2 otherwise. That is, the expected squared error is $\alpha^2/2$. We optimize over the α value to minimize the cost

$$\min_{\alpha \in [0,1]} \max \left\{ \alpha^2/2, (1 - (1 - \alpha))^2/2, \mathbb{E}\left[(1/2 - (\alpha \cdot h + (1 - \alpha) \cdot (1 - h)))^2 \right] \right\}.$$

This is optimized for $\alpha = 1 - 1/\sqrt{2}$, yielding a cost of $3/4 - 1/\sqrt{2} \approx 0.04289$, which is *optimal* according to our Section 4 lower bound, even if unbounded shared randomness is allowed.

6.2 The $x \in [0,1]$ Case

An important observation regarding optimal biased algorithms is that they, without loss of generality, can be expressed as a pair of monotone increasing functions $T, Z_0 : [0,1] \to [0,1]$ as follows. Here T is a threshold function that determines whether 0 or 1 is sent, Z_0 is the estimator when 0 is received, and $Z_1 : [0,1] \to [0,1]$, given by $Z_1(h) = 1 - Z_0(1-h)$, is the estimator when 1 is received. That is, Buffy sends

$$X = \begin{cases} 1 & \text{if } x \geq T(h) \\ 0 & \text{otherwise} \end{cases}.$$

In turn, Angel estimates $\widehat{x} = Z_X(h)$. We further explain this representation in Supplement 9.4. Based on this observation, we next lay out a sequence of algorithmic improvements over deterministic rounding that leverage the shared randomness to reduce the cost. We visualize the algorithms resulting from each improvement in Figure 1.

6.2.1 Subtractive dithering adaptations

As subtractive dithering provides the lowest cost (albeit using unbounded shared randomness) of the previously mentioned unbiased algorithms, one may wonder if it is possible to adapt it to the biased scenario. Accordingly, we first briefly overview two natural adjustments that use unbounded shared randomness and improve over the $1/16$ cost of deterministic rounding. We then propose improved protocols that reduce the cost further despite using only a small number (e.g., $\ell = 3$) of random bits.

Intuitively, subtractive dithering may produce estimates that are outside the $[0,1]$ range. Therefore, by *truncating* the estimates to $[0,1]$ one may only reduce the expected squared error for any $x \neq 1/2$. However, it does not reduce the expected squared error for $x = 1/2$, and thus the cost would remain $1/12$.

To reduce the cost, one may further truncate the estimates to $[z, 1-z]$ for some $z \in [0, 1/2]$. Indeed, we show in the full version [4] that this truncation reduces the cost to ≈ 0.0602, for z satisfying $1/24 + z^2/2 + (2z^3)/3 = 0$ ($z \approx 0.17349$).

A better adaptation strategy is obtained by changing the estimation to a linear combination of X and h. Specifically, consider the protocol where Buffy sends (for a shared $h \sim U[0,1]$)

$$X = \begin{cases} 1 & \text{if } x \geq h \\ 0 & \text{otherwise} \end{cases}$$

and Angel estimates, for some $\alpha \in [0,1]$, $\alpha \cdot h + (1-\alpha) \cdot X$. Optimizing the parameters, we show in the full version [4] that this algorithm achieves a cost of $5/3 - \phi \approx 0.04863$, which is obtained for $\alpha = 2 - \phi \approx 0.382$. Interestingly, this cost is achieved for *all* $x \in [0,1]$.

In Figure 1, we illustrate this algorithm. As shown, the subtractive dithering adaption has $T(h) = h$ and $Z_0(h) = \alpha \cdot h$. This means that Buffy sends $X = 1$ if $x \geq h$ and Angel estimates $\widehat{x} = \alpha \cdot h$ if $X = 0$ and $\widehat{x} = (1 - \alpha \cdot (1-h)) = (1-\alpha) + \alpha \cdot h$ otherwise.

6.2.2 Deterministically rounding extreme values

We now show how to leverage a finite number of shared random bits ℓ to design improved algorithms. As we show, it is possible to benefit from *deterministically* rounding values that are "close" to 0 or 1 and use the shared randomness otherwise.

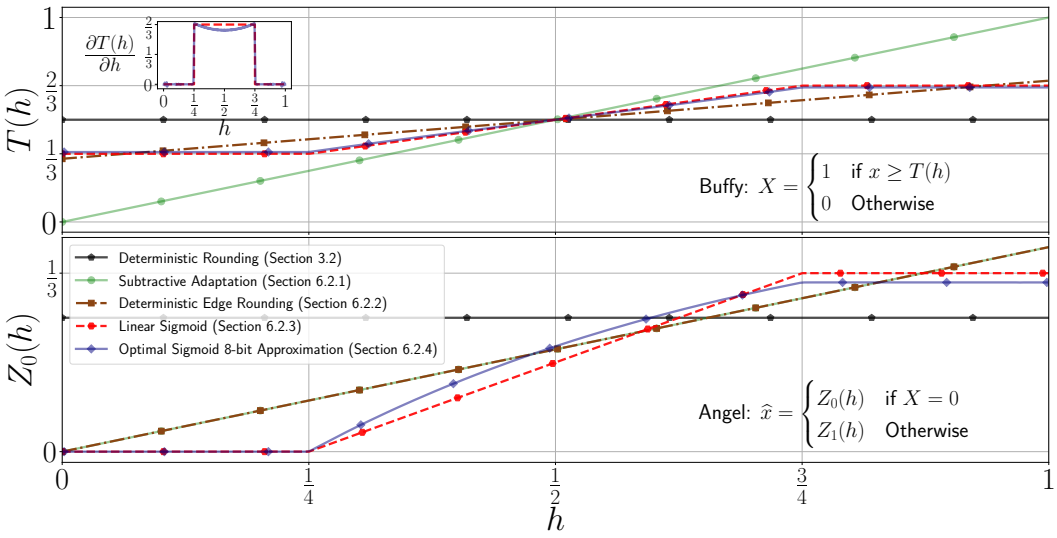


Figure 1 Illustration of the different biased algorithms. While deterministic rounding does not use the shared randomness and is thus constant, the other algorithms have both the threshold and estimation be monotone functions of h.

Similarly to the subtractive dithering adaptation above, Angel estimates x using a linear combination of h (with weight α) and X (with a weight of $1-\alpha$), where $\alpha \in [0,1]$ is chosen later. For all $i \in [2^\ell - 1]$, define the interval

$$\mathcal{I}_i = \left[(1-\alpha)/2 + i \cdot \frac{\alpha}{2^\ell - 1},\ (1-\alpha)/2 + (i+1) \cdot \frac{\alpha}{2^\ell - 1} \right). \tag{2}$$

In our algorithm, Buffy sends

$$X = \begin{cases} 0 & \text{if } x < (1-\alpha)/2 \\ \mathbb{1}_{h \leq i} & \text{if } x \in \mathcal{I}_i,\ i \in [2^\ell - 1] \\ 1 & \text{if } x \geq (1+\alpha)/2 \end{cases},$$

and Angel estimates: $\widehat{x} = \alpha \cdot h/(2^\ell - 1) + (1-\alpha) \cdot X$.

Note that we deterministically partition the range $[(1-\alpha)/2, (1+\alpha)/2]$ into $2^\ell - 1$ equally spaced intervals. Intuitively, the chosen intervals are designed to make the expected squared error a continuous function of x, as our analysis, given in the full version [4], indicates.

As we show, minimizing $cost = \min_\alpha \max_x \mathbb{E}[(\widehat{x} - x)^2]$ yields cumbersome expressions. For example, we get that with one shared random bit ($\ell = 1$), our algorithm has a cost of $1/18 \approx 0.05556$ (obtained for $\alpha = 1/3$, different than the α value used for the $x \in \{0, 1/2, 1\}$ case), lower than that of deterministic rounding (i.e., $1/16$). For $\ell = 2$, we obtain a cost of $\frac{259-140\sqrt{3}}{338} \approx 0.04885$ (reached for $\alpha = \frac{15-6\sqrt{3}}{13}$), and $\ell = 3$ bits further reduces the cost to $35/722 \approx 0.04848$ (when $\alpha = 7/19$). Additionally, with $\ell = 3$ bits, this improves over the subtractive dithering adaptions (that use unbounded shared randomness) for all $x \in [0,1]$. Notice that these costs are ≈21%, ≈6.4%, and ≈5.6% from the ≈ 0.0459 lower bound (see Section 4.1.2), and thus from the optimal algorithm. For completeness, we give the limiting algorithm (as $\ell \to \infty$) in the full version [4]. For intuition, we illustrate the limiting algorithm ($h \in [0,1]$) in Figure 1. As shown, we have $T(h) = \frac{1-\alpha}{2} + \alpha \cdot h$ (where $\alpha = 2 - \phi \approx 0.38$) and $Z_0(h) = \alpha \cdot h$. Observe that Angel uses the same estimation function as in Section 6.2.1,

but Buffy's threshold function is different. Intuitively, the new threshold function ensures that each x is mapped to the closest estimate value. For example, if $x = 0.1$ and $h = 0$, the subtractive adaptation would have $X = 1$ and thus $\widehat{x} = 1 - \alpha \approx 0.62$ while here we get $X = 0$ and $\widehat{x} = 0$.

Interestingly, the cost slightly and monotonically *increases* when increasing the number of bits ℓ beyond 3. This phenomenon suggests that we need more complex algorithms to leverage additional available random bits. We explore several approaches; in the full version [4], we show that by probabilistically selecting between the above algorithm (for $\ell \to \infty$) and the $\{0, 1/2, 1\}$ algorithm from Section 6.1, we can reduce the error to $\frac{6\sqrt{10}+11\sqrt{5}-18\sqrt{2}-17}{24} \approx 0.04644$. Intuitively, Buffy and Angel can implicitly agree on the chosen algorithm using the shared randomness. Here, we proceed by analyzing the potential benefits of non-uniform partitioning of the h values, which reduces the error further.

6.2.3 Non-uniform partitioning

Intuitively, the above algorithms have a threshold function that is linear in h; i.e., it takes the form $T(h) = a \cdot h + b$. We now show that this can be improved by looking at *sigmoid*-like functions. For ease of exposition, in this section, we consider $h \in [0, 1]$ to represent unbounded shared randomness, although the algorithm can be discretized given sufficient random bits. Recall from Section 6.2 that an algorithm can be expressed as a pair of functions $T, Z_0 : [0, 1] \to [0, 1]$ such that Buffy sends 1 if $x \geq T(h)$ while Angel estimates $Z_0(h)$ when receiving $X = 0$ and $Z_1(h) = 1 - Z_0(1 - h)$ otherwise. Here, we consider a *linear sigmoid* function (also illustrated in Figure 1), which, for some $h_0 \in [0, 1/2]$, is defined as

$$T(h) = \begin{cases} \alpha & \text{if } h < h_0 \\ \alpha + \frac{(1-2\alpha)(h-h_0)}{1-2h_0} & \text{if } h \in [h_0, 1-h_0] \\ 1 - \alpha & \text{otherwise} \end{cases}, \quad Z_0(h) = \begin{cases} 0 & \text{if } h < h_0 \\ \frac{(1-2\alpha)(h-h_0)}{1-2h_0} & \text{if } h \in [h_0, 1-h_0] \\ 1 - 2\alpha & \text{otherwise} \end{cases}.$$

Notice that in this algorithm we have $Z_0(h) = T(h) - \alpha$.

Our analysis, given in Supplement 9.5, shows that the cost is minimized for $h_0 = 1/4, \alpha = 1/3$, where the error is:

$$\mathbb{E}[(\widehat{x} - x)^2] = \mathbb{E}[(\widehat{x})^2] - 2x\mathbb{E}[\widehat{x}] + x^2 = \begin{cases} 5/108 - x/3 + x^2 & \text{if } x < 1/3 \\ 5/108 & \text{if } x \in [1/3, 2/3] \\ 77/108 - 5x/3 + x^2 & \text{otherwise} \end{cases}.$$

Therefore, the cost is $5/108 \approx 0.0463$, which is less than 0.9% higher than the 0.0459 lower bound (Section 4.1.2). The algorithm has two interesting properties. First, its expected squared error is constant for all $x \in \{0, 1\} \cup [1/3, 2/3]$ and, second, its expectation is not continuous as a function of x, as shown in Figure 2.

6.2.4 Towards the optimal algorithm

We now consider more general algorithms that have arbitrary estimate function Z_0. To that end, we use a numerical solver that approximates the optimal solution. Clearly, to define the input problem, we need to limit the number of variables and constraints. We achieve this using several observations:

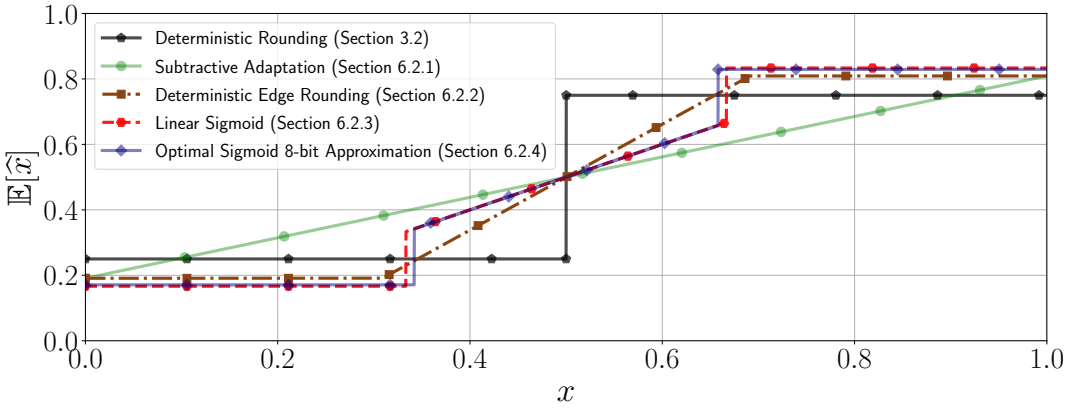


Figure 2 Illustration of the expectation of the different biased algorithms. Deterministic rounding does not use randomization and is therefore a step function, while others increase gradually in x. Notice that the expectations of the linear sigmoid and optimal approximation are not continuous.

- We consider bounded shared randomness $h \in [2^\ell]$ for $\ell \in \mathbb{N}$ bits. In fact, bounded shared randomness is precisely what allows us to develop this numerical approach.
- We use the observation that an optimal algorithm's T and Z_0 functions are not independent and satisfy $\forall h \in [0,1] : T(h) = \frac{Z_0(h)+Z_1(h)}{2} = \frac{Z_0(h)+(1-Z_0(1-h))}{2}$; this is because, that way, every $x \in [0,1]$ is estimated using the value closer to it between $Z_0(h)$ and $Z_1(h) = 1 - Z_0(1-h)$. In fact, the algorithms in sections 6.2.2-6.2.3 follow this rule, while the subtractive adaptation (Section 6.2.1) does not. As a result, we can define the variables $\{z_h | h \in [2^\ell]\}$ and derive the thresholds from the solver's output by $Z_0(h) = z_h$.
- For computing the maximal error for any $x \in [0,1]$, it is enough to look at a discrete set of points. This is because the number of possible estimates is $2^{\ell+1}$. Therefore, given two estimates $z_h, 1 - z_{2^\ell-1-h}$ that correspond to the values Angel uses given h and $X = 0$ or $X = 1$, the worst expected squared error (for this h) is obtained for $y_h \triangleq \frac{z_h+1-z_{2^\ell-1-h}}{2}$. Therefore, by checking all $x \in \{y_h \mid h \in [2^\ell]\}$, we can compute the cost.

Using these observations, we formulate the input as:

$$\begin{aligned} \underset{\{z_h|h\in[2^\ell]\}}{\text{minimize}} \quad & C \\ \text{subject to} \quad & C \geq \sum_{j=0}^{h} \left(y_h - (1 - z_{2^\ell-1-h})\right)^2 + \sum_{j=h+1}^{2^\ell-1} \left(y_h - z_h\right)^2, \quad h = 0, \ldots, 2^\ell - 1 \\ & y_h = \frac{z_h+1-z_{2^\ell-1-h}}{2}, \qquad z_h \in [0,1] \qquad\qquad h = 0, \ldots, 2^\ell - 1 \end{aligned}$$

In the above, we express the expected squared error at y_h by considering the h values for which $x \geq T(h)$ ($j \in [h]$) and those that $x < T(h)$. The output for the above problem does not seem to follow a compact representation. However, it is still possible to implement using a simple lookup table. For example, if $\ell = 8$, we can store all z_h when implementing Buffy and Angel. This algorithm's cost is lower than that of the linear sigmoid (that uses unbounded randomness) when using $\ell \geq 4$ bits. Specifically, using 4 shared random bits, the cost is ≈ 0.04611, while using 8 bits, it further reduces to ≈ 0.04599. Notice that these are less than 0.5% and 0.2% higher than the lower bound of Section 4.1.2. We note that this approach yields improvement even for a small number of shared random bits; for example, using $\ell = 1$ bit ($h \in \{0,1\}$), we get a cost of $1/20$ for $z_0 = 0.1, z_1 = 0.3$ which is equivalent to the following algorithm:

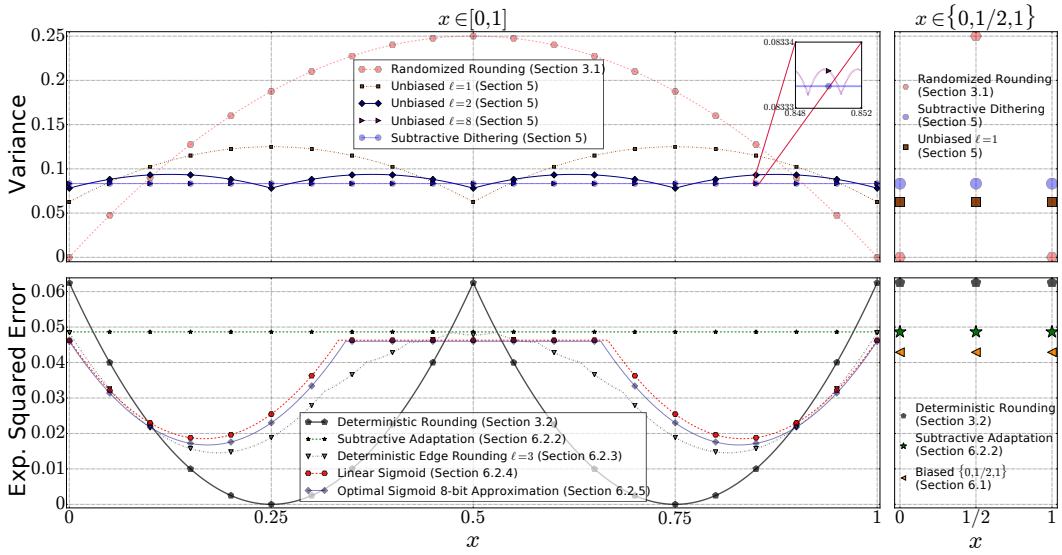


Figure 3 An illustration of the variance and expected squared error of the different algorithms. As shown, our unbiased algorithm is competitive with subtractive dithering despite using a single shared random byte, while our single-bit algorithm improves over subtractive on $\{0, 1/2, 1\}$. For the biased case, in addition to improving the $\{0, 1/2, 1\}$ case, our optimal sigmoid approximation algorithm achieves the lowest cost (less than 0.2% of the optimum!) while using a single shared random byte.

$$X = \begin{cases} 1 & \text{if } x \geq 0.4 + 0.2h \\ 0 & \text{otherwise} \end{cases}, \qquad \widehat{x} = 0.1 + 0.2h + 0.6X \ .$$

We visualize the resulting algorithm, for $\ell = 8$, in Figures 1 and 2. Notice that while the algorithm looks almost similar to our linear sigmoid, looking that the derivative $\frac{\partial T(h)}{\partial h}$ (Figure 1) shows that this optimal solution is not piece-wise linear.

7 Visual Comparison of the Algorithm Costs

We illustrate the various algorithms in Figure 3. In the unbiased case, notice how a single ($\ell = 1$) shared random bit significantly improves over randomized rounding (which is optimal when Buffy and Angel are restricted to private randomness). This further improves for larger ℓ values, where for $\ell = 8$ we have a cost that is only 0.02% higher than that of subtractive dithering, which uses unbounded shared randomness (the difference shown in zoom). When x is known to be in $\{0, 1/2, 1\}$ (right-hand side of the figure), it is evident how our unbiased $\ell = 1$ algorithm improves over both randomized rounding and subtractive dithering.

In the biased case, our adaptation to the subtractive dithering estimation (termed Subtractive Adaptation) improves over the cost of deterministic rounding. This is further improved by the algorithm of Section 6.2.2, termed Deterministic Edge Rounding, which is depicted using $\ell = 3$ bits as it minimizes its cost. Next, the Linear sigmoid (Section 6.2.3) shows how to lower the cost (using unbounded shared randomness) by non-uniform partitioning of the h values. Additionally, we show the optimal 8-bit algorithm (Section 6.2.4) that gets within 0.2% from the lower bound while using a single shared random byte. Finally, if x is known to be in $\{0, 1/2, 1\}$, our (optimal) biased $\{0, 1/2, 1\}$ algorithm improves over all other solutions while using only a single shared random bit.

8 Discussion

In this paper, we studied upper and lower bounds for the problem of sending a real number using a single bit. The goal is to minimize the cost, which is the worst-case variance for unbiased algorithms, or the worst-case expected squared error for biased ones. For all cases, we demonstrated how shared randomness helps to reduce the cost. Motivated by real-world applications, we derived algorithms with a bounded number of random bits that can be as low as a single shared bit. For example, in the unbiased case, using just one shared random bit reduces the variance two-fold compared to randomized rounding (which is optimal when no shared randomness is available). Further, using a single byte of shared randomness, our algorithm's variance is within 0.02% from the state of the art, which uses unbounded shared randomness. Our results are also near-optimal in the biased case, with a gap lower than 0.2% between the upper and lower bounds with a single shared random byte. Our upper bound is presented as a sequence of algorithms, each generalizing the previous while reducing the cost further.

For the special case where x is known to be in $\{0, 1/2, 1\}$, we give optimal unbiased and biased algorithms, together with matching lower bounds. Our algorithms use a single shared random bit, and the lower bounds show that the cost cannot be improved even when unbounded shared randomness is allowed.

We conclude by identifying directions for future research, beyond settling the correct bounds. First, our lower bounds apply for algorithms that use unbounded shared randomness, and new techniques for developing sharper bounds for other cases are of interest. Another direction is looking into optimizing the cost when sending k bits, for some $k > 1$. We make a first small step in Supplement 9.3, where we provide simple generalizations of our unbiased algorithm and lower bound to sending k bits. Finally, we are unclear on whether private randomness can help improve biased algorithms (see Table 1).

9 Supplementary Material

9.1 Optimality of Deterministic Rounding

We show that without shared randomness, deterministic rounding is an optimal biased solution. Notice that, in such a case, any protocol is defined by the probability of sending 1, denoted $Y(x)$, and the reconstruction distributions $V_0, V_1 \in \Delta([0,1])$.

Let us examine $\mathbb{E}[V_0]$ and $\mathbb{E}[V_1]$. We assume, without lost of generality, that $\mathbb{E}[V_0] \leq \mathbb{E}[V_1]$.

We have that:

$$\mathbb{E}[\widehat{x}] = Y(x)\mathbb{E}[V_1] + (1 - Y(x))\mathbb{E}[V_0].$$

That is, we have that *for any* $x \in [0,1]$: $\mathbb{E}[V_0] \leq \mathbb{E}[\widehat{x}] \leq \mathbb{E}[V_1]$. Next, we have that the cost, $\mathbb{E}[(\widehat{x} - x)^2]$, is bounded as

$$\mathbb{E}[(\widehat{x} - x)^2] \geq (\mathbb{E}[(\widehat{x} - x)])^2 .$$

In particular, for $x = 0$, we get that

$$\mathbb{E}[(\widehat{x} - x)^2 | x = 0] \geq (\mathbb{E}[\widehat{x} | x = 0])^2 \geq (\mathbb{E}[V_0])^2 .$$

Similarly, for $x = 1$, we have

$$\mathbb{E}[(\widehat{x} - x)^2 | x = 1] \geq (\mathbb{E}[(\widehat{x}) | x = 1] - 1)^2 \geq (1 - \mathbb{E}[V_1])^2 .$$

Notice that if $\mathbb{E}[V_0] \geq 0.25$ then $\mathbb{E}[(\widehat{x}-x)^2|x=0] \geq 1/16$, and similarly, if $\mathbb{E}[V_1] \leq 0.75$ then $\mathbb{E}[(\widehat{x}-x)^2|x=1] \geq 1/16$. Assume to the contrary that there exists an algorithm with a with a worst-case expected squared error lower than $1/16$, then we have $\mathbb{E}[V_0] \leq 0.25$ and $\mathbb{E}[V_1] \geq 0.75$. However, we have that $x = 0.5$ gives:

$$\begin{aligned}
\mathbb{E}[(\widehat{x}-x)^2|x=0.5] &= \mathbb{E}[\widehat{x}^2|x=0.5] - 2x\mathbb{E}[\widehat{x}|x=0.5] + 0.25 \\
&= Y(0.5)\mathbb{E}[V_1^2] + (1-Y(0.5))\mathbb{E}[V_0^2] - (Y(0.5)\mathbb{E}[V_1] + (1-Y(0.5))\mathbb{E}[V_0]) + 0.25 \\
&\geq Y(0.5)(\mathbb{E}[V_1])^2 + (1-Y(0.5))(\mathbb{E}[V_0])^2 - (Y(0.5)\mathbb{E}[V_1] + (1-Y(0.5))\mathbb{E}[V_0]) + 0.25 \\
&= Y(0.5)\cdot\mathbb{E}[V_1]\cdot(\mathbb{E}[V_1]-1) + (1-Y(0.5))\cdot\mathbb{E}[V_0]\cdot(\mathbb{E}[V_0]-1) + 0.25 \\
&\geq Y(0.5)\cdot 0.75\cdot(-0.25) + (1-Y(0.5))\cdot 0.25\cdot(-0.75) + 0.25 = 0.25 - 3/16 = 1/16.
\end{aligned}$$

In the first inequality, we used that fact that for any random variable V: $\mathbb{E}[V^2] \geq (\mathbb{E}[V])^2$, and in the second we used $\mathbb{E}[V_0] \leq 0.25$ and $\mathbb{E}[V_1] \geq 0.75$. This concludes the proof and establishes the optimality of deterministic rounding when no shared randomness is used.

9.2 Proof of the Biased $\{0, 1/2, 1\}$ Lower Bound

We recall Lemma 4:

▶ **Lemma 4.** *Any deterministic algorithm must incur a cost of at least* $\frac{q(0)\cdot q(1/2)}{4(q(0)+q(1/2))}$.

Proof. We denote by X_0 the set of values in $\{0, 1/2, 1\}$ that are closer to v_0 than to v_1. We assume without loss of generality that $v_0 \leq v_1$ and $q(0) \leq q(1)$ and prove that an optimal algorithm would set $v_0 = \frac{q(1/2)}{2(q(0)+q(1/2))}, v_1 = 1$, which incurs a cost of $\frac{q(0)\cdot q(1/2)}{4(q(0)+q(1/2))}$. Indeed, for this choice of v_0, v_1 we have that $X_0 = \{0, 1/2\}$, and we get a cost of

$$\begin{aligned}
q(0)\left(\frac{q(1/2)}{2(q(0)+q(1/2))}\right)^2 + q(1/2)\left(\frac{1}{2} - \frac{q(1/2)}{2(q(0)+q(1/2))}\right)^2 &= q(0)\left(\frac{q(1/2)}{2(q(0)+q(1/2))}\right)^2 \\
+ q(1/2)\left(\frac{q(0)}{2(q(0)+q(1/2))}\right)^2 &= \frac{q(0)q(1/2)^2 + q(1/2)q(0)^2}{4\left(q(0)+q(1/2)\right)^2} = \frac{q(0)\cdot q(1/2)}{4\left(q(0)+q(1/2)\right)}.
\end{aligned}$$

We now bound the performance of the optimal algorithm. We first notice that an optimal algorithm should have $0 \in X_0$ and $1 \notin X_0$. Next, notice that v_0 should be at most $1/2$ and v_1 should be at least $1/2$. Otherwise, one can improve the error for $x = 0$ or $x = 1$, respectively, without increasing the error at $1/2$. Further, observe that an optimal algorithm must have $v_0 = 0$ or $v_1 = 1$. That is because if $1/2 \in X_0$, we can reduce the error for $x = 1$ by setting $v_1 = 1$. Similarly, when $1/2 \notin X_0$, choosing $v_0 = 0$ decreases the error for $x = 0$. Now, we claim that there exists an optimal algorithm for which $v_1 = 1$. Consider some solution, and set $v_0' = 1 - v_1$ and $v_1' = 1$. This does not affect the error of $x = 1/2$, and does not increase the cost as $q(0) \leq q(1)$. We are left with choosing v_0; let us denote by $c(v_0) = q(0)v_0^2 + q(1/2)(1/2 - v_0)^2$ the resulting cost. This function has a minimum at $v_0 = \frac{q(1/2)}{2(q(0)+q(1/2))}$, which gives a cost of $\frac{q(0)\cdot q(1/2)}{4(q(0)+q(1/2))}$. ◀

This cost is maximized for $q(1/2) = \sqrt{2} - 1$ and $q(0) = q(1) = \frac{2-\sqrt{2}}{2}$, giving a lower bound of $3/4 - 1/\sqrt{2} \approx 0.04289$. In fact, one can verify that this is the best attainable lower bound for *any* discrete distribution on three points. Further, in Section 6.1, we show that this is *an optimal* lower bound when x is known to be in $\{0, 1/2, 1\}$, by giving an algorithm with a matching cost.

9.3 Generalization to k Bits

9.3.1 General Quantized Algorithm

We use a hash function h such that $h \in \{0,1\}^\ell$ is uniformly distributed. Let $A \sim U[0,1]$ be independent of h.

$$C = \lfloor (2^k - 1) \cdot x \rfloor$$

$$p = (2^k - 1) \cdot x - \lfloor (2^k - 1) \cdot x \rfloor$$

$$R = 2^k - 1$$

We then set

$$X \triangleq \begin{cases} C+1 & \text{if } p \geq (A+h)2^{-\ell} \\ C & \text{otherwise} \end{cases}$$

We send X to Angel which estimates

$$\widehat{x} = \frac{X + (h - 0.5(2^\ell - 1)) \cdot 2^{-\ell}}{R}.$$

To show that our protocol is unbiased, notice that: $\mathbb{E}[X] = R \cdot x$ and that $\mathbb{E}[h] = 0.5(2^\ell - 1)$.

9.3.2 Lower Bounds

Similarly to the 1-bit case, we consider the discrete distribution over

$$\left\{ i \cdot \left(\frac{1}{3 \cdot 2^{k-1} - 1} \right) \mid i \in \left\{0, 1, \ldots, 3 \cdot 2^{k-1} - 1\right\} \right\}.$$

We set $a_{1/2} = \frac{\sqrt{2}-1}{2^{k-1}}$ and $a_0 = a_1 = \frac{1 - a_{1/2}}{2^k}$ and

$$\forall i : q\left(i \cdot \left(\frac{1}{3 \cdot 2^{k-1} - 1} \right) \right) = a_{(i \mod 3)/2}.$$

When each consecutive set of three points has the same probability, one can derive an optimal algorithm with precisely two values between each such triplet. The optimal choice of locations of the values in each triplet is similar to our single-bit analysis of the previous subsection, i.e., one should have a values at

$$\left\{ \frac{\sqrt{2} - 1 + i}{2^{k-1}} \;\middle|\; i \in \left\{0, 1, \ldots, 2^{k-1} - 1\right\} \right\} \bigcup \left\{ \frac{i}{2^{k-1}} \middle|\; i \in \left\{1, \ldots, 2^{k-1}\right\} \right\}.$$

We turn into calculating the cost. Notice that every triplet has a width of $\frac{2}{3 \cdot 2^{k-1} - 1}$. Therefore, the cost now reduces, compared to the 1-bit analysis, by a factor of $\left(\frac{2}{3 \cdot 2^{k-1} - 1} \right)^2$. That is, we get a lower bound of $\frac{3 - 2\sqrt{2}}{(3 \cdot 2^{k-1} - 1)^2} = \frac{3 - 2\sqrt{2}}{2.25(2^k - 2/3)^2}$. We note that, for large k and ℓ values, our variance is within 10% of the lower bound, as

$$\lim_{k,\ell \to \infty} \frac{\frac{1}{12(2^k - 1)^2}}{\frac{3 - 2\sqrt{2}}{2.25(2^k - 2/3)^2}} = \frac{9 + 6\sqrt{2}}{16} \approx 1.093.$$

9.4 Reducing Algorithms to Monotone T, Z_0 Functions

We now show how an algorithm can be represented as described in Section 6.2. Fixing the shared randomness value h, Angel estimates $\widehat{x}$ solely based on the sent bit X; denote these values by $A_X(h)$. Without loss of generality, assume that $\forall h : A_1(h) \geq A_0(h)$.[2] This means that, in an optimal algorithm, Buffy should send $X = 1$ if $x \geq \frac{A_0(h)+A_1(h)}{2}$, as otherwise the error would be suboptimal for any x not satisfying the condition. In particular, this means that we can express Buffy's algorithm using a threshold function T.

Next, we claim that the threshold function can be considered monotone, without increasing the cost. To that end, we first consider a finite shared randomness $h \in [2^\ell]$. In such a case, if there exists some $h_1 > h_2 \in [2^\ell]$ such that $T(h_1) < T(h_2)$, we can modify the algorithm as follows: For all $h \notin \{h_1, h_2\}$, no modification is made. If $h = h_1$, then the modified algorithm works as if $h = h_2$, and vice versa. Following this process, we can sort T until it becomes monotone. A similar argument can be made for the continuous ($h \in [0, 1]$) case (possibly with an additional ϵ discretization cost).

We proceed with showing that there exists an optimal algorithm in which $Z_1(h) = 1 - Z_0(1 - h)$. This is achieved using a symmetry observation. Specifically, if an algorithm does not satisfy the above, consider its "dual algorithm": instead of sending x using $T(h)$, we send $x' = 1 - x$ using $T'(h) = 1 - T(1 - h)$; similarly, Angel estimates $\widehat{x}' = 1 - \widehat{x}$. Then, if both Buffy and Angel use the shared randomness to implicitly agree on whether to run the original or dual algorithms, each with probability half, the cost can only decrease. Additional details are given in the full version [4].

9.5 Analysis of the Linear Sigmoid (Section 6.2.3)

First, we have (see Section 6.2) that the estimate function for $X = 1$ is:

$$Z_1(h) = 1 - Z_0(1 - h) = \begin{cases} 2\alpha & \text{if } h < h_0 \\ 2\alpha + (1 - 2\alpha) \cdot \frac{h - h_0}{1 - 2h_0} & \text{if } h \in [h_0, 1 - h_0] \\ 1 & \text{otherwise} \end{cases}.$$

For $x \in [\alpha, 1 - \alpha]$, denote by $T^{-1}(x) = \frac{(1-2h_0)(x-\alpha)}{1-2\alpha} + h_0$ the value such that $T(T^{-1}(x)) = x$. We proceed with computing the expectation:

$$\mathbb{E}[\widehat{x}|x < \alpha](\implies X = 0) = h_0 \cdot (1 - 2\alpha) + \int_{h_0}^{1-h_0} \left((1 - 2\alpha) \cdot \frac{h - h_0}{1 - 2h_0} \right) dh = 1/2 - \alpha$$

$$\begin{aligned} \mathbb{E}[\widehat{x}|x > 1 - \alpha](\implies X = 1) &= h_0 \cdot (1 + 2\alpha) + \int_{h_0}^{1-h_0} \left(2\alpha + (1 - 2\alpha) \cdot \frac{h - h_0}{1 - 2h_0} \right) dh \\ &= 1/2 + \alpha \end{aligned}$$

[2] If for some h, $A_0(h) > A_1(h)$, there exists an equivalent algorithm that replaces the role of $X = 0$ and $X = 1$ for this specific h.

$$\begin{aligned}\mathbb{E}[\widehat{x}|x\in[\alpha,1-\alpha]] &= h_0\cdot(2\alpha)+h_0(1-2\alpha)\\ &\quad+\int_{h_0}^{T^{-1}(x)}\left(2\alpha+(1-2\alpha)\cdot\frac{h-h_0}{1-2h_0}\right)dh\\ &\quad+\int_{T^{-1}(x)}^{1-h_0}\left((1-2\alpha)\cdot\frac{h-h_0}{1-2h_0}\right)dh\\ &= h_0+2\alpha(T^{-1}(x)-h_0)+\int_{h_0}^{1-h_0}\left((1-2\alpha)\cdot\frac{h-h_0}{1-2h_0}\right)dh\\ &= 1/2+(-1+2h_0)\alpha+2\alpha(T^{-1}(x)-h_0)=1/2-\alpha+2\alpha T^{-1}(x)\end{aligned}$$

Therefore, we have:

$$\mathbb{E}[\widehat{x}]=\begin{cases}1/2-\alpha & \text{if } x<\alpha\\ 1/2-\alpha+2\alpha\left(\frac{(1-2h_0)(x-\alpha)}{1-2\alpha}+h_0\right) & \text{if } x\in[\alpha,1-\alpha]\\ 1/2+\alpha & \text{otherwise}\end{cases}.$$

Next, we calculate the second moment of the estimate:

$$\begin{aligned}\mathbb{E}[(\widehat{x})^2|x<\alpha](\implies X=0) &= h_0\cdot(1-2\alpha)^2+\int_{h_0}^{1-h_0}\left((1-2\alpha)\cdot\frac{h-h_0}{1-2h_0}\right)^2dh\\ &= 1/3+h_0/3-4\alpha/3-4h_0\alpha/3+4\alpha^2/3+4h_0\alpha^2/3\end{aligned}$$

$$\begin{aligned}\mathbb{E}[(\widehat{x})^2|x>1-\alpha](\implies X=1) &= h_0\cdot(1+(2\alpha)^2)\\ &\quad+\int_{h_0}^{1-h_0}\left(2\alpha+(1-2\alpha)\cdot\frac{h-h_0}{1-2h_0}\right)^2dh\\ &= 1/3+h_0/3+2\alpha/3-4h_0\alpha/3+4\alpha^2/3+4h_0\alpha^2/3\end{aligned}$$

$$\begin{aligned}\mathbb{E}[(\widehat{x})^2|x\in[\alpha,1-\alpha]] &= h_0\cdot((1-2\alpha)^2+(2\alpha)^2)\\ &\quad+\int_{h_0}^{T^{-1}(x)}\left(2\alpha+(1-2\alpha)\cdot\frac{h-h_0}{1-2h_0}\right)^2dh+\int_{T^{-1}(x)}^{1-h_0}\left((1-2\alpha)\cdot\frac{h-h_0}{1-2h_0}\right)^2dh\\ &= h_0\cdot((1-2\alpha)^2+(2\alpha)^2)+\frac{(1-2h_0)(x-\alpha)(x^2+4x\alpha+7\alpha^2)}{3-6\alpha}\\ &\quad+\frac{(2h_0-1)(-1+x+\alpha)(1+x+x^2-4x\alpha+\alpha(-5+7\alpha))}{3-6\alpha}\\ &= \frac{1-6\alpha-6\alpha x^2(-1+2h_0)+12\alpha^2-14\alpha^3+h_0(1-6\alpha+24\alpha^2-20\alpha^3)}{3-6\alpha}\end{aligned}$$

Putting it together, we get:

$$\mathbb{E}[\widehat{x}^2]=\begin{cases}1/3+h_0/3-4\alpha/3-4h_0\alpha/3+4\alpha^2/3+4h_0\alpha^2/3 & \text{if } x<\alpha\\ \frac{1-6\alpha-6\alpha x^2(-1+2h_0)+12\alpha^2-14\alpha^3+h_0(1-6\alpha+24\alpha^2-20\alpha^3)}{3-6\alpha} & \text{if } x\in[\alpha,1-\alpha]\\ 1/3+h_0/3+2\alpha/3-4h_0\alpha/3+4\alpha^2/3+4h_0\alpha^2/3 & \text{otherwise}\end{cases}.$$

By solving

$$\min_{h_0\in[0,1/2],\alpha\in[0,1/2]}\max_{x\in[0,1]}\mathbb{E}[(\widehat{x}-x)^2]=\min_{h_0\in[0,1/2],\alpha\in[0,1/2]}\max_{x\in[0,1]}\mathbb{E}[\widehat{x}^2]-2x\mathbb{E}[\widehat{x}]+x^2,$$

we get that the algorithm is optimized for $\alpha=1/3$ and $h_0=1/4$, where the resulting cost is:

$$\mathbb{E}[(\widehat{x}-x)^2] = \mathbb{E}[(\widehat{x})^2] - 2x\mathbb{E}[\widehat{x}] + x^2$$
$$= \begin{cases} 5/108 - x/3 + x^2 & \text{if } x < 1/3 \\ 5/108 & \text{if } x \in [1/3, 2/3] \\ 77/108 - 5x/3 + x^2 & \text{otherwise} \end{cases}.$$

Therefore, the cost is $5/108 \approx 0.0463$, which is less than 0.9% higher than the 0.0459 lower bound (Section 4.1.2).

References

1. Afshin Abdi and Faramarz Fekri. Indirect Stochastic Gradient Quantization and Its Application in Distributed Deep Learning. In *AAAI*, 2020.
2. Ran Ben Basat, Michael Mitzenmacher, and Shay Vargaftik. Biased [0,1] proof. URL: `https://gist.github.com/ranbenbasat/9959d1c70471fe870424eefbecd3e13c`.
3. Ran Ben-Basat, Gil Einziger, Isaac Keslassy, Ariel Orda, Shay Vargaftik, and Erez Waisbard. Memento: Making Sliding Windows Efficient for Heavy Hitters. In *ACM CoNEXT*, 2018.
4. Ran Ben-Basat, Michael Mitzenmacher, and Shay Vargaftik. How to Send a Real Number Using a Single Bit (and Some Shared Randomness). *CoRR*, abs/2010.02331, 2020. `arXiv:2010.02331`.
5. Ran Ben Basat, Sivaramakrishnan Ramanathan, Yuliang Li, Gianni Antichi, Minian Yu, and Michael Mitzenmacher. PINT: Probabilistic In-band Network Telemetry. In *Proceedings of the Annual conference of the ACM Special Interest Group on Data Communication on the applications, technologies, architectures, and protocols for computer communication*, pages 662–680, 2020.
6. Jeremy Bernstein, Yu-Xiang Wang, Kamyar Azizzadenesheli, and Animashree Anandkumar. signSGD: Compressed Optimisation for Non-Convex Problems. In *International Conference on Machine Learning*, pages 560–569, 2018.
7. Zarathustra Elessar Brady (`https://cstheory.stackexchange.com/users/50608/zeb`). Is Subtractive Dithering the Optimal Algorithm for Sending a Real Number Using One Bit? URL: `https://cstheory.stackexchange.com/questions/48281`.
8. MR Garey, David Johnson, and Hans Witsenhausen. The complexity of the generalized Lloyd-Max problem (Corresp.). *IEEE Transactions on Information Theory*, 28(2):255–256, 1982.
9. Robert M. Gray and David L. Neuhoff. Quantization. *IEEE transactions on information theory*, 44(6):2325–2383, 1998.
10. Allan Grønlund, Kasper Green Larsen, Alexander Mathiasen, Jesper Sindahl Nielsen, Stefan Schneider, and Mingzhou Song. Fast Exact K-Means, K-Medians and Bregman Divergence Clustering in 1D. *arXiv preprint*, 2017. `arXiv:1701.07204`.
11. Rob Harrison, Qizhe Cai, Arpit Gupta, and Jennifer Rexford. Network-Wide Heavy Hitter Detection with Commodity Switches. In *Proceedings of the Symposium on SDN Research*, pages 1–7, 2018.
12. Russell Impagliazzo and David Zuckerman. How to Recycle Random Bits. In *FOCS*, volume 30, pages 248–253, 1989.
13. Peter Kairouz, H. Brendan McMahan, Brendan Avent, Aurélien Bellet, Mehdi Bennis, Arjun Nitin Bhagoji, Keith Bonawitz, Zachary Charles, Graham Cormode, Rachel Cummings, Rafael G. L. D'Oliveira, Salim El Rouayheb, David Evans, Josh Gardner, Zachary Garrett, Adrià Gascón, Badih Ghazi, Phillip B. Gibbons, Marco Gruteser, Zaid Harchaoui, Chaoyang He, Lie He, Zhouyuan Huo, Ben Hutchinson, Justin Hsu, Martin Jaggi, Tara Javidi, Gauri Joshi, Mikhail Khodak, Jakub Konečný, Aleksandra Korolova, Farinaz Koushanfar, Sanmi Koyejo, Tancrède Lepoint, Yang Liu, Prateek Mittal, Mehryar Mohri, Richard Nock, Ayfer Özgür, Rasmus Pagh, Mariana Raykova, Hang Qi, Daniel Ramage, Ramesh Raskar, Dawn Song, Weikang Song, Sebastian U. Stich, Ziteng Sun, Ananda Theertha Suresh, Florian Tramèr, Praneeth Vepakomma, Jianyu Wang, Li Xiong, Zheng Xu, Qiang Yang, Felix X. Yu, Han Yu, and Sen Zhao. Advances and Open Problems in Federated Learning, 2019. `arXiv:1912.04977`.

14 Sai Praneeth Karimireddy, Quentin Rebjock, Sebastian Stich, and Martin Jaggi. Error Feedback Fixes SignSGD and other Gradient Compression Schemes. In *International Conference on Machine Learning*, pages 3252–3261, 2019.

15 Jakub Konečnỳ, Brendan McMahan, and Daniel Ramage. Federated Optimization: Distributed Optimization Beyond the Datacenter. *arXiv preprint*, 2015. `arXiv:1511.03575`.

16 Michael Mitzenmacher. Queues with Small Advice. *arXiv preprint*, 2020. `arXiv:2006.15463`.

17 Ilan Newman. Private vs. Common Random bits in Communication Complexity. *Information processing letters*, 39(2):67–71, 1991.

18 Lawrence Roberts. Picture Coding Using Pseudo-Random Noise. *IRE Transactions on Information Theory*, 8(2):145–154, 1962.

19 Leonard Schuchman. Dither Signals and Their Effect on Quantization Noise. *IEEE Transactions on Communication Technology*, 12(4):162–165, 1964.

20 Frank Seide, Hao Fu, Jasha Droppo, Gang Li, and Dong Yu. 1-Bit Stochastic Gradient Descent and its Application to Data-Parallel Distributed Training of Speech DNNs. In *Fifteenth Annual Conference of the International Speech Communication Association*, 2014.

21 Nir Shlezinger, Mingzhe Chen, Yonina C Eldar, H Vincent Poor, and Shuguang Cui. UVeQFed: Universal Vector Quantization for Federated Learning. *arXiv preprint*, 2020. `arXiv:2006.03262`.

22 Shay Vargaftik, Isaac Keslassy, and Ariel Orda. LSQ: Load Balancing In Large-Scale Heterogeneous Systems With Multiple Dispatchers. *IEEE/ACM Transactions on Networking*, 28(3):1186–1198, 2020.

23 Wei Wen, Cong Xu, Feng Yan, Chunpeng Wu, Yandan Wang, Yiran Chen, and Hai Li. Terngrad: Ternary Gradients to Reduce Communication in Distributed Deep Learning. In *Advances in neural information processing systems*, pages 1509–1519, 2017.

24 Andrew Chi-Chin Yao. Probabilistic Computations: Toward a Unified Measure of Complexity. In *IEEE FOCS*, 1977.

25 Guoxia Yu, Tanya Vladimirova, and Martin N Sweeting. Image Compression Systems on Board Satellites. *Acta Astronautica*, 64(9-10):988–1005, 2009.

26 Pengyu Zhang and Deepak Ganesan. Enabling Bit-by-Bit Backscatter Communication in Severe Energy Harvesting Environments. In *11th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 14)*, pages 345–357, 2014.

Using a Geometric Lens to Find k Disjoint Shortest Paths

Matthias Bentert
Faculty IV, Algorithmics and Computational Complexity, Technische Universität Berlin, Germany

André Nichterlein
Faculty IV, Algorithmics and Computational Complexity, Technische Universität Berlin, Germany

Malte Renken
Faculty IV, Algorithmics and Computational Complexity, Technische Universität Berlin, Germany

Philipp Zschoche
Faculty IV, Algorithmics and Computational Complexity, Technische Universität Berlin, Germany

Abstract

Given an undirected n-vertex graph and k pairs of terminal vertices $(s_1, t_1), \ldots, (s_k, t_k)$, the k-Disjoint Shortest Paths (k-DSP) problem asks whether there are k pairwise vertex-disjoint paths $P_1, \ldots, P_k$ such that P_i is a shortest s_i-t_i-path for each $i \in [k]$. Recently, Lochet [SODA '21] provided an algorithm that solves k-DSP in $n^{O\left(k^{5^k}\right)}$ time, answering a 20-year old question about the computational complexity of k-DSP for constant k. On the one hand, we present an improved $O(kn^{16k \cdot k! + k + 1})$-time algorithm based on a novel geometric view on this problem. For the special case $k = 2$, we show that the running time can be further reduced to $O(nm)$ by small modifications of the algorithm and a further refined analysis. On the other hand, we show that k-DSP is W[1]-hard with respect to k, showing that the dependency of the degree of the polynomial running time on the parameter k is presumably unavoidable.

2012 ACM Subject Classification Theory of computation → Graph algorithms analysis

Keywords and phrases graph algorithms, dynamic programming, W[1]-hardness, geometry

Digital Object Identifier 10.4230/LIPIcs.ICALP.2021.26

Category Track A: Algorithms, Complexity and Games

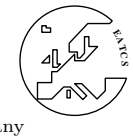
Related Version *Full Version*: `https://arxiv.org/abs/2007.12502`

Funding This work was started at the research retreat of the group *Algorithmics and Computational Complexity* in September 2019, held in Schloss Neuhausen (Prignitz, Germany).
Malte Renken: Supported by the DFG project MATE (NI 369/17).

1 Introduction

The k-Disjoint Paths problem is a fundamental and well-studied combinatorial problem. Given an undirected graph G and k terminal pairs $(s_i, t_i)_{i \in [k]}$, the question is whether there are pairwise disjoint [1] s_i-t_i-paths P_i for each $i \in [k]$. The problem was shown to be NP-hard by Karp [9] when k is part of the input. On the positive side, Robertson and Seymour [13] provided an algorithm running in $O(n^3)$ time for any constant k. Later, Kawarabayashi et al. [10] improved the running time to $O(n^2)$, again for fixed k. On directed graphs, in contrast, the problem is NP-hard even for $k = 2$ [7]. However, on directed acyclic graphs, the problem becomes again polynomial-time solvable for constant k [7] and linear-time solvable for $k = 2$ [14].

[1] We only consider the vertex-disjoint setting to which the edge-disjoint version can be reduced.



Focusing on the undirected case, we study the problem variant where all paths in the solution have to be shortest paths. This variant was introduced by Eilam-Tzoreff [5].

k Disjoint Shortest Paths (k-DSP)

Input: A graph $G = (V, E)$ and $k \in \mathbb{N}$ pairs $(s_i, t_i)_{i \in [k]}$ of vertices.

Task: Find k disjoint paths P_i such that P_i is a shortest s_i-t_i-path for each $i \in [k]$.

Throughout the whole paper, we assume that the input graph is connected. Eilam-Tzoreff [5] showed the NP-hardness of k-DSP when k is part of the input. Moreover, Eilam-Tzoreff provided a dynamic-programming based $O(n^8)$-time algorithm for 2-DSP. This algorithm for 2-DSP works also for positive edge lengths. Recently, Gottschau et al. [8] and Kobayashi and Sako [11] independently extended this result by providing polynomial-time algorithms for the case that the edge lengths are non-negative. As for directed graphs, Berczi and Kobayashi [2] provided a polynomial-time algorithm for strictly positive edge length. Very recently, Akhmedov [1] presented an algorithm solving 2-DSP in $O(n^6)$ time for unweighted graphs and in $O(n^7)$ for strictly positive edge lengths. Note that allowing zero-length edges generalizes 2-Disjoint Path on directed graphs, which is NP-hard [7]. Extending the problem to finding two disjoint s_i-t_i-paths of minimal total length (in undirected graphs), Björklund and Husfeldt [4] provided a randomized algorithm with running time $O(n^{11})$.

The existing algorithms for 2-DSP are based on dynamic programming with tedious case distinctions. We provide a new algorithm using a simple and elegant geometric perspective:

▶ **Theorem 1.** *2-DSP can be solved in $O(nm)$ time.*

Whether or not k-DSP for constant $k \geq 3$ is polynomial-time solvable was posed as a research challenge [6, open problem 4.6]. Recently, Lochet [12] settled this long standing open question by showing that k-DSP can be solved in $n^{O(k^{5^k})}$ time. Using our geometric approach, we provide an improved $O(k \cdot n^{16k \cdot k! + k + 1})$-time algorithm.

▶ **Theorem 2.** *k-DSP can be solved in $O(k \cdot n^{16k \cdot k! + k + 1})$ time.*

We describe the basic idea of our algorithms and the new geometric tools in Section 2. In Section 3, we formalize these geometric tools for two paths and prove Theorem 1. In Section 4, we lift these arguments to $k > 2$ paths. In Section 5, we present our algorithm for Theorem 2 and prove its correctness.

Finally, we show that k-DSP is W[1]-hard with respect to k. Hence, under standard assumptions from parameterized complexity, there is no algorithm with running time $f(k)n^{O(1)}$ for any function f. Thus, polynomial-time algorithms where k does not appear in the exponent (as the $O(n^2)$-time algorithm for k-Disjoint Path for any constant k [13, 10]) are unlikely to exist for k-DSP. Furthermore, under the Exponential-Time Hypothesis (ETH), we show that there is no algorithm with running time $f(k) \cdot n^{o(k)}$ for any computable function f.

▶ **Proposition 3** ($\star$[2])**.** *k-DSP is W[1]-hard with respect to k. Moreover, assuming ETH, there is no $f(k) \cdot n^{o(k)}$-time algorithm for k-DSP.*

Preliminaries. We set $\mathbb{N} := \{0, 1, 2, \dots, \}$ and $[n] := \{1, 2, \dots, n\}$. We always denote by $G = (V, E)$ a graph (undirected and connected unless said otherwise) and by n and m the number of vertices and edges in G, respectively. A *path* of length $\ell \geq 0$ in a graph G is a

[2] Some proofs (marked with a $\star$) are deferred to a full version.

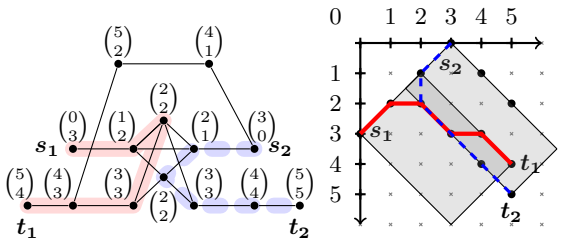


Figure 1 *Left side:* A graph with distinguished vertices s_1, s_2, t_1, t_2; vertex coordinates written next to the vertices. Two shortest paths are highlighted. *Right side:* The 2D-arrangement of the vertices. The two gray rectangles spanned by s_1 and t_1 and by s_2 and t_2 contain the two shortest paths s_1-t_1 (solid red path) and s_2-t_2 (dashed blue path).

sequence of distinct vertices $v_0v_1 \dots v_\ell$ such that each pair v_{i-1}, v_i is connected by an edge in G. The first and last vertex v_0 and v_ℓ are called the *end vertices* or *ends* of P and are denoted by s_P and t_P. We also say that P is a path *from* v_0 *to* v_ℓ, a path *between* v_0 and v_ℓ, or a v_0-v_ℓ-path. When no ambiguity arises, we do not distinguish between a path and its set of vertices. For $v, w \in P$, we denote by $P[v, w]$ the subpath of P with end vertices v and w. For two vertices v, w, we denote the length of a shortest v-w-path in G by $\mathrm{dist}_G(v, w)$ or $\mathrm{dist}(v, w)$ if the graph G is clear from the context. If for all $i \in [k]$ there is a path P_i that is a shortest s_i-t_i-path and disjoint with P_j for all $j \in [k] \setminus \{i\}$, then we say that the paths $(P_i)_{i\in[k]}$ are a solution for an instance $(G, (s_i, t_i)_{i\in[k]})$ of k-DSP.

2 The Key Concepts behind our Polynomial-Time Algorithm

In this section, we describe our approach to solve k-DSP in polynomial-time for any fixed k. As a warm-up, we start with sketching an algorithm for 2-DSP based on the same approach.

Solving 2-DSP in the plane. Before describing the algorithm, we show the central geometric idea behind it. Recall that we want to find two shortest paths P_1 and P_2 from s_1 to t_1 and from s_2 to t_2, respectively. We now arrange the vertices on a 2-dimensional grid where the first coordinate of each vertex is the distance to s_1 and the second coordinate the distance to s_2; see left side of Figure 1 for an example graph with the corresponding coordinates and the right side for an arrangement of the vertices in a grid with a continuous drawing of the paths (drawing straight lines between points occurring in the paths). Clearly, with two breadth-first searches from s_1 and s_2 we can compute the coordinates of all vertices in linear time. Note that there might be multiple vertices with the same coordinates. However, at most one vertex per coordinate can be part of a shortest s_1-t_1- or s_2-t_2-path.

This arrangement of the vertices allows the following simple geometric observation: The drawing of each shortest s_1-t_1-path has to be within a rectangle with angles of 45° to the coordinate axes and with corner points s_1 and t_1 (see gray rectangles in right side of Figure 1). As a consequence, shortest paths that have to stay within two disjoint such rectangles cannot intersect. We use this argument extensively in the subsequently sketched algorithm.

We assume that the drawings of P_1 and P_2 cross as displayed in the right side of Figure 1 (the non-crossing case is easier to deal with). Our algorithm solving 2-DSP in this case is as follows: We distinguish whether the intersection of the drawings of P_1 and P_2 contain a point with integer coordinates (that is, our algorithm tries both possibilities).

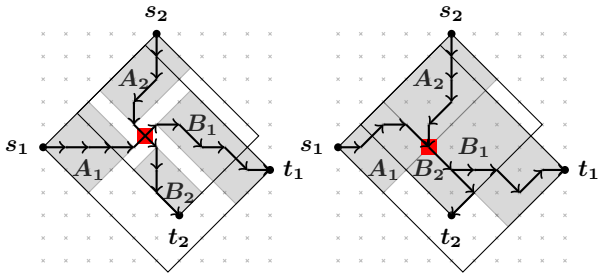


Figure 2 An illustration of the directed acyclic graph used for Theorem 1. The first point p in the intersection of the drawing of P_1 and P_2 is marked by a red square. *Left side:* Straigh-line drawings of P_1 and P_2 intersect, but not in points with integer coordinates. *Right side:* Both P_1 and P_2 use vertices with same coordinates.

If the intersection does not have a point with integer coordinates, then it is easy to see that the intersection of the drawing of P_1 and P_2 has to be a single point p (with non-integer coordinates). We guess[3] the four coordinate pairs $(x, y), (x, y+1), (x+1, y)$, and $(x+1, y+1)$, $x, y \in \mathbb{N}$, surrounding the intersection point of the drawings of P_1 and P_2. Note that this can be done in $O(n)$ time by guessing the vertex on the coordinate pair (x, y). The goal is now to turn the input graph G into a directed acyclic graph D such that each shortest s_i-t_i-path in G corresponds to an s_i-t_i-path in D. To this end, we partition the grid into four areas A_1, B_1, A_2, B_2 (each area is defined by one of the guessed points and the closer endpoint of the path going through the point) and orient each edge according to the area it lies in (see left side of Figure 2 for an illustration). An edge $\{v, w\}$ in the area A_i or B_i, $i \in [2]$, is oriented towards the vertex w with the larger i coordinate. Edges between A_i and B_i are oriented towards the vertex in B_i. All remaining (unoriented) edges are removed. Note that this results in a directed acyclic graph. Furthermore, a shortest s_i-t_i-path in G induces an s_i-t_i-path in D and each s_i-t_i-path in D is a shortest path in G because it is strictly monotone increasing in the i-coordinate and all strictly monotone increasing paths have the same length as each path contains one vertex for each integer i-coordinate between the i-coordinates of s_i and t_i. Observe that in this case the two paths cannot intersect as P_1 can only reach vertices with coordinates in A_1 and B_1 and P_2 can only use vertices with coordinates in A_2 and B_2. Hence, one can find P_1 and P_2 in linear time. Altogether, this gives a running time of $O(nm)$ in this case.

Assume now that there is a point with integer coordinates in this intersection; this case requires more work. We assume that if there are two points (x_1, y_1) and (x_2, y_2) in the intersection of the drawings of P_1 and P_2, then we have $x_1 < x_2 \Leftrightarrow y_1 < y_2$. If this is not the case, then repeat the algorithm below with swapped s_2 and t_2. We guess the first point p in the intersection (note that it has integer coordinates). This can be done in $O(n)$ time by guessing a vertex on p. Now we arrange the areas slightly different. The areas are defined by p and one coordinate of s_1, t_1, s_2, t_2; see the right side of Figure 2 for an illustration. Edges in the area $A_i \setminus B_j$ or $B_i \setminus A_j$ are oriented towards the vertex with the larger i-coordinate. Note that edges on the line in $A_i \cap B_j, i \neq j$, could either be used by P_1 or P_2 (but not both), meaning that we have to direct the edges towards the vertex with either the larger 1- or 2-coordinate. Since there are only two possibilities for orienting the edges in $A_1 \cap B_2$, there are only four different possibilities to orient the edges on $A_1 \cap B_2$ and $A_2 \cap B_1$. We try

[3] Whenever we pretend to guess something, the algorithm actually exhaustively tests all possible choices.

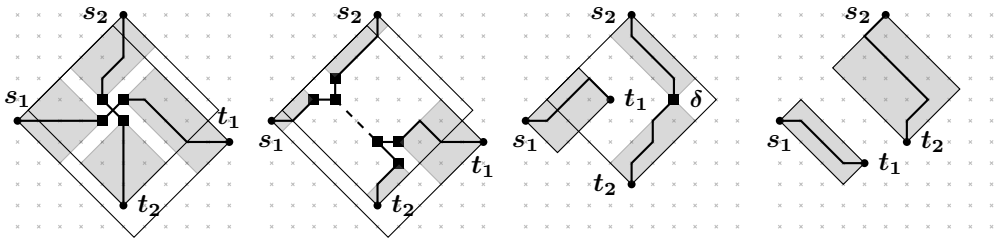


Figure 3 The four cases for the projection of two paths P_1 and P_2 in the two-dimensional grid. From left to right: (1) The projection of the paths cross in one point with non-integer coordinates. (2) The projection of the paths cross in at least one point with integer coordinates. (3) The rectangles defined by the endpoints of P_1 and P_2 intersect, but the projections of P_1 and P_2 do not intersect. (4) The rectangles defined by the endpoints of P_1 and P_2 do not intersect. For each of the two paths our algorithm guesses the vertices on the positions marked by squares.

all four possible orientations and if at least one of them yields a solution, then we know that there is a solution. All other (unorientated) edge are removed – a shortest s_i-t_i-path cannot use it. Note that this results for each of the four described cases in a directed acyclic graph. Furthermore, again a shortest s_i-t_i-path in G induces a s_i-t_i-path in D and each s_i-t_i-path in D is an shortest path in G.

Finally, we use a $O(n+m)$-time algorithm of Tholey [14] for 2-Disjoint Paths on a DAG to find P_1 and P_2. Since there are $O(n)$ possibilities for the point p and 4 possibilities for directing the edges between A_i and B_j, $i \neq j$, we call $O(n)$ instances of the algorithm of Tholey [14]. Thus, we obtain Theorem 1 which is formally proven in Section 3.

Generalizing to k-DSP. We now discuss how to generalize the ideas from above to k-DSP, where $k > 2$. One central idea for $k = 2$ is that the subpaths within the areas A_1, A_2, B_1, B_2 (see Figure 2) can hardly overlap. The only overlap is possible along the borders. In our approach for $k > 2$, we simplify this even further by guessing the vertices on each path before and after the intersection (thus incurring a higher running time). This results in four cases; see Figure 3 for an overview of the cases and the guessed vertices (marked by black squares). It is easy to see in Figure 3 that in each case no subpath within one gray area can possibly intersect with a subpath within another gray area. As can be seen in Figure 3, there is only one case where subpaths of P_1 and P_2 have to be computed carefully due to possible intersections: Both paths use the dashed line in the second case from the left. However, this is the part where both paths are strictly monotone in both coordinates. This is what allowed us for $k = 2$ to transform the graph into a DAG while preserving the solutions (both paths being strictly monotone in at least one coordinate is actually sufficient for this transformation).

Considering k paths, we associate with each vertex $v \in V$ a position in the k-dimensional Euclidean vector space. For brevity, we say that a path has color i if it is strictly monotone in its i-coordinate. Thus, each path P_j has color j. The problem k-Disjoint Paths on a DAG is solvable in polynomial time for constant k [7]. Thus, if we want to find k subpaths from u_i to v_i, $i \in [k]$, that all have the same color (i. e. for each $i \in [k]$ we have that the difference of the i-th coordinate of u_i and v_i is $\text{dist}(u_i, v_i)$), then we can use the algorithm of Fortune et al. [7]. For completeness, we provide a dynamic program with a precise running time analysis as Fortune et al. [7] only state "polynomial time". The general approach to solve the given k-DSP instance is thus as follows: Split the paths $P_1, \ldots, P_k$ into $f(k)$ subpaths (i. e. guess the endpoints of the subpaths) and find a partition of the subpaths such that

(i) subpaths in the same part of the partition share a common color and hence can be computed by the algorithm of Fortune et al. [7] or our dynamic program, and

(ii) subpaths in different parts of the partition cannot intersect.

We remark that this is essentially the same general approach used by Lochet [12]. However, he does not use the geometric view of the paths (as we do). As a result, even for $k = 2$ he only bounds the number of created subpaths by 9^{91} (cf. Lochet [12, Lemma 4.2]). While this constant is certainly not optimized, one can easily see in Figure 3 that our approach splits the two paths in at most five parts (in the second case). Moreover, our geometric view allows us to use a more efficient way of splitting the paths for general k, which we describe below.

Recall that for $k = 2$ the two paths P_1 and P_2 have at most one intersection (point or straight line); see Figure 3. However, in three dimensions $k > 2$ this is no longer true as neither P_1 nor P_2 needs to be monotone in a third dimension. Thus, to exploit the properties shown in Figure 3 for two paths P_i and P_j, we need to project into two dimensions using the i and j coordinate. Hence, we need to be careful with using proper projections to 2D; see Section 3 for details on the geometric arguments. However, whenever two paths P_i and P_j intersect, then we know that the two subpaths in the intersection have both colors i and j. Thus, we can use for these subpaths the two-dimensional observations behind Figure 3 with new projections. We store for each subpath P' of P_i the set Φ of all indices of paths P' intersects, that is, Φ is a subset of all colors that P' has. Now, if P' and P_j intersect, then there is a subpath of P_j that has colors $\Phi \cup \{i\}$. Hence this set Φ of colors can be seen as a "tower of colors" that is transferred to other paths. Our algorithm transfers these towers from one path to another as long as possible. These towers will be defined over permutations of subsets of $[k]$ that encode how these color-towers are produced; see Section 4. As there are at most $k \cdot k!$ such permutations, this explains the exponent of our algorithm. In the end, we arrive at Theorem 2, see Section 5 for details.

3 The Geometry of Two Shortest Paths

In this section, we formalize and generalize the idea behind the geometric view (visualized in Figures 1–3). We start by introducing our notation for projections. For any $\emptyset \subset I \subseteq [k]$ and any vector $x \in \mathbb{R}^k$ we denote with $x^I \in \mathbb{R}^{|I|}$ the orthogonal projection of x to the coordinates in I. That is, x^I is the $|I|$-dimensional vector obtained by deleting all dimensions in x that are not in I. We usually drop the brackets in the exponent, thus writing e.g., $(5,6,7,8,9)^{1,3,4} = (5,7,8)$ or $(5,6,7)^2 = 6$. Similarly, for $R \subseteq \mathbb{R}^k$ we define $R^I := \{x^I \mid x \in R\} \subseteq \mathbb{R}^{|I|}$.

We associate with each vertex $v \in V$ a position in the k-dimensional Euclidean vector space. Formally, $\vec{v} := (\vec{v}^i)_{i\in[k]} := (\mathrm{dist}(s_i, v))_{i\in[k]} \in \mathbb{N}^k$ and for $U \subseteq V$ we use $\vec{U} := \{\vec{u} \mid u \in U\}$. For the k-DSP-instance at hand, one can compute the positions of each vertex in $O(km)$ using simple breadth-first-search from each vertex s_i.

In the following, we will use the following notations for any index set $\emptyset \subset I \subseteq [k]$:
$v \circ^I w \Leftrightarrow \forall a \in I\colon \vec{v}^a \circ \vec{w}^a$, for any $\circ \in \{<, \leq, =, \geq, >\}$ and vertices v, w.
$V \circ^I W \Leftrightarrow \{\vec{v}^I \mid v \in V\} \circ \{\vec{w}^I \mid w \in W\}$, for any $\circ \in \{\subset, \subseteq, =, \supseteq, \supset\}$ and vertex sets V, W.
We further write $x \in^I X$ if there is a vertex $x' \in X$ with $x' =^I x$ and $x \notin^I X$ otherwise.

▶ **Lemma 4.** *For any pair of vertices $v, w \in V$, we have $\|\vec{v} - \vec{w}\|_\infty \leq \mathrm{dist}(v, w)$.*

Proof. Let P be a shortest v-w-path with edge set E_P. Each edge $\{p, q\}$ of P has $\|\vec{p} - \vec{q}\|_\infty = 1$ and thus by the triangle inequality $\|\vec{v} - \vec{w}\|_\infty \leq \sum_{e\in E_P} 1 = \mathrm{dist}(v, w)$. ◀

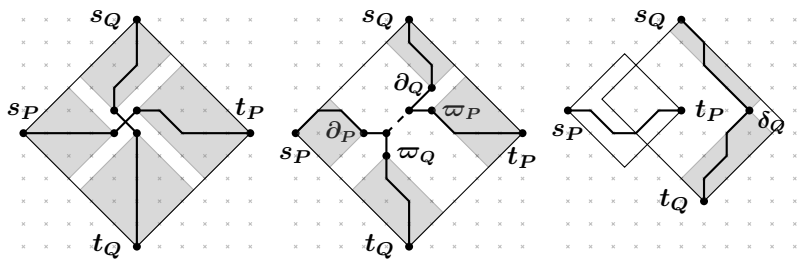


Figure 4 Picture of the a, b-projection of an a-colored path P and a b-colored path Q. The labels are abbreviated as $\partial_Q = \partial_Q^{a,b}(P)$, $\Delta = \Delta^{a,b}(P, Q)$, etc. *Left side:* The case that the two paths a, b-cross but do not share vertices with common a, b-coordinates (thus $\alpha_P(Q) = \omega_P(Q) = \bot$, see Definition 8). *Middle:* Illustration of Lemma 10. The dashed black edge is the a, b-crossing of P and Q. The rectangle areas $\overrightarrow{s_P} \diamond \overrightarrow{\partial_P}$, $\overrightarrow{\varpi_P} \diamond \overrightarrow{t_P}$, $\overrightarrow{s_Q} \diamond \overrightarrow{\partial_Q}$, and $\overrightarrow{\varpi_Q} \diamond \overrightarrow{t_Q}$ are highlighted in gray. These areas are pairwise disjoint. *Right side:* Illustration of Lemma 14. If P and Q are a, b-noncrossing and $\delta_Q \neq \bot$, then the two shaded areas are disjoint from $\overrightarrow{s_P} \diamond \overrightarrow{t_P}$.

For two vertices $u, w \in V$, define $u \diamond w := \{v \in V \mid \mathrm{dist}(u, v) + \mathrm{dist}(v, w) = \mathrm{dist}(u, w)\}$ to be the set of all vertices that lie on a shortest u-w-path. Similarly, for any $x, y \in \mathbb{N}^k$ define $x \diamond y := \{z \in \mathbb{R}^k \mid \|x - z\|_\infty + \|z - y\|_\infty = \|x - y\|_\infty\}$ which is a rectangle whose sides form an angle of 45° with the coordinate axes (see right side of Figure 1).

▶ **Definition 5.** *Let $s, t \in V$ with $\mathrm{dist}(s, t) = \|\vec{s} - \vec{t}\|_\infty$ and P be any shortest s-t-path. We then call the pair (s, t) and the path P* colored*. Furthermore, we define $C(P) := C(s, t) := \{a \in [k] \mid |\vec{s}^a - \vec{t}^a| = \|\vec{s} - \vec{t}\|_\infty\}$ and call (s, t) and P a-*colored *if $a \in C(s, t)$.*

Note that, if P is an a-colored path, then P is strictly monotone in its a-coordinate. Note that for arbitrary $u, w \in V$ we do *not* always have $\overrightarrow{u \diamond w} \subseteq \vec{u} \diamond \vec{w}$, that is the coordinates of all vertices on shortest u-w-paths are not necessarily contained in the set of coordinates "spanned" by $\vec{u}$ and $\vec{w}$. However, this inclusion holds for colored vertex pairs as shown next:

▶ **Lemma 6** (⋆). *Let $v, w \in V$ be an a-colored pair for any color a. Then, $\overrightarrow{v \diamond w} \subseteq \vec{v} \diamond \vec{w}$.*

We will usually be concerned with the projection of $\vec{v} \diamond \vec{w}$ to some set of coordinates $I \subseteq [k]$. Note in this context that $(\vec{v} \diamond \vec{w})^I = \vec{v}^I \diamond \vec{w}^I$.

For some $a \neq b$, consider the projections of an a-colored path P_a and a b-colored path P_b to the $\{a, b\}$-plane, that is, the coordinates of the vertices of the paths are projected to their a- and b-coordinate and edges are drawn as straight lines between the projected vertices. To this end, for any path P we define $\zeta(P) \subset \mathbb{R}^k$ as the piecewise linear curve connecting the points of $\vec{P}$ in the order given by P. It is not hard to see (e.g. in Figure 1 (right) and Figure 3 case (2)), that the intersection of P_a and P_b in the a, b-projection is also a straight line segment with an angle of 45° to the coordinate axes.

▶ **Lemma 7** (⋆). *Let P be an a-colored path and Q be a b-colored path. Then $\zeta(P)^{a,b} \cap \zeta(Q)^{a,b}$ is a (possibly empty) straight line segment.*

Note that even if $\zeta(P)^{a,b} \cap \zeta(Q)^{a,b}$ is non-empty, it needs not contain points from $\mathbb{N}^2$ as can be seen in the left side of Figure 4. In the following, we define the first and last vertices of P and Q on their crossing as well as the coordinates of the vertices before and after their crossing.

▶ **Definition 8.** *Let P be an a-colored path and Q be a b-colored path. We say P and Q are a,b-*crossing *if the intersection $X := \zeta(P)^{a,b} \cap \zeta(Q)^{a,b}$ is non-empty. If $X = \emptyset$, they are called a,b-noncrossing.*

If $\overrightarrow{P}^{a,b} \cap X \neq \emptyset$, then we define $\alpha_P^{a,b}(Q)$ (resp. $\omega_P^{a,b}(Q)$) as the first (resp. last) vertex v of P with $v^{a,b} \in X$. In all other cases set $\alpha_P^{a,b}(Q) := \omega_P^{a,b}(Q) := \perp$.

If P and Q are a,b-crossing, we further define $\partial_P^{a,b}(Q)$ (resp. $\varpi_P^{a,b}(Q)$) as the last (resp. first) vertex of P before (resp. after) that intersection. If no such vertex exists, we set $\partial_P^{a,b}(Q) := \perp$ (resp. $\varpi_P^{a,b}(Q) := \perp$). In all these notations we will omit a, b, and Q if it is clear from context.

▶ **Observation 9.** *If P, Q are two paths with $\alpha_P^{a,b}(Q) \neq \perp$, then $P[\alpha_P^{a,b}(Q), \omega_P^{a,b}(Q)] =^{a,b} Q[\alpha_Q^{a,b}(P), \omega_Q^{a,b}(P)]$. In particular, both of these subpaths are a,b-colored.*

If P and Q are a,b-crossing, then Observation 9 characterizes the behavior of the "crossing subpaths". Let us now consider the remaining path segments before and after the crossing. By Lemma 6 these segments have to lie in the rectangle areas $\overrightarrow{s_P} \diamond \overrightarrow{\partial_P^{a,b}(Q)}$, $\overrightarrow{\varpi_P^{a,b}(Q)} \diamond \overrightarrow{t_P}$, $\overrightarrow{s_Q} \diamond \overrightarrow{\partial_Q^{a,b}(P)}$, and $\overrightarrow{\varpi_Q^{a,b}(P)} \diamond \overrightarrow{t_Q}$ which are displayed in Figure 4 (left and middle). We can show that these areas are indeed pairwise disjoint.

▶ **Lemma 10** (⋆). *Let P and Q be two a,b-crossing paths. Then the sets $\left(\overrightarrow{s_P} \diamond \overrightarrow{\partial_P^{a,b}(Q)}\right)^{a,b}$, $\left(\overrightarrow{\varpi_P^{a,b}(Q)} \diamond \overrightarrow{t_P}\right)^{a,b}$, $\left(\overrightarrow{s_Q} \diamond \overrightarrow{\partial_Q^{a,b}(P)}\right)^{a,b}$, and $\left(\overrightarrow{\varpi_Q^{a,b}(P)} \diamond \overrightarrow{t_Q}\right)^{a,b}$ are pairwise disjoint (or undefined).*

Unfortunately, when P and Q are a,b-noncrossing, then $s_P \diamond t_P$ and $s_Q \diamond t_Q$ are not disjoint in general, see for example Figure 4 (right). To deal with this case, we show that when splitting one path into two subpaths at the vertex δ_Q (see Figure 4 (right)), then we get the desired properties that the respective rectangles do not intersect.

▶ **Definition 11.** *Let P, Q be two colored paths and $a, b \in [k]$. The* common a,b-area *of P and Q is $\Delta^{a,b}(P,Q) := (\overrightarrow{s_P} \diamond \overrightarrow{t_P})^{a,b} \cap (\overrightarrow{s_Q} \diamond \overrightarrow{t_Q})^{a,b}$.*

▶ **Definition 12.** *Let P be an a-colored path and Q a b-colored path where (without loss of generality) $s_P <^a t_P$. Define $B := \{v \in V \mid v =^b s_P \wedge v <^a s_P\} \cup \{v \in V \mid v =^b t_P \wedge v >^a t_P\}$. Define $\delta_Q^{b,a}(P)$ as the unique vertex in $Q \cap B$ or as $\perp$ if that intersection is empty.*

▶ **Lemma 13** (⋆). *If P, Q are a,b-noncrossing paths with $\Delta^{a,b}(P,Q) \neq \emptyset$, then $\delta_P^{a,b}(Q) \neq \perp$ or $\delta_Q^{a,b}(P) \neq \perp$.*

The next lemma shows that if P and Q are not crossing but have a common area $\Delta_{a,b}$, then the path whose end vertex lies not in the common area $\Delta_{a,b}$ does not enter $\Delta_{a,b}$ at all.

▶ **Lemma 14** (⋆). *If P is an a-colored path and Q a b-colored path with $\delta_P^{a,b}(Q) \neq \perp$, then $\left(\overrightarrow{s_Q} \diamond \overrightarrow{t_Q}\right)^{a,b}$ is disjoint from $\left(\overrightarrow{s_P} \diamond \overrightarrow{\delta_P^{a,b}(Q)}\right)^{a,b} \cup \left(\overrightarrow{\delta_P^{a,b}(Q)} \diamond \overrightarrow{t_P}\right)^{a,b}$.*

▶ **Definition 15.** *Let P be an a-colored s_P-t_P-path and Q a b-colored s_Q-t_Q-path. We then define $\mathcal{C}_P^{a,b}(Q) := \{s_P, t_P, \mu_P^{a,b}(Q) \mid \mu \in \{\alpha, \omega, \partial, \varpi, \delta\}\} \setminus \{\perp\}$.*

The next proposition shows the sets $\mathcal{C}_P^{a,b}(Q)$ and $\mathcal{C}_Q^{a,b}(P)$ "characterize" the crossing of P and Q in the sense that any two shortest paths using these vertices have exactly the same vertex coordinates in the crossing.

▶ **Proposition 16** (⋆). *Let P and P' be a-colored s_P-t_P-paths, and let Q and Q' be b-colored s_Q-t_Q-paths. If $\mathcal{C}_P^{a,b}(Q) \subseteq P'$ and $\mathcal{C}_Q^{a,b}(P) \subseteq Q'$, then $\{v \in P' \mid v \in^{a,b} Q'\} =^{a,b} \{v \in P \mid v \in^{a,b} Q\}$.*

We now have all ingredients for the proof of Theorem 1.

Proof of Theorem 1. Let $I := (G = (V, E), k, ((s_1, t_1), (s_2, t_2))$ be an instance of 2-DSP. Compute $\vec{v}$ for all $v \in V$ via two breadth-first searches in $O(n + m)$ time. We assume without loss of generality that G is connected. To ensure that we report I being a *yes*-instance only if I is indeed a *yes*-instance, we perform a sanity-check in the very end to verify that our guesses were correct. Hence, we only need to show that we find in $O(nm)$ time a solution to I if there is one. To this end, assume there are disjoint shortest s_i-t_i-paths P_i for $i \in [2]$. By Lemma 7 we have three cases.

(Case 1): $\zeta(P_1) \cap \zeta(P_2)$ is empty. If $\Delta^{1,2}(P_1, P_2) = \emptyset$, then, by Lemma 6, a solution can easily be found by two independent breadth-first-searches. Otherwise, we guess in $O(n)$ time the vertex $\delta_{P_1}^{1,2}(P_2)$ on P_1 or $\delta_{P_2}^{2,1}(P_1)$ on P_2 from Definition 12. By Lemma 13, at least one of them exists (assume without loss of generality that $\delta_{P_1}^{1,2}(P_2)$ exists). By Lemma 14, $\delta_{P_1}^{1,2}(P_2) \neq \perp$ and any shortest s_1-$\delta_{P_1}^{1,2}(P_2)$-path ($\delta_{P_1}^{1,2}(P_2)$-t_1-path) is vertex disjoint from any shortest s_2-t_2-path. Hence, we now can check in $O(m)$ time whether I is a *yes*-instance.

(Case 2): $\zeta(P_1) \cap \zeta(P_2)$ has no point with integer coordinates. Then, we guess the four points surrounding $\zeta(P_1) \cap \zeta(P_2)$ in $O(n)$ time. This can be done in $O(n)$ time by guessing the vertex $\partial_{P_1}^{1,2}(P_2)$ and branch into two cases. Let p_{s_i} and p_{t_i} be the guessed points used by P_i such that $p_{s_i}^i + 1 = p_{t_i}^i$, for $i \in [2]$. Now we construct a directed graph D on the vertices V such that there is an arc (v, w) if $\{v, w\} \in E$, $\vec{v}^i + 1 = \vec{w}^i$, and $\overrightarrow{\{v, w\}} \subseteq \vec{s_i} \diamond p_{s_i} \cup p_{t_i} \diamond \vec{t_i}$ for some $i \in [2]$. Note that D is acyclic and that each s_i-t_i-path in D corresponds (same set of vertices) to a shortest s_i-t_i-path in G, and has an arc (v, w) such that $\vec{v} = p_{s_i}$ and $\vec{w} = p_{t_i}$. Hence, by Lemma 10 we can simply use two breadth-first-searches from s_1 and s_2 to find a solution.

(Case 3): $\zeta(P_1) \cap \zeta(P_2)$ has at least one point with integer coordinates. Without loss of generality, we assume that $\overrightarrow{\alpha_{P_1}^{1,2}(P_2)} = \overrightarrow{\alpha_{P_2}^{1,2}(P_1)}$, otherwise we swap the terminal pairs in the input instance. We guess in $O(n)$ time the discrete point $p \in \zeta(P_1)^{1,2} \cap \zeta(P_2)^{1,2}$ such that $\vec{p}^1$ is minimized. Let $A_i := \vec{s_i} \diamond \vec{p}$ and $B_i := \vec{p} \diamond \vec{t_i}$, for all $i \in [2]$. Now we construct a directed acyclic graph D on the vertices V such that there is an arc (v, w) if for some $i \in [2]$ we have (1) $\{v, w\} \in E$, (2) $\vec{v}^i + 1 = \vec{w}^i$, and (3) (a) $\vec{v} \in A_i \setminus B_i$ and $\vec{w} \in A_i$ or (b) $\vec{v} \in B_i$ and $\vec{w} \in B_i \setminus A_i$.

To add the edges with coordinates in $A_1 \cap B_2, A_2 \cap B_1$ to D, we observe that our assumption implies that $\zeta(P_1) \cap \zeta(P_2) \subseteq B_1 \cap B_2$. Hence, all edges where the vertices have coordinates in $A_i \cap B_j$ can only be used by either P_1 or P_2, for each $\{i, j\} = [2]$. Thus, we branch in four cases and add the edges accordingly. Note that D is acyclic and that each s_i-t_i-path in D corresponds to a shortest s_i-t_i-path in G going through point p. Hence, by Lemma 10 I is *yes*-instance if and only if there are disjoint s_i-t_i-path in D, for all $i \in [2]$. Thus, we apply an $O(n+m)$-time algorithm of Tholey [14] for 2-Disjoint Paths on a DAG. This yields a total running time of $O(nm)$. ◀

4 The Geometry of Many Shortest Paths

In the previous section, we looked at two shortest paths P and Q from s_P to t_P and s_Q and t_Q respectively. We showed that selecting at most six vertices from P and Q (three per path; see Definition 15) is sufficient to ensure that each pair of shortest s_P-t_P- and s_Q-t_Q-paths

that also contain the vertices $\mathcal{C}_P^{a,b}(Q)$ and $\mathcal{C}_Q^{a,b}(P)$, respectively, "behave" like P and Q in the sense of using the same coordinates or not (see Proposition 16). In this section, we define a set $\mathcal{C}$, $|\mathcal{C}| \in O(k \cdot k!)$, that basically ensures the same properties for k paths. To formalize our goal in this section, we introduce the concept of *avoiding* paths.

▶ **Definition 17** (I-avoiding). *Let $\emptyset \subset I \subseteq [k]$. We say that two paths P and Q are* I-avoiding *if $p \notin^I Q$ holds for every internal vertex p of P and $q \notin^I P$ for every internal vertex q of Q.*

We further call two vertex pairs (s_p, t_p) and (s_q, t_q) I-avoiding, if $(\vec{s_p}^I \diamond \vec{t_p}^I) \cap (\vec{s_q}^I \diamond \vec{t_q}^I) \subseteq \{\vec{s_p}^I, \vec{t_p}^I\} \cap \{\vec{s_q}^I, \vec{t_q}^I\}$.

Note that being I-avoiding implies being I'-avoiding for all $I' \supseteq I$. We use *avoiding* as a shorthand for $[k]$-avoiding. The reason for defining *avoiding* in such a way that the endpoints of the paths play a special role is as follows: When partitioning a colored path $P = v_1 \dots v_\ell$ into two subpaths $P' = v_1 \dots v_j$ and $P'' = v_j v_{j+1} \dots v_n$, then these two subpaths obviously share exactly one vertex, namely v_j. However, we still want to call the pairs $(\vec{v_1}, \vec{v_j})$ and $(\vec{v_j}, \vec{v_\ell})$ avoiding since these two subpaths cannot have any other intersection besides $\vec{v_j}$.

Two paths P_1 and P_2 are internally disjoint if neither of them contains an internal vertex of the other path. Avoiding paths are clearly internally disjoint.

▶ **Observation 18.** *Let P, Q be two avoiding paths. Then P is internally disjoint from Q.*

Moreover, avoiding vertex pairs (s, t) and (u, w) ensure that the corresponding shortest s-t- and u-v-paths are internally disjoint.

▶ **Lemma 19** ($\star$). *Let (s, t) and (u, w) be two colored pairs of vertices. If (s, t) and (u, w) are avoiding, then each shortest s-t-path is internally disjoint from each shortest u-w-path.*

With the notation of avoiding pairs, we can formulate our goal for this section. To this end, fix a solution $\mathcal{P} = (P_i)_{i \in [k]}$ for the k-DSP instance $(G, (s_i, t_i)_{i \in [k]})$, that is, P_i is the s_i-t_i-path in this solution. Essentially, we want to partition the paths in $\mathcal{P}$ into subpaths and assign labels (subsets of $[k]$) to each subpath such that the following holds:

(1.) Let P be a subpath with labels $\Phi \subseteq [k]$. For each $a \in \Phi$, P is a-colored.

(2.) Let P and Q be subpaths from s_P to t_P and s_Q and t_Q with labels $\Phi_P, \Phi_Q \subseteq [k]$ respectively. If $\Phi_P \neq \Phi_Q$, then (s_P, t_P) and (s_Q, t_Q) are avoiding.

Note that (2.) will be the central argument in our algorithm for k-DSP. The algorithm guesses the endpoints of these subpaths and based on (2.), the algorithm can compute the interior points of subpaths with different label sets independently of each other.

Note that for $k = 2$ the partition of P_1 and P_2 along the sets $\mathcal{C}_{P_1}^{1,2}(P_2)$ and $\mathcal{C}_{P_2}^{1,2}(P_1)$, respectively, satisfies the above two points: Each subpath of P_i, $i \in [2]$, has label i. Moreover, the subpaths between the α and ω-vertices have both labels 1 and 2. Hence, (1.) above is satisfied. Furthermore, (2.) essentially follows from Proposition 16.

We now give some intuition on how to lift this to arbitrary fixed k leading to Definition 20, a generalization of Definition 15. Initially, each path P_i has label i. Whenever two paths P_i and P_j in the solution intersect in the (i, j)-projection (that is, we have α and ω vertices), then the subpaths in the intersection gets both labels i and j. If a third path P' also intersects with the subpath of P_j that intersects with P_i, then we try to use the intersections to move the label i via path P_j to some subpath of P'. Generalizing this, we consider for each sequence $\sigma = (\ell_1, \dots, \ell_h)$ whether label ℓ_1 could be "transported" from P_{ℓ_1} to P_{ℓ_2}, from P_{ℓ_2} to P_{ℓ_3}, ..., and from $P_{\ell_{h-1}}$ to P_{ℓ_h}. The reason for doing it this way is that we will have for each such sequence $\sigma = (\ell_1, \dots, \ell_h)$ at most one consecutive subpath on P_{ℓ_h} that has label ℓ_1 transported via σ. While the idea of transporting labels would also work with triplets (transport label a via path P_b to path P_c), we do not have any bound on the number of resulting subpaths (as for each triplets there might be many such subpaths).

In order to formalize this idea and define the *crossing set* $\mathcal{C}$ of these paths (Definition 20), we need some notation. Let $\sigma = (\ell_1, \dots, \ell_h)$ be a sequence. We define $\text{set}(\sigma) := \{\ell_1, \dots, \ell_h\}$ to be the set with all entries of σ. For a path $P = v_0 \dots v_h$ and $1 \leq i < j \leq h$ let $P[v_i, v_j] := v_i \dots v_j$ be the subpath of P with endpoints v_i and v_j. The set $\mathcal{C}$ for each permutation σ of each $\Phi \subseteq [k]$ is recursively defined as follows. Therein, $\mathcal{T}$ describes the first and last vertex on the respective subpath having exactly the set of labels in Φ.

▶ **Definition 20.** *For each $\Phi \subseteq [k]$ and each permutation $\sigma = (\ell_1, \dots, \ell_h)$ of Φ set:*

- *If $h = 1$ with $\sigma = (i)$, then set $\mathcal{C}^\sigma := \mathcal{T}(\sigma) := \{s_i, t_i\}$.*
- *If $h = 2$ with $\sigma = (i, j)$, then set $\mathcal{T}(\sigma) := \{\alpha^{i,j}_{P_j}(P_i), \omega^{i,j}_{P_j}(P_i)\}$, and $\mathcal{C}^\sigma := \mathcal{C}^{i,j}_{P_j}(P_i)$.*
- *If $h \geq 3$, then let $\sigma_{start} := (\ell_1, \dots, \ell_{h-1})$, $\sigma_{end} := (\ell_2, \dots, \ell_h)$. If $\mathcal{T}(\sigma_{start}) = \{\perp\}$ or $\mathcal{T}(\sigma_{end}) = \{\perp\}$ or $Q := P_{\ell_{h-1}}[\mathcal{T}(\sigma_{start})] \cap P_{\ell_{h-1}}[\mathcal{T}((\ell_h, \ell_{h-1}))] = \emptyset$, then set $\mathcal{T}(\sigma) := \mathcal{C}^\sigma := \{\perp\}$. Otherwise, let $P := P_{\ell_h}[\mathcal{T}(\sigma_{end})]$. Then set $\mathcal{T}(\sigma) := \{\alpha^{\ell_1,\ell_h}_P(Q), \omega^{\ell_1,\ell_h}_P(Q)\}$. Moreover, set $\mathcal{C}^\sigma := \mathcal{C}^{\ell_1,\ell_h}_P(Q) \cup \mathcal{C}^{\ell_1,\ell_h}_Q(P)$.*

The set $\mathcal{C} := \bigcup_\sigma \mathcal{C}^\sigma$ is the crossing set *of $\mathcal{P}$.*

As subsequently shown, when transporting the labels via a sequence $\sigma = (\ell_1, \dots, \ell_h)$, the intersecting subpath in the target path P_{ℓ_h} agrees in all coordinates in $\text{set}(\sigma)$ with the subpath of $P_{\ell_{h-1}}$ where the label is transported from.

▶ **Lemma 21 (⋆).** *Let $\sigma := (\ell_1, \dots, \ell_h)$ be any permutation of any $\Phi \subseteq [k]$ with $|\Phi| = h \geq 2$. If $\mathcal{T}(\sigma) \neq \{\perp\}$, then $P_{\ell_h}[\mathcal{T}(\sigma)] =^\Phi Q'$ for some subpath Q' of $Q := P_{\ell_{h-1}}[\mathcal{T}(\sigma_{start})] \cap P_{\ell_{h-1}}[\mathcal{T}((\ell_h, \ell_{h-1}))]$ where $\sigma_{start} := (\ell_1, \dots, \ell_{h-1})$.*

We next formalize the notions used in the context of the intersection of $\mathcal{C}$ with the paths $\mathcal{P}$.

▶ **Definition 22.** *An* i-marble path *T is a set of vertices such that $\{s_i, t_i\} \subseteq T$ and for each $u, v \in T$ the pair (u, v) is i-colored. A* segment *S of a i-marble path T is a subset of T containing two vertices denoted* $\text{start}(S)$ *and* $\text{end}(S)$ *and all vertices $v \in T$ with* $\text{start}(S) <^i v <^i \text{end}(S)$*. A segment is minimal if it contains exactly two vertices.*

We say a segment is i-colored, if $(\text{start}(S), \text{end}(S))$ is i-colored. We say two segments S and S' are avoiding if the minimal subsegments of S and S' are pairwise avoiding.

We say a path P follows *S if it is i-colored, has end vertices* $\text{start}(S)$ *and* $\text{end}(S)$*, and $S \subseteq V(P)$.*

To prove the central statement of this section, we need to formalize the labels of a segment. To this end, let S_i be a segment of P_i. Then set

$$\mathit{labels}[S_i] := \{a \mid \exists \sigma = (a = \ell_1, \dots, \ell_h = i), h \geq 1 \colon S_i \subseteq P_i[\mathcal{T}(\sigma)]\}.$$

▶ **Proposition 23 (⋆).** *For $i, j \in [k]$ let $S_i \subseteq V(P_i) \cap \mathcal{C}$ and $S_j \subseteq V(P_j) \cap \mathcal{C}$ be two minimal segments. If $\mathit{labels}[S_i] \neq \mathit{labels}[S_j]$, then S_i and S_j are avoiding.*

5 The Algorithm: Utilizing the Geometry

In this section, we finally present the algorithm behind Theorem 2. Pseudo-code for this algorithm is listed in Algorithm 1. In a nutshell, we first guess all marble paths T_i and the map $\mathcal{T}$ corresponding to the crossing set $\mathcal{C}$ of some solution (if one exists). Then, we find all minimal segments of each marble path T_i and partition them such that (1) all minimal segments in the same part of the partition are strictly monotone in the same set of

Algorithm 1 Our algorithm for k-DSP.

```
1  function solve(G, (s_i, t_i)_{i∈[k]})
2      foreach guess (T_i)_{i∈[k]}, Ends of the crossing set do
           /* We assume now that the guess is correct, that is, for a
              solution P = (P_i)_{i∈[k]} we have T_i = C ∩ P_i, i ∈ [k], & Ends = T   */
3          P_i = ∅, for all i ∈ [k]
4          foreach minimal segment S of some T_i, i ∈ [k] do
5              marks[S] ← ∅
6              foreach sequence σ = (ℓ_1, ℓ_2, ..., i) with S ⊆ Ends(σ) do
7                  marks[S] ← marks[S] ∪ set(σ)
8              j ← min marks[S]
9              x ← arg min{ v⃗^j | v ∈ {start(S), end(S)} }
10             y ← arg max{ v⃗^j | v ∈ {start(S), end(S)} }
11             P_j = P_j ∪ {(x, y)}
12         foreach j ∈ [k] do Order P_j = ((x_1, y_1), (x_2, y_2), ...) so that x⃗_1^j ≤ x⃗_2^j ≤ ...
13         if all instances (D(G, i), P_i), i ∈ [k] of |P_i|-Disjoint Paths on i-layered
            DAGs are yes-instances and the combined solutions form a solution of
            k-DSP then return yes
14     return no
```

coordinates, and (2) two minimal segments in distinct parts of the partition are avoiding. The crucial improvement over the algorithm of Lochet [12] is that our partition is much smaller. Afterwards, we find shortest disjoint paths for each part of our partition separately via dynamic programming.

To this end, we introduce *c-layered DAGs* and the p-Disjoint Paths on DAGs problem. For a graph G with coordinates $\vec{v}$ (as defined in Section 3) for all $v \in V$, the c-layered DAG $D(G, c)$ of G is the directed graph $(V(G), A)$, where $(x, y) \in A$ if and only if $\{x, y\} \in E(G)$ and $\vec{y}^c - \vec{x}^c = 1$. Crucial here is the following simple observation.

▶ **Observation 24.** *A path P in G is c-colored if and only if $(V(P), \{(u, v) \mid \{u, v\} \in E(P), \vec{v}^c - \vec{u}^c = 1\})$ is a path in the c-layered DAG of G.*

In p-Disjoint Paths on DAGs we are given a directed acyclic graph D and a list $(s_i, t_i)_{i\in[p]}$ of (possibly intersecting) terminal pairs, and ask whether there are pairwise internally disjoint s_i-t_i-path in D, for each $i \in [p]$. Fortune et al. [7] showed a $n^{O(p)}$-time algorithm for p-Disjoint Path on DAGs. A straight-forward dynamic program yields a more specific running time of $O(n^{p+1})$.

▶ **Lemma 25** (⋆). *An instance of p-Disjoint Paths on DAGs on a graph with n vertices can be solved in $O(n^{p+1})$ time.*

▶ **Lemma 26** (⋆). *Algorithm 1 runs in $O(k \cdot n^{16k \cdot k! + k + 1})$ time.*

For the correctness of Algorithm 1, we need to show that each part of the partition of minimal segments can be solved independently. This follows from Proposition 23 together with the fact that Algorithm 1 exhaustively tries all possibilities for the crossing set $\mathcal{C}$. Together with Lemma 26, this implies Theorem 2.

6 Conclusion

We provided an improved polynomial-time for k-DSP. However, while the running time of our algorithm can certainly be slightly improved by some case distinctions and a more careful analysis, the algorithm is still far from being practical. Reducing the factor in the exponent to a polynomial in k is a clear challenge for future work. Considering the fine-grained complexity of 2-DSP, it would be interesting to know whether there are running time barriers based on e. g. the Strong Exponential Time Hypothesis.

Concerning generalizations of k-DSP, we believe that we can modify our algorithm in a straight-forward way to work with positive edge-lengths. However, the case of non-negative edge-lengths seems much more difficult. Our basic geometric observations made in Section 3 crucially depend on the fact that we are looking for shortest paths. Thus, if there are no k disjoint shortest paths, then computing k disjoint paths minimizing their total length in polynomial time is still an open problem for $k \geq 3$ (for $k = 2$ Björklund and Husfeldt [4] provided a randomized $O(n^{11})$ time algorithm and moreover proved that this problem is contained in NC if the graph is planar and cubic [3]).

References

1 Maxim Akhmedov. Faster 2-disjoint-shortest-paths algorithm. In *Proceedings of the 15th International Computer Science Symposium in Russia (CSR '20)*, volume 12159 of *Lecture Notes in Computer Science*, pages 103–116. Springer, 2020. doi:10.1007/978-3-030-50026-9_7.

2 Kristof Berczi and Yusuke Kobayashi. The Directed Disjoint Shortest Paths Problem. In *Proceedings of the 25th Annual European Symposium on Algorithms (ESA '17)*, volume 87 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 13:1–13:13. Schloss Dagstuhl – Leibniz-Zentrum fuer Informatik, 2017. doi:10.4230/LIPIcs.ESA.2017.13.

3 Andreas Björklund and Thore Husfeldt. Counting shortest two disjoint paths in cubic planar graphs with an NC algorithm. In *Proceedings of the 29th International Symposium on Algorithms and Computation (ISAAC '18)*, volume 123 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 19:1–19:13. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2018. doi:10.4230/LIPIcs.ISAAC.2018.19.

4 Andreas Björklund and Thore Husfeldt. Shortest two disjoint paths in polynomial time. *SIAM Journal on Computing*, 48(6):1698–1710, 2019. doi:10.1137/18M1223034.

5 Tali Eilam-Tzoreff. The disjoint shortest paths problem. *Discrete Applied Mathematics*, 85(2):113–138, 1998. doi:10.1016/S0166-218X(97)00121-2.

6 Fedor V. Fomin, Dániel Marx, Saket Saurabh, and Meirav Zehavi. New Horizons in Parameterized Complexity (Dagstuhl Seminar 19041). *Dagstuhl Reports*, 9(1):67–87, 2019. doi:10.4230/DagRep.9.1.67.

7 Steven Fortune, John E. Hopcroft, and James Wyllie. The directed subgraph homeomorphism problem. *Theoretical Computer Science*, 10:111–121, 1980. doi:10.1016/0304-3975(80)90009-2.

8 Marinus Gottschau, Marcus Kaiser, and Clara Waldmann. The undirected two disjoint shortest paths problem. *Operations Research Letters*, 47(1):70–75, 2019. doi:10.1016/j.orl.2018.11.011.

9 Richard M Karp. On the computational complexity of combinatorial problems. *Networks*, 5(1):45–68, 1975. doi:10.1002/net.1975.5.1.45.

10 Ken-ichi Kawarabayashi, Yusuke Kobayashi, and Bruce A. Reed. The disjoint paths problem in quadratic time. *Journal of Combinatorial Theory. Series B*, 102(2):424–435, 2012. doi:10.1016/j.jctb.2011.07.004.

11 Yusuke Kobayashi and Ryo Sako. Two disjoint shortest paths problem with non-negative edge length. *Operations Research Letters*, 47(1):66–69, 2019. doi:10.1016/j.orl.2018.11.012.

12 William Lochet. A polynomial time algorithm for the k-disjoint shortest paths problem. In *Proceedings of the 32nd ACM-SIAM Symposium on Discrete Algorithms (SODA '21)*, pages 169–178. SIAM, 2021. doi:10.1137/1.9781611976465.12.

13 Neil Robertson and Paul D. Seymour. Graph minors. XIII: the disjoint paths problem. *Journal of Combinatorial Theory. Series B*, 63(1):65–110, 1995. doi:10.1006/jctb.1995.1006.

14 Torsten Tholey. Linear time algorithms for two disjoint paths problems on directed acyclic graphs. *Theoretical Compututer Science*, 465:35–48, 2012. doi:10.1016/j.tcs.2012.09.025.

Deterministic Rounding of Dynamic Fractional Matchings

Sayan Bhattacharya
Department of Computer Science, University of Warwick, Coventry, UK

Peter Kiss
Department of Computer Science, University of Warwick, Coventry, UK

Abstract

We present a framework for *deterministically* rounding a dynamic fractional matching. Applying our framework in a black-box manner on top of existing fractional matching algorithms, we derive the following new results: (1) The first deterministic algorithm for maintaining a $(2-\delta)$-approximate maximum matching in a fully dynamic bipartite graph, in arbitrarily small polynomial update time. (2) The first deterministic algorithm for maintaining a $(1+\delta)$-approximate maximum matching in a decremental bipartite graph, in polylogarithmic update time. (3) The first deterministic algorithm for maintaining a $(2+\delta)$-approximate maximum matching in a fully dynamic general graph, in *small* polylogarithmic (specifically, $O(\log^4 n)$) update time. These results are respectively obtained by applying our framework on top of the fractional matching algorithms of Bhattacharya et al. [STOC'16], Bernstein et al. [FOCS'20], and Bhattacharya and Kulkarni [SODA'19].

Previously, there were two known general-purpose rounding schemes for dynamic fractional matchings. Both these schemes, by Arar et al. [ICALP'18] and Wajc [STOC'20], were randomized.

Our rounding scheme works by maintaining a good *matching-sparsifier* with bounded arboricity, and then applying the algorithm of Peleg and Solomon [SODA'16] to maintain a near-optimal matching in this low arboricity graph. To the best of our knowledge, this is the first dynamic matching algorithm that works on general graphs by using an algorithm for low-arboricity graphs as a black-box subroutine. This feature of our rounding scheme might be of independent interest.

2012 ACM Subject Classification Theory of computation → Design and analysis of algorithms

Keywords and phrases Matching, Dynamic Algorithms, Data Structures

Digital Object Identifier 10.4230/LIPIcs.ICALP.2021.27

Category Track A: Algorithms, Complexity and Games

Related Version *Full Version*: `https://arxiv.org/abs/2105.01615`

Funding *Sayan Bhattacharya*: Supported by EPSRC Grant EP/S03353X/1.

1 Introduction

The central question in the area of *dynamic algorithms* is to understand how can we efficiently maintain a good solution to a computational problem, when the underlying input changes over time [26, 28]. In the past decade, an extensive body of work in this area has been devoted to the study of *dynamic matching* [1, 5, 6, 7, 8, 10, 11, 17, 22, 24, 34, 35, 38, 39].

A matching $M \subseteq E$ in G is a set of edges that do not share any common endpoint. In the dynamic matching problem, the input is a graph $G = (V, E)$ that keeps getting updated via edge insertions/deletions, and the goal is to maintain an approximately *maximum matching* in G with small (preferably polylogarithmic) update time, where the phrase "update time" refers to the time it takes to handle an "update" (edge insertion/deletion) in G.[1] From the current landscape of dynamic matching, we can identify a common template that underpins a number of existing algorithms for this problem. This template consists of three steps.

[1] An algorithm has an "amortized" update time of $O(\tau)$ iff starting from an empty graph, it can handle any sequence of κ edges insertions/deletions in $O(\tau \cdot \kappa)$ total time.

Step (I). Design an efficient dynamic algorithm that maintains an approximately maximum *fractional matching*[2] $w : E \to [0, 1]$ in the input graph $G = (V, E)$. All the known algorithms for this first step are deterministic [9, 12, 13, 15, 14, 16, 23].

Step (II). Maintain a *sparse* (bounded-degree) subgraph $S = (V, E_S)$ of the input graph, with $E_S \subseteq E$, that approximately preserves the size of maximum matching [3, 40]. In a bit more details, the subgraph S should have the property that $\mu(S)$ is very close to $size(w)$, where $\mu(S)$ denotes the size of maximum (integral) matching in S, and $size(w) = \sum_{e \in E} w(e)$ denotes the size of the fractional matching w from the previous step. Such a subgraph S is often referred to as a *matching-sparsifier* of G [4]. There are two known algorithms for this second step and both of them are randomized, in sharp contrast to Step (I). Specifically, Arar et al. [3] designed a randomized rounding scheme for sparsifying a dynamic fractional matching. Their algorithm works only in the *oblivious adversary* setting, where the future updates cannot depend on the past actions taken by the algorithm. This result was very recently improved upon by Wajc [40], who presented an elegant dynamic rounding scheme for Step (II) that, although randomized, works in a much more general *adaptive adversary* setting, where the future updates to the algorithm can depend on all its past random bits.

Step (III). Maintain a near-optimal matching in the (bounded-degree) sparsifier S from the previous step, using a known algorithm by Gupta et al. [25], which has $O(\Delta)$ update time on dynamic graphs with maximum degree $\leq \Delta$. Since S has bounded degree, the third step incurs only a small overhead in the update time. The algorithm in [25] is also deterministic.

A natural question arises from the preceding discussion. Can we design an efficient *deterministic* dynamic algorithm for Step (II)? Since Step (I) and Step (III) are already deterministic, an efficient deterministic algorithm for Step (II) will allow us to derandomize multiple existing results in the literature on dynamic matching. We resolve this question in the affirmative. Specifically, our main result is summarized in the theorem below.

▶ **Theorem 1.** *Fix any small constant $\delta > 0$. Consider a dynamic graph $G = (V, E)$ on n nodes and a (dynamic) fractional matching w in G. In this setting, an update either inserts/deletes an edge in $G = (V, E)$ or changes the weight $w(e)$ of an existing edge $e \in E$. We can deterministically maintain a subgraph $S = (V, E_S)$ of G, with $E_S \subseteq E$, such that:*

1. *There exists a fractional matching $h' : E_S \to [0, 1]$ in S with $size(w) \leq (1 + \delta) \cdot size(h')$.*
2. *If w is a (δ, δ)-approximate maximal matching in G, then $\mu(G) \leq (2 + \delta) \cdot \mu(S)$.*
3. *The arboricity of S is $O(\log^2 n)$.*
4. *Every update in G or w, on average, leads to $O(\log^2 n)$ updates in S.*
5. *Our dynamic algorithm for maintaining S has $O(\log^2 n)$ amortized update time.*

Bounded arboricity matching-sparsifiers. We will shortly explain part-(2) of Theorem 1, which uses the notion of a (δ, δ)-*approximate maximal matching* that has not been defined yet. For now, we focus on an intriguing feature of Theorem 1, namely, that it only maintains a subgraph S with bounded *arboricity*.[3] This is in sharp contrast to all previous work on dynamic matching-sparsifiers: they satisfy the strictly stronger requirement of bounded

[2] A fractional matching w in G assigns a weight $w(e) \in [0, 1]$ to every edge $e \in E$, ensuring that the total weight assigned to all the edges incident on any given node is ≤ 1.

[3] Informally, an undirected graph $G' = (V', E')$ has arboricity κ if we can assign a direction to each of its edges $e \in E'$ in such a way that every node $v \in V'$ gets an out-degree of at most $O(\kappa)$. If a graph has maximum degree at most κ, then its arboricity is also $O(\kappa)$, but not vice versa.

maximum degree [3, 40]. Our algorithm exploits this feature in a crucial manner, allowing certain nodes to have large degrees in S while ensuring that the arboricity of S remains at most $O(\log^2 n)$. This does not cause any problem in the overall scheme of things, however, because Peleg and Solomon [37] have shown how to deterministically maintain a $(1+\delta)$-approximate maximum matching in $O(\Delta)$ update time in a dynamic graph with arboricity $\leq \Delta$. Their algorithm allows us to efficiently maintain a near-optimal matching in S.

To summarize, there is an existing line of work on dynamic matching which deal with the special class of low-arboricity graphs [29, 34, 37]. Theorem 1 shows that if we have a good dynamic matching algorithm for low-arboricity graphs, then we can use it in a black-box manner to design better dynamic matching algorithms for *general graphs* as well.

Implications of Theorem 1. We start by focussing on bipartite graphs. If a graph G is bipartite, then the size of a maximum fractional matching in G is equal to $\mu(G)$. Accordingly, part-(1) of Theorem 1 implies that if the input graph G is bipartite, then our dynamic algorithm maintains a sparsifier $S = (V, E_S)$ such that $size(w) \leq (1+\delta) \cdot \mu(S)$. We can now run the dynamic algorithm from [37] on S, which has small arboricity, to efficiently maintain a near-optimal (integral) matching $M \subseteq E_S$ with $size(w) \leq (1+\delta) \cdot |M|$.

Bhattacharya et al. [14] gave a deterministic algorithm for maintaining $(2-\epsilon)$-approximate maximum *fractional* matchings in bipartite graphs with arbitrarily small polynomial update time. Applying our dynamic rounding framework on top of this result from [14], we get the *first deterministic* algorithm for dynamic (integral) matchings in bipartite graphs with the same approximation ratio and similar update time, as summarized in the theorem below.

▶ **Theorem 2.** *For every constant $k \geq 10$, there exists a $\beta_k \in (1,2)$, and a* deterministic *dynamic algorithm that maintains a β_k-approximate maximum matching in an n-node bipartite graph with $O(n^{1/k} \cdot \log^4 n)$ amortized update time.*

Next, very recently Bernstein et al. [9] showed how to maintain a $(1+\delta)$-approximate maximum *fractional* matching in a bipartite graph with $O(\log^3 n)$ amortized update time in the *decremental* setting, where the input graph only undergoes edge-deletions. Applying our dynamic rounding framework on top of their result, we get the *first deterministic* algorithm for maximum (integral) matching in an analogous decremental setting, with the same approximation ratio and similar update time. This is stated in the theorem below.

▶ **Theorem 3.** *We can* deterministically *maintain a $(1+\delta)$-approximate maximum matching in a decremental bipartite graph on n nodes with $O(\log^7 n)$ amortized update time.*

Moving on to general graphs, we note that if a graph $G = (V, E)$ is non-bipartite, then the size of a maximum fractional matching can be as large as $(3/2) \cdot \mu(G)$. Thus, if we are to naively apply our dynamic rounding framework based on the guarantee given to us by part-(1) of Theorem 1, then we will lose out on a factor of $3/2$ in the approximation ratio. This is where part-(2) of Theorem 1 comes in handy. Specifically, as in [3, 40], we invoke the notion of an (α, β)-*approximate maximal matching* (see Definition 6).

To see why this notion is useful for us, consider the result of Bhattacharya and Kulkarni [16], who designed a deterministic dynamic algorithm for $(2+\delta)$-approximate maximum *fractional* matching, for small constant $\delta > 0$, in general graphs with $O(1)$ amortized update time. Furthermore, the fractional matching maintained by [16] is (δ, δ)-approximately maximal. Thus, applying Theorem 1 on top of this result from [16], we can deterministically maintain a sparsifier $S = (V, E_S)$ of the input graph G with $\mu(G) \leq (2+\delta) \cdot \mu(S)$. We can now maintain a near-optimal maximum matching $M \subseteq E_S$ in S, using the algorithm of [37]. Since $\mu(G) \leq (2+\delta) \cdot \mu(S)$, M will be a $(2+\delta)$-approximate maximum (integral) matching in G. Putting everything together, we get the result summarized in the theorem below.

▶ **Theorem 4.** *We can* deterministically *maintain a* $(2+\delta)$*-approximate maximum matching in an* n*-node dynamic graph with* $O(\log^4 n)$ *amortized update time, for small constant* $\delta > 0$.

Prior to our work, the only *deterministic* dynamic algorithm for $(2+\delta)$-approximate maximum matching in general graphs with polylogarithmic update time was due to Bhattacharya et al. [14]. The exact polylogarithmic factor in the update time of [14] was huge (more than $\log^{20} n$), and the algorithm of [14] was significantly more complicated than ours.

Perspective. Existing techniques for proving update-time lower bounds for dynamic problems cannot distinguish between deterministic and randomized algorithms [2, 27, 30, 31, 36]. Thus, understanding the power of randomization in the dynamic setting is an important research agenda, which comprises of two separate strands of work. (1) Studying the power of the *oblivious adversary* assumption while designing a randomized algorithm for a given dynamic problem. (2) Studying the separation between randomized algorithms that work against *adaptive adversaries* on the one hand, and deterministic algorithms on the other. Our work falls under the second category. A recent breakthrough result under this category has been a deterministic algorithm for dynamic minimum spanning forest with worst-case subpolynomial update time [18, 21]. This improves upon earlier work which achieved the same update time guarantee for dynamic minimum spanning forest, but using a randomized algorithm that works against adaptive adversary [32]. There are other well-studied dynamic problems where currently we have polynomial gaps between the update times of the best-known deterministic algorithm and the best-known randomized algorithm against adaptive adversary [19, 20]. Bridging these gaps remain challenging open questions.

Our Techniques. The key ingredient in our rounding scheme is a simple *degree-split* procedure. Given any graph $G' = (V', E')$ as input, this procedure runs in linear time and outputs a subgraph $G'' = (V', E'')$ where the degree of every node $v \in V'$ drops by a factor of $(1/2) \cdot (1 \pm \epsilon)$, provided the initial degree of v in G' was larger than $(1/\epsilon)$. See Algorithm 3.

Using this degree-split procedure, we first design a simple *static* algorithm for sparsifying a *uniform* fractional matching w (which assigns the same weight to every edge) in an input graph $G = (V, E)$. This works in rounds. In each round, we start by repeatedly removing the nodes with degree at most $(1/\epsilon)$, until we are left with a graph $G' = (V', E')$ where every remaining node has degree larger than $(1/\epsilon)$. We now apply the degree-split procedure on $G' = (V', E')$ to obtain a subset of edges $E'' \subseteq E'$, double the weight of every edge $e \in E''$, and discard the edges $e \in E'' \setminus E'$ from the support of the fractional matching. Since the degree-split procedure reduces the degree of every node in V' by (approximately) a factor of $1/2$, it follows that we (approximately) preserve the total weight received by every node while implementing a given round. We can show that if we continue with this process for (roughly) logarithmic many rounds, then we end up with a bounded-arboricity subgraph of the input graph G that approximately preserves the size of the fractional matching w. In the dynamic setting, we try to mimic this static algorithm in a natural lazy manner.

When the input is a dynamic graph G and a (not necessarily uniform) fractional matching w, then, roughly speaking, we first discretize w and then decompose it into $O(\log n)$ many uniform fractional matchings, defined on mutually edge-disjoint subgraphs of G. We run a dynamic algorithm for sparsifying a uniform fractional matching on each of these subgraphs, and we maintain the union of the outputs of all these $O(\log n)$ many dynamic sparsifiers.

2 Notations and Preliminaries

Throughout this paper, we let $G = (V, E)$ denote the input graph, and $n = |V|$ will be the number of nodes in G. For any $v \in V$, $E' \subseteq E$ and $V' \subseteq V$, we let $E'(v, V') = \{(u, v) \in E' : u \in V'\}$ denote the set of edges in E' that are incident on v and have their other endpoints in V'. To ease notations, we define $E'(v) := E'(v, V)$. We also define $\deg_{E'}(v, V') := |E'(v, V')|$ and $\deg_{E'}(v) := |E'(v)|$. Furthermore, given any subset of edges $E' \subseteq E$, we let $V(E') = \bigcup_{(u,v)\in E'} \{u, v\}$ denote the set of endpoints of the edges in E'. Throughout the rest of this paper, we will consider δ to be some small constant, and we will fix two more parameters β and ϵ as stated below.

$$10^{-3} \geq \delta = 20 \cdot \beta = 5000 \cdot \epsilon \cdot \log n > 0. \tag{1}$$

Given any subset of edges $E' \subseteq E$, a *weight-function* $w' : E' \to [0, 1]$ assigns a (possibly fractional) weight $0 \leq w'(e) \leq 1$ to every edge $e \in E'$. We say that E' is the *support* of w' and write $\textsc{Support}(w') := E'$. The *size* of the weight-function w' is defined as $size(w') := \sum_{e\in E'} w'(e)$. For any node $v \in V$, let $w'(v) := \sum_{(u,v)\in E'} w'(u, v)$ denote the total weight received by v from all its incident edges under the weight-function w'. We say that w' is a *fractional matching* in the graph $G' := (V, E')$ iff $w'(v) \leq 1$ for all $v \in V$. Since $E' \subseteq E$, we often abuse notation to say that such a weight-function w' is a fractional matching in $G = (V, E)$ as well. For any $0 \leq \lambda \leq 1$, we say that w' is a λ-*uniform* weight-function (or, fractional matching, if $w'(v) \leq 1$ for all $v \in V$) iff $w'(e) = \lambda$ for all edges $e \in E'$.

Let $\mu(G')$ and $\mu_f(G')$ respectively denote the size of maximum matching and the size of maximum fractional matching in a graph G'. We will use the following well-known theorem.

▶ **Theorem 5.** *Consider any graph G'. If G' is bipartite, then $\mu(G') = \mu_f(G')$. Otherwise, we have $\mu(G) \leq \mu_f(G') \leq (3/2) \cdot \mu(G)$.*

We will use the notion of an *approximately maximal* matching as in Arar et al. [3].

▶ **Definition 6.** *Consider any graph $G' = (V', E')$ and a fractional matching w' in G'. We say that w' is a (α, β)-approximately maximal matching in G' iff the following holds. For every edge $(u, v) \in E'$, either (1) $\{w'(u, v) \geq \beta\}$, or (2) $\{$there is at least one endpoint $x \in \{u, v\}$ such that $w'(x) \geq 1 - \alpha$ and $w'(x, y) < \beta$ for all edges $(x, y) \in E'$ incident on $x\}$.*

An *orientation* of a graph $G' = (V', E')$ assigns a direction to every edge $(u, v) \in E'$. For the rest of this paper, whenever we say that a graph G' has *arboricity* $O(\kappa)$, we mean that G' admits an orientation of its edges where the maximum out-degree of a node is $O(\kappa)$ [33].

3 A Static Algorithm for Sparsifying a Uniform Fractional Matching

In this section, we present a simple static algorithm for sparsifying a uniform fractional matching. This will form the basis of our dynamic algorithm in Section 4 and Section 5.

As input, we receive a graph $G = (V, E)$ and a λ-uniform fractional matching $w : E \to [0, 1]$ in G, for some $\lambda \in [\delta/n^2, \beta)$. Define $L = L(\lambda)$ to be the largest integer k such that $\beta/2 \leq 2^k\lambda < \beta$. Since δ is a constant and $\delta/n^2 \leq \lambda < \beta$, from (1) we infer that $L = O(\log n)$.

The algorithm proceeds in *rounds* $i \in \{0, \ldots, L - 1\}$. Before the start of round $i = 0$, we initialize $E^{(\geq 0)} := E$, $V^{(\geq 0)} := V$, $G^{(\geq 0)} := \left(V^{(\geq 0)}, E^{(\geq 0)}\right)$ and $\gamma^{(0)} := w$. Thus, $\gamma^{(0)}$ is a λ-uniform fractional matching in G. In each round i, we identify a subset of edges $F^{(i)} \subseteq E^{(\geq i)}$ that get *frozen*, in the sense that they are *not* considered in subsequent rounds. Define $\mathcal{F}^{(i)} := E^{(\geq i)} \bigcup_{j=0}^{i-1} F^{(j)}$ and $\mathcal{H}^{(i)} := (V, \mathcal{F}^{(i)})$ for all $i \in [0, L]$. The following invariant will be satisfied in the beginning of each round $i \in [0, L]$. The weight-function $\gamma^{(i)} : \mathcal{F}^{(i)} \to [0, 1]$ ensures that $\gamma^{(i)}(v) \simeq w(v)$ for all $v \in V$. Clearly, this invariant holds for $i = 0$.

Implementing a given round $i \in [0, L-1]$. Initialize $G' = (V', E') := G^{(\geq i)}$. There are two distinct *steps* in this round. During the first step, we keep iteratively removing the nodes with degree $\leq (1/\epsilon)$ from G'. Let $V^{(i)} \subseteq V^{(\geq i)}$ be the collection of nodes that get removed from G' in this manner, and let $F^{(i)} \subseteq E^{(\geq i)}$ denote the set of edges incident on $V^{(i)}$. Define $V^{(\geq i+1)} := V^{(\geq i)} \setminus V^{(i)}$. At the end of this first step, the status of $G' = (V', E')$ is as follows: $V' = V^{(\geq i+1)}$ and $E' = E^{(\geq i)} \setminus F^{(i)}$. Intuitively, we can afford to remove the nodes $V^{(\geq i)}$ from G' because the edges in $F^{(i)}$ admit an orientation with maximum out-degree $(1/\epsilon)$. At the end of this first step every node in G' has degree $\geq (1/\epsilon)$.

In the second step, we call a subroutine Degree-Split(E'), which returns a subset of edges $E'' \subseteq E'$ with the following property: $\deg_{E''}(v) \simeq (1/2) \cdot \deg_{E'}(v)$ for all nodes $v \in V'$. We will shortly see how to implement this Degree-Split subroutine. For now, we move ahead with the description of round i. We set $E^{(\geq i+1)} := E''$ and $G^{(\geq i+1)} := \big(V^{(\geq i+1)}, E^{(\geq i+1)}\big)$. Next, we discard the edges in $E' \setminus E^{(\geq i+1)}$ from the support of γ and double the weights on the remaining edges in $E^{(\geq i+1)}$. This leads us to a new weight-function $\gamma^{(i+1)} : \mathcal{F}^{(i+1)} \to [0, 1]$ in $\mathcal{H}^{(i+1)} = (V, \mathcal{F}^{(i+1)})$ which is defined as follows. For every edge $e \in \mathcal{F}^{(\geq i+1)}$, we have:

$$\gamma^{(i+1)}(e) = \begin{cases} 2 \cdot \gamma^{(i)}(e) & \text{if } e \in E^{(\geq i+1)}; \\ \gamma^{(i)}(e) & \text{else if } e \in \bigcup_{j=0}^{i} F^{(j)}. \end{cases} \tag{2}$$

At this point, if $i < L-1$, then we are ready to proceed to the next round $i+1$. Otherwise, if $i = L-1$, then we terminate the algorithm after setting $F^{(L)} := E^{(\geq L)}$ and $V^{(L)} := V^{(\geq L)}$.

Define $F := \bigcup_{i=0}^{L} F^{(i)}$, $H := (V, F)$, and $h := \gamma^{(L)}$. We will show that $H = (V, F)$ is a good matching-sparsifier for the λ-uniform matching w in the input graph $G = (V, E)$. The relevant pseudocodes are summarized in Algorithm 1, Algorithm 2 and Algorithm 3. For clarity of exposition, in some of these pseudocodes we use one additional notation $h^{(i)}$, which is basically a weight-function $h^{(i)} : F^{(i)} \to [0, 1]$ such that $h^{(i)}(e) = 2^i \lambda$ for all $e \in F^{(i)}$.

Implementing the Degree-Split(E')subroutine. Consider any graph $G^* = (V^*, E^*)$. A *walk* W in G^* is a set of distinct edges $\{(u_0, v_0), \ldots, (u_k, v_k)\} \subseteq E^*$ such that $v_i = u_{i+1}$ for all $i \in [0, k-1]$. Let $W^{(even)} = \{(u_{2i}, v_{2i}) : i \in [0, \lfloor k/2 \rfloor]\}$ denote the collection of even numbered edges from this walk W. The walk W is said to be *maximal* in G^* iff $\deg_{E^* \setminus W}(u_0) = \deg_{E^* \setminus W}(u_k) = 0$. When we call Degree-Split(E'), it first partitions the edge-set E' into a collection of walks $\mathcal{W}$ as specified in Algorithm 3. It then returns the set $E'' \subseteq E'$, which consists of all the even numbered edges from all the walks $W \in \mathcal{W}$.

▷ Claim 7. The subroutine Degree-Split(E') runs in $O(|E'|)$ time.

Proof. We can compute a maximal walk W in a given graph $G^* = (V^*, E^*)$ in $O(|W|)$ time. Hence, the total running time of Algorithm 3 is given by $\sum_{W \in \mathcal{W}} O(|W|) = O(|E'|)$. ◁

▷ Claim 8. In Algorithm 3, $\deg_{E''}(v) \in \left[\frac{\deg_{E'}(v)}{2} - 1, \frac{\deg_{E'}(v)}{2} + 1\right]$ for all nodes $v \in V(E')$.

Proof. Consider any graph $G^* = (V^*, E^*)$ and a walk $W = \{(u_0, v_0), \ldots, (u_k, v_k)\}$ in G^*. Let End$(W) = \{u_0, v_k\}$ denote the endpoints of this walk W, and let $V(W) = \bigcup_{i=0}^{k} \{u_i\} \bigcup_{i=0}^{k} \{v_i\}$ denote the set of all nodes touched by W. Note that $\deg_{W^{(even)}}(v) = (1/2) \cdot \deg_W(v)$ for all nodes $v \in V(W) \setminus$ End(W), and $\deg_{W^{(even)}}(v) \in [(1/2) \cdot \deg_W(v) - 1, (1/2) \cdot \deg_W(v) + 1]$ for all nodes $v \in$ End(W). Now, fix any node $v \in V(E')$, and observe that:

1. $\deg_{E''}(v) = \sum_{W \in \mathcal{W}} \deg_{W^{(even)}}(v)$ and $\deg_{E'}(v) = \sum_{W \in \mathcal{W}} \deg_W(v)$.
2. The definition of a maximal walk implies that $v \in$ End(W) for at most one $W \in \mathcal{W}$.

These observations, taken together, imply the claim. ◁

Algorithm 1 STATIC-UNIFORM-SPARSIFY($G = (V, E), \lambda$), where $\delta/n^2 \leq \lambda < \beta$.

Let $w : E \to [0, 1]$ be a λ-uniform fractional matching in G.
Initialize $V^{(\geq 0)} := V$ and $E^{(\geq 0)} := E$.
Initialize a weight-function $h^{(0)} : E^{(\geq 0)} \to [0, 1]$ so that $h^{(0)}(e) := \lambda$ for all $e \in E^{(\geq 0)}$.
Let $L := L(\lambda)$ be the unique nonnegative integer k such that $\beta/2 \leq 2^k\lambda < \beta$.
Call the subroutine REBUILD$(0, \lambda)$. (see Algorithm 2)
Define $F := \bigcup_{i=0}^{L} F^{(i)}$, and $H := (V, F)$.
Define $h : F \to [0, 1]$ such that for all $i \in [0, L]$ and $e \in F^{(i)}$ we have $h(e) := h^{(i)}(e)$.

Algorithm 2 REBUILD(i', λ).

for $i = i'$ to $(L - 1)$ **do**:
  $V^{(i)} \leftarrow \emptyset$.
  while there is some node $v \in V^{(\geq i)} \setminus V^{(i)}$ with $\deg_{E^{(\geq i)}}\left(v, V^{(\geq i)} \setminus V^{(i)}\right) \leq (1/\epsilon)$ **do**:
    $V^{(i)} \leftarrow V^{(i)} \cup \{v\}$.
  $V^{(\geq i+1)} \leftarrow V^{(\geq i)} \setminus V^{(i)}$.
  $F^{(i)} \leftarrow \{(u, v) \in E^{(\geq i)} : \text{ either } u \in V^{(i)} \text{ or } v \in V^{(i)}\}$.
  $E^{(\geq i+1)} \leftarrow$ DEGREE-SPLIT$(E^{(\geq i)} \setminus F^{(i)})$.
  for all edges $e \in E^{(\geq i+1)}$ **do**:
    $h^{(i+1)}(e) \leftarrow 2 \cdot h^{(i)}(e)$.
$F^{(L)} \leftarrow E^{(\geq L)}$.
$V^{(L)} \leftarrow V^{(\geq L)}$.

Algorithm 3 DEGREE-SPLIT(E').

Initialize $E^* \leftarrow E'$ and $\mathcal{W} \leftarrow \emptyset$.
while $E^* \neq \emptyset$ **do**
  Let $G^* := (V(E^*), E^*)$, where $V(E^*)$ is the set of endpoints of the edges in E^*.
  Compute a *maximal* walk W in G^*.
  Set $\mathcal{W} \leftarrow \mathcal{W} \cup \{W\}$, and $E^* \leftarrow E^* \setminus W$.
Return the set of edges $E'' := \bigcup_{W \in \mathcal{W}} W^{(even)}$.

Note that $h^{(i)}$ and $F^{(i)}$ represent global variables in the pseudocodes above.

▶ **Lemma 9.** *Algorithm 1 runs in $O(|E|)$ time.*

Proof. The runtime of Algorithm 1 is dominated by the call to REBUILD$(0, \lambda)$. Accordingly, focus on any given iteration $i \in [0, L - 1]$ of the outer FOR loop in Algorithm 2. During this iteration, using appropriate data structures the inner WHILE loop takes $O(|F^{(i)}|)$ time, the call to DEGREE-SPLIT$(E^{(\geq i)} \setminus F^{(i)})$ takes $O(|E^{(\geq i)} \setminus F^{(i)}|)$ time as per Claim 7, and the inner FOR loop takes $O(|E^{(\geq i+1)}|)$ time. Hence, the total time taken to implement iteration i of the outer FOR loop is at most $O(|E^{(\geq i)}|) + O(|E^{(\geq i)} \setminus F^{(i)}|) + O(|E^{(\geq i+1)}|) = O(|E^{(\geq i)}|)$.

Summing over all $i \in [0, L - 1]$, we derive that:

$$\text{The total runtime of Algorithm 1 is at most } \sum_{i=0}^{L-1} O\left(|E^{(\geq i)}|\right). \tag{3}$$

Next, fix an iteration i of the outer For loop in Algorithm 2, and focus on the call to Degree-Split($E^{(\geq i)} \setminus F^{(i)}$). The inner While loop ensures that $\deg_{E^{(\geq i)} \setminus F^{(i)}}(v) > (1/\epsilon)$ for all $v \in V^{(\geq i+1)}$. By Claim 8, the degree of every concerned node is (roughly) halved by a call to Degree-Split(.). Hence, $|E^{(\geq i+1)}| = O\left((1/2) \cdot |E^{(\geq i)}|\right)$ for all $i \in [0, L-1]$. Plugging this back into (3), the lemma follows since $\sum_{i=0}^{L-1} O\left(|E^{(\geq i)}|\right) = O\left(|E^{(\geq 0)}|\right) = O(|E|)$. ◀

We will next show that the subgraph $H = (V, F)$ returned by our algorithm is a good matching-sparsifier.Towards this end, we first derive the following important claim.

▷ Claim 10 (Informal). For all $v \in V$ and $i \in [0, L-1]$, we have $\gamma^{(i+1)}(v) \simeq (1+\epsilon) \cdot \gamma^{(i)}(v)$.

Proof. Let us track how starting from $\gamma^{(i)}$, the weight-function $\gamma^{(i+1)}$ is constructed during round i. Recall that Support $\left(\gamma^{(i)}\right) := E^{(\geq i)} \bigcup_{j=0}^{i-1} F^{(j)}$. During round i, we first identify the subset $F^{(i)} \subseteq E^{(\geq i)}$, and then identify another subset $E^{(\geq i+1)} \subseteq E^{(\geq i)} \setminus F^{(i)}$. As we switch from $\gamma^{(i)}$ to $\gamma^{(i+1)}$, the following three events occur: (1) The weights of the edges $e \in F^{(i)} \bigcup_{j=0}^{i-1} F^{(j)}$ do not change. (2) The edges $e \in \left(E^{(\geq i)} \setminus F^{(i)}\right) \setminus E^{(\geq i+1)}$ get discarded from the support of $\gamma^{(i+1)}$. (3) The weights of the remaining edges $e \in E^{(\geq i+1)}$ get doubled.

Now, fix any node $v \in V$. The claim follows from our analysis of the two cases below.

Case 1: $v \in V^{(\geq i+1)}$. In this case, the inner While loop in Algorithm 2 ensures that $\deg_{E^{(\geq i)} \setminus F^{(i)}}(v) > (1/\epsilon)$. Hence, applying Claim 8, we get: $\deg_{E^{(\geq i+1)}}(v) \simeq (1 \pm \epsilon) \cdot (1/2) \cdot \deg_{E^{(\geq i)} \setminus F^{(i)}}(v)$. To summarize, (about) half the edges in $E^{(\geq i)} \setminus F^{(i)}$ that are incident on v get discarded from the support of $\gamma^{(i+1)}$, while the remaining edges in $E^{(\geq i)} \setminus F^{(i)}$ double their weights. In contrast, the edges in $F^{(i)} \bigcup_{j=0}^{i-1} F^{(j)}$ that are incident on v do not change their weights at all. This implies that $\gamma^{(i+1)}(v) \simeq (1 \pm \epsilon) \cdot \gamma^{(i)}(v)$.

Case 2: $v \notin V^{(\geq i+1)}$. In this case, every edge $(u, v) \in$ Support $\left(\gamma^{(i)}\right)$ continues to remain in the support of $\gamma^{(i+1)}$ with the same weight. Hence, we get: $\gamma^{(i+1)}(v) = \gamma^{(i)}(v)$. ◁

▶ **Lemma 11** (Informal). *The weight-function $h : F \to [0, 1]$ satisfies three properties:*

1. *$h(e) < \beta$ for all edges $e \in F$.*
2. *$w(v) \simeq (1 \pm \epsilon L) \cdot h(v)$ for all nodes $v \in V$.*
3. *$size(w) \simeq (1 \pm \epsilon L) \cdot size(h)$.*

Proof. Before the start of round 0, we have $\gamma^{(0)}(e) = \lambda$ for all edges $e \in E^{(\geq 0)}$. Subsequently, in each round $i \in [0, L-1]$, the weight of each edge $e \in E^{(\geq i+1)}$ gets doubled. Hence, we have $h(e) = \gamma^{(L)}(e) \leq 2^L \lambda < \beta$ for all $e \in F$. This proves part-(1) of the lemma.

Next, fix any node $v \in V$. Before the start of round 0, we have $\gamma^{(0)} = w$ and hence $\gamma^{(0)}(v) = w(v)$. Subsequently, after each round $i \in [0, L-1]$, Claim 10 guarantees that $\gamma^{(i+1)}(v) \simeq (1 \pm \epsilon) \cdot \gamma^{(i)}(v)$. This gives us: $h(v) = \gamma^{(L)}(v) \simeq (1 \pm \epsilon)^L \cdot \gamma^{(0)}(v) \simeq (1 \pm \epsilon L) \cdot \gamma^{(0)}(v) \simeq (1 \pm \epsilon L) \cdot w(v)$. Finally, summing this (approximate) equality over all nodes $v \in V$, we get: $size(w) \simeq (1 \pm \epsilon L) \cdot size(h)$. This proves part-(2) and part-(3) of the lemma. ◀

Levels of nodes and edges. The *level* of a node $v \in V$ is defined as $\ell(v) := \max\{i \in [0, L] : v \in V^{(\geq i)}\}$. Similarly, the *level* of an edge $e \in E$ is defined as $\ell(e) := \max\{i \in [0, L] : e \in E^{(\geq i)}\}$. Since $V^{(i)} = V^{(\geq i)} \setminus V^{(\geq i+1)}$ for all $i \in [0, L]$, it follows that $\ell(v) = i$ iff $v \in V^{(i)}$.

▶ **Observation 12.** *For all $(u, v) \in F$, we have $\ell(u, v) = \min(\ell(u), \ell(v))$ and $(u, v) \in F^{(\ell(u,v))}$.*

Proof. Consider any edge $(u,v) \in F = \bigcup_{i=0}^{L} F^{(i)}$. W.l.o.g., suppose that $(u,v) \in F^{(j)}$ for some $j \in [0,L]$. Before the start of round 0, we have $(u,v) \in E^{(\geq 0)}$. During each round $i \in [0, j-1]$, the edge (u,v) gets included in the set $E^{(\geq i+1)}$, and both its endpoints u, v get included in the set $V^{(\geq i+1)}$. At round j, one of its endpoints (say, u) gets included in $V^{(j)}$, and the edge (u,v) also gets included in $F^{(j)}$. Since $F^{(j)} \subseteq E^{(\geq j+1)} \setminus E^{(\geq j)}$, we infer that $\ell(u) = j$, $\ell(v) \geq j$, and $\ell(u,v) = \max\left\{i \in [0,L] : (u,v) \in E^{(\geq i)}\right\} = j = \min(\ell(u), \ell(v))$. ◀

▶ **Observation 13.** *For every edge $(u,v) \in F$, we have $h(u,v) = 2^{\ell(u,v)} \cdot \lambda$.*

Proof. Suppose that $\ell(u,v) = i \in [0,L]$, and hence $(u,v) \in F^{(i)}$. Before the start of round 0, we have $(u,v) \in E^{(\geq 0)}$ and $\gamma^{(0)}(u,v) = \lambda$. During each round $j \in [0, i-1]$, the edge (u,v) gets included in $E^{(\geq j+1)}$ and we double its weight, i.e., we set $\gamma^{(j+1)}(u,v) := 2 \cdot \gamma^{(j)}(u,v)$. Thus, at the start of round i, we have $(u,v) \in E^{(\geq i)}$ and $\gamma^{(i)}(u,v) = 2^i \cdot \lambda$. During round i, the edge (u,v) gets included in the set $F^{(i)}$ and its weight is frozen for the subsequent rounds, so that we get: $2^i \cdot \lambda = \gamma^{(i)}(u,v) = \gamma^{(i+1)}(u,v) = \cdots = \gamma^{(L)}(u,v) = h(u,v)$. ◀

▶ **Lemma 14.** *The graph $H = (V,F)$ has arboricity at most $O\left(\epsilon^{-1} + \beta^{-1}\right) = O(\log n)$.*

Proof. For any nodes $u, v \in V$ with $\ell(u) = \ell(v) = i < L$, we say that u was assigned its level *before* v iff we had $u \in V^{(i)}$ just before the iteration of the inner WHILE loop in Algorithm 2 which adds v to $V^{(i)}$. We now define the following orientation of the graph $H = (V,F)$:

- Consider any edge $(u,v) \in F$. W.l.o.g. suppose that $\ell(u) \leq \ell(v)$. If $\ell(u) < \ell(v)$, then the edge is orientated *from u towards v*. Otherwise, if $\ell(u) = \ell(v) = L$, then the edge is oriented in any arbitrary direction. Finally, if $\ell(u) = \ell(v) < L$ and (say) the node u was assigned its level before the node v, then the edge is oriented *from u towards v*.

Fix any node $x \in V$. Define $\text{Out}_F(x) := \{(x,y) \in F :$ the edge (x,y) is oriented away from $x\}$. We will show that $|\text{Out}_F(x)| \leq O\left(\epsilon^{-1} + \beta^{-1}\right)$. The lemma will then follow from (1).

Case 1: $\ell(x) = i < L$. Let $X^- \subseteq V^{(\geq i)}$ be the set of nodes in $V^{(\geq i)}$ that are assigned the level i *before* the node x. In words, the symbol X^- denotes the status of the set $V^{(i)}$ just before x gets added to $V^{(i)}$ in Algorithm 2. For every edge $(x,y) \in \text{Out}_F(x)$, we have $y \in V^{\geq i} \setminus X^-$ and $(x,y) \in E^{(\geq i)}$. Hence, it follows that $|\text{Out}_F(x)| \leq \deg_{E^{(\geq i)}}(x, V^{(\geq i)} \setminus X^-) \leq \epsilon^{-1}$.

Case 2: $\ell(x) = L$. Consider any edge $(x,y) \in \text{Out}_F(x)$. Clearly, this implies that $\ell(y) = L$, and hence $\ell(x,y) = L$ by Observation 12. Thus, by Observation 13 we have $h(x,y) = 2^L \cdot \lambda \geq \beta/2$. In other words, $h(x,y) \geq \beta/2$ for all $(x,y) \in \text{Out}_F(x)$. Now, part-(2) of Lemma 11 implies that: $w(x) = \Omega(h(x)) = \Omega\left(\sum_{(x,y) \in \text{Out}_F(x)} h(x,y)\right) = \Omega\left(|\text{Out}_F(x)| \cdot (\beta/2)\right)$. Accordingly, we get: $w(x) = \Omega\left(|\text{Out}_F(x)| \cdot \beta\right)$, and hence: $|\text{Out}_F(x)| = O\left(\beta^{-1} \cdot w(x)\right) = O(\beta^{-1})$. The last inequality holds since $w(x) \leq 1$. ◀

To summarize, our static algorithm runs in linear time (Lemma 9), returns a subgraph $H = (V,F)$ with bounded arboricity (Lemma 14), and this subgraph H admits a fractional matching that closely approximates the input λ-uniform matching w in G (Lemma 11).

4 Dynamically Sparsifying a Uniform Fractional Matching

In this section, we will present a dynamic algorithm for sparsifying a uniform fractional matching, which will be referred to as DYNAMIC-UNIFORM-SPARSIFY$(G = (V,E), \lambda)$. The input to this algorithm is a dynamic graph $G = (V,E)$ that keeps changing via a sequence of updates (edge insertions/deletions), and a fixed parameter $\delta/n^2 \leq \lambda < \beta$. Throughout this

sequence of updates, it is guaranteed that the graph G admits a valid λ-uniform fractional matching w. We will show how to maintain a subgraph $H_{(a)} = (V, F_{(a)})$ of this dynamic graph $G = (V, E)$, with $F_{(a)} \subseteq E$, that is a good matching-sparsifier of G with respect to w.

Our dynamic algorithm will be heavily based on the static algorithm from Section 3. We now introduce a couple of (informal) terms that relate to various aspects of this static algorithm. These terms will be very useful in the ensuing discussion. First, for each $i \in [0, L]$, the term *level-i-structure* will refer to the following sets: $E^{(\geq i)}, V^{(\geq i)}, V^{(i)}$ and $F^{(i)}$. Second, the term *hierarchy* will refer to the union of the level-i-structures over all $i \in [0, L]$.

We will maintain a partition of the edge-set E into two subsets: $E_{(a)}$ and $E_{(p)}$. The edges in $E_{(a)}$ (resp., $E_{(p)}$) will be called *active* (resp., *passive*). We will let $G_{(a)} := (V, E_{(a)})$ and $G_{(p)} := (V, E_{(p)})$ respectively denote the active and passive subgraphs of the input graph $G = (V, E)$. Our dynamic algorithm will make a *lazy attempt* at mimicking the static algorithm from Section 3, when the latter receives the active subgraph $G_{(a)}$ as input.

Preprocessing. At preprocessing, we set $E_{(p)} := \emptyset$ and $E_{(a)} := E$, and then call Static-Uniform-Sparsify$(G_{(a)} = (V, E_{(a)}), \lambda)$, as described in Algorithm 1. It returns the hierarchy, where for each $i \in [0, L]$ the level-i-structure consists of $E^{(\geq i)}, V^{(\geq i)}, F^{(i)}, V^{(i)}$. Finally, for each $i \in [0, L]$, we initialize a set $D^{(\geq i)} := \emptyset$. This concludes the preprocessing step.

Handling an edge-insertion. When an edge e gets inserted into the input graph $G = (V, E)$, we call the subroutine Handle-Insertion(e, λ), as described in Algorithm 4. This classifies the edge e as passive, and sets $E_{(p)} \leftarrow E_{(p)} \cup \{e\}$. If the previous step does not violate Invariant 1, then we are done. Otherwise, if Invariant 1 gets violated, then we throw away the existing hierarchy and all its associated structures (such as the sets $D^{(\geq i)}$), and perform the preprocessing step again on the current input graph G.

▶ **Invariant 1.** $|E_{(p)}| \leq \epsilon \cdot |E_{(a)}|$.

Handling an edge-deletion. When an edge e gets deleted from G, we call the subroutine Handle-Deletion(e, λ), as described in Algorithm 6. If e was already passive, then it simply gets removed from the set $E_{(p)}$, and we are done. Henceforth, we assume that e was active, and at level $\ell(e) = k$, just before getting deleted.[4]

First, we remove e from the set $E_{(a)}$, because the edge is no longer present in G. Next, for every $i \in [0, k]$, we insert e into the set $D^{(\geq i)}$. From now on, we will refer to e as a *dead edge*. Intuitively, the edge e, even after getting deleted, continues to be present in the level-i-structure for each $i \in [0, k]$. Thus, up until this point, the hierarchy does not change.

Next, we check if the previous steps lead to a violation of Invariant 2. If Invariant 2 continues to remain satisfied, then we are done. Otherwise, we find the minimum index $j \in [0, L]$ such that $\left|D^{(\geq j)}\right| > \epsilon \cdot |E^{(\geq j)}|$, and then perform the following operations: (1) For every $i \in [j, L]$, we delete the dead edges $D^{(\geq i)}$ from the level-i-structure and reset $D^{(\geq i)} \leftarrow \emptyset$. (2) Finally, we call the subroutine Rebuild(j, λ) as described in Algorithm 2.

▶ **Invariant 2.** $\left|D^{(\geq i)}\right| \leq \epsilon \cdot \left|E^{(\geq i)}\right|$ *for all* $i \in [0, L]$.

[4] See the paragraph just before Observation 12 for the definition of the level of an edge.

Algorithm 4 HANDLE-INSERTION(e, λ).

$E_{(p)} \leftarrow E_{(p)} \cup \{e\}$.
if $\left|E_{(p)}\right| > \epsilon \cdot \left|E_{(a)}\right|$ **then**
  Call the subroutine CLEAN-UP$(0, \lambda)$.
  $E_{(a)} \leftarrow E_{(a)} \cup E_{(p)}$.
  $E_{(p)} \leftarrow \emptyset$.
  Call the subroutine STATIC-UNIFORM-SPARSIFY$\big(G_{(a)} := (V, E_{(a)}), \lambda\big)$.

Algorithm 5 CLEAN-UP(j, λ).

for all $i = j$ to L **do**
  $E^{(\geq i)} \leftarrow E^{(\geq i)} \setminus D^{(\geq i)}$.
  $F^{(i)} \leftarrow F^{(i)} \setminus D^{(\geq i)}$.
  $D^{(\geq i)} \leftarrow \emptyset$.

Algorithm 6 HANDLE-DELETION(e, λ).

if $e \in E_{(p)}$ **then**
  $E_{(p)} \leftarrow E_{(p)} \setminus \{e\}$.
else
  $k \leftarrow \ell(e) := \max\left\{i \in [0, L] : e \in E^{(\geq i)}\right\}$.
  $E_{(a)} \leftarrow E_{(a)} \setminus \{e\}$.
  for $i = 0$ to k **do**
    $D^{(\geq i)} \leftarrow D^{(\geq i)} \cup \{e\}$.
  if $\left|D^{(\geq i)}\right| > \epsilon \cdot \left|E^{(\geq i)}\right|$ for some index $i \in [0, L]$ **then**
    Let j be the minimum index $i \in [0, L]$ for which $\left|D^{(\geq i)}\right| > \epsilon \cdot \left|E^{(\geq i)}\right|$.
    Call the subroutine CLEAN-UP(j, λ).
    Call the subroutine REBUILD(j, λ).

Note that $E_{(p)}, E_{(a)}, E^{(\geq i)}, D^{(\geq i)}, F^{(i)}$ represent global variables in these pseudocodes.

To summarize, we satisfy Invariant 1 and Invariant 2, and handle the updates to G in a lazy manner. Newly inserted edges are classified as passive, and they are completely ignored in the hierarchy unless their number becomes sufficiently large compared to the total number of active edges, at which point we rebuild everything from scratch. In contrast, when an active edge gets deleted from some level $i \in [0, L]$, it is classified as dead and it continues to be present in the level-j-structure for all $j \in [0, i]$. Finally, if we notice that for some $k \in [0, L]$ the level-k-structure has too many dead edges $D^{(\geq k)}$, then we remove all the dead edges from every level-j-structure with $j \in [k, L]$, and rebuild these structures from scratch.

From Section 3, recall that $F := \bigcup_{i=0}^{L} F^{(i)}$. For any set of edges E', we will use the notation $E'_{(a)} := E' \cap E_{(a)}$ to denote the subset of edges in E' that are active in the current input graph $G = (V, E)$. Accordingly, we define $F_{(a)} := F \cap E_{(a)}$ and $H_{(a)} := (V, F_{(a)})$.

Lemma 15 and Lemma 16 below should respectively be seen as analogues of Lemma 14 and Lemma 11 from Section 3. They show that the subgraph $H_{(a)} = (V, F_{(a)})$ is a good matching sparsifier of the input dynamic graph $G = (V, E)$. Intuitively, Lemma 15 and Lemma 16 hold because Invariant 1 and Invariant 2 ensure that throughout the sequence of updates, the hierarchy maintained by our dynamic algorithm is *very close* to the hierarchy

constructed by the algorithm from Section 3 when it receives the current graph $G = (V, E)$ as input. Due to space constraints, the proofs of these two lemmas are deferred to the full version.

▶ **Lemma 15.** *The graph $H_{(a)} = (V, F_{(a)})$ has arboricity at most $O(\epsilon^{-1} + \beta^{-1}) = O(\log n)$.*

▶ **Lemma 16.** *The graph $H_{(a)}$ admits a fractional matching $h' : F_{(a)} \to [0, 1]$ such that:*
1. *For every edge $e \in F_{(a)}$, we have $h'(e) < \beta$.*
2. *For every node $v \in V$, we have $h'(v) \leq w(v)$.*
3. *We have $size(w) \leq (1 + 60\epsilon \cdot \log(\beta/\lambda)) \cdot size(h')$.*

▶ **Lemma 17.** *The dynamic algorithm* Dynamic-Uniform-Sparsify(G, λ) *has an amortized update time of $O\left(\epsilon^{-1} \cdot \log(\beta/\lambda)\right) = O(\log^2 n)$.*

Proof. Define a potential function $\Phi := |E_{(p)}| + \sum_{i=0}^{L} \left|D^{(\geq i)}\right|$. Insertion of an edge increases the potential Φ by at most one unit, as the newly inserted edge gets classified as passive. On the other hand, deletion of an edge e increases the potential Φ by at most $L+1$ units, since e gets added to each of the sets $D^{(\geq 0)}, \ldots, D^{(\geq \ell(e))}$, and $\ell(e) \leq L$. To summarize, each update in G creates at most $O(L)$ units of new potential. We will show that whenever our dynamic algorithm spends T units of time, the potential Φ drops by at least $\Omega(\epsilon \cdot T)$. Since Φ is always ≥ 0, this implies the desired amortized update time of $O(\epsilon^{-1} \cdot L) = O\left(\epsilon^{-1} \cdot \log(\beta/\lambda)\right)$.

Consider the insertion of an edge e into $G = (V, E)$, and suppose that we call Static-Uniform-Sparsify$(G_{(a)}, \lambda)$ while handling this insertion. Let $m_{(a)} := |E_{(a)}|$, $m_{(p)} := |E_{(p)}|$, $m_{(d)} := |D^{(\geq 0)}|$ and $m := |E|$, just before e gets inserted. Invariant 1 and Invariant 2 respectively ensure that $m_{(p)} = \epsilon \cdot m$ and $m_{(d)} \leq \epsilon \cdot m$. By Lemma 9, the call to Static-Uniform-Sparsify$(G_{(a)}, \lambda)$ takes $O(m)$ time. Thus, the total time to handle this edge insertion is given by $T := O(m + m_d) = O(m)$. On the other hand, when our algorithm finishes handling this edge insertion, we have $E_{(p)} = \emptyset$, and hence the potential Φ decreases by at least $m_{(p)} = \epsilon \cdot m$ units. In other words, the drop in the potential Φ is at least $\Omega(\epsilon \cdot T)$.

Next, consider the deletion of an edge e from $G = (V, E)$, and suppose that while handling this deletion we call the subroutine Rebuild(k, λ) for some $k \in [0, L]$. Just before e gets deleted, let $m_{(d)}^{(\geq k)} := \left|D^{(\geq k)}\right|$ and $m^{(\geq k)} := E^{(\geq k)}$. Invariant 2 ensures that $m_{(d)}^{(\geq k)} = \epsilon \cdot m^{(\geq k)}$. By Lemma 9, the call to Rebuild(k, λ) takes $O\left(m^{(\geq k)}\right)$ time. Hence, excluding the time it takes to identify the level k, which is $O(L) = O(\log(\beta/\lambda)) = O(\log n)$ in the worst-case, our dynamic algorithm spends $T = O\left(m^{(\geq k)}\right)$ time to handle this edge deletion. On the other hand, this decreases the potential Φ by at least $m_{(d)}^{(\geq k)} = \epsilon \cdot m^{(\geq k)}$, since once we are done processing this edge deletion, we have $D^{(\geq k)} = \emptyset$. So the potential Φ drops by $\Omega(\epsilon \cdot T)$. ◀

5 Dynamically Sparsifying an Arbitrary Fractional Matching

In this section, we briefly sketch our dynamic algorithm for maintaining a matching-sparsifier as specified by Theorem 1.

The input is a dynamic graph $G = (V, E)$ with n nodes, and a (*not* necessarily uniform) fractional matching $w : E \to [0, 1]$ in G. An "update" either inserts/deletes an edge in G or changes the weight $w(e)$ of an existing edge e in G. Our algorithm works in three steps.

Step I (Discretizing w). For every integer $j \geq 0$, define $\lambda_j := (\beta/n^2) \cdot (1 + \beta)^j$. Let K be the largest integer j such that $\lambda_j < \beta$. We now discretize w to get a new fractional matching $\hat{w} : E \to [0, 1]$, which is defined as follows. Consider any edge $e \in E$. If $w(e) < \lambda_0$, then $\hat{w}(e) := 0$. Else if $\lambda_0 \leq w(e) < \beta$, then $\hat{w}(e) := \lambda_i$ where i is the unique integer such that $\lambda_i \leq w(e) < \lambda_{i+1}$. Otherwise, if $w(e) \geq \beta$, then $\hat{w}(e) := w(e)$.

For each $i \in [0, K]$, let E_i denote the subset of edges $e \in E$ with $\hat{w}(e) = \lambda_i$, let $G_i := (V, E_i)$, and let $w_i : E_i \to [0, 1]$ be the restriction of the fractional matching $\hat{w}$ onto the set E_i (i.e., w_i is a λ_i-uniform fractional matching in G_i). Finally, define the subset of edges $E_{\geq\beta} := \{e \in E : \hat{w}(e) = w(e) \geq \beta\}$, and let $G_{\geq\beta} := (V, E_{\geq\beta})$. In the dynamic setting, we can easily maintain the subgraphs $G_0, \ldots, G_K, G_{\geq\beta}$ of the input graph G *on the fly*.

Step II (Sparsifying each G_i). For each $i \in [0, K]$, we maintain a sparsifier of G_i with respect to w_i, with the help of the dynamic algorithm from Section 4. Specifically, let $H_i = (V, F_i)$ denote the sparsifier $H_{(a)} = (V, F_{(a)})$ maintained by the algorithm Dynamic-Uniform-Sparsify(G_i, λ_i) from Section 4 (thus, we have $F_i \subseteq E_i$).

Step III (Putting everything together). Let $E_S := \bigcup_{i=0}^{K} F_i \bigcup E_{\geq\beta}$. In the full version of the paper, we show that the subgraph $S := (V, E_S)$ of G satisfies all the five conditions stated in Theorem 1.

References

1. Amir Abboud, Raghavendra Addanki, Fabrizio Grandoni, Debmalya Panigrahi, and Barna Saha. Dynamic set cover: improved algorithms and lower bounds. In *STOC*, 2019.
2. Amir Abboud and Virginia Vassilevska Williams. Popular conjectures imply strong lower bounds for dynamic problems. In *FOCS*, 2014.
3. Moab Arar, Shiri Chechik, Sarel Cohen, Cliff Stein, and David Wajc. Dynamic matching: Reducing integral algorithms to approximately-maximal fractional algorithms. In *ICALP*, 2018.
4. Sepehr Assadi and Aaron Bernstein. Towards a unified theory of sparsification for matching problems. In *SOSA*, 2019.
5. S. Baswana, M. Gupta, and S. Sen. Fully dynamic maximal matching in $O(\log n)$ update time. In *FOCS*, 2011.
6. Soheil Behnezhad, Mahsa Derakhshan, MohammadTaghi Hajiaghayi, Cliff Stein, and Madhu Sudan. Fully dynamic maximal independent set with polylogarithmic update time. In *FOCS*, 2019.
7. Soheil Behnezhad, Jakub Lacki, and Vahab S. Mirrokni. Fully dynamic matching: Beating 2-approximation in $O(\Delta^{\epsilon})$ update time. In *SODA*, 2020.
8. Aaron Bernstein, Sebastian Forster, and Monika Henzinger. A deamortization approach for dynamic spanner and dynamic maximal matching. In *SODA*, 2019.
9. Aaron Bernstein, Maximilian Probst Gutenberg, and Thatchaphol Saranurak. Deterministic decremental reachability, scc, and shortest paths via directed expanders and congestion balancing. In *FOCS*, 2020.
10. Aaron Bernstein and Cliff Stein. Fully dynamic matching in bipartite graphs. In *ICALP*, 2015.
11. Aaron Bernstein and Cliff Stein. Faster fully dynamic matchings with small approximation ratios. In *SODA*, 2016.
12. Sayan Bhattacharya, Deeparnab Chakrabarty, and Monika Henzinger. Deterministic fully dynamic approximate vertex cover and fractional matching in O(1) amortized update time. In *IPCO*, 2017.
13. Sayan Bhattacharya, Monika Henzinger, and Giuseppe F. Italiano. Deterministic fully dynamic data structures for vertex cover and matching. In *SODA*, 2015.
14. Sayan Bhattacharya, Monika Henzinger, and Danupon Nanongkai. New deterministic approximation algorithms for fully dynamic matching. In *STOC*, 2016.
15. Sayan Bhattacharya, Monika Henzinger, and Danupon Nanongkai. Fully dynamic approximate maximum matching and minimum vertex cover in $O(\log^3 n)$ worst case update time. In *SODA*, 2017.

16 Sayan Bhattacharya and Janardhan Kulkarni. Deterministically maintaining a $(2+\epsilon)$-approximate minimum vertex cover in $O(1/\epsilon^2)$ amortized update time. In *SODA*, 2019.

17 Moses Charikar and Shay Solomon. Fully dynamic almost-maximal matching: Breaking the polynomial barrier for worst-case time bounds. In *ICALP*, 2018.

18 Julia Chuzhoy, Yu Gao, Jason Li, Danupon Nanongkai, Richard Peng, and Thatchaphol Saranurak. A deterministic algorithm for balanced cut with applications to dynamic connectivity, flows, and beyond. In *FOCS*, 2020.

19 Ran Duan, Haoqing He, and Tianyi Zhang. Dynamic edge coloring with improved approximation. In Timothy M. Chan, editor, *SODA*, 2019.

20 Jacob Evald, Viktor Fredslund-Hansen, Maximilian Probst Gutenberg, and Christian Wulff-Nilsen. Decremental APSP in directed graphs versus an adaptive adversary. *CoRR*, abs/2010.00937, 2020. `arXiv:2010.00937`.

21 Gramoz Goranci, Harald Räcke, Thatchaphol Saranurak, and Zihan Tan. The expander hierarchy and its applications to dynamic graph algorithms. In *SODA*, 2021.

22 Fabrizio Grandoni, Stefano Leonardi, Piotr Sankowski, Chris Schwiegelshohn, and Shay Solomon. $(1+\epsilon)$-approximate incremental matching in constant deterministic amortized time. In *SODA*, 2019.

23 Anupam Gupta, Ravishankar Krishnaswamy, Amit Kumar, and Debmalya Panigrahi. Online and dynamic algorithms for set cover. In *STOC*, 2017.

24 Manoj Gupta. Maintaining approximate maximum matching in an incremental bipartite graph in polylogarithmic update time. In *FSTTCS*, 2014.

25 Manoj Gupta and Richard Peng. Fully dynamic $(1+\epsilon)$-approximate matchings. In *FOCS*, 2013.

26 Monika Henzinger and Valerie King. Randomized dynamic graph algorithms with polylogarithmic time per operation. In *STOC*, 1995.

27 Monika Henzinger, Sebastian Krinninger, Danupon Nanongkai, and Thatchaphol Saranurak. Unifying and strengthening hardness for dynamic problems via the online matrix-vector multiplication conjecture. In *STOC*, 2015.

28 Jacob Holm, Kristian de Lichtenberg, and Mikkel Thorup. Poly-logarithmic deterministic fully-dynamic algorithms for connectivity, minimum spanning tree, 2-edge, and biconnectivity. *J. ACM*, 48(4):723–760, 2001.

29 Tsvi Kopelowitz, Robert Krauthgamer, Ely Porat, and Shay Solomon. Orienting fully dynamic graphs with worst-case time bounds. In *ICALP*, 2014.

30 Tsvi Kopelowitz, Seth Pettie, and Ely Porat. Higher lower bounds from the 3sum conjecture. In *SODA*, 2016.

31 Kasper Green Larsen. The cell probe complexity of dynamic range counting. In Howard J. Karloff and Toniann Pitassi, editors, *STOC*, 2012.

32 Danupon Nanongkai, Thatchaphol Saranurak, and Christian Wulff-Nilsen. Dynamic minimum spanning forest with subpolynomial worst-case update time. In *FOCS*, 2017.

33 C St JA Nash-Williams. Edge-disjoint spanning trees of finite graphs. *Journal of the London Mathematical Society*, 1(1):445–450, 1961.

34 Ofer Neiman and Shay Solomon. Simple deterministic algorithms for fully dynamic maximal matching. In *STOC*, 2013.

35 Krzysztof Onak and Ronitt Rubinfeld. Maintaining a large matching and a small vertex cover. In *STOC*, 2010.

36 Mihai Patrascu. Towards polynomial lower bounds for dynamic problems. In *STOC*, 2010.

37 David Peleg and Shay Solomon. Dynamic $(1+\epsilon)$-approximate matchings: A density-sensitive approach. In *SODA*, 2016.

38 Piotr Sankowski. Faster dynamic matchings and vertex connectivity. In *SODA*, 2007.

39 Shay Solomon. Fully dynamic maximal matching in constant update time. In *FOCS*, 2016.

40 David Wajc. Rounding dynamic matchings against an adaptive adversary. In *STOC*, 2020.

Traveling Repairperson, Unrelated Machines, and Other Stories About Average Completion Times

Marcin Bienkowski
Institute of Computer Science, University of Wrocław, Poland

Artur Kraska
Institute of Computer Science, University of Wrocław, Poland

Hsiang-Hsuan Liu
Utrecht University, The Netherlands

Abstract

We present a unified framework for minimizing average completion time for many seemingly disparate *online* scheduling problems, such as the traveling repairperson problems (TRP), dial-a-ride problems (DARP), and scheduling on unrelated machines.

We construct a simple algorithm that handles all these scheduling problems, by computing and later executing auxiliary schedules, each optimizing a certain function on already seen prefix of the input. The optimized function resembles a prize-collecting variant of the original scheduling problem. By a careful analysis of the interplay between these auxiliary schedules, and later employing the resulting inequalities in a factor-revealing linear program, we obtain improved bounds on the competitive ratio for all these scheduling problems.

In particular, our techniques yield a 4-competitive deterministic algorithm for all previously studied variants of online TRP and DARP, and a 3-competitive one for the scheduling on unrelated machines (also with precedence constraints). This improves over currently best ratios for these problems that are 5.14 and 4, respectively. We also show how to use randomization to further reduce the competitive ratios to $1 + 2/\ln 3 < 2.821$ and $1 + 1/\ln 2 < 2.443$, respectively. The randomized bounds also substantially improve the current state of the art. Our upper bound for DARP contradicts the lower bound of 3 given by Fink et al. (Inf. Process. Lett. 2009); we pinpoint a flaw in their proof.

2012 ACM Subject Classification Theory of computation → Online algorithms; Theory of computation → Scheduling algorithms

Keywords and phrases traveling repairperson problem, dial-a-ride, machine scheduling, unrelated machines, minimizing completion time, competitive analysis, factor-revealing LP

Digital Object Identifier 10.4230/LIPIcs.ICALP.2021.28

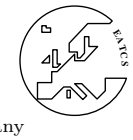
Category Track A: Algorithms, Complexity and Games

Funding Supported by Polish National Science Centre grant 2016/22/E/ST6/00499.

1 Introduction

In the traveling repairperson problem (TRP) [37], requests arrive in time at points of a metric space and they need to be eventually serviced. In the same metric, there is a mobile server, that can move at a constant speed. The server starts at a distinguished point called the origin. A request is considered serviced once the server reaches its location; we call such time its *completion time*. The goal is to minimize the sum (or equivalently the average) of all completion times. We focus on a weighted variant, where all requests have non-negative weights and the goal is to minimize the weighted sum of completion times.

A natural and well-studied extension of the TRP problem is a so-called *dial-a-ride problem* (DARP) [20], where each request has a source and a destination and the goal is to transport an object between these two points. There, the server may have a fixed capacity limiting



the number of objects it may carry simultaneously; this capacity may be also infinite. For the finite-capacity case, one can also distinguish between preemptive variant, where objects can be unloaded at some points of the metric space (different than their destination) and non-preemptive variant, where such unloading is not allowed.

A seemingly disparate problem is scheduling on m unrelated machines [23]. There, weighted jobs arrive in time, each with a vector of size m describing execution times of the job when assigned to a given machine. A single machine can execute at most one job at a time. The goal is to assign each job (at or after its arrival) to one of the machines to minimize the weighted sum of completion times. This problem comes in two flavors: in the preemptive one, job execution may be interrupted and picked up later, while in the non-preemptive one, such interruption is not possible. As an extension, each job may have precedence constraints, i.e., can be executed only once some other jobs are completed.

Online Algorithms. Our focus is on natural *online* scenarios of TRP, DARP [21], and machine scheduling [24]. There, an online algorithm ALG, at time t, knows only requests/jobs that arrived before or at time t. The number of requests/jobs is also not known by an algorithm a priori. We say that an online algorithm ALG is c-competitive if for any request/job sequence $\mathcal{I}$ it holds that $\text{COST}_{\text{ALG}}(\mathcal{I}) \leq c \cdot \text{COST}_{\text{OPT}}(\mathcal{I})$, where OPT is a cost-optimal *offline* solution for $\mathcal{I}$. For a randomized algorithm ALG, we replace its cost by its expectation. The competitive ratio of ALG is the infimum over all values c such that ALG is c-competitive [15].

In this paper, we present a unified framework for handling such online scheduling problems where the cost is the weighted sum of completion times. We present an algorithm MIMIC that yields substantially improved competitive ratios for all the problems described above.

1.1 Previous Work

The currently best algorithms for the TRP, the DARP, and machine scheduling on unrelated machines share a common framework. Namely, each of these algorithms works in phases of geometrically increasing lengths. In each phase, it computes and executes an auxiliary schedule for the requests presented so far. (In the case of the TRP and DARP, the server additionally returns to the origin afterward.) The auxiliary schedule optimizes a certain function, such as maximizing the weight of served requests [8,16,24,28,32,33] or minimizing the sum of completion times with an additional penalty for non-served requests [27].[1] Moreover, known randomized algorithms are also based on a common idea: they delay the execution of the deterministic algorithm by a random offset [16, 27, 32, 33]. We call these approaches *phase based*. The currently best results are gathered in Table 1.

Traveling Repairperson and Dial-a-Ride Problems. The online variant of the TRP has been first investigated by Feuerstein and Stougie [21]. By adapting an algorithm for the cow-path problem problem [7], they gave a 9-competitive solution for line metrics. The result has been improved by Krumke et al. [32], who gave a phase-based deterministic algorithm INTERVAL attaining competitive ratio of $3 + 2\sqrt{2} < 5.829$ for an arbitrary metric space. A slightly different algorithm with the same competitive ratio was given by Jaillet and Wagner [28]. Bienkowski and Liu [8] applied postprocessing to auxiliary schedules, serving heavier requests

[1] Computing such auxiliary schedule usually involves optimally solving an NP-hard task. This is typical for the area of online algorithms, where the focus is on information-theoretic aspects and not on computational complexity. Algorithms presented in this paper also aim at minimizing the achievable competitive ratio rather than minimizing the running time.

earlier, and improved the ratio to 5.429 on line metrics. Finally, Hwang and Jaillet proposed a phase-based algorithm Plan-And-Commit [27]. They give a computer-based upper bound of 5.14 for the competitive ratio and an analytical upper bound of 5.572.

Randomized counterparts of algorithms Interval and Plan-And-Commit achieve ratios of 3.874 [32, 33] and 3.641 [27], respectively. Interestingly, the latter bound is not a direct randomization of the deterministic algorithm, but uses a different parameterization, putting more emphasis on penalizing requests not served by auxiliary schedules.

The phase-based algorithm Interval extends in a straightforward fashion to the DARP problem with an arbitrary assumption on the server capacity, both for the preemptive and non-preemptive variants: all the details of the solved problem are encapsulated in the computations of auxiliary schedules [32]. In the same manner, Interval can be enhanced to handle k-TRP and k-DARP variants, where an algorithm has k servers at its disposal (also for any k, any server capacities, and any preemptiveness assumptions) [14]. Although this was not explicitly stated in [27], the algorithm Plan-And-Commit can be extended in the same way.

From the impossibility side, Feuerstein and Stougie [21] gave a lower bound for the TRP (that also holds already for a line) of $1 + \sqrt{2} > 2.414$, while the bound of 7/3 for randomized algorithms was presented by Krumke et al. [32]. For the variant of the TRP with multiple servers, the deterministic lower bound is only 2 [14] (it holds for any number of servers). Clearly, all these lower bounds hold also for any variant of DARP. For the DARP with a single server of capacity 1, the deterministic lower bound can be improved to 3 [21] and the randomized one to 2.410 [32].

The authors of [22] claimed a lower bound of 3 for randomized k-DARP (for any k). This contradicts the upper bound we present in this paper. In Section 7, we pinpoint a flaw in their argument.

TRP and DARP: Related Results. Both online TRP and DARP problems were considered under different objectives, such as minimizing the total makespan (when the TRP becomes online TSP) [3–6, 9–11, 13, 18, 29, 30, 35] or maximum flow time [25, 31, 34].

The offline variants of TRP and DARP have been extensively studied both from the computational hardness (see, e.g., [20, 37]) and approximation algorithms perspectives. In particular, the TRP, also known as the *minimum latency problem* problem, is NP-hard already on weighted trees [40] (where the closely related traveling salesperson problem [12] becomes trivial) and the best known approximation factor in general graphs is 3.59 [17]. For some metrics (Euclidean plane, planar graphs or weighted trees) the TRP admits a PTAS [2, 42].

Machine Scheduling on Unrelated Machines. The first online algorithm for the scheduling on unrelated machines ($R|r_j|\sum w_jC_j$ in the Graham et al. notation [23]) was given by Hall et al. [24]. They gave 8-competitive polynomial-time algorithm, which would be 4-competitive if the polynomial-time requirement was lifted. Chakrabarti et al. showed how to randomize this algorithm, achieving the ratio of $2/\ln 2 < 2.886$ [16]. They also observe that both algorithms can handle precedence constraints. The currently best deterministic lower of 1.309 is due to Vestjens [45], and the best randomized one of 1.157 is due to Seiden [39].

Machine Scheduling: Related Results. While for unrelated machines, the results have not been beaten for the last 25 years, the competitive ratios for simpler models were improved substantially. For example, for parallel identical machines, a sequence of papers lowered the ratio to 1.791 [19, 36, 38, 41].

Table 1 Previous and current bounds on the competitive ratios for the TRP and the DARP problems. Asterisked results were not given in the referenced papers, but they are immediate consequences of the arguments therein. All upper bounds for the TRP/DARP variants hold for any number k of servers, any server capacities, both in the preemptive and the non-preemptive case. Upper bounds for scheduling hold also in the presence of precedence constraints. Bounds proven in the current paper are given in boldface.

	deterministic		randomized	
	lower	upper	lower	upper
TRP	2.414 [21]	5.14 [27]	2.333 [32]	3.641 [27]
DARP	3 [21]	5.14* [27]	2.410 [32]	3.641* [27]
k-TRP	2 [14]	5.14* [27]	2 [14]	3.641* [27]
k-DARP	2 [14]	5.14* [27]	2 [14]	3.641* [27]
k-TRP, k-DARP (all variants)		**4**		**2.821**
scheduling on unrelated machines	1.309 [45]	4 [24] **3**	1.157 [39]	2.886 [16] **2.443**

The problem has also been studied intensively in the offline regime. Both weighted preemptive and non-preemptive variants were shown to be APX-hard [26, 43]. On the positive side, a 1.698-approximation for the preemptive case was given by Sitters [43], and a 1.5-approximation for the non-preemptive case by Skutella [44]. A PTAS for a constant number of machines is due to Afrati et al. [1].

1.2 Resettable Scheduling

The phase-based algorithms for DARP variants and machine scheduling on unrelated machines both execute auxiliary schedules, but the ones for the DARP variants need to bring the server back to the origin between schedules. We call the latter action *resetting*. To provide a single algorithm for all these scheduling variants, we define a class of *resettable scheduling* problems.

We assume that jobs are handled by an *executor*, which has a set of possible states. And at time 0, it is in a distinguished *initial state*. An input to the problem consists of a sequence of jobs $\mathcal{I}$ released over time. Each job r is characterized by its arrival time $a(r)$, its weight $w(r)$, and possibly other parameters that determine its execution time. The executor cannot start executing job r before its arrival time $a(r)$. We will slightly abuse the notation and use $\mathcal{I}$ to also denote the *set* of all jobs from the input sequence. There is a problem-specific way of executing jobs and we use $s_{\text{ALG}}(r)$ to denote the *completion time* of a job by an algorithm ALG. The cost of an algorithm is defined as the weighted sum of job completion times, $\text{COST}_{\text{ALG}}(\mathcal{I}) = \sum_{r\in\mathcal{I}} w(r) \cdot s_{\text{ALG}}(r)$.

For any time τ, let $\mathcal{I}_\tau$ be the set of jobs that appear till τ. An *auxiliary τ-schedule* is a problem-specific way of feasibly executing a subset of jobs from $\mathcal{I}_\tau$. Such schedule starts at time 0, terminates at time τ, and leaves no job partially executed. We require that the following properties hold for any resettable scheduling problem.

Delayed execution. At any time t, if the executor is in the initial state, it can execute an arbitrary auxiliary τ-schedule (for $\tau \le t$). Such action takes place in time interval $[t, t+\tau)$. Any job r that would be completed at time $z \in [0, \tau)$ by the τ-schedule started at time 0 is now completed exactly at time $t + z$ (unless it has been already executed before).

Resetting executor. Assume that at time t, the executor was in the initial state, and then executed a τ-schedule, ending at time $t+\tau$. Then, it is possible to *reset* the executor using extra $\gamma \cdot \tau$ time, where γ is a parameter characteristic to the problem. That is, at time $t + (1+\gamma) \cdot \tau$, the executor is again in its initial state.

Learning minimum. We define $\min(\mathcal{I})$ to be the earliest time at which Opt may complete some job. We require that the value of $\min(\mathcal{I})$ is learned by an online algorithm at or before time $\min(\mathcal{I})$ and that $\min(\mathcal{I}) > 0$.

We call scheduling problems that obey these restrictions γ-resettable.

Example 1: Machine Scheduling is 0-Resettable. For the machine scheduling problem, the executor is always in the initial state, and no resetting is necessary. As we may assume that processing of any job takes positive time, $\min(\mathcal{I}) > 0$ holds for any input $\mathcal{I}$.

Example 2: DARP Problems are 1-Resettable. For the DARP variants, the executor state is the position of the algorithm server, with the origin used as the initial state.[2] Jobs are requests for transporting objects and an auxiliary τ-schedule is a fixed path of length τ starting at the origin, augmented with actions of picking up and dropping particular objects.[3] It is feasible to execute a τ-schedule starting at any time t when the server is at the origin. In such case, jobs are completed with an extra delay of t. Furthermore, right after serving the τ-schedule, the distance between the server and the origin is at most τ. Thus, it is possible to reset the executor to the initial state within extra time $1 \cdot \tau$.

Finally, as we may assume that there are no requests that arrive at time 0 with both start and destination at the origin, $\min(\mathcal{I}) > 0$ for any input $\mathcal{I}$.

1.3 Our Contribution

In this paper, we provide a deterministic routine Mimic and its randomized version that solves any γ-resettable scheduling problem. It achieves a deterministic ratio of $3+\gamma$ and a randomized one of $1 + (1+\gamma)/\ln(2+\gamma)$.

That is, for 1-resettable scheduling problems (the DARP variants with arbitrary server capacity, an arbitrary number of servers, and both in the preemptive and non-preemptive setting, or the TRP problem with an arbitrary number of servers), this gives solutions whose ratios are at most 4 and $1+2/\ln 3 < 2.821$, respectively. For 0-resettable scheduling problems (that include scheduling on unrelated machines with or without precedence constraints), the ratios of our solutions are 3 and $1 + 1/\ln 2 < 2.443$.

In both cases, our results constitute a substantial improvement over currently best ratios as illustrated in Table 1. Our result for the scheduling on unrelated machines is the first improvement in the last 25 years for this problem.

Challenges and Techniques. Mimic works in phases of geometrically increasing lengths. At the beginning of each phase, at time τ, it computes an auxiliary τ-schedule that optimizes the total completion time of jobs seen so far with an additional penalty for non-completed jobs: they are penalized as if they were completed at time τ. Then, within the phase it executes this schedule and afterward it resets the executor. We obtain a randomized variant by delaying the start of Mimic by an offset randomly chosen from a *continuous* distribution.

[2] In the variants with k servers, the executor state is a k-tuple describing the positions of all servers.
[3] In the preemptive variants, preemption is allowed *inside* an auxiliary schedule, provided that after a τ-schedule terminates, each job is either completed or untouched.

Admittedly, this idea is not new, and in fact, when we apply MIMIC to the TRP problem, it becomes a slightly modified variant of PLAN-AND-COMMIT [27]. Hence, the main technical contribution of our paper is a careful and exact analysis of such an approach. The crux here is to observe several structural properties and relations among schedules produced by MIMIC in consecutive phases, carefully tracking the overlaps of the job sets completed by them. On this basis, and for a fixed number Q of phases, we construct a maximization linear program (LP), whose optimal value upper-bounds the competitive ratio of MIMIC. Roughly speaking, the LP encodes, in a sparse manner, an adversarially created input. To upper bound its value, we explicitly construct a solution to its dual (minimization) program and show that its value is at most 4 for any number of phases Q.

Bounding the competitive ratio for the randomized version of MIMIC is substantially more complicated as we need to combine the discrete world of an LP with uncountably many random choices of the algorithm. To tackle this issue, we consider an intermediate solution DISC which approximates the random choice of MIMIC to a given precision, choosing an offset randomly from a discrete set of M values. This way, we upper-bound the ratio of MIMIC by $1 + (1/M) \cdot \sum_{j=1}^{M} (2+\gamma)^{j/M}$. This bound holds for an arbitrary value of M, and thus by taking the limit, we obtain the desired bound on the competitive ratio. Interestingly, we use the same LP for analyzing both the deterministic and the randomized solution.

2 Deterministic and Randomized Algorithms: Routine MIMIC

To describe our approach for γ-resettable scheduling, we start with defining auxiliary schedules used by our routine MIMIC. The parameter γ will be used to define partitioning of time into phases. Both our deterministic and randomized solutions will run MIMIC, however, the randomized one will execute it for a random choice of parameters.

Auxiliary Schedules. As introduced already in Subsection 1.2, an (auxiliary) τ-schedule A describes a sequence of job executions, has the total duration τ, and may be executed whenever the executor is in the initial state. For the preemptive variants, we assume that once such a schedule terminates, each job is processed either completely or not at all.

For a fixed input $\mathcal{I}$, and a τ-schedule A, we use $R(A)$ to denote the set of jobs that would be served by A if it was executed from time 0, i.e., in the interval $[0, \tau)$. For any set of jobs $R \subseteq R(A)$, let

$$w(R) = \sum_{r \in R} w(r) \qquad \text{and} \qquad \textsc{cost}_A(R) = \sum_{r \in R} w(r) \cdot s_A(r). \tag{1}$$

Note that if a schedule A serves all jobs from the input ($R(A) = \mathcal{I}$), then $\textsc{cost}_A(R(A))$ coincides with the cost of an algorithm that executes schedule A at time 0.

Recall that $\mathcal{I}_\tau \subseteq \mathcal{I}$ denotes the set of jobs that arrive till time τ. For any τ-schedule A, we define its value as

$$\textsc{val}_\tau(A) = \textsc{cost}_A(R(A)) + \tau \cdot w\left(\mathcal{I}_\tau \setminus R(A)\right). \tag{2}$$

The value corresponds to the actual cost of completing jobs from $\mathcal{I}_\tau$ by schedule A in interval $[0, \tau)$, but we charge A for unprocessed jobs as if they were completed at time τ.

▶ **Definition 1.** *For any $\tau \geq 0$, let S_τ be the τ-schedule minimizing function $\textsc{val}_\tau$. Ties are broken arbitrarily, but in a deterministic fashion.*

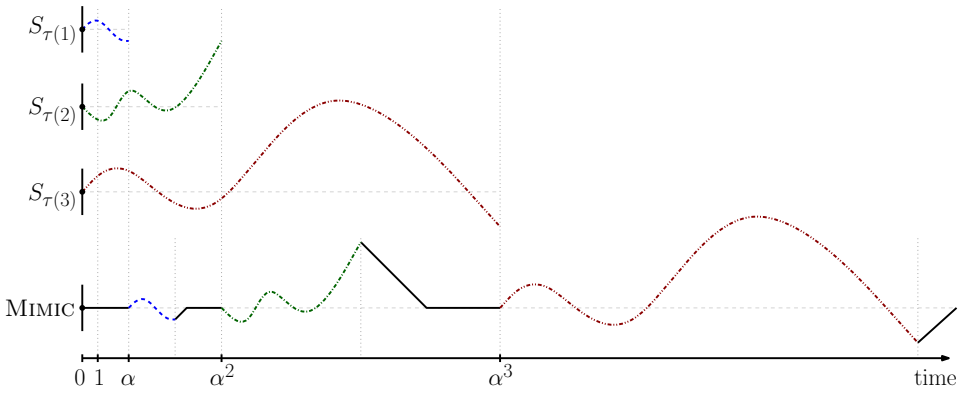


Figure 1 An example execution of algorithm MIMIC$(1,0)$ applied for the TRP problem (i.e., we use $\alpha = 3$). We assume that $\min(\mathcal{I}) = 1$. Within time interval $[\tau(k) = \alpha^k, 2 \cdot \alpha^k)$ of phase $k+1$, MIMIC executes a $\tau(k)$-schedule $S_{\tau(k)}$ that optimizes function $\text{VAL}_{\tau(k)}$. Afterwards within time interval $[2 \cdot \alpha^k, \tau(k+1) = 3 \cdot \alpha^k)$, MIMIC resets its state to the initial one (the server of TRP returns to the origin).

Routine MIMIC. For solving the γ-resettable scheduling problem, we define routine MIMIC(γ, ω), where $\omega \in (-1, 0]$ is an additional parameter that controls the initial delay.

- Our deterministic algorithm is simply MIMIC$(\gamma, 0)$.
- Our randomized algorithm first chooses a value ω uniformly at random from the range $(-1, 0]$. Then, it executes MIMIC(γ, ω).

Internally, MIMIC(γ, ω) uses a parameter $\alpha = 2 + \gamma$. It splits time into phases in the following way. For any k, let $\tau_k = \tau(k) = \min(\mathcal{I}) \cdot \alpha^{k+\omega}$. The k-th phase (for $k \geq 1$) starts at time $\tau_{k-1} = \min(\mathcal{I}) \cdot \alpha^{k-1+\omega}$ and ends at time $\tau_k = \min(\mathcal{I}) \cdot \alpha^{k+\omega}$. The time interval $[0, \tau_0) = [0, \alpha^\omega \cdot \min(\mathcal{I}))$ does not belong to any phase. As $\alpha^\omega \cdot \min(\mathcal{I}) \leq \min(\mathcal{I})$, no jobs can be completed within this interval, by the definition of $\min(\mathcal{I})$ (see Subsection 1.2).

MIMIC does nothing till the end of phase 1 (till time $\tau_1 = \alpha^{1+\omega} \cdot \min(\mathcal{I})$). Since $\omega \geq -1$, we have $\tau_1 \geq \min(\mathcal{I})$. As MIMIC learns the value of $\min(\mathcal{I})$ latest at time $\min(\mathcal{I})$, it can thus correctly identify the value of τ_1 before or at time τ_1.

For a phase $k+1$, where $k \geq 1$, MIMIC behaves in the following way. We ensure that at time τ_k, at the beginning of phase $k+1$, MIMIC is in its initial state. At this time, MIMIC computes the τ_k-schedule $S_{\tau(k)}$ (see Definition 1), executes it within time interval $[\tau_k, 2 \cdot \tau_k)$ and afterwards, it resets its state to the initial one. The execution of $S_{\tau(k)}$ will not be interrupted or modified when new jobs arrive within phase $k+1$. Furthermore, MIMIC serves only those requests from $S_{\tau(k)}$ it has not yet served earlier. The resetting part takes time $\gamma \cdot \tau_k$, and is thus finished at time $(2+\gamma) \cdot \tau_k = \alpha \cdot \tau_k = \tau_{k+1}$ when the next phase starts. An illustration is given in Figure 1.

3 Intermediate Algorithm DISC

As mentioned in the introduction, we introduce an additional intermediate algorithm DISC, whose analysis will allow us to bound the competitive ratios of both our deterministic and randomized solution. For an integer ℓ, we use $[\ell]$ to denote the set $\{0, \ldots, \ell - 1\}$.

DISC(γ, M, β) solves the γ-resettable scheduling problem, and is additionally parameterized by a positive integer M, and a real number $\beta \in (0, 1/M]$. DISC(γ, M, β) first chooses a random integer $m \in [M]$. Then, it executes MIMIC$(\gamma, \omega = -1 + m/M + \beta)$. The main result of this paper is the following bound, whose proof is will be given in the next two sections.

▶ **Theorem 2.** *For any γ, any positive integer M, and any $\beta \in (0, 1/M]$, the competitive ratio of $\textsc{Disc}(\gamma, M, \beta)$ for the γ-resettable scheduling is at most $1 + (1/M) \cdot \sum_{j=1}^{M} (2+\gamma)^{j/M}$.*

▶ **Corollary 3.** *For any γ, the competitive ratio of our Mimic-based deterministic solution is at most $3 + \gamma$ and the ratio of randomized one at most $1 + (1+\gamma)/\ln(2+\gamma)$.*

Proof. Let $\xi_M = 1 + (1/M) \cdot \sum_{j=1}^{M} \alpha^{j/M}$. First, we note that $\textsc{Disc}(\gamma, M = 1, \beta = 1)$ chooses deterministically $m = 0$ and executes $\textsc{Mimic}(\gamma, \omega = -1 + 0 + 1 = 0)$, i.e., is equivalent to our deterministic algorithm. Hence, by Theorem 2, the corresponding competitive ratio is at most $\xi_1 = 3 + \gamma$.

For analyzing our randomized algorithm, we observe that instead of choosing a random $\omega \in (-1, 0]$, we may choose a random integer $m \in [M]$ and a random real $\beta \in (0, 1/M]$ and set $\omega = -1 + m/M + \beta$. Thus, for any fixed integer M, our randomized algorithm is equivalent to choosing random $\beta \in (0, 1/M]$ and running $\textsc{Disc}(\gamma, M, \beta)$.

Fix any input $\mathcal{I}$. By Theorem 2, $\mathbf{E}_m[\textsc{cost}_{\textsc{Disc}(\gamma,M,\beta)}(\mathcal{I})] \le \xi_M \cdot \textsc{cost}_{\textsc{Opt}}(\mathcal{I})$ holds for any $\beta \in (0, 1/M]$, where the expected value is taken over random choice of m. Clearly, this relation holds also when β is chosen randomly, i.e., $\mathbf{E}_\omega[\textsc{cost}_{\textsc{Mimic}(\gamma,\omega)}] = \mathbf{E}_\gamma \mathbf{E}_m[\textsc{cost}_{\textsc{Disc}(\gamma,M,\beta)}(\mathcal{I})] \le \xi_M \cdot \textsc{cost}_{\textsc{Opt}}(\mathcal{I})$. As the bound is valid for any M, and the competitive ratio of our randomized algorithm is at most $\inf_{M \in \mathbb{N}}\{\xi_M\} = \lim_{M\to\infty} \xi_M = 1 + (1+\gamma)/\ln(2+\gamma)$. ◀

4 Structural Properties of DISC

In this section, we build relations useful for analyzing the performance of $\textsc{Disc}(\gamma, M, \beta)$ on any instance $\mathcal{I}$ of the γ-resettable scheduling problem.

We start by presenting structural properties of schedules S_τ. We note that even if there exists a τ-schedule A that completes all jobs from $\mathcal{I}$, S_τ may leave some jobs untouched. However, a sufficiently long schedule S_τ completes all jobs.

▶ **Lemma 4.** *Fix any input $\mathcal{I}$. There exists a value $T_\mathcal{I}$, such that for any $\tau \ge T_\mathcal{I}$, S_τ completes all jobs of $\mathcal{I}$ and is an optimal (cost-minimal) solution for $\mathcal{I}$.*

Proof. Let $\textsc{Opt}$ be a cost-optimal schedule for $\mathcal{I}$ and let t be its length. Let w be the weight of the lightest job from $\mathcal{I}$. We fix $T_\mathcal{I} = \max\{t, (\textsc{val}_t(\textsc{Opt}) + 1)/w\}$. Now, we pick any $\tau \ge T_\mathcal{I}$, and investigate properties of S_τ.

As $\tau \ge T_\mathcal{I} \ge t$, the schedule of $\textsc{Opt}$ can be trivially extended to a τ-schedule A that does nothing in its suffix of length $\tau - t$. Both A and $\textsc{Opt}$ complete all jobs, and thus $\textsc{val}_\tau(A) = \textsc{val}_t(\textsc{Opt})$. Moreover, as S_τ minimizes function $\textsc{val}_\tau$, $\textsc{val}_\tau(S_\tau) \le \textsc{val}_\tau(A) = \textsc{val}_t(\textsc{Opt}) < T_\mathcal{I} \cdot w \le \tau \cdot w$, and thus S_τ completes all jobs (as otherwise $\textsc{val}_\tau$ would include a penalty of at least $\tau \cdot w$). As S_τ and $\textsc{Opt}$ complete all jobs, $\textsc{cost}_{S_\tau}(\mathcal{I}) = \textsc{val}_\tau(S_\tau) \le \textsc{val}_t(\textsc{Opt}) = \textsc{cost}_{\textsc{Opt}}(\mathcal{I})$, i.e., S_τ is an optimal solution for $\mathcal{I}$. ◀

Sub-phases. Recall that the algorithm $\textsc{Disc}(\gamma, M, \beta)$ chooses a random integer $m \in [M]$, and executes $\textsc{Mimic}(\gamma, \omega = -1 + m/M + \beta)$. To compare $\textsc{Disc}$ executions for different random choices, we introduce sub-phases. Recall that $\alpha = 2 + \gamma$; let $\delta = \alpha^{1/M}$.

Recall that the k-th phase of $\textsc{Mimic}$ starts at time τ_{k-1} and ends at time τ_k, where $\tau_k = \min(\mathcal{I}) \cdot \alpha^{k-1+m/M+\beta} = \min(\mathcal{I}) \cdot \alpha^{\beta-1} \cdot \delta^{m+k\cdot M}$. For any q, we define

$$\eta_q = \eta(q) = \min(\mathcal{I}) \cdot \alpha^{\beta-1} \cdot \delta^q. \tag{3}$$

In these terms, $\tau_k = \eta_{m+k\cdot M}$. We define the q-th sub-phase (for $q \ge 0$) as the time interval starting at time η_{q-1} and ending at time η_q. Then, phase k of $\textsc{Disc}(\gamma, M, \beta)$ consists of exactly M sub-phases, numbered from $(k-1) \cdot M + m + 1$ to $k \cdot M + m$. An example of

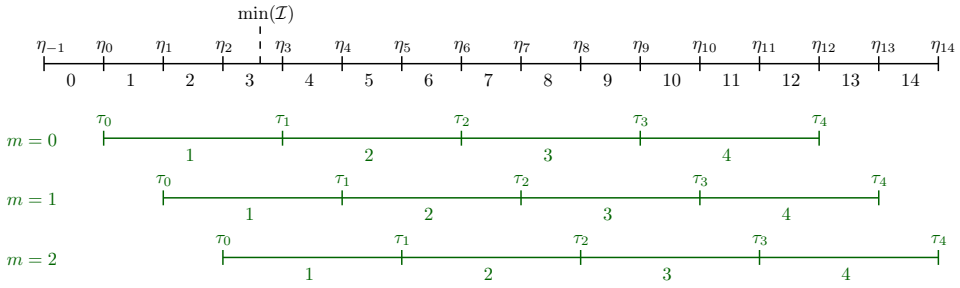


Figure 2 Example of phases (green) and sub-phases (black) of algorithm Disc$(\gamma, M = 3, \beta)$ for all possible choices of m. The time interval lengths are in logarithmic scale. The starts and ends of sub-phases are deterministic functions of γ, M, and β, but the start of a phase depends additionally on the integer $m \in [M]$ chosen randomly by Disc. Sub-phase 0 is not contained in any phase, but will be used in our analysis.

phases and sub-phases is given in Figure 2. We emphasize that the start and the end of a sub-phase is a deterministic function of the parameters of Disc, while the start and end of a phase depend additionally on the value $m \in [M]$ that Disc chooses randomly.

Recall that our deterministic algorithm is equivalent to Mimic$(\gamma, 0) \equiv$ Disc$(\gamma, 1, 1)$. In this case $m = 0$, and thus $\eta_q = \tau_q$ for any q, i.e., each phase consists of one sub-phase, and their indexes coincide.

Sub-phases vs Auxiliary Schedules. We now identify the times when auxiliary schedules are computed by Disc(γ, M, β). Recall that at the beginning of any phase $k+1$ (where $k \geq 1$), i.e., at time $\tau_k = \eta_{m+k \cdot M}$, Disc computes and executes schedule $S_{\eta(m+k \cdot M)}$. Let $T_{\mathcal{I}}$ be the threshold guaranteed by Lemma 4 and we define $K_{\mathcal{I}}$ as the smallest integer satisfying $\eta(K_{\mathcal{I}} \cdot M) \geq T_{\mathcal{I}}$. Note that $K_{\mathcal{I}}$ is a deterministic function of input $\mathcal{I}$.

For any choice of $m \in [M]$, the schedule $S_{\eta(m+K_{\mathcal{I}} \cdot M)}$ completes all jobs. This schedule is executed by Disc in phase $K_{\mathcal{I}} + 1$, and thus Disc terminates latest in phase $K_{\mathcal{I}} + 1$. Summing up, Disc(γ, M, β) executes schedules $S_{\eta(m+M)}, S_{\eta(m+2M)}, \ldots, S_{\eta(m+K_{\mathcal{I}} \cdot M)}$. At the beginning of the first phase, Disc does nothing, but for notational ease, we assume that in the first phase, it also computes and executes a dummy schedule $S_{\eta(m)}$, which does not complete any job. For succinctness, we use $A_q = S_{\eta(q)}$. In these terms, Disc(γ, M, β) executes schedules $A_{m+k \cdot M}$ for $k \in [K_{\mathcal{I}} + 1]$.

Let $Q = K_{\mathcal{I}} \cdot M + (M - 1)$: possible schedule indexes used by Disc range from 0 to Q. For any schedule A_q, we define the set of indexes of *preceding schedules* $P(q) = \{q', q' + M, \ldots, q - M\}$, where $q' = q \mod M$.

Fresh and Stale Requests. We assume that no jobs are completed by the online algorithm while it is resetting the executor, and we assume that the execution of schedule A_q may complete only jobs from set $R(A_q)$. It is however important to note that $R(A_q)$ and $R(A_{q-M})$ may overlap significantly, in which case the execution of schedule A_q serves only these jobs from $R(A_q)$ that have not been served already. To further quantify this effect, for $q \in [Q+1]$, we define the set of *fresh* jobs of schedule A_q as

$$R^{\mathrm{F}}(A_q) = R(A_q) \setminus \bigcup_{\ell \in P(q)} R(A_\ell). \tag{4}$$

The remaining jobs from $R(A_q)$ are called *stale* and are denoted $R^{\mathrm{S}}(A_q) = R(A_q) \setminus R^{\mathrm{F}}(A_q)$. For succinctness, we define the following shorthand notations for their weights:

$$w_q^{\mathrm{F}} = w(R^{\mathrm{F}}(A_q)), \qquad w_q^{\mathrm{S}} = w(R^{\mathrm{S}}(A_q)), \qquad w_q = w(R(A_q)) = w_q^{\mathrm{F}} + w_q^{\mathrm{S}}. \tag{5}$$

▶ **Lemma 5.** *For any $q \in [Q+1]$, it holds that $w_q^{\mathrm{S}} \le \sum_{\ell \in P(q)} w_\ell^{\mathrm{F}}$. This relation becomes equality for $q \ge K_{\mathcal{I}} \cdot M$.*

Proof. By a simple induction, it can be shown that $\biguplus_{\ell \in P(q)} R^{\mathrm{F}}(A_\ell) = \bigcup_{\ell \in P(q)} R(A_\ell)$ for any $q \in [Q+1]$. Then, using the definition of stale jobs, $R^{\mathrm{S}}(A_q) \subseteq \bigcup_{\ell \in P(q)} R(A_\ell) = \biguplus_{\ell \in P(q)} R^{\mathrm{F}}(A_\ell)$. Applying weight to both sides yields $w_q^{\mathrm{S}} \le \sum_{\ell \in P(q)} w_\ell^{\mathrm{F}}$.

Next, we show that this relation can be reversed for $q \ge K_{\mathcal{I}} \cdot M$ (i.e., for the schedule executed in the last phase of Disc). For such q, A_q completes all jobs, and thus $\bigcup_{\ell \in P(q)} R(A_\ell) \subseteq R(A_q) = R^{\mathrm{F}}(A_q) \uplus R^{\mathrm{S}}(A_q)$. By the definition of fresh jobs, $R^{\mathrm{F}}(A_q)$ does not contain any job from $\bigcup_{\ell \in P(q)} R(A_\ell)$, and thus $\bigcup_{\ell \in P(q)} R(A_\ell) \subseteq R^{\mathrm{S}}(A_q)$. This implies that $\biguplus_{\ell \in P(q)} R^{\mathrm{F}}(A_\ell) = \bigcup_{\ell \in P(q)} R(A_\ell) \subseteq R^{\mathrm{S}}(A_q)$. After applying weights to both sides, we obtain $w_q^{\mathrm{S}} \ge \sum_{\ell \in P(q)} w_\ell^{\mathrm{F}}$ as desired. ◀

Jobs Completed in Sub-phases. For further analysis, we refine our notions when a job is completed. For a η_q-schedule A_q, let $R_j(A_q)$ be the set of jobs completed in sub-phase $j \le q$, i.e., within interval $[\eta_{j-1}, \eta_j)$. As $\eta_{-1} \le \eta_{m-1} \le \min(\mathcal{I})$ (cf. (3)), no job can be completed within the interval $[0, \eta_{-1})$ (before sub-phase 0). Hence, $R(A_q) = \biguplus_{j=0}^{q} R_j(A_q)$.

We partition sets $R^{\mathrm{F}}(A_q)$ and $R^{\mathrm{S}}(A_q)$ analogously, defining sets $R_j^{\mathrm{F}}(A_q)$ and $R_j^{\mathrm{S}}(A_q)$ (for $0 \le j \le q$), such that $R^{\mathrm{F}}(A_q) = \biguplus_{j=0}^{q} R_j^{\mathrm{F}}(A_q)$ and $R^{\mathrm{S}}(A_q) = \biguplus_{j=0}^{q} R_j^{\mathrm{S}}(A_q)$. For succinctness, for $0 \le j \le q$, we introduce the following shorthand notations:

- $w_{qj}^{\mathrm{F}} = w(R_j^{\mathrm{F}}(A_q))$, $w_{qj}^{\mathrm{S}} = w(R_j^{\mathrm{S}}(A_q))$, and $w_{qj} = w(R_j(A_q)) = w_{qj}^{\mathrm{F}} + w_{qj}^{\mathrm{S}}$;
- $g_{qj}^{\mathrm{F}} = \textsc{cost}_{A_q}(R_j^{\mathrm{F}}(A_q))$, $g_{qj}^{\mathrm{S}} = \textsc{cost}_{A_q}(R_j^{\mathrm{S}}(A_q))$, and $g_{qj} = \textsc{cost}_{A_q}(R_j(A_q)) = g_{qj}^{\mathrm{F}} + g_{qj}^{\mathrm{S}}$.

▶ **Lemma 6.** *For any $0 \le q < \ell \le Q$, it holds that $\sum_{j=0}^{q}(g_{qj} - g_{\ell j}) + \sum_{j=0}^{q} \eta_q \cdot (w_{\ell j} - w_{qj}) \le 0$.*

Proof. For any η_q-schedule B, it holds that

$$\begin{aligned}\textsc{val}_{\eta(q)}(B) &= \textsc{cost}_B(R(B)) + \eta_q \cdot w\left(\mathcal{I}_{\eta(q)} \setminus R(B)\right) \\ &= \textstyle\sum_{j=0}^{q} \textsc{cost}_B(R_j(B)) + \eta_q \cdot w(\mathcal{I}_{\eta(q)}) - \eta_q \cdot \sum_{j=0}^{q} w(R_j(B)).\end{aligned}$$

Fix any $\ell \le Q$ and let A_ℓ^q be the η_q-schedule consisting of the first q sub-phases of η_ℓ-schedule A_ℓ. Since A_q is a minimizer of $\textsc{val}_{\eta(q)}$, it holds that $\textsc{val}_{\eta(q)}(A_q) \le \textsc{val}_{\eta(q)}(A_\ell^q)$. Thus, $\sum_{j=0}^{q} g_{qj} - \eta_q \cdot \sum_{j=0}^{q} w_{qj} \le \sum_{j=0}^{q} g_{\ell j} - \eta_q \cdot \sum_{j=0}^{q} w_{\ell j}$. ◀

Costs of DISC and OPT. Finally, we can express costs of Disc and Opt using the newly introduced notions.

▶ **Lemma 7.** *For any input $\mathcal{I}$, parameters M and $\beta \in (0, 1/M]$, it holds that $\mathbf{E}[\mathit{COST}_{\textsc{Disc}}(\mathcal{I})] = (1/M) \cdot \sum_{q=0}^{Q} \sum_{j=0}^{q} \left(\eta_q \cdot w_{qj}^{\mathrm{F}} + g_{qj}^{\mathrm{F}}\right)$.*

Proof. Recall that Disc chooses random $m \in [M]$ and then at time η_q it executes schedule A_q, for all $q \in \{m, m+M, \ldots, m+K_{\mathcal{I}} \cdot M\}$. When Disc executes A_q, it completes jobs from $R^{\mathrm{F}}(A_q)$. By the delayed execution property of the resettable scheduling (cf. Subsection 1.2), each job $r \in R^{\mathrm{F}}(A_q)$ is completed at time $\eta_q + s_{A_q}(r)$. Thus, the cost of executing A_q by Disc is equal to

$$\sum_{r\in R^{\mathrm{F}}(A_q)} w(r)\cdot(\eta_q + s_{A_q}(r)) = \eta_q \cdot w(R^{\mathrm{F}}(A_q)) + \textsc{cost}_{A_q}(R^{\mathrm{F}}(A_q))$$
$$= \eta_q \cdot w_q^{\mathrm{F}} + \sum_{j=0}^{q} g_{qj}^{\mathrm{F}} = \sum_{j=0}^{q} \left(\eta_q \cdot w_{qj}^{\mathrm{F}} + g_{qj}^{\mathrm{F}}\right).$$

For any $q \in [Q+1]$, the probability that Disc executes A_q is equal to $1/M$, and thus the lemma follows. ◀

▶ **Lemma 8.** *For any input $\mathcal{I}$ and any $q \in \{Q-M+1, Q-M+2, \ldots, Q\}$, it holds that $\textsc{cost}_{\textsc{Opt}}(\mathcal{I}) = \sum_{j=0}^{q} g_{qj}$.*

Proof. Recall that for such choice of q, schedules A_q serve all jobs of $\mathcal{I}$ achieving optimal cost. Therefore, $\textsc{cost}_{\textsc{Opt}}(\mathcal{I}) = \textsc{cost}_{A_q}(R(A_q)) = \sum_{j=0}^{q} \textsc{cost}_{A_q}(R_j(A_q)) = \sum_{j=0}^{q} g_{qj}$. ◀

5 Factor-Revealing Linear Program

Now we show that the Disc-to-Opt cost ratio on an arbitrary input $\mathcal{I}$ can be upper-bounded by a value of a linear (maximization) program.

Assume we fixed γ and any input $\mathcal{I}$ to the γ-resettable scheduling problem. We also fix parameters of Disc: an integer M and $\beta \in (0, 1/M]$. These choices imply the values of Q and η_q for any q. This allows us to define the linear program $\mathcal{P}_{\gamma,\mathcal{I},M,\beta}$ whose goal is to maximize

$$\sum_{q=0}^{Q} \sum_{j=0}^{q} \eta_q \cdot w_{qj}^{\mathrm{F}} + g_{qj}^{\mathrm{F}} \tag{6}$$

subject to the following constraints:

$$\sum_{j=0}^{q} g_{qj} \le 1 \qquad \text{for all } Q-M+1 \le q \le Q \tag{7}$$
$$\sum_{j=0}^{q} w_{qj}^{\mathrm{S}} - \sum_{\ell\in P(q)} \sum_{j=0}^{\ell} w_{\ell j}^{\mathrm{F}} \le 0 \qquad \text{for all } 0 \le q \le Q-M \tag{8}$$
$$\sum_{\ell\in P(q)} \sum_{j=0}^{\ell} w_{\ell j}^{\mathrm{F}} - \sum_{j=0}^{q} w_{qj}^{\mathrm{S}} \le 0 \qquad \text{for all } Q-M+1 \le q \le Q \tag{9}$$
$$\sum_{j=0}^{q} (g_{qj} - g_{\ell j}) + \sum_{j=0}^{q} \eta_q \cdot (w_{\ell j} - w_{qj}) \le 0 \qquad \text{for all } 0 \le q < \ell \le Q \tag{10}$$
$$\eta_{j-1} \cdot w_{qj}^{\mathrm{S}} - g_{qj}^{\mathrm{S}} \le 0 \qquad \text{for all } 0 \le j \le q \le Q \tag{11}$$
$$g_{qj}^{\mathrm{F}} - \eta_j \cdot w_{qj}^{\mathrm{F}} \le 0 \qquad \text{for all } 0 \le j \le q \le Q \tag{12}$$
$$\eta_{j-1} \cdot w_{qj}^{\mathrm{F}} - g_{qj}^{\mathrm{F}} \le 0 \qquad \text{for all } 0 \le j \le q \le Q \tag{13}$$

and non-negativity of all variables. In (10), we treat w_{qj} and g_{qj} not as variables, but as shorthand notations for $w_{qj}^{\mathrm{F}} + w_{qj}^{\mathrm{S}}$ and $g_{qj}^{\mathrm{F}} + g_{qj}^{\mathrm{S}}$, respectively.

The intuition behind this LP formulation is that instead of creating the whole input $\mathcal{I}$, the adversary only chooses the values of variables w_{qj}^{F}, w_{qj}^{S}, g_{qj}^{F} and g_{qj}^{S} that satisfy some subset of inequalities (inequalities that have to be satisfied if these variables were created on the basis of actual input $\mathcal{I}$). This intuition is formalized below.

▶ **Lemma 9.** *Fix any γ, any input $\mathcal{I}$ for γ-resettable scheduling, and parameters of Disc: integer M and $\beta \in (0, 1/M]$. Then, $\mathbf{E}[\textsc{cost}_{\textsc{Disc}}(\mathcal{I})]/\textsc{cost}_{\textsc{Opt}}(\mathcal{I}) \le P^*_{\gamma,\mathcal{I},M,\beta}/M$, where $P^*_{\gamma,\mathcal{I},M,\beta}$ is the value of the optimal solution to $\mathcal{P}_{\gamma,\mathcal{I},M,\beta}$.*

Proof. By scaling all variables by the same value, $\mathcal{P}_{\gamma,\mathcal{I},M,\beta}$ is equivalent to the (non-linear) optimization program $\mathcal{P}'_{\gamma,\mathcal{I},M,\beta}$, whose objective is to maximize $(\sum_{q=0}^{Q} \sum_{j=0}^{q} \eta_q \cdot w_{qj}^{\mathrm{F}} + g_{qj}^{\mathrm{F}}) / \max_{Q-M+1\le q\le Q} \sum_{j=0}^{q} g_{qj}$, subject to constraints (8)–(13). In particular, the optimal values of these programs, $P^*_{\gamma,\mathcal{I},M,\beta}$ and $P'^*_{\gamma,\mathcal{I},M,\beta}$ are equal.

Next, we set the values of variables w^{F}_{qj}, w^{S}_{qj}, g^{F}_{qj} and g^{S}_{qj} on the basis of input $\mathcal{I}$, and parameters M and β. (Note that the variables depend on these parameters, but not on the random choices of Disc.) We now show that they satisfy the constraints of $\mathcal{P}'^{*}_{\gamma,\mathcal{I},M,\beta}$ and we relate $\mathbf{E}[\textsc{cost}_{\textsc{Disc}}(\mathcal{I})]/\textsc{cost}_{\textsc{Opt}}(\mathcal{I})$ to $P^{*}_{\gamma,\mathcal{I},M,\beta}$.

By Lemma 5 and the relations $w^{\mathrm{F}}_{q} = \sum_{j=0}^{q} w^{\mathrm{F}}_{qj}$ and $w^{\mathrm{S}}_{q} = \sum_{j=0}^{q} w^{\mathrm{S}}_{qj}$, the variables satisfy (8) and (9). Next, Lemma 6 implies (10). Inequalities (11), (12) and (13) follow directly by the definition of costs and weights. Finally, by Lemma 7 and Lemma 8, for any $q \in \{Q-M+1,\ldots,Q\}$, it holds that $\mathbf{E}[\textsc{cost}_{\textsc{Disc}}(\mathcal{I})]/\textsc{cost}_{\textsc{Opt}}(\mathcal{I}) = (1/M)\cdot(\sum_{q=0}^{Q}\sum_{j=0}^{q}\eta_q\cdot w^{\mathrm{F}}_{qj} + g^{\mathrm{F}}_{qj})/(\sum_{j=0}^{q} g_{qj})$, and thus $\mathbf{E}[\textsc{cost}_{\textsc{Disc}}(\mathcal{I})]/\textsc{cost}_{\textsc{Opt}}(\mathcal{I}) \le P'^{*}_{\gamma,\mathcal{I},M,\beta}/M = P^{*}_{\gamma,\mathcal{I},M,\beta}/M$. ◀

5.1 Dual Program and Competitive Ratio

By Lemma 9, the optimal value of $\mathcal{P}_{\gamma,\mathcal{I},M,\beta}$ is an upper bound on the competitive ratio of Disc. By weak duality, an upper-bound is given by any feasible solution to the dual program $\mathcal{D}_{\gamma,\mathcal{I},M,\beta}$ that we present below.

$\mathcal{D}_{\gamma,\mathcal{I},M,\beta}$ uses variables $\xi_q, B_q, C_q, D_{\ell q}, F_{qj}, G_{qj}$, and H_{qj}, corresponding to inequalities (7)–(13) from $\mathcal{P}_{\gamma,\mathcal{I},M,\beta}$, respectively. In the formulas below, we use $L_q = M\cdot K + (q \mod M)$ and $S(q) = \{q+M, q+2\cdot M, \ldots, L_q - M\}$. For succinctness of the description, we introduce two auxiliary variables for any $0 \le j \le q \le Q$:

$$U_{qj} = \sum_{\ell=q+1}^{Q} D_{\ell q} - \sum_{\ell=j}^{q-1} D_{q\ell} \quad \text{and} \quad V_{qj} = \sum_{\ell=j}^{q-1} \eta_\ell \cdot D_{q\ell} - \sum_{\ell=q+1}^{Q} \eta_q \cdot D_{\ell q}. \tag{14}$$

The goal of $\mathcal{D}_{\gamma,\mathcal{I},M,\beta}$ is to minimize

$$\sum_{q=Q-M+1}^{Q} \xi_q \tag{15}$$

subject to the following constraints (in all of them, we omitted the statement that they hold for all $j \in \{0,\ldots,q\}$):

$$U_{qj} + G_{qj} - H_{qj} \ge 1 \qquad \text{for all } 0 \le q \le Q-M \tag{16}$$
$$U_{qj} - F_{qj} \ge 0 \qquad \text{for all } 0 \le q \le Q-M \tag{17}$$
$$U_{qj} + G_{qj} - H_{qj} + \xi_q \ge 1 \qquad \text{for all } Q-M+1 \le q \le Q \tag{18}$$
$$U_{qj} - F_{qj} + \xi_q \ge 0 \qquad \text{for all } Q-M+1 \le q \le Q \tag{19}$$
$$V_{qj} + \eta_{j-1}\cdot H_{qj} - \eta_j \cdot G_{qj} + C_{L_q} - \textstyle\sum_{\ell\in S(q)} B_\ell \ge \eta_q \qquad \text{for all } 0 \le q \le Q-M \tag{20}$$
$$V_{qj} + \eta_{j-1}\cdot F_{qj} + B_q \ge 0 \qquad \text{for all } 0 \le q \le Q-M \tag{21}$$
$$V_{qj} - \eta_j \cdot G_{qj} + \eta_{j-1}\cdot H_{qj} \ge \eta_q \qquad \text{for all } Q-M+1 \le q \le Q \tag{22}$$
$$V_{qj} + \eta_{j-1}\cdot F_{qj} - C_q \ge 0 \qquad \text{for all } Q-M+1 \le q \le Q \tag{23}$$

and non-negativity of all variables.

▶ **Lemma 10.** *For any γ, any input $\mathcal{I}$ for γ-resettable scheduling, any positive integer M, and any $\beta \in (0, 1/M]$, there exists a feasible solution to $\mathcal{D}_{\gamma,\mathcal{I},M,\beta}$ of value at most $M + \sum_{j=1}^{M}(2+\gamma)^{j/M}$.*

We defer the proof to the next subsection, first arguing how it implies the main theorem of the paper (the competitive ratio of Disc).

Proof of Theorem 2. Fix any γ, and consider algorithm $\textsc{Disc}(\gamma, M, \beta)$ for any positive integer M, and any $\beta \in (0, 1/M]$. Fix any input $\mathcal{I}$ to the γ-resettable scheduling problem. Let $P^{*}_{\gamma,\mathcal{I},M,\beta}$ be the value of an optimal solution to $\mathcal{P}_{\gamma,\mathcal{I},M,\beta}$. By weak duality and Lemma 10, $P_{\gamma,\mathcal{I},M,\beta} \le M + \sum_{j=1}^{M}(2+\gamma)^{j/M}$. Hence, by Lemma 9, $\mathbf{E}[\textsc{cost}_{\textsc{Disc}}(\mathcal{I})]/\textsc{cost}_{\textsc{Opt}}(\mathcal{I}) \le P^{*}_{\gamma,\mathcal{I},M,\beta}/M \le 1 + (1/M)\cdot\sum_{j=1}^{M}(2+\gamma)^{j/M}$, as desired. ◀

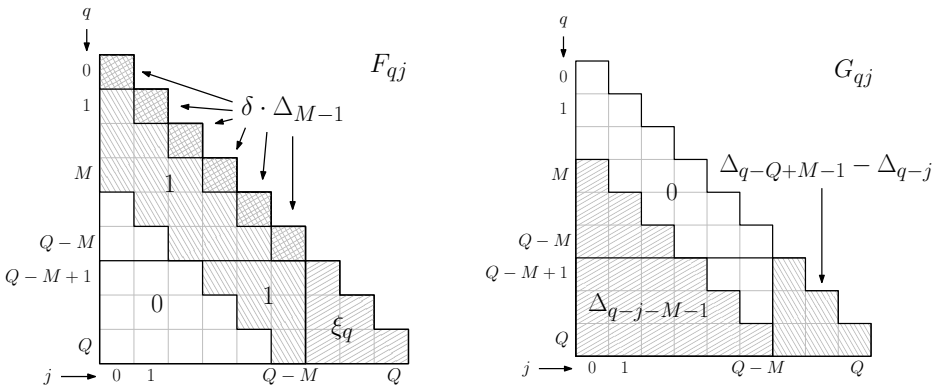


Figure 3 Visual presentation of values assigned to dual variables F_{qj} (left) and G_{qj} (right) for $M = 3$ and $Q = 8$.

5.2 Proof of Lemma 10

Let

$$\Delta_k = \sum_{i=0}^{k} \delta^i = \left(\delta^{k+1} - 1\right) / \left(\delta - 1\right).$$

In particular $\Delta_{-1} = 0$. We choose the following values of the dual variables:

$$\xi_q = 1 + \delta^{q-Q+M} \quad \text{for } Q - M + 1 \le q \le Q,$$

$$F_{qj} = \begin{cases} \xi_q & \text{for } Q - M + 1 \le j \le q \le Q, \\ \delta \cdot \Delta_{M-1} & \text{for } 0 \le j \le Q - M \text{ and } q = j, \\ 1 & \text{for } 0 \le j \le Q - M \text{ and } q \in \{j+1, \ldots, j+M\}, \\ 0 & \text{otherwise,} \end{cases}$$

$$G_{qj} = \begin{cases} \Delta_{q-Q+M-1} - \Delta_{q-j} & \text{for } Q - M + 1 \le j \le q \le Q, \\ \Delta_{q-j-M-1} & \text{for } j \le q - M \\ 0 & \text{otherwise,} \end{cases}$$

$$\begin{aligned} B_q &= \eta_{q-M-1} \cdot \left(\delta^{M+1} + 1\right) \cdot \left(\delta^M - 1\right) && \text{for } 0 \le q \le Q - M, \\ C_q &= \eta_{q-M-1} \cdot \left(\delta^{M+1} + 1\right) && \text{for } Q - M + 1 \le q \le Q, \\ D_{qj} &= F_{q,j+1} - F_{qj} && \text{for } 0 \le j < q \le Q, \\ H_{qj} &= F_{qj} + G_{qj} - 1 && \text{for } 0 \le j \le q \le Q. \end{aligned}$$

The values of F_{qj} and G_{qj} (for $0 \le j \le q \le Q$) are depicted in Figure 3 for an easier reference. We will extensively use the property that $\eta_i \cdot \delta^j = \eta_{i+j}$ for any i and j.

Objective Value. With the above assignment of dual variables the objective value of $\mathcal{D}_{\gamma,\mathcal{I},M,\beta}$ is equal to $\sum_{q=Q-M+1}^{Q} \xi_q = M + \sum_{j=1}^{M} \delta^j = M + \sum_{j=1}^{M} (2+\gamma)^{j/M}$ as desired.

Non-negativity of Variables. Variables ξ_q, C_q, B_q, F_{qj} and G_{qj} are trivially non-negative (for those q and j for which they are defined). The non-negativity of $D_{qj} = F_{q,j+1} - F_{qj}$ follows as F_{qj} is a non-decreasing function of its second argument (cf. Figure 3).

Finally, for showing non-negativity of variable H_{qj}, we consider two cases. If $j \geq q - M$, then $F_{qj} \geq 1$. Otherwise, $j \leq q - M - 1$, and then $G_{qj} = \Delta_{q-j-M-1} \geq 1$. Thus, in either case $H_{qj} = F_{qj} + G_{qj} - 1 \geq 0$.

Helper Bounds. It remains to show that the given values of dual variables satisfy all constraints (16)–(23) of the dual program $\mathcal{D}_{\gamma,\mathcal{I},M,\beta}$. We define a few helper notions and identities that are used throughout the proof of dual feasibility. For any $q \in [Q+1]$, let

$$R_q = \textstyle\sum_{\ell=q+1}^{Q} D_{\ell q} = \sum_{\ell=q+1}^{Q} (F_{\ell,q+1} - F_{\ell q})\,.$$

▶ **Lemma 11.** *$R_q = \delta \cdot \Delta_{M-1}$ for $q \leq Q - M$ and $R_q = 0$ otherwise.*

Proof. We consider three cases.

1. $q \in \{0, \ldots, Q - M - 1\}$. Then, $R_q = F_{q+1,q+1} + \sum_{\ell=q+1}^{Q} (F_{\ell+2,\ell+1} - F_{\ell+1,\ell}) - F_{Qq} = \delta \cdot \Delta_{M-1} + \sum_{\ell=q+1}^{Q} 0 - 0 = \delta \cdot \Delta_{M-1}$.
2. $q = Q - M$. Then, $R_q = \sum_{\ell=Q-M+1}^{Q} (\xi_\ell - 1) = \sum_{j=1}^{M} \delta^j = \delta \cdot \Delta_{M-1}$.
3. $q \in \{Q - M + 1, \ldots, Q\}$. Then, $R_q = \sum_{\ell=q+1}^{Q} (\xi_\ell - \xi_\ell) = 0$. ◀

Next, we investigate the values of V_{qj} for different q and j. Using its definition (cf. (14)),

$$V_{qj} = \textstyle\sum_{\ell=j}^{q-1} \eta_\ell \cdot D_{q\ell} - \sum_{\ell=q+1}^{Q} \eta_q \cdot D_{\ell q} = \sum_{\ell=j}^{q-1} \eta_\ell \cdot (F_{q,\ell+1} - F_{q\ell}) - \eta_q \cdot R_q. \tag{24}$$

Additionally, using $H_{qj} = F_{qj} + G_{qj} - 1$, we obtain

$$\eta_j \cdot G_{qj} - \eta_{j-1} \cdot H_{qj} = (\eta_j - \eta_{j-1}) \cdot G_{qj} + \eta_{j-1} - \eta_{j-1} \cdot F_{qj}. \tag{25}$$

Using the chosen values of G_{qj}, we observe that

$$(\eta_j - \eta_{j-1}) \cdot G_{qj} = \begin{cases} \eta_{q+j-Q+M-1} - \eta_q & \text{for } Q - M + 1 \leq j \leq q, \\ \eta_{q-M-1} - \eta_{j-1} & \text{for } j \leq q - M - 1, \\ 0 & \text{otherwise.} \end{cases} \tag{26}$$

Furthermore, in all the cases, it can be verified that

$$(\eta_j - \eta_{j-1}) \cdot G_{qj} + \eta_{j-1} - \eta_{q-M-1} \geq 0. \tag{27}$$

5.2.1 Showing inequalities (16)–(19)

We prove that relations (16)–(19) hold with equality. In fact, it suffices to show (17) and (19): inequalities (16) and (18) follow immediately as we chose $H_{qj} = F_{qj} + G_{qj} - 1$. Using the definition of U_{qj} (cf. (14)), we obtain

$$U_{qj} = \textstyle\sum_{\ell=q+1}^{Q} D_{\ell q} - \sum_{\ell=j}^{q-1} D_{q\ell} = R_q - \sum_{\ell=j}^{q-1} (F_{q,\ell+1} - F_{q\ell}) = R_q - F_{qq} + F_{qj}.$$

Now, we observe that for $q \leq Q - M$, it holds that $R_q - F_{qq} = \delta \cdot \Delta_{M-1} - \delta \cdot \Delta_{M-1} = 0$, and thus $U_{qj} - F_{qj} = 0$, which implies (17). On the other hand, for $q > Q - M$, it holds that $R_q - F_{qq} = 0 - \xi_q$, and hence $U_{qj} - F_{qj} + \xi_q = 0$, which implies (19).

5.2.2 Showing inequalities (20)–(21)

Within this part, we assume $q \le Q - M$. We start with evaluating some terms that are present in (20) and (21). First, we observe that

$$B_q = \eta_{q-M-1} \cdot \left(\delta^{M+1} + 1\right) \cdot \left(\delta^M - 1\right) = \eta_{q+M} - \eta_q + \eta_{q-1} - \eta_{q-M-1}. \tag{28}$$

Second, we compute the term $C_{L_q} - \sum_{\ell \in S(q)} B_\ell$. Recall that $S(q) = \{q+M, q+2\cdot M, \ldots, L_q - M\}$. Thus,

$$\begin{aligned} C_{L_q} - \textstyle\sum_{\ell \in S(q)} B_\ell &= \left(\delta^{M+1} + 1\right) \cdot \left[\eta(L_q - M - 1) - (\delta^M - 1) \cdot \eta(-M-1) \cdot \textstyle\sum_{\ell \in S(q)} \delta^\ell\right] \\ &= (\delta^{M+1} + 1) \cdot \left[\eta(L_q - M - 1) - \eta(-M-1) \cdot \left(\delta^{L_q} - \delta^{q+M}\right)\right] \\ &= (\delta^{M+1} + 1) \cdot \eta_{q-1} = \eta_{q+M} + \eta_{q-1}. \end{aligned} \tag{29}$$

▶ **Lemma 12.** *Fix any $0 \le j \le q \le Q - M$. Then,*

$$V_{qj} = \eta_q - \eta_{q-1} - \eta_{q+M} - \eta_{j-1} \cdot F_{qj} + (\eta_j - \eta_{j-1}) \cdot G_{qj} + \eta_{j-1}.$$

Proof. By the definition, $\Delta_{M-1} = \sum_{i=0}^{M-1} \delta^i$, and therefore $(\eta_{q-1} - \eta_q) \cdot \delta \cdot \Delta_{M-1} = \eta_q - \eta_{q+M}$. Thus, it suffices to show the following relation

$$V_{qj} = \eta_{q-1} \cdot (\delta \cdot \Delta_{M-1} - 1) - \eta_q \cdot \delta \cdot \Delta_{M-1} - \eta_{j-1} \cdot F_{qj} + (\eta_j - \eta_{j-1}) \cdot G_{qj} + \eta_{j-1}.$$

To evaluate V_{qj} using (24), it is useful to trace values $F_{qj}, F_{q,j+1}, \ldots, F_{qq}$ (cf. Figure 3), noting that only the increases of these values contribute to V_{qj}. We also note that for $q \le Q - M$, possible increases are from 0 to 1 (between $F_{q,q-M-1}$ and $F_{q,q-M}$) and from 1 to $\delta \cdot \Delta_{M-1}$ (between $F_{q,q-1}$ and F_{qq}). We consider three cases, using $R_q = \delta \cdot \Delta_{M-1}$ below.

1. $j \le q - M - 1$. Then, $F_{qj} = 0$ and

$$\begin{aligned} V_{qj} &= \eta_{q-1} \cdot (F_{qq} - F_{q,q-1}) + \eta_{q-M-1} \cdot (F_{q,q-M} - F_{q,q-M-1}) - \eta_q \cdot R_q \\ &= \eta_{q-1} \cdot (\delta \cdot \Delta_{M-1} - 1) - \eta_q \cdot \delta \cdot \Delta_{M-1} + \eta_{q-M-1} - \eta_{j-1} + \eta_{j-1} - \eta_{j-1} \cdot F_{qj}. \end{aligned}$$

   The lemma follows as $(\eta_j - \eta_{j-1}) \cdot G_{qj} = \eta_{q-M-1} - \eta_{j-1}$ (see (26)).
2. $j \in \{q - M, \ldots, q - 1\}$. Then $F_{qj} = 1$, and

$$\begin{aligned} V_{qj} &= \eta_{q-1} \cdot (F_{qq} - F_{q,q-1}) - \eta_q \cdot R_q \\ &= \eta_{q-1} \cdot (\delta \cdot \Delta_{M-1} - 1) - \eta_q \cdot \delta \cdot \Delta_{M-1} \\ &= \eta_{q-1} \cdot (\delta \cdot \Delta_{M-1} - 1) - \eta_q \cdot \delta \cdot \Delta_{M-1} - \eta_{j-1} \cdot F_{qj} + \eta_{j-1}. \end{aligned}$$

   The lemma follows as $(\eta_j - \eta_{j-1}) \cdot G_{qj} = 0$ (see (26)).
3. $j = q$. Then $F_{qj} = \delta \cdot \Delta_{M-1}$, and thus

$$\begin{aligned} V_{qj} &= -\eta_q \cdot R_q \\ &= \eta_{q-1} \cdot \delta \cdot \Delta_{M-1} - \eta_q \cdot \delta \cdot \Delta_{M-1} - \eta_{j-1} \cdot F_{qj} \\ &= \eta_{q-1} \cdot (\delta \cdot \Delta_{M-1} - 1) - \eta_q \cdot \delta \cdot \Delta_{M-1} - \eta_{j-1} \cdot F_{qj} + \eta_{j-1}. \end{aligned}$$

   As in the previous case, the lemma follows as $(\eta_j - \eta_{j-1}) \cdot G_{qj} = 0$. ◀

Showing Inequality (20). We show that (20) holds with equality. Using Lemma 12, (29), and (25) yields

$$\begin{aligned} &V_{qj} + \eta_{j-1} \cdot H_{qj} - \eta_j \cdot G_{qj} + C_{L_q} - \textstyle\sum_{\ell \in S(q)} B_\ell \\ &\quad = \eta_q - \eta_{q-1} - \eta_{q+M} - \eta_{j-1} \cdot F_{qj} + (\eta_j - \eta_{j-1}) \cdot G_{qj} + \eta_{j-1} \\ &\qquad - (\eta_j - \eta_{j-1}) \cdot G_{qj} - \eta_{j-1} + \eta_{j-1} \cdot F_{qj} + \eta_{q+M} + \eta_{q-1} = \eta_q. \end{aligned}$$

Showing Inequality (21). Using Lemma 12, (29), and (25) yields

$$\begin{aligned} &V_{qj} + \eta_{j-1} \cdot F_{qj} + B_q \\ &\quad = \eta_q - \eta_{q-1} - \eta_{q+M} - \eta_{j-1} \cdot F_{qj} + (\eta_j - \eta_{j-1}) \cdot G_{qj} + \eta_{j-1} \\ &\qquad + \eta_{j-1} \cdot F_{qj} + \eta_{q+M} - \eta_q + \eta_{q-1} - \eta_{q-M-1} \\ &\quad = (\eta_j - \eta_{j-1}) \cdot G_{qj} + \eta_{j-1} - \eta_{q-M-1} \geq 0. \end{aligned}$$

where the last inequality follows by (27).

5.2.3 Showing inequalities (22)–(23)

Within this part, we assume that $q \geq Q - M + 1$.

▶ **Lemma 13.** *Fix any $q \geq Q - M + 1$ and $0 \leq j \leq q$. Then,*

$$V_{qj} = \eta_q + (\eta_j - \eta_{j-1}) \cdot G_{qj} - \eta_{j-1} \cdot F_{qj} + \eta_{j-1}.$$

Proof. As $g \geq Q - M + 1$, it holds that $R_q = 0$, and thus (24) reduces to

$$V_{qj} = \textstyle\sum_{\ell=j}^{q-1} \eta_\ell \cdot (F_{q,\ell+1} - F_{q\ell}) .$$

As in the proof of Lemma 12, to further evaluate V_{qj}, it is useful to trace values $F_{qj}, F_{q,j+1}, \ldots, F_{qq}$ (cf. Figure 3), where the increases of these values contribute to V_{qj}. We also note that for $q \geq Q - M + 1$, the possible increases are from 0 to 1 (between $F_{q,q-M-1}$ and $F_{q,q-M}$) and from 1 to ξ_q (between $F_{q,Q-M}$ and $F_{q,Q-M+1}$). We consider three cases.

1. $j \leq q - M - 1$. Then $F_{qj} = 0$, and

$$\begin{aligned} V_{qj} &= \eta_{Q-M} \cdot (F_{qq} - F_{q,q-1}) + \eta_{q-M-1} \cdot (F_{q,q-M} - F_{q,q-M-1}) \\ &= \eta_{Q-M} \cdot (\xi_q - 1) + \eta_{q-M-1} \\ &= \eta_q + \eta_{q-M-1} - \eta_{j-1} + \eta_{j-1} - \eta_{j-1} \cdot F_{qj}. \end{aligned}$$

   The lemma follows as $(\eta_j - \eta_{j-1}) \cdot G_{qj} = \eta_{q-M-1} - \eta_{j-1}$ (see (26)).
2. $j \in \{q - M, \ldots, Q - M\}$. Then $F_{qj} = 1$, and

$$\begin{aligned} V_{qj} &= \eta_{Q-M} \cdot (F_{qq} - F_{q,q-1}) \\ &= \eta_{Q-M} \cdot (\xi_q - 1) \\ &= \eta_q - \eta_{j-1} \cdot F_{qj} + \eta_{j-1}. \end{aligned}$$

   The lemma follows as $(\eta_j - \eta_{j-1}) \cdot G_{qj} = 0$ (see (26)).
3. $j \in \{q - M, \ldots, Q - M\}$. Then $F_{qj} = \xi_q = 1 + \delta^{q-Q+M}$, and

$$\begin{aligned} V_{qj} = 0 &= \eta_q + \eta_{q+j-Q+M-1} - \eta_q - \eta_{j-1} \cdot (1 + \delta^{q-Q+M}) + \eta_{j-1} \\ &= \eta_q + \eta_{q+j-Q+M-1} - \eta_q - \eta_{j-1} \cdot F_{qj} + \eta_{j-1}. \end{aligned}$$

   The lemma follows as $(\eta_j - \eta_{j-1}) \cdot G_{qj} = \eta_{q+j-Q+M-1} - \eta_q$ (see (26)). ◀

Showing Inequality (22). We show that (22) holds with equality. Using Lemma 13 and (25), we obtain

$$\begin{aligned} &V_{qj} + \eta_{j-1} \cdot H_{qj} - \eta_j \cdot G_{qj} \\ &\quad = \eta_q + (\eta_j - \eta_{j-1}) \cdot G_{qj} - \eta_{j-1} \cdot F_{qj} + \eta_{j-1} - (\eta_j - \eta_{j-1}) \cdot G_{qj} - \eta_{j-1} + \eta_{j-1} \cdot F_{qj} \\ &\quad = \eta_q. \end{aligned}$$

Showing Inequality (23). Using Lemma 13, (25), and the definition of C_q, we obtain

$$\begin{aligned} &V_{qj} + \eta_{j-1} \cdot F_{qj} - C_q \\ &\quad = \eta_q + (\eta_j - \eta_{j-1}) \cdot G_{qj} - \eta_{j-1} \cdot F_{qj} + \eta_{j-1} + \eta_{j-1} \cdot F_{qj} - \eta_q - \eta_{q-M-1} \\ &\quad = (\eta_j - \eta_{j-1}) \cdot G_{qj} + \eta_{j-1} - \eta_{q-M-1} \geq 0. \end{aligned}$$

where the last inequality follows by (27).

6 Tightness of the Analysis

The analysis of our algorithms is tight as proven below. For the deterministic one, we additionally show that choosing ω different from 0 does not help.

▶ **Theorem 14.** *For any γ, there are γ-resettable scheduling problems, such that for any $\omega \in (-1, 0]$, the competitive ratio of* Mimic(γ, ω) *is at least $3 + \gamma$.*

Proof. We fix a small $\varepsilon > 0$ and let $\alpha = 2 + \gamma$. The input $\mathcal{I}$ contains two jobs: the first one of weight ε that arrives at time 1, and second one of weight 1 that arrives at time $\alpha^{1+\omega} + \varepsilon$. We assume that there exists a schedule S_1 that serves the first job at the time of its arrival and a schedule S_2 that serves both jobs at the times of their arrivals. Therefore, $\textsc{cost}_{\textsc{Opt}}(\mathcal{I}) = \varepsilon \cdot 1 + 1 \cdot (\alpha^{1+\omega} + \varepsilon) = \alpha^{1+\omega} + 2 \cdot \varepsilon$.

For analyzing the cost of Mimic, note that at at time 1, Mimic observes the first job and learns the value of $\min(\mathcal{I}) = 1$. This is the sole purpose of the first job: setting $\min(\mathcal{I}) = 1$ makes the algorithm miss the opportunity to serve the second job early. At time $\tau_1 = \alpha^{1+\omega}$, Mimic executes the τ_1-schedule S'_1, which is schedule S_1 prolonged trivially to length τ_1. Next, at time $\tau_2 = \alpha^{2+\omega}$, Mimic executes the τ_2-schedule S'_2, which is schedule S_2 prolonged trivially to length τ_2. This way it completes the second job at time $\tau_2 + (\alpha^{1+\omega} + \varepsilon)$, and thus $\textsc{cost}_{\textsc{Mimic}}(\mathcal{I}) \geq \tau_2 + \alpha^{1+\omega} + \varepsilon = \alpha^{1+\omega} \cdot (1 + \alpha) + \varepsilon$. By taking appropriately small $\varepsilon > 0$, the ratio between $\textsc{cost}_{\textsc{Mimic}}(\mathcal{I})$ and $\textsc{cost}_{\textsc{Opt}}(\mathcal{I})$ becomes arbitrarily close to $1 + \alpha = 3 + \gamma$. ◀

▶ **Theorem 15.** *For any γ, there are γ-resettable scheduling problems, such that the competitive ratio of a randomized algorithm that runs* Mimic(γ, ω) *with a random $\omega \in (-1, 0]$ is at least $1 + (1 + \gamma)/\ln(2 + \gamma)$.*

Proof. Let $\alpha = 2 + \gamma$. The input $\mathcal{I}$ contains a single job of weight 1 arriving at time 1. We also assume that for any $\tau \geq 1$, there exists a τ-schedule S_τ that completes this job at time 1. Clearly, $\textsc{cost}_{\textsc{Opt}}(\mathcal{I}) = 1 \cdot 1 = 1$.

At time 1, Mimic observes the only job of $\mathcal{I}$ and learns that $\min(\mathcal{I}) = 1$. Its sets $\tau_1 = \alpha^{1+\omega}$ and at time τ_1 it executes schedule S_{τ_1}, thus completing the job at time $\tau_1 + 1$. Therefore, $\textsc{cost}_{\textsc{Mimic}}(\mathcal{I}) = \int_{-1}^{0} \tau_1 + 1 \, d\omega = \int_{-1}^{0} \alpha^{1+\omega} + 1 \, d\omega = 1 + (\alpha - 1)/\ln \alpha = 1 + (1 + \gamma)/\ln(2 + \gamma)$. This implies the desired lower bound. ◀

7 Flaw in the Randomized Lower Bound for DARP

The authors of [22] claim a lower bound of 3 for randomized k-DARP (for any $k \geq 1$), see Theorem 4 of [22]. Below we show a flaw in their argument.

The construction given in the proof of their Theorem 4 uses Yao min-max principle and is parameterized with a few variables, in particular with an integer m and with a real number $v \in [0, 1]$. Towards the end of the proof, they show that the competitive ratio of any randomized algorithm for the k-DARP problem is at least

$$L_{m,v} = \frac{3m - 4km - 4km^2 + v^{\frac{m+1}{2-m}} \cdot (3 + (4km^2 + 4km + 6m + 6) \cdot v)}{-4 - m - 2km - 2km^2 + v^{\frac{m+1}{2-m}} \cdot (3 + (2km^2 + 2km + 4m + 4) \cdot v)}$$

and they claim that there exists v, such that $L_{m,v} = 3$ when m tends to infinity. However, for any fixed k and any v (also being a function of m), by dividing numerator and denominator by m^2, we obtain that

$$\lim_{m\to\infty} L_{m,v} = \frac{-4k + v^{\frac{m+1}{2-m}} \cdot 4k \cdot v}{-2k + v^{\frac{m+1}{2-m}} \cdot 2k \cdot v} = 2.$$

That is, the proven lower bound is 2 instead of 3.

References

1 Foto N. Afrati, Evripidis Bampis, Chandra Chekuri, David R. Karger, Claire Kenyon, Sanjeev Khanna, Ioannis Milis, Maurice Queyranne, Martin Skutella, Clifford Stein, and Maxim Sviridenko. Approximation schemes for minimizing average weighted completion time with release dates. In *Proc. 40th IEEE Symp. on Foundations of Computer Science (FOCS)*, pages 32–44, 1999. doi:10.1109/SFFCS.1999.814574.

2 Sanjeev Arora and George Karakostas. Approximation schemes for minimum latency problems. *SIAM Journal on Computing*, 32(5):1317–1337, 2003. doi:10.1137/S0097539701399654.

3 Norbert Ascheuer, Sven Oliver Krumke, and Jörg Rambau. Online dial-a-ride problems: Minimizing the completion time. In *Proc. 17th Symp. on Theoretical Aspects of Computer Science (STACS)*, pages 639–650, 2000. doi:10.1007/s10951-005-6811-3.

4 Giorgio Ausiello, Esteban Feuerstein, Stefano Leonardi, Leen Stougie, and Maurizio Talamo. Serving requests with on-line routing. In *Proc. 4th Scandinavian Symposium and Workshops on Algorithm Theory (SWAT)*, pages 37–48, 1994. doi:10.1007/3-540-58218-5_4.

5 Giorgio Ausiello, Esteban Feuerstein, Stefano Leonardi, Leen Stougie, and Maurizio Talamo. Competitive algorithms for the on-line traveling salesman. In *Proc. 4th Int. Workshop on Algorithms and Data Structures (WADS)*, pages 206–217, 1995. doi:10.1007/3-540-60220-8_63.

6 Giorgio Ausiello, Esteban Feuerstein, Stefano Leonardi, Leen Stougie, and Maurizio Talamo. Algorithms for the on-line travelling salesman. *Algorithmica*, 29(4):560–581, 2001. doi:10.1007/s004530010071.

7 Ricardo A. Baeza-Yates, Joseph C. Culberson, and Gregory J. E. Rawlins. Searching in the plane. *Information and Computation*, 106(2):234–252, 1993.

8 Marcin Bienkowski and Hsiang-Hsuan Liu. An improved online algorithm for the traveling repairperson problem on a line. In *Proc. 44th Int. Symp. on Mathematical Foundations of Computer Science (MFCS)*, pages 6:1–6:12, 2019. doi:10.4230/LIPIcs.MFCS.2019.6.

9 Alexander Birx and Yann Disser. Tight analysis of the smartstart algorithm for online dial-a-ride on the line. *SIAM Journal on Discrete Mathematics*, 34(2):1409–1443, 2020. doi:10.1137/19M1268513.

10 Alexander Birx, Yann Disser, and Kevin Schewior. Improved bounds for open online dial-a-ride on the line. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques (APPROX/RANDOM)*, pages 21:1–21:22, 2019. doi:10.4230/LIPIcs.APPROX-RANDOM.2019.21.

11 Antje Bjelde, Yann Disser, Jan Hackfeld, Christoph Hansknecht, Maarten Lipmann, Julie Meißner, Kevin Schewior, Miriam Schlöter, and Leen Stougie. Tight bounds for online TSP on the line. In *Proc. 28th ACM-SIAM Symp. on Discrete Algorithms (SODA)*, pages 994–1005, 2017. doi:10.1137/1.9781611974782.63.

12 Markus Bläser. Metric TSP. In *Encyclopedia of Algorithms*, pages 1276–1279. Springer, 2016. doi:10.1007/978-1-4939-2864-4_230.

13 Michiel Blom, Sven Oliver Krumke, Willem de Paepe, and Leen Stougie. The online TSP against fair adversaries. *INFORMS Journal on Computing*, 13(2):138–148, 2001. doi:10.1287/ijoc.13.2.138.10517.

14 Vincenzo Bonifaci and Leen Stougie. Online k-server routing problems. *Theory of Computing Systems*, 45(3):470–485, 2009. doi:10.1007/s00224-008-9103-4.

15 Allan Borodin and Ran El-Yaniv. *Online Computation and Competitive Analysis*. Cambridge University Press, 1998.

16 Soumen Chakrabarti, Cynthia A. Phillips, Andreas S. Schulz, David B. Shmoys, Clifford Stein, and Joel Wein. Improved scheduling algorithms for minsum criteria. In *Proc. 23rd Int. Colloq. on Automata, Languages and Programming (ICALP)*, pages 646–657, 1996. doi:10.1007/3-540-61440-0_166.

17 Kamalika Chaudhuri, Brighten Godfrey, Satish Rao, and Kunal Talwar. Paths, trees, and minimum latency tours. In *Proc. 44th IEEE Symp. on Foundations of Computer Science (FOCS)*, pages 36–45, 2003. doi:10.1109/SFCS.2003.1238179.

18 Pei-Chuan Chen, Erik D. Demaine, Chung-Shou Liao, and Hao-Ting Wei. Waiting is not easy but worth it: the online TSP on the line revisited. Unpublished, 2019. arXiv:1907.00317.

19 José R. Correa and Michael R. Wagner. Lp-based online scheduling: from single to parallel machines. *Mathematical Programming*, 119(1):109–136, 2009. doi:10.1007/s10107-007-0204-7.

20 Willem de Paepe, Jan Karel Lenstra, Jirí Sgall, René A. Sitters, and Leen Stougie. Computer-aided complexity classification of dial-a-ride problems. *INFORMS Journal on Computing*, 16(2):120–132, 2004. doi:10.1287/ijoc.1030.0052.

21 Esteban Feuerstein and Leen Stougie. On-line single-server dial-a-ride problems. *Theoretical Computer Science*, 268(1):91–105, 2001. doi:10.1016/S0304-3975(00)00261-9.

22 Irene Fink, Sven Oliver Krumke, and Stephan Westphal. New lower bounds for online k-server routing problems. *Information Processing Letters*, 109(11):563–567, 2009. doi:10.1016/j.ipl.2009.01.024.

23 R.L. Graham, E.L. Lawler, J.K. Lenstra, and A.H.G.Rinnooy Kan. Optimization and approximation in deterministic sequencing and scheduling: a survey. In *Discrete Optimization II*, volume 5 of *Annals of Discrete Mathematics*, pages 287–326. Elsevier, 1979. doi:10.1016/S0167-5060(08)70356-X.

24 Leslie A. Hall, Andreas S. Schulz, David B. Shmoys, and Joel Wein. Scheduling to minimize average completion time: Off-line and on-line approximation algorithms. *Mathematics of Operations Research*, 22(3):513–544, 1997. doi:10.1287/moor.22.3.513.

25 Dietrich Hauptmeier, Sven Oliver Krumke, and Jörg Rambau. The online dial-a-ride problem under reasonable load. In *Proc. 4th Int. Conf. on Algorithms and Complexity (CIAC)*, pages 125–136, 2000. doi:10.1007/3-540-46521-9_11.

26 Han Hoogeveen, Petra Schuurman, and Gerhard J. Woeginger. Non-approximability results for scheduling problems with minsum criteria. *INFORMS Journal on Computing*, 13(2):157–168, 2001. doi:10.1287/ijoc.13.2.157.10520.

27 Dawsen Hwang and Patrick Jaillet. Online scheduling with multi-state machines. *Networks*, 71(3):209–251, 2018. doi:10.1002/net.21799.

28 Patrick Jaillet and Michael R. Wagner. Online routing problems: Value of advanced information as improved competitive ratios. *Transp. Sci.*, 40(2):200–210, 2006. doi:10.1287/trsc.1060.0147.

29 Patrick Jaillet and Michael R. Wagner. Generalized online routing: New competitive ratios, resource augmentation, and asymptotic analyses. *Oper. Res.*, 56(3):745–757, 2008. doi:10.1287/opre.1070.0450.

30 Vinay A. Jawgal, V. N. Muralidhara, and P. S. Srinivasan. Online travelling salesman problem on a circle. In *Proc. 15th Theory and Applications of Models of Computation (TAMC)*, pages 325–336, 2019. doi:10.1007/978-3-030-14812-6_20.

31 Sven Oliver Krumke, Willem de Paepe, Diana Poensgen, Maarten Lipmann, Alberto Marchetti-Spaccamela, and Leen Stougie. On minimizing the maximum flow time in the online dial-a-ride problem. In *Proc. 3rd Workshop on Approximation and Online Algorithms (WAOA)*, pages 258–269, 2005. doi:10.1007/11671411_20.

32 Sven Oliver Krumke, Willem de Paepe, Diana Poensgen, and Leen Stougie. News from the online traveling repairman. *Theoretical Computer Science*, 295:279–294, 2003. doi:10.1016/S0304-3975(02)00409-7.

33 Sven Oliver Krumke, Willem de Paepe, Diana Poensgen, and Leen Stougie. Erratum to "news from the online traveling repairman" [TCS 295 (1-3) (2003) 279-294]. *Theoretical Computer Science*, 352(1-3):347–348, 2006. doi:10.1016/j.tcs.2005.11.036.

34 Sven Oliver Krumke, Luigi Laura, Maarten Lipmann, Alberto Marchetti-Spaccamela, Willem de Paepe, Diana Poensgen, and Leen Stougie. Non-abusiveness helps: An O(1)-competitive algorithm for minimizing the maximum flow time in the online traveling salesman problem. In *Proc. 5th Int. Workshop on Approximation Algorithms for Combinatorial Optimization (APPROX)*, pages 200–214, 2002. doi:10.1007/3-540-45753-4_18.

35 Maarten Lipmann, Xiwen Lu, Willem de Paepe, René Sitters, and Leen Stougie. On-line dial-a-ride problems under a restricted information model. *Algorithmica*, 40(4):319–329, 2004. doi:10.1007/s00453-004-1116-z.

36 Nicole Megow and Andreas S. Schulz. On-line scheduling to minimize average completion time revisited. *Operations Research Letters*, 32(5):485–490, 2004. doi:10.1016/j.orl.2003.11.008.

37 Sartaj Sahni and Teofilo F. Gonzalez. P-complete approximation problems. *Journal of the ACM*, 23(3):555–565, 1976. doi:10.1145/321958.321975.

38 Andreas S. Schulz and Martin Skutella. Scheduling unrelated machines by randomized rounding. *SIAM Journal on Discrete Mathematics*, 15(4):450–469, 2002. doi:10.1137/S0895480199357078.

39 Steven S. Seiden. A guessing game and randomized online algorithms. In *Proc. 32nd ACM Symp. on Theory of Computing (STOC)*, pages 592–601, 2000. doi:10.1145/335305.335385.

40 René Sitters. The minimum latency problem is NP-hard for weighted trees. In *Proc. 9th Int. Conf. on Integer Programming and Combinatorial Optimization (IPCO)*, pages 230–239, 2002. doi:10.1007/3-540-47867-1_17.

41 René Sitters. Efficient algorithms for average completion time scheduling. In *Proc. 14th Int. Conf. on Integer Programming and Combinatorial Optimization (IPCO)*, pages 411–423, 2010. doi:10.1007/978-3-642-13036-6_31.

42 René Sitters. Polynomial time approximation schemes for the traveling repairman and other minimum latency problems. In *Proc. 25th ACM-SIAM Symp. on Discrete Algorithms (SODA)*, pages 604–616, 2014. doi:10.1137/1.9781611973402.46.

43 René Sitters. Approximability of average completion time scheduling on unrelated machines. *Mathematical Programming*, 161(1-2):135–158, 2017. doi:10.1007/s10107-016-1004-8.

44 Martin Skutella. Semidefinite relaxations for parallel machine scheduling. In *Proc. 39th IEEE Symp. on Foundations of Computer Science (FOCS)*, pages 472–481, 1998. doi:10.1109/SFCS.1998.743498.

45 Arjen P. A. Vestjens. *On-line Machine Scheduling*. PhD thesis, Eindhoven University of Technology, 1997.

Counting Short Vector Pairs by Inner Product and Relations to the Permanent

Andreas Björklund[1]
Lund, Sweden

Petteri Kaski
Department of Computer Science, Aalto University, Espoo, Finland

Abstract

Given as input two n-element sets $\mathcal{A}, \mathcal{B} \subseteq \{0,1\}^d$ with $d = c \log n \leq (\log n)^2/(\log\log n)^4$ and a target $t \in \{0,1,\ldots,d\}$, we show how to count the number of pairs $(x,y) \in \mathcal{A} \times \mathcal{B}$ with integer inner product $\langle x,y\rangle = t$ deterministically, in $n^2/2^{\Omega\left(\sqrt{\log n \log\log n/(c\log^2 c)}\right)}$ time. This demonstrates that one can solve this problem in deterministic subquadratic time almost up to $\log^2 n$ dimensions, nearly matching the dimension bound of a subquadratic randomized detection algorithm of Alman and Williams [FOCS 2015]. We also show how to modify their randomized algorithm to count the pairs w.h.p., to obtain a fast randomized algorithm.

Our deterministic algorithm builds on a novel technique of reconstructing a function from sum-aggregates by prime residues, or *modular tomography*, which can be seen as an additive analog of the Chinese Remainder Theorem.

As our second contribution, we relate the fine-grained complexity of the task of counting of vector pairs by inner product to the task of computing a zero-one matrix permanent over the integers.

2012 ACM Subject Classification Theory of computation → Design and analysis of algorithms

Keywords and phrases additive reconstruction, Chinese Remainder Theorem, counting, inner product, modular tomography, orthogonal vectors, permanent

Digital Object Identifier 10.4230/LIPIcs.ICALP.2021.29

Category Track A: Algorithms, Complexity and Games

Related Version *Previous Version*: `https://arxiv.org/abs/2007.14092`

Acknowledgements We thank Virginia Vassilevska Williams and Ryan Williams for many useful discussions. We also thank the anonymous reviewers for their useful remarks.

1 Introduction

The Inner Product and the Size of Preimages. The inner product map $\langle x,y\rangle = \sum_{i=1}^{d} x_i y_i$ of two d-dimensional vectors $x = (x_1, x_2, \ldots, x_d)$ and $y = (y_1, y_2, \ldots, y_d)$ is one of the cornerstones of linear algebra and its applications. For example, when x and y are vectors of observations normalized to zero mean and unit standard deviation, $\langle x,y\rangle$ is the Pearson correlation between x and y. As such, it is a fundamental computational and data-analytical task to efficiently gain information about the *preimages* of the inner product map; for example, to highlight pairs of similar or dissimilar observables between two families of n observables.

[1] This work was carried out while A.B. was employed as a researcher at Lund University, Department of Computer Science, and the major part of the writeup was carried out while A.B. was employed as a researcher at Ericsson Research

The protagonist of this paper is the following counting problem (#InnerProduct):

Given as input a target $t \in \{0, 1, \ldots, d\}$ and two n-element sets $\mathcal{A} \subseteq \{0,1\}^d$ and $\mathcal{B} \subseteq \{0,1\}^d$, count the number of vector pairs $(x, y) \in \mathcal{A} \times \mathcal{B}$ with integer inner product $\langle x, y \rangle = t$.

From a complexity-theoretic standpoint, this problem generalizes many conjectured-hard problems in the study of fine-grained complexity – such as the $t = 0$ special case, the *orthogonal vector counting* (#OV) problem – as well as generalizing fundamental application settings, such as similarity search in Hamming spaces. While it is immediate that subquadratic scalability in n is obtainable when $d = o(\log n)$, our interest in this paper is to obtain an improved understanding of the fine-grained complexity landscape for *moderately short* vectors, specifically for d at most poly-logarithmic in n.

Subquadratic Scaling for Moderately Short Vectors. Our main positive result establishes deterministic subquadratic scalability for #InnerProduct up to d growing essentially as the square of the logarithm of n:

▶ **Theorem 1** (Main; Subquadratic Scaling for #InnerProduct)**.** *There exists a deterministic algorithm that, given as input a target $t \in \{0, 1, \ldots, c \log n\}$ and two n-element sets $\mathcal{A}, \mathcal{B} \subseteq \{0,1\}^{c \log n}$ with $4 \leq c \leq \frac{\log n}{(\log \log n)^4}$, outputs the number of pairs $(x, y) \in \mathcal{A} \times \mathcal{B}$ with $\langle x, y \rangle = t$ in time*

$$n^2 / 2^{\Omega\left(\sqrt{\frac{\log n \log \log n}{c \log^2 c}}\right)}. \tag{1}$$

The algorithm in Theorem 1 is based on a novel technique of reconstructing a function from its sum-aggregates by prime residue, which can be seen as an *additive* analog of the Chinese Remainder Theorem and may be of independent interest (cf. Sect. 2).

We also show how a randomized algorithm for the decision problem of checking for a pair of vectors whose Hamming distance is less than a target by Alman and Williams [6], can with a small modification be turned into an algorithm for #InnerProduct.

▶ **Theorem 2** (Randomized Subquadratic Scaling for #InnerProduct)**.** *There exists a randomized algorithm that w.h.p., given as input a target $t \in \{0, 1, \ldots, c \log n\}$ and two n-element sets $\mathcal{A}, \mathcal{B} \subseteq \{0,1\}^{c \log n}$ with $4 \leq c \leq \frac{\log n}{(\log \log n)^3}$, outputs the number of pairs $(x, y) \in \mathcal{A} \times \mathcal{B}$ with $\langle x, y \rangle = t$ in time*

$$n^2 / 2^{\Omega\left(\frac{\log n}{c \log^2 c}\right)}. \tag{2}$$

While the randomized algorithm in Theorem 2 is faster than the deterministic one in Theorem 1, we stress that as far as we know no deterministic algorithm in subquadratic time was previously known for #InnerProduct, even for $O(\log n)$ dimensions. In particular, derandomizing Theorem 2 while retaining subquadratic time seems challenging, even though some progress on the amount of randomness needed in the algorithm has been made, cf. Theorem 1.1 in [3].

Our further objective is to understand the fine-grained complexity of #InnerProduct in relation to that of #OV and other counting problems. For $d = O(\log n)$, it is known that these problems are truly-subquadratically related; indeed, Chen and Williams [15] give a parsimonious reduction for the detection variants of these two problems. That is, if #OV can be solved in $n^{2-\omega(1)}$ time, then so can #InnerProduct. However, while

there is a subquadratic time algorithm for #OV whose running time scales as good as $n^{2-\Omega(1/\log c)}$ [14], the reduction of Chen and Williams [15] does not immediately give a non-trivial algorithm for #InnerProduct. Indeed, the fastest known algorithm for the decision version InnerProduct utilizes probabilistic polynomials for symmetric Boolean functions with optimal dependence on the degree and error [6], and does not go via fast OV algorithms and the reduction above. In Theorem 2, we show how a simple modification to the algorithm in Alman and Williams [6] can turn their algorithm into a counting one. We note that while Alman, Chan, and Williams [3] later presented a deterministic algorithm based on Chebyshev polynomials over the reals for minimum/maximum Hamming weight pair, with the same running time as the randomized one in [6], that deterministic algorithm, or the even faster randomized one they presented, can not be turned into one for #InnerProduct by our suggested modification alone.

Lower Bounds via the Permanent. The running times (1) and (2) would, at least at first, appear to leave room for improvement. Indeed, the running time (2) is considerably worse than the running time $n^{2-\Omega(1/\log c)}$ obtained by Chan and Williams [14] for #OV. We proceed to show that this intuition might be misleading, since such scalability would imply the existence of considerably faster algorithms for a canonical hard problem in exponential-time complexity. Accordingly, to gain insight into the complexity of #InnerProduct and #OV when $d = \omega(\log n)$, we introduce our second protagonist (R-Permanent):

> Given as input an $n \times n$ matrix M with entries m_{ij} in a ring R for $i, j \in [n]$, compute the *permanent*
>
> $$\operatorname{per} M = \sum_{\sigma \in S_n} \prod_{i \in [n]} m_{i,\sigma(i)} \,,$$
>
> where S_n is the group of all permutations of $[n] = \{1, 2, \ldots, n\}$.

Ryser's algorithm from 1963 computes the permanent with $O(n2^n)$ arithmetic operations in R [22]. It is a major open problem whether this can be improved to $O(c^n)$ for some constant $c < 2$. Even improving the running time to less than 2^n operations has been noted as a challenge by Knuth in the *Art of Computer Programming* [19, Exercise 4.6.4.11]. Valiant in 1979 famously proved that the permanent is #P-complete even when restricted to $m_{ij} \in \{0, 1\}$ and evaluated over the ring of integers [24]; this version of the problem can be interpreted as counting the perfect matchings in a balanced bipartite graph having the matrix as its biadjacency matrix. For zero-one inputs over the integers, *somewhat* faster algorithms are known (cf. related work below); to the best of our knowledge, the current champion for zero-one matrices computes the permanent in $2^{n-\Omega\left(\sqrt{n/\log\log n}\right)}$ time [12].

As our second contribution, we relate the fine-grained scalability of #InnerProduct and #OV to the task of computing the permanent of a zero-one matrix over the integers. In particular, our first result shows that if we could solve #InnerProduct as fast as the fastest currently known algorithms for #OV [14], then we would immediately obtain a much faster algorithm for the zero-one matrix permanent:

▶ **Theorem 3** (Lower Bound for #InnerProduct via Integer Permanent). *If there exists an algorithm for solving #InnerProduct for N vectors from $\{0,1\}^{c \log N}$ in time $N^{2-\Omega(1/\log c)}$, then there exists an algorithm solving the permanent of an $n \times n$ zero-one matrix over the integers in time $2^{n-\Omega(n/\log n)}$.*

Thus, despite the true-subquadratic equivalence for $d = O(\log n)$ [15], it would appear that #InnerProduct and #OV have different complexity characteristics when $d = \omega(\log n)$.

Our next result shows that a modest improvement in fine-grained scalability of #OV would likewise imply much faster algorithms for the permanent.

▶ **Theorem 4** (Lower Bound for #OV via Integer Permanent)**.** *If there exists an algorithm for solving #OV for N vectors from $\{0,1\}^{c \log N}$ in time $N^{2-\Omega(1/\log^{1-\epsilon} c)}$ for some $\epsilon > 0$, then there exists an algorithm solving the permanent of an $n \times n$ zero-one matrix over the integers in time $2^{n-\Omega(n/\log^{2/\epsilon-2} n)}$.*

We note that such fast algorithms for #OV would already disprove the so-called Super Strong ETH, that k-CNFSAT on n variables has no $2^{n-n/o(k)}$-time algorithm, by the reduction to OV by Williams [26] after sparsification [17]. The present result merely adds to the list of consequences of faster algorithms for #OV.

Methodology and Organization of the Paper. The key methodological contribution underlying our main algorithmic result (Theorem 1) is a novel additive analog of the Chinese Remainder Theorem (Lemma 5 developed in Sect. 2 independently of the application), which enables us to recover the number of pairs $(x, y) \in \mathcal{A} \times \mathcal{B}$ with $\langle x, y \rangle = t$ from counts of pairs (x, y) satisfying $\langle x, y \rangle \equiv r \pmod{p}$ for multiple small primes p and residues $r \in \{0, 1, \ldots, p-1\}$. In particular, the crux of the algorithmic speedup lies in the observation that to recover the count associated with a target $0 \leq t \leq d$, primes up to roughly $\sqrt{d}$ suffice by Lemma 5. To obtain the counts of pairs in each residue class r modulo p, we employ the polynomial method with modulus-amplifying polynomials of Beigel and Tarui [9] to accommodate the counts under a prime-power modulus, with fast rectangular matrix multiplication of Coppersmith [16] as the key subroutine implementing the count; this latter part of the algorithm design developed in Sect. 3 follows well-known techniques in fine-grained algorithm design (e.g. [3]). Similarly, the randomized algorithm design in Theorem 2 follows by a minor adaptation of the probabilistic-polynomial techniques of Alman and Williams [6] to a counting context; a proof is relegated to Sect. 4.

Our two lower-bound reductions, Theorem 3 and Theorem 4, rely on reducing an $m \times m$ integer permanent first via the Chinese Remainder Theorem into permanents modulo multiple primes p with $p \leq m \ln m$, and then using algebraic splitting via Ryser's formula [22] to obtain short-vector instances of #InnerProduct and #OV, respectively. For #InnerProduct and Theorem 3, the split employs a novel discrete-logarithm version of Ryser's formula modulo p to arrive at two collections of vectors whose counts of pairs with specific inner products enable recovery of the permanent modulo p; the proof is presented in Sect. 5. For #OV and Theorem 4, the split analogously employs Ryser's formula modulo p but with a more intricate vector-coding of group residues modulo p to obtain the desired correspondence with counts of pairs of orthogonal vectors; we relegate the proof to Sect. 6.

Related Work and Further Applications. As regards exact and approximate inner products, Abboud, Williams, and Yu [1] used the polynomial method to construct a randomized subquadratic time algorithm for OV. Chan and Williams [14] derandomized the algorithm and showed that it could also solve the counting problem #OV. The first result that addressed an inner product different from zero, was the randomized algorithm for minimum Hamming weight pair by Alman and Williams [6]. Subsequently, Alman, Chan, and Williams [3] found an even faster randomized as well as a deterministic subquadratic algorithm matching [6].

A number of studies address approximate versions of inner-product counting in subquadratic time, such as the detection of outlier correlations and offline computation of approximate nearest neighbors, including Valiant [23], Karppa, Kaski, and Kohonen [18], Alman [2], and Alman, Chan, and Williams [4]. All the algorithms use fast rectangular matrix multiplication.

As regards permanents, Bax and Franklin presented a randomised $2^{n-\Omega(n^{1/3}/\log n)}$ expected time algorithm for the 0/1-matrix permanent [8]. Björklund [10] derived a faster and deterministic $2^{n-\Omega(\sqrt{n/\log n})}$ time algorithm. The algorithm was subsequently improved to a deterministic $2^{n-\Omega(\sqrt{n/\log\log n})}$ time algorithm by Björklund, Kaski, and Williams [12].

For the computation of an integer matrix permanent modulo a prime power $p^{\lambda n/p}$ for any constant $\lambda < 1$, Björklund, Husfeldt, and Lyckberg [11] derived a $2^{n-\Omega(n/(p\log p))}$ time algorithm. For the computation of a matrix permanent over an arbitrary ring R on r elements, Björklund and Williams [13] gave a deterministic $2^{n-\Omega(\frac{n}{r})}$ time algorithm.

The problem #InnerProduct has various applications in combinatorial algorithms. For example, it can be used to count the satisfying assignments to a Sym∘And formula (cf. Sect. 7), or compute the weight enumerator polynomial of a linear code (cf. Sect. 7). We would also like to highlight a recent application of deterministic algorithms for #OV [14] in efficient construction of rigid matrices [5].

2 Reconstruction from Sum-Aggregates by Prime Residue

This section develops the main methodological contribution of this work. Namely, we show that a complex-valued function $f : D \to \mathbb{C}$ can be reconstructed from its sum-aggregates by prime residue when the domain D is a prefix of the set of nonnegative integers. In essence, reconstruction of a function from its sum-aggregates can be viewed as an *additive* analog of the Chinese Remainder Theorem; that is, we obtain reconstruction up to the *sum* of the prime moduli – in the precise sense of (3) below – whereas the Chinese Remainder Theorem enables reconstruction up to the product of the moduli.[2]

In our application of counting pairs of vectors by inner product, we let f be a counting function such that $f(\ell)$ counts the number of pairs $(x, y) \in \mathcal{A} \times \mathcal{B}$ with $\langle x, y\rangle = \ell$. Reconstruction from sum-aggregates then enables us to recover f by counting the number of pairs (x, y) with $\langle x, y\rangle \equiv r \pmod p$ for small primes p and residues $r \in \{0, 1, \ldots, p-1\}$; we postpone the details of this application to Sect. 3 and first proceed to study reconstructibility.

Sum-Aggregation by Prime Residue. Let $p_1, p_2, \ldots, p_m$ be distinct prime numbers and let

$$D \subseteq \{0, 1, \ldots, s_m - 1\} \qquad \text{where} \qquad s_m = 1 + \sum_{b=1}^{m} (p_b - 1)\,. \tag{3}$$

Letting f_ℓ be shorthand for $f(\ell)$, we show that we can recover f from the sequence of its *sum-aggregates*

$$F_{br} = \sum_{\substack{\ell \in D \\ \ell \equiv r \pmod{p_b}}} f_\ell \quad \text{for each residue } r \in \{0, 1, \ldots, p_b - 1\} \text{ and each } b \in \{1, 2, \ldots, m\}. \tag{4}$$

[2] Here it should be noted that the scope of the classical Chinese Remainder Theorem is also somewhat more restricted than our present setting; indeed, the Chinese Remainder Theorem does not enable the reconstruction of an arbitrary function f but rather is restricted to reconstruction in the case when f is known to vanish in all but one point of D. We also note that the present additive setting can be captured algebraically through generalized Chinese remaindering in polynomial rings; here we refer to the alternative proof of Lemma 5 at the end of this section.

To start with, we observe that this sequence is linearly dependent. Indeed, define

$$F_{01} = \sum_{\ell \in D} f_\ell \tag{5}$$

and observe that for each $b \in \{1, 2, \ldots, m\}$ we have the linear relation $F_{01} = \sum_{r=0}^{p_b - 1} F_{br}$.

To obtain an equivalent and – as we will shortly show – linearly independent sequence, take the sequence formed by the sum F_{01} followed by F_{br} for each *nonzero* residue $r \in \{1, 2, \ldots, p_b - 1\}$ and each $b \in \{1, 2, \ldots, m\}$. Let us write F for this sequence of length s_m. By extending the domain of the function f with zero-values $f_\ell = 0$ as needed, we can also assume that $D = \{0, 1, \ldots, s_m - 1\}$ in what follows.

Sum-Aggregation as a Linear System – Modular Tomography. Let us now study reconstruction of f from F. From (4) and (5) we observe that to reconstruct f from F it suffices to solve the linear system

$$F = Af\,, \tag{6}$$

where A is the $s_m \times s_m$ *nonzero residue aggregation matrix* whose entries are defined for all $b \in \{0, 1, 2, \ldots, m\}$, $i \in \{1, 2, \ldots, p_b - 1\}$, and $\ell \in \{0, 1, \ldots, s_m - 1\}$ by the rule

$$A_{bi,\ell} = \begin{cases} 1 & \text{if } b = 0; \\ 1 & \text{if } b \geq 1 \text{ and } i \equiv \ell \pmod{p_b}; \\ 0 & \text{if } b \geq 1 \text{ and } i \not\equiv \ell \pmod{p_b}, \end{cases} \tag{7}$$

where we have assumed for convenience that $p_0 = 2$. Indeed, we readily verify from (4), (5), and (7) that

$$F_{bi} = \sum_{\ell=0}^{s_m - 1} A_{bi,\ell} f_\ell$$

holds for each $b \in \{0, 1, \ldots, m\}$ and $i \in \{0, 1, \ldots, p_b - 1\}$. When we want to stress the m selected primes, we write $A^{p_1, p_2, \ldots, p_m}$ for the matrix A.

The row-banded structure given by (7) is perhaps easiest illustrated with a small example. Below we display the matrix A for the primes $p_1 = 2$, $p_2 = 3$, and $p_3 = 5$:

$$A^{2,3,5} = \left[\begin{array}{cccccccc} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ \hline 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 \\ \hline 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \\ \hline 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \end{array}\right]. \tag{8}$$

Observe in particular that the first band $b = 0$ corresponds to the sum (5) and the subsequent bands $b \in \{1, 2, \ldots, m\}$ each correspond to one of the primes $p_1, p_2, \ldots, p_m$ so that the $p_b - 1$ rows inside each band correspond to the sum-aggregates (4) of the $p_b - 1$ *nonzero* residue classes modulo p_b.

Our main technical lemma establishes that the matrix A is invertible, thus enabling reconstruction of f from F. Or, what is the same, we can recover f by *modular tomography* from the sequence F of *tomographs*; indeed, from a tomographic standpoint we can by (7) and (8) view each $b \in \{1, 2, \ldots, m\}$ as corresponding to a bundle of pairwise disjoint parallel lines aggregating the data f, with $i \in \{1, 2, \ldots, p_b - 1\}$ indexing the lines in a bundle.

▶ **Lemma 5** (Reconstruction from Sum-Aggregates by Prime Residue). *The nonzero residue aggregation matrix $A^{p_1,p_2,\ldots,p_m}$ is invertible whenever $p_1, p_2, \ldots, p_m$ are distinct primes.*

The key idea in the proof is to decompose $A^{p_1,p_2,\ldots,p_m}$ over the complex numbers into the product of a near-block-diagonal matrix with near-Vandermonde blocks and a Vandermonde matrix, both of which are then shown to have nonzero determinant. We will also present an alternative proof using elementary commutative algebra.

Proof of the Reconstruction Lemma. We now prove Lemma 5. We show that for distinct primes $p_1, p_2, \ldots, p_m$ the matrix $A = A^{p_1,p_2,\ldots,p_m}$ is invertible over rational numbers. Our strategy is to show that $A = UV$ for two complex matrices U and V that both have nonzero determinant. Indeed, the near-cyclic banded structure of A suggests that one should pursue a decomposition in terms of block-structured near-Vandermonde matrices. Let us first define the matrices U and V, then present a small example, and then complete the proof.

We will use the following standard facts about complex roots of unity. For a positive integer N, let us write $\omega_N = \exp\left(\frac{2\pi\mathfrak{i}}{N}\right)$, where $\mathfrak{i} = \sqrt{-1}$ is the imaginary unit. For all $m \in \mathbb{Z}$ we have

$$\frac{1}{N}\sum_{j=0}^{N-1} \omega_N^{km} = \begin{cases} 1 & \text{if } k \equiv 0 \pmod{N}; \\ 0 & \text{if } k \not\equiv 0 \pmod{N}. \end{cases} \tag{9}$$

The matrix U will use a $(m+1) \times (m+1)$ block structure that is similar to the $(m+1)$-band structure of A, but now the structure is used both for rows and columns. Again for convenience we assume $p_0 = 2$. The matrix U is defined for all $b \in \{0, 1, 2, \ldots, m\}$, $i \in \{1, 2, \ldots, p_b - 1\}$, $d \in \{0, 1, \ldots, m\}$, and $k \in \{1, 2, \ldots, p_d - 1\}$ by the rule

$$U_{bi,dk} = \begin{cases} 1 & \text{if } d = 0 \text{ and } b = 0; \\ \frac{1}{p_b} & \text{if } d = 0 \text{ and } b \geq 1; \\ 0 & \text{if } d \geq 1 \text{ and } b \neq d; \\ \frac{1}{p_b}\omega_{p_b}^{-ik} & \text{if } d \geq 1 \text{ and } b = d. \end{cases} \tag{10}$$

The matrix V is a Vandermonde matrix with $(m+1)$-banded structure defined for all $d \in \{0, 1, \ldots, m\}$, $k \in \{1, 2, \ldots, p_d - 1\}$, and $\ell \in \{0, 1, \ldots, s_m - 1\}$ by the rule

$$V_{dk,\ell} = \begin{cases} 1 & \text{if } d = 0; \\ \omega_{p_d}^{k\ell} & \text{if } d \geq 1\,. \end{cases} \tag{11}$$

Before proceeding with the proof that $A = UV$, let us present an example for the primes $p_1 = 2$, $p_2 = 3$, and $p_3 = 5$; recall (8). We have

$$\left[\begin{array}{cccccccc} 1&1&1&1&1&1&1&1\\ \hline 0&1&0&1&0&1&0&1\\ \hline 0&1&0&0&1&0&0&1\\ 0&0&1&0&0&1&0&0\\ \hline 0&1&0&0&0&0&1&0\\ 0&0&1&0&0&0&0&1\\ 0&0&0&1&0&0&0&0\\ 0&0&0&0&1&0&0&0 \end{array}\right]_{A^{2,3,5}} =$$

$$= \left[\begin{array}{c|c|cc|cccc} 1&0&0&0&0&0&0&0\\ \hline \frac{1}{2}&\frac{1}{2}\omega_2^{-1\cdot 1}&0&0&0&0&0&0\\ \hline \frac{1}{3}&0&\frac{1}{3}\omega_3^{-1\cdot 1}&\frac{1}{3}\omega_3^{-1\cdot 2}&0&0&0&0\\ \frac{1}{3}&0&\frac{1}{3}\omega_3^{-2\cdot 1}&\frac{1}{3}\omega_3^{-2\cdot 2}&0&0&0&0\\ \hline \frac{1}{5}&0&0&0&\frac{1}{5}\omega_5^{-1\cdot 1}&\frac{1}{5}\omega_5^{-1\cdot 2}&\frac{1}{5}\omega_5^{-1\cdot 3}&\frac{1}{5}\omega_5^{-1\cdot 4}\\ \frac{1}{5}&0&0&0&\frac{1}{5}\omega_5^{-2\cdot 1}&\frac{1}{5}\omega_5^{-2\cdot 2}&\frac{1}{5}\omega_5^{-2\cdot 3}&\frac{1}{5}\omega_5^{-2\cdot 4}\\ \frac{1}{5}&0&0&0&\frac{1}{5}\omega_5^{-3\cdot 1}&\frac{1}{5}\omega_5^{-3\cdot 2}&\frac{1}{5}\omega_5^{-3\cdot 3}&\frac{1}{5}\omega_5^{-3\cdot 4}\\ \frac{1}{5}&0&0&0&\frac{1}{5}\omega_5^{-4\cdot 1}&\frac{1}{5}\omega_5^{-4\cdot 2}&\frac{1}{5}\omega_5^{-4\cdot 3}&\frac{1}{5}\omega_5^{-4\cdot 4} \end{array}\right]_{U^{2,3,5}} \cdot$$

$$\cdot \left[\begin{array}{cccccccc} 1&1&1&1&1&1&1&1\\ \hline \omega_2^{1\cdot 0}&\omega_2^{1\cdot 1}&\omega_2^{1\cdot 2}&\omega_2^{1\cdot 3}&\omega_2^{1\cdot 4}&\omega_2^{1\cdot 5}&\omega_2^{1\cdot 6}&\omega_2^{1\cdot 7}\\ \hline \omega_3^{1\cdot 0}&\omega_3^{1\cdot 1}&\omega_3^{1\cdot 2}&\omega_3^{1\cdot 3}&\omega_3^{1\cdot 4}&\omega_3^{1\cdot 5}&\omega_3^{1\cdot 6}&\omega_3^{1\cdot 7}\\ \omega_3^{2\cdot 0}&\omega_3^{2\cdot 1}&\omega_3^{2\cdot 2}&\omega_3^{2\cdot 3}&\omega_3^{2\cdot 4}&\omega_3^{2\cdot 5}&\omega_3^{2\cdot 6}&\omega_3^{2\cdot 7}\\ \hline \omega_5^{1\cdot 0}&\omega_5^{1\cdot 1}&\omega_5^{1\cdot 2}&\omega_5^{1\cdot 3}&\omega_5^{1\cdot 4}&\omega_5^{1\cdot 5}&\omega_5^{1\cdot 6}&\omega_5^{1\cdot 7}\\ \omega_5^{2\cdot 0}&\omega_5^{2\cdot 1}&\omega_5^{2\cdot 2}&\omega_5^{2\cdot 3}&\omega_5^{2\cdot 4}&\omega_5^{2\cdot 5}&\omega_5^{2\cdot 6}&\omega_5^{2\cdot 7}\\ \omega_5^{3\cdot 0}&\omega_5^{3\cdot 1}&\omega_5^{3\cdot 2}&\omega_5^{3\cdot 3}&\omega_5^{3\cdot 4}&\omega_5^{3\cdot 5}&\omega_5^{3\cdot 6}&\omega_5^{3\cdot 7}\\ \omega_5^{4\cdot 0}&\omega_5^{4\cdot 1}&\omega_5^{4\cdot 2}&\omega_5^{4\cdot 3}&\omega_5^{4\cdot 4}&\omega_5^{4\cdot 5}&\omega_5^{4\cdot 6}&\omega_5^{4\cdot 7} \end{array}\right]_{V^{2,3,5}} . \tag{12}$$

The main technical aspect of the proof that $A = UV$ is to partition the index $\ell \in \{0, 1, \ldots, s_m - 1\}$ to the $m + 1$ bands. Towards this end, define for each $c \in \{0, 1, \ldots, m\}$ the prefix-sum

$$\underline{s}_c = \begin{cases} 0 & \text{if } c = 0; \\ 1 + \sum_{\ell=1}^{c-1} \big(p_c - 1\big) & \text{if } c \geq 1. \end{cases}$$

In particular, for every $\ell \in \{0, 1, \ldots, s_m - 1\}$, we observe that there exist unique $c \in \{0, 1, \ldots, m\}$ and $j \in \{1, 2, \ldots, p_c - 1\}$ such that

$$\ell = j - 1 + \underline{s}_c \,. \tag{13}$$

We are now ready to show that $A = UV$. Let $b \in \{0, 1, \ldots, m\}$, $i \in \{0, 1, \ldots, p_b - 1\}$, and $\ell \in \{0, 1, \ldots, s_k - 1\}$ be arbitrary. Let $c \in \{0, 1, \ldots, m\}$ and $j \in \{1, 2, \ldots, p_c - 1\}$ be uniquely determined from ℓ by (13). From (10), (11), (9), and (7) we observe that

$$\sum_{d=0}^{m} \sum_{k=0}^{p_d - 1} U_{bi,dk} V_{dk,\ell} =$$

$$= \begin{cases} 1 & \text{if } b = 0; \\ \frac{1}{p_b}\big(1 + \sum_{k=1}^{p_b-1} \omega_{p_b}^{-ik+k(j-1+\underline{s}_b)}\big) & \\ \qquad = \frac{1}{p_b} \sum_{k=0}^{p_b-1} \omega_{p_b}^{k(j-i-1+\underline{s}_b)} = 1 & \text{if } b \geq 1 \text{ and } i \equiv j - 1 + \underline{s}_b = \ell \pmod{p_b}; \\ \frac{1}{p_b}\big(1 + \sum_{k=1}^{p_b-1} \omega_{p_b}^{-ik+k(j-1+\underline{s}_b)}\big) & \\ \qquad = \frac{1}{p_b} \sum_{k=0}^{p_b-1} \omega_{p_b}^{k(j-i-1+\underline{s}_b)} = 0 & \text{if } b \geq 1 \text{ and } i \not\equiv j - 1 + \underline{s}_b = \ell \pmod{p_b}. \end{cases}$$

$$= A_{bi,\ell} \,.$$

Thus, $A = UV$ holds. It remains to show that both matrices U and V have nonzero determinant over the complex numbers. Starting with the Vandermonde matrix V, let $\nu_0 = 1$ and $\nu_\ell = \omega_{p_c}^j$ for $\ell \in \{1, 2, \ldots, s_m - 1\}$, where $c \in \{0, 1, \ldots, m\}$ and $j \in \{1, 2, \ldots, p_c - 1\}$ are uniquely determined from ℓ by (13). In particular, we observe that V is a Vandermonde matrix with $D = s_m - 1$ and $V = \big[\nu_i^j\big]_{i=0,1,\ldots,D}^{j=0,1,\ldots,D}$. The Vandermonde determinant formula

thus gives $\det V = \sum_{0 \le k < \ell \le D} (\nu_\ell - \nu_k)$. Furthermore, this determinant is nonzero because $p_1, p_2, \ldots, p_m$ are distinct primes and thus $\nu_0, \nu_1, \ldots, \nu_D$ are distinct. Next, let us consider the matrix U defined by (10). At this point it may be useful to revisit the structure of U via the example (12). We observe that the block-diagonal of U with $b = c \ge 1$ consists of matrices that each decompose into the product of a $(p_b - 1) \times (p_b - 1)$ diagonal matrix with diagonal entries $\frac{1}{p_b}\omega_{p_b}^{-i}$ for $i \in \{1, 2, \ldots, p_b - 1\}$ and a $(p_b - 1) \times (p_b - 1)$ Vandermonde matrix with a nonzero determinant since $\omega_{p_b}^{-i}$ for $i \in \{1, 2, \ldots, p_b - 1\}$ are distinct. Thus, since the determinant of U is the product of the determinants of the block-matrices on the diagonal, each of which is nonzero, the determinant of U is nonzero. It follows that A is invertible and thus given F we can solve for f via (6). This completes the proof of Lemma 5. ◀

An Alternative Proof. Let us sketch an alternative algebraic proof for Lemma 5. Let us view $f = \sum_{\ell=0}^{s_m - 1} f_\ell x^\ell \in \mathbb{C}[x]$ as a univariate polynomial in the indeterminate x over the complex numbers. For each $b \in \{1, 2, \ldots, m\}$, we observe that the polynomial remainder of f divided by $x^{p_b} - 1$ satisfies $f \operatorname{rem} (x^{p_b} - 1) = \sum_{i=0}^{p_b - 1} F_{bi} x^i \in \mathbb{C}[x]$. Furthermore, for distinct $b, c \in \{1, 2, \ldots, m\}$ we have that the sets of complex roots of $x^{p_b} - 1$ and $x^{p_c} - 1$ are disjoint apart from the common root $x = 1$; that is, $(x^{p_b} - 1)/(x - 1)$ and $(x^{p_c} - 1)/(x - 1)$ are coprime polynomials. Thus, by (3) and the Chinese Remainder Theorem in $\mathbb{C}[x]$, we can recover f from the sequence of remainders $f \operatorname{rem} (x^{p_b} - 1)$ for $b \in \{1, 2, \ldots, m\}$; cf. von zur Gathen and Gerhard [25] for the algebraic background and associated near-linear-time algorithmic toolbox. ◀

3 Counting Pairs of Zero-One Vectors by Inner Product

This section documents our main algorithm and proves Theorem 1. Let κ be a parameter that satisfies, with foresight,

$$4 \le \kappa \le \frac{\log n}{(\log \log n)^4}\,. \tag{14}$$

Let $a^{(1)}, a^{(2)}, \ldots, a^{(n)} \in \{0,1\}^d$ and $b^{(1)}, b^{(2)}, \ldots, b^{(n)} \in \{0,1\}^d$ be given as input with $d \le \kappa \log n$. We want to compute for each $t \in \{0, 1, \ldots, d\}$ the count

$$f_t = |\{(i,j) \in \{1,2,\ldots,n\}^2 : \langle a^{(i)}, b^{(j)} \rangle = t\}|\,.$$

Our high-level approach will be to use Lemma 5 and (6) to solve for the counts $f_0, f_1, \ldots, f_d$ using as input counts that have been sum-aggregated by prime residue. More precisely, we will work with prime moduli $p_1, p_2, \ldots, p_m$ and develop an algorithm that computes, for given further input $p \in \{p_1, p_2, \ldots, p_m\}$ and $r \in \{0, 1, \ldots, p-1\}$, the sum-aggregated count

$$F_{pr} = |\{(i,j) \in \{1,2,\ldots,n\}^2 : \langle a^{(i)}, b^{(j)} \rangle \equiv r \pmod p\}|\,.$$

The detailed choices for m and the primes $p_1, p_2, \ldots, p_m$ will be presented later.

The Residue-Indicator Polynomial. Assume p and r have been given. We will rely on the polynomial method, and accordingly we first build a standard polynomial that indicates the residue r modulo p in a pair of vectors.

Let $x = (x_1, x_2, \ldots, x_d)$ and $y = (y_1, y_2, \ldots, y_d)$ be two vectors of indeterminates. By Fermat's little theorem, the $2d$-indeterminate polynomial

$$G_{p,r}(x,y) = 1 - \left(\sum_{k=1}^{d} x_k y_k - r\right)^{p-1} \tag{15}$$

satisfies for all $i, j \in \{1, 2, \ldots, n\}$ the indicator property

$$G_{p,r}\big(a^{(i)}, b^{(j)}\big) \equiv \begin{cases} 1 \pmod p & \text{if } \langle a^{(i)}, b^{(j)} \rangle \equiv r \pmod p; \\ 0 \pmod p & \text{if } \langle a^{(i)}, b^{(j)} \rangle \not\equiv r \pmod p. \end{cases} \tag{16}$$

We observe that $G_{p,r}$ has degree $2p - 2$.

Modulus Amplification for Zero-One Residues. To enable taking the sum of a large number of indicators, we make use of the *modulus amplifying polynomials* of Beigel and Tarui [9].

▶ **Theorem 6** (Modulus amplification; Beigel and Tarui [9]). *For $h \in \mathbb{Z}_{\geq 1}$, define the polynomial*

$$A_h(z) = 1 - (1 - z)^h \sum_{j=0}^{h-1} \binom{h + j - 1}{j} z^j \,. \tag{17}$$

Then, for all $m \in \mathbb{Z}_{\geq 2}$ and $s \in \mathbb{Z}$, we have

(i) *$s \equiv 0 \pmod m$ implies $A_h(s) \equiv 0 \pmod{m^h}$, and*

(ii) *$s \equiv 1 \pmod m$ implies $A_h(s) \equiv 1 \pmod{m^h}$.*

We observe that A_h has degree $2h - 1$. Composing (17) and (15), we obtain the amplified residue-indicator polynomial

$$G^h_{p,r}(x, y) = A_h\big(G_{p,r}(x, y)\big) \,. \tag{18}$$

From (16) and Theorem 6, we observe the amplified indicator property

$$G^h_{p,r}\big(a^{(i)}, b^{(j)}\big) \equiv \begin{cases} 1 \pmod{p^h} & \text{if } \langle a^{(i)}, b^{(j)} \rangle \equiv r \pmod p; \\ 0 \pmod{p^h} & \text{if } \langle a^{(i)}, b^{(j)} \rangle \not\equiv r \pmod p. \end{cases} \tag{19}$$

Furthermore, we observe that $G^h_{p,r}$ has degree $(2h - 1)(2p - 2)$.

Multilinear Reduct and Bounding the Number of Monomials. For a nonnegative integer e, define $\underline{e} = 0$ if $e = 0$ and $\underline{e} = 1$ if $e \geq 1$. For a monomial $x_1^{e_1} x_2^{e_2} \cdots x_d^{e_d} y_1^{f_1} y_2^{f_2} \cdots y_d^{f_d}$, define the *multilinear reduct* by

$$\underline{x_1^{e_1} x_2^{e_2} \cdots x_d^{e_d} y_1^{f_1} y_2^{f_2} \cdots y_d^{f_d}} = x_1^{\underline{e}_1} x_2^{\underline{e}_2} \cdots x_d^{\underline{e}_d} y_1^{\underline{f}_1} y_2^{\underline{f}_2} \cdots y_d^{\underline{f}_d} \,.$$

For a polynomial $Q(x, y)$, define the multilinear reduct $\underline{Q}(x, y)$ by taking the multilinear reduct of each monomial $Q(x, y)$ and simplifying. Since $a^{(i)}$ and $b^{(j)}$ are $\{0, 1\}$-valued vectors, over the integers we have

$$Q\big(a^{(i)}, b^{(j)}\big) = \underline{Q}\big(a^{(i)}, b^{(j)}\big) \,. \tag{20}$$

Furthermore, if Q has degree D, then $\underline{Q}$ has at most $\sum_{j=0}^{D} \binom{2d}{j}$ monomials. In particular, we observe that $\underline{G}^h_{p,r}$ has at most $\sum_{j=0}^{4hp} \binom{2d}{j}$ monomials.

Split-Monomial Form of the Multilinear Reduct. Suppose that the multilinear reduct $\underline{G}^h_{p,r}(x,y)$ has exactly M monomials with the representation

$$\underline{G}^h_{p,r}(x,y) = \sum_{k=1}^{M} \gamma^{(k)}\, x_1^{e_1^{(k)}} x_2^{e_2^{(k)}} \cdots x_d^{e_d^{(k)}} y_1^{f_1^{(k)}} y_2^{f_2^{(k)}} \cdots y_d^{f_d^{(k)}}\,. \tag{21}$$

For $I, J \subseteq \{1,2,\ldots,n\}$ and $k \in \{1,2,\ldots,M\}$, define

$$L_{I,k} = \sum_{i\in I} \big(a_1^{(i)}\big)^{e_1^{(k)}} \big(a_2^{(i)}\big)^{e_2^{(k)}} \cdots \big(a_d^{(i)}\big)^{e_d^{(k)}} \gamma^{(k)}\,,\ R_{J,k} = \sum_{j\in J} \big(b_1^{(j)}\big)^{f_1^{(k)}} \big(b_2^{(j)}\big)^{f_2^{(k)}} \cdots \big(b_d^{(j)}\big)^{f_d^{(k)}}\,. \tag{22}$$

From (22), (21), (20), and (19), we have

$$\begin{aligned}\sum_{k=1}^{M} L_{I,k} R_{J,k} &= \sum_{i\in I}\sum_{j\in J} \underline{G}^h_{p,r}\big(a^{(i)}, b^{(j)}\big) \\ &\equiv \big|\big\{(i,j) \in I\times J : \langle a^{(i)}, b^{(j)}\rangle \equiv r \pmod p\big\}\big| \pmod{p^h}\,.\end{aligned} \tag{23}$$

In particular, assuming that $|I||J| \leq p^h - 1$, from (23) it follows that $\sum_{k=1}^{M} L_{I,k}R_{J,k}$ computed modulo p^h recovers the number of pairs $(i,j) \in I \times J$ with $\langle a^{(i)}, b^{(j)}\rangle \equiv r \pmod p$.

We now move from deriving the polynomial and its properties to describing the algorithm.

Algorithm for the Prime-Residue Count. The algorithm will rely on (23) via fast rectangular matrix multiplication to count the number of pairs $(i,j) \in \{1,2,\ldots,n\}^2$ that satisfy $\langle a^{(i)}, b^{(j)}\rangle \equiv r \pmod p$.

The algorithm first computes the explicit M-monomial representation of the polynomial $\underline{G}^h_{p,r}$ in (21). More precisely, the algorithm evaluates (15), (17), and (18) in explicit monomial representation, taking multilinear reducts with respect to the variables $x_1, x_2, \ldots, x_d, y_1, y_2, \ldots, y_d$ whenever possible. This results in the set

$$\{(k, \gamma^{(k)}, e_1^{(k)}, e_2^{(k)}, \ldots, e_d^{(k)}, f_1^{(k)}, f_2^{(k)}, \ldots, f_d^{(k)}) : k \in \{1,2,\ldots,M\}\}\,. \tag{24}$$

Next, the algorithm constructs two rectangular matrices S and T, with the objective of making use of the following algorithm of Coppersmith [16].

▶ **Theorem 7** (Coppersmith [16]). *Given an $N \times \lfloor N^{0.17}\rfloor$ matrix S and an $\lfloor N^{0.17}\rfloor \times N$ matrix T as input, the matrix product ST over the integers can be computed in $O(N^2 \log^2 N)$ arithmetic operations.*

Towards this end, let g be a positive integer whose value we will fix later. Introduce two set partitions of $\{1,2,\ldots,n\}$ with cells

$$I_1, I_2, \ldots, I_{\lceil n/g\rceil} \subseteq \{1,2,\ldots,n\} \qquad \text{and} \qquad J_1, J_2, \ldots, J_{\lceil n/g\rceil} \subseteq \{1,2,\ldots,n\}\,,$$

respectively, so that $|I_u| = g$ and $|I_v| = g$ for $u, v \in \{1,2,\ldots,\lfloor n/g\rfloor\}$. Indeed, we thus have $|I_u||J_v| \leq g^2$ for all $u, v \in \{1,2,\ldots,\lceil n/g\rceil\}$, so (23) applied to I_u and J_v modulo p^h recovers the number of pairs $(i,j) \in I_u \times J_v$ with $\langle a^{(i)}, b^{(j)}\rangle \equiv r \pmod p$, assuming that $g^2 \leq p^h - 1$, which will be justified by our eventual choice of g.

Now let $N = \lceil n/g\rceil$ and define the $N \times M$ and $M \times N$ matrices S and T by setting $S_{uk} = L_{I_u,k}$ and $T_{kv} = R_{I_v,k}$ for $u, v \in \{1,2,\ldots,\lceil n/g\rceil\}$ and $k \in \{1,2,\ldots,M\}$. Concretely, the algorithm computes S and T from the given input one entry at a time using the computed monomial list (24) and the formulas (22) for $I = I_u$ and $J = J_v$ for each $u, v \in \{1,2,\ldots,\lceil n/g\rceil\}$ and $k = 1,2,\ldots,M$. The algorithm then multiplies S and T to obtain the product matrix ST modulo p^h, where we assume that each entry of ST is reduced to $\{0,1,\ldots,p^h-1\}$. Finally, the algorithm outputs the sum $F_{pr} = \sum_{u=1}^{\lceil n/g\rceil}\sum_{v=1}^{\lceil n/g\rceil} (ST)_{uv}$.

Parameterizing the Algorithm. Let us now fix precise values for the parameters g, h, and p that were left open. First, to apply the algorithm in Theorem 7 to the matrices S and T, we need $M \leq N^{0.17} = \lceil n/g \rceil^{0.17}$. Subject to the assumption $g \leq n^{0.1}$ – to be justified later – it will be sufficient to show that $M \leq n^{0.15}$. We recall that $M \leq \sum_{j=0}^{4hp} \binom{2d}{j}$ and $d \leq \kappa \log n$. With foresight, let us set

$$\beta_\kappa = \frac{K}{\log \kappa}\,, \tag{25}$$

where $K > 0$ is a small constant that will be fixed later. In particular, since $\kappa \geq 4$, we have the upper bound

$$\beta_\kappa \log \frac{\kappa}{\beta_\kappa} = K - \frac{K \log K}{\kappa} + \frac{K \log \log \kappa}{\kappa} \leq \frac{5K}{4} \tag{26}$$

which we can make an arbitrarily small and positive by choosing a small enough K. Let us assume – to be justified later – that $p = o(\beta_\kappa \log n)$. Taking

$$h = \left\lfloor \beta_\kappa \frac{\log n}{p} \right\rfloor \tag{27}$$

we have, for all large enough n,

$$\begin{aligned} M &\leq \sum_{j=0}^{4hp} \binom{2d}{j} \leq 4hp \binom{2d}{4hp} \leq 4hp \left(\frac{2ed}{4hp}\right)^{4hp} \\ &\leq 4\left(\beta_\kappa \frac{\log n}{p} + 1\right) p \left(\frac{2e\kappa \log n}{4\big(\beta_\kappa \frac{\log n}{p} - 1\big)p}\right)^{4\left(\beta_\kappa \frac{\log n}{p} + 1\right)p} \\ &= 4\big(\beta_\kappa \log n + p\big) \left(\frac{2e\kappa \log n}{4\big(\beta_\kappa \log n - p\big)}\right)^{4(\beta_\kappa \log n + p)} \\ &\leq \big(5\beta_\kappa \log n\big) \left(\frac{2e\kappa}{3\beta_\kappa}\right)^{5\beta_\kappa \log n} \leq n^{0.15}\,, \end{aligned} \tag{28}$$

where the last inequality follows by (26) and choosing K small enough. Thus, Theorem 7 applies, subject to the assumptions $g \leq n^{0.1}$, $g^2 \leq p^h - 1$, and $p = o(\beta_\kappa \log n)$, which still need to be established. Before this, we digress to further preliminaries to enable reconstruction.

Preliminaries on Asymptotics of Primes. In what follows let us write p_j for the jth prime number with $j = 1, 2, \ldots$; that is, $p_1 = 2$, $p_2 = 3$, $p_3 = 5$, and so forth. Asymptotically, from the Prime Number Theorem we have $p_m \sim m \ln m$ (e.g. Rosser and Schoenfeld [21]), and the sum of the first m primes satisfies $\sum_{j=1}^{m} p_j \sim \frac{1}{2} m^2 \ln m$ (cf. Bach and Shallit [7]), where we write $f(m) \sim g(m)$ if $\lim_{m\to\infty} \frac{f(m)}{g(m)} = 1$.

When evaluated for the first m primes, the reconstruction parameter (3) thus satisfies

$$s_m = 1 + \sum_{j=1}^{m} (p_j - 1) \sim \frac{p_m^2}{2 \ln p_m}\,. \tag{29}$$

We are now ready to continue parameterization of the algorithm.

Further Parameterization of the Algorithm. Let m be a positive integer whose value will be fixed shortly. The algorithm will work with $p_1, p_2, \ldots, p_m$, the first m prime numbers. To reconstruct inner products of length-d zero-one vectors over the integers, we need $d+1 \leq s_m$, which for $d \leq \kappa \log n$ and (29) means

$$\frac{p_m^2}{2 \ln p_m} \sim \kappa \log n \,.$$

From Bertrand's postulate it thus follows that choosing the least m so that

$$2\sqrt{\kappa (\ln n) \ln \ln n} \leq p_m \leq 4\sqrt{\kappa (\ln n) \ln \ln n} \tag{30}$$

implies that we have $d+1 \leq s_m$ for all large enough n and thus reconstruction is feasible. The choice (30) also justifies our ealier assumption made in the context of (27) and (28) that $p_j = o(\beta_\kappa \log n)$ for all $j \in \{1, 2, \ldots, m\}$; indeed, from (14) and (25), we have

$$\beta_\kappa \log n = \frac{K \log n}{\log \kappa}$$

and thus from (14) and (30) we observe that

$$\frac{p_j}{\beta_\kappa \log n} \leq \frac{4\kappa^{1/2} (\log \kappa)(\ln n)^{1/2} (\ln \ln n)^{1/2}}{K \log n} = o(1) \,.$$

Let us next choose the parameter g. Using $p_j = o(\beta_\kappa \log n)$ again, we have

$$p_j^{h_j} = p_j^{\left\lfloor \beta_\kappa \frac{\log n}{p_j} \right\rfloor} \geq p_j^{\beta_\kappa \frac{\log n}{p_j} - 1} \geq p_j^{\beta_\kappa \frac{\log n}{2p_j}} = 2^{\beta_\kappa \frac{\log n}{2p_j} \log p_j} = n^{\beta_\kappa \frac{\log p_j}{2p_j}} \,.$$

Since $p_1 < p_2 < \cdots < p_m$, for $j \in \{1, 2, \ldots, m\}$ thus $p_j^{h_j} \geq n^{\beta_\kappa \frac{\log p_m}{2p_m}}$. It follows that choosing

$$g = \left\lfloor \sqrt{n^{\beta_\kappa \frac{\log p_m}{2p_m}} - 1} \right\rfloor \tag{31}$$

justifies our assumption $g^2 \leq p_j^{h_j} - 1$ for $j \in \{1, 2, \ldots, m\}$. The final assumption $g \leq n^{0.1}$ is justified by observing that $\frac{\log p_m}{2p_m}$ is a decreasing function of m and observing that $\beta_\kappa = o(1)$ by (14) and (25). The algorithm is now parameterized. Let us next analyse its running time.

Running Time Analysis. First, let us seek control on N as a function of n. From (30) and (31), we have

$$g \geq \sqrt{n^{\beta_\kappa \frac{2 \log 2 + \log \kappa + \log \ln n + \log \ln \ln n}{16 \sqrt{\kappa \ln n \ln \ln n}}} - 1} - 1 \,.$$

This together with (14) gives us the crude lower bound

$$g = \exp\left(\Omega\left(\beta_\kappa \sqrt{\frac{(\ln n) \ln \ln n}{\kappa}} \right) \right) .$$

We thus have

$$N^2 = \lceil n/g \rceil^2 = n^{2 - \Omega\left(\beta_\kappa \sqrt{\frac{\ln \ln n}{\kappa \ln n}}\right)} \,.$$

Recalling (28), we observe that the time to compute the M-monomial list (24) can be bounded by $n^{0.31}$ because the algorithm is careful to take multilinear reducts and thus at no stage of evaluating (15), (17), and (18) the number of monomials increases above $(n^{0.15})^2 = n^{0.30}$. Since $\log p_j^{h_j} = h_j \log p_j = \lfloor \beta_\kappa \frac{\log n}{p_j} \rfloor \log p_j = O(\log n)$, the arithmetic over the integers and modulo $p_j^{h_j}$ for each $j = 1, 2, \ldots, m$ runs in time polylogarithmic in n for each arithmetic operation executed by the algorithm. Because the algorithm in Theorem 7 runs in $O(N^2 \log^2 N)$ arithmetic operations, we observe that the polylogarithmic terms are subsumed by the asymptotic notation and the entire algorithm for computing F_{pr} for given $p \in \{p_1, p_2, \ldots, p_m\}$ and $r \in \{0, 1, \ldots, p-1\}$ runs in time

$$n^{2-\Omega\left(\beta_\kappa \sqrt{\frac{\log\log n}{\kappa \log n}}\right)} = n^{2-\Omega\left(\sqrt{\frac{\log\log n}{\kappa (\log \kappa)^2 \log n}}\right)}. \tag{32}$$

From (30) we observe that the required repeats for different p and r result in multiplicative polylogarithmic terms in n and are similarly subsumed to result in total running time of the form (32). This completes the proof of Theorem 1. ◀

4 A Faster Randomized Algorithm for #InnerProduct

This section sketches a proof for Theorem 2. We follow the algorithm outlined in Alman and Williams [6]. We note that by their Theorem 1.2, there are probabilistic polynomials over any field with error ϵ of degree $O(\sqrt{n \log(1/\epsilon)})$. In their Theorem 4.2, they have a probabilistic OR-construction that takes the disjunction of a random set of s^2 pairs of vector inner products as

$$q(x_1, y_1, x_2, y_2, \ldots, x_s, y_s) = 1 + \prod_{k=1}^{2} \left(1 + \sum_{(i,j) \in R_k} \left(1 + p(x_{i,1} + y_{i,1}, x_{i,2} + y_{j,2}, \ldots, x_{i,s} + y_{j,s})\right)\right),$$

where p is a probabilistic threshold polynomial over $\mathbb{F}_2$ of error $\epsilon = s^{-3}$, and $R_k \subseteq [s]^2$ for $k = 1, 2$ are sieve subsets drawn uniformly at random. This construction can be used to detect w.h.p. if there is a pair in the s^2-sized batch whose difference Hamming weight is less than the threshold. By repeated computations with new p's and R_k's, a majority vote for the batch can be chosen as the correct answer, again w.h.p. for all batches.

We implement the following change of q to get an #InnerProduct algorithm. We take p to be a probabilistic polynomial of error $\epsilon = s^{-3}$ for the symmetric function $[\![\sum_{i=1}^{n} z_i = t]\!]$, over a field of characteristic $> s^2$. We then construct q as

$$q(x_1, y_1, x_2, y_2, \ldots, x_s, y_s) = \sum_{(i,j) \in [s]^2} p(x_{i,1} y_{i,1}, x_{i,2} y_{j,2}, \ldots, x_{i,s} y_{j,s}). \tag{33}$$

Since the characteristic of the field is large enough, (33) is equal to the number of pairs in the s^2-sized batch that has inner product equal to t with probability at least $1 - s^2\epsilon \geq 1 - \frac{1}{s}$, a similar bound on the probability as in Theorem 4.2. Also, the degree of the polynomials is only a factor 2 larger. As with the original algorithm, if we repeat this enough times and take the majority in each batch, we get the correct number of pairs with t as inner product in all batches. By summing these final majority numbers over the integers, we obtain the output. We note that the parameters of the error and the degree has only changed by a constant, and hence that all calculations of the running time and the error bound of the original algorithm carries through also for our modification of the algorithm. This completes the proof sketch. ◀

5 A Lower Bound for #InnerProduct via Zero-One Permanents

This section proves Theorem 3; the proof of Theorem 4 is presented in Sect. 6.

Throughout this section we let M be an $n \times n$ matrix with entries $m_{ij} \in \{0,1\}$ for $i,j \in \{1,2,\ldots,n\}$. For convenience, let us write $[n] = \{1,2,\ldots,n\}$. Recalling Ryser's formula, we have

$$\operatorname{per} M = (-1)^n \sum_{S \subseteq [n]} (-1)^{|S|} \prod_{i \in [n]} \sum_{j \in S} m_{ij} \,. \tag{34}$$

First Reduction: Chinese Remaindering. Since it is immediate that $0 \leq \operatorname{per} M \leq n!$, it suffices to compute the permanent modulo small primes p and then assemble the result over the integers via the Chinese Remainder Theorem. Let us first state and prove a crude upper bound on the size of the primes needed. For a positive integer m, let us write $m\#$ for the product of all prime numbers at most m.

▶ **Lemma 8.** *For all sufficiently large n, we have $n! \leq (n \ln n)\#$.*

Proof. Recall that for a positive integer m we write write $m\#$ for the product of all prime numbers at most m. For $m \geq 563$, we have $\ln m\# > m\big(1 - \frac{1}{2 \ln m}\big)$; cf. Rosser and Schoenfeld [21]. For the factorial function, for $n \geq 1$, Robbins [20] shows that $n! = \sqrt{2\pi n}\big(\frac{n}{e}\big)^n e^{\alpha_n}$ with $\frac{1}{12n+1} < \alpha_n < \frac{1}{12n}$, which gives us the comparatively crude upper bound $\ln n! < \big(n + \frac{1}{2}\big) \ln n - n + 1$ for $n \geq 1$. We want $\ln n! < \ln m\#$. Accordingly, it suffices to have $m \geq 563$ and $\big(n + \frac{1}{2}\big) \ln n - n + 1 < m\big(1 - \frac{1}{2 \ln m}\big)$. It is immediate that $m \geq n \ln n$ suffices for $m \geq 563$, which completes the proof. ◀

Thus, it suffices to work with all primes p with $p \leq n \ln n$ in what follows.

A Reduction from Zero-One Permanent to #InnerProduct. This section starts our work towards Theorem 3 without yet parameterizing the reduction in detail. Let a prime $2 \leq p \leq n \ln n$ be given. We seek to compute $\operatorname{per} M$ modulo p. Fix a primitive root $g \in \{1,2,\ldots,p-1\}$ modulo p. For an integer a with $a \not\equiv 0 \pmod p$, let us write $\operatorname{dlog}_{p,g} a$ for the *discrete logarithm* of a relative to g modulo p. That is, $\operatorname{dlog}_{p,g} a$ is the unique integer in $\{0,1,\ldots,p-2\}$ that satisfies $g^{\operatorname{dlog}_{p,g} a} \equiv a \pmod p$. Working modulo p and collecting the outer sum in (34) by the sign $\sigma \in \{-1,1\}$ and the *nonzero* products by their discrete logarithm, we have

$$\operatorname{per} M \equiv (-1)^n \sum_{e=0}^{p-2} g^e \big(w_1^{(e)} - w_{-1}^{(e)}\big) \pmod p \,,$$

where

$$w_\sigma^{(e)} = \left| \left\{ S \subseteq [n] \,:\, (-1)^{|S|} = \sigma \text{ and } \operatorname{dlog}_{p,g} \prod_{i \in [n]} \sum_{j \in S} m_{ij} \equiv e \pmod{p-1} \right\} \right|$$

for $\sigma \in \{-1,1\}$ and $e \in \{0,1,\ldots,p-2\}$. Thus, to compute $\operatorname{per} M$ modulo p it suffices to compute the coefficients $w_\sigma^{(e)}$. Towards this end, suppose that $n \geq 4$ is even and let

$$L = \{1,2,\ldots,n/2\} \qquad \text{and} \qquad R = \{n/2, n/2+1, \ldots, n\} \,.$$

For $\sigma_L, \sigma_R \in \{1,-1\}$, let

$$w_{\sigma_L,\sigma_R}^{(e)} = \left| \left\{ S \subseteq [n] : (-1)^{|S \cap L|} = \sigma_L,\ (-1)^{|S \cap R|} = \sigma_R \text{ and } \operatorname{dlog}_{p,g} \prod_{i \in [n]} \sum_{j \in S} m_{ij} \equiv e \pmod{p-1} \right\} \right|$$

Clearly $w_\sigma^{(e)} = \sum_{\substack{\sigma_L,\sigma_R\in\{-1,1\}\\ \sigma_L\sigma_R=\sigma}} w_{\sigma_L,\sigma_R}^{(e)}$, so it suffices to focus on computing $w_{\sigma_L,\sigma_R}^{(e)}$ in what follows. Define the set families

$$\mathcal{L}_{\sigma_L} = \big\{A \subseteq L : (-1)^{|A|} = \sigma_L\big\} \qquad \text{and} \qquad \mathcal{R}_{\sigma_R} = \big\{B \subseteq R : (-1)^{|B|} = \sigma_R\big\}$$

with $|\mathcal{L}_{\sigma_L}| = |\mathcal{R}_{\sigma_R}| = 2^{n/2-1}$. Next we will define two families of length-d zero-one vectors whose pair counts by inner product will enable us to recover the coefficients $w_{\sigma_L,\sigma_R}^{(e)}$. The structure of the vectors will be slightly elaborate, so let us first define an index set D for indexing the $|D| = d$ dimensions. Let

$$D = \big\{(i,\ell,r,k) \in [n]\times\{0,1,\ldots,p-1\}\times\{0,1,\ldots,p-1\}\times[np] : \\ \ell + r \not\equiv 0 \pmod p \text{ implies } k \le \operatorname{dlog}_{p,g}\big(\ell+r\big)\big\}.$$

We have $d = n^2p^2 + np(p-1)(p-2)/2 < n^4(\ln n)^3$.

For $A \in \mathcal{L}_{\sigma_L}$ and $B \in \mathcal{R}_{\sigma_R}$, define the vectors $\lambda(A) \in \{0,1\}^D$ and $\rho(B) \in \{0,1\}^D$ for all $(i,\ell,r,k) \in D$ by the rules

$$\lambda(A)_{i\ell rk} = \begin{cases} 1 & \text{if } \ell \equiv \sum_{j\in A} m_{ij} \pmod p; \\ 0 & \text{otherwise;}\end{cases} \quad \text{and} \quad \rho(B)_{i\ell rk} = \begin{cases} 1 & \text{if } r \equiv \sum_{j\in B} m_{ij} \pmod p; \\ 0 & \text{otherwise.}\end{cases}$$

To study the inner product $\langle\lambda(A),\rho(B)\rangle$ it will be convenient to work with Iverson's bracket notation. Namely, for a logical proposition P, let

$$[\![P]\!] = \begin{cases} 1 & \text{if } P \text{ is true;} \\ 0 & \text{if } P \text{ is false.}\end{cases}$$

Over the integers, we now have

$$\begin{aligned}
\langle\lambda(A),\rho(B)\rangle &= \sum_{(i,\ell,r,k)\in D} \lambda(A)_{i\ell rk}\rho(B)_{i\ell rk} \\
&= \sum_{(i,\ell,r,k)\in D} [\![\ell \equiv \sum_{j\in A} m_{ij} \pmod p]\!][\![r \equiv \sum_{j\in B} m_{ij} \pmod p]\!] \\
&= \sum_{i\in[n]} \sum_{\substack{\ell,r=0\\ \ell+r\not\equiv 0 \pmod p}}^{p-1} [\![\ell \equiv \sum_{j\in A} m_{ij} \pmod p]\!][\![r \equiv \sum_{j\in B} m_{ij} \pmod p]\!] \operatorname{dlog}_{p,g}\big(\ell+r\big) \\
&\qquad + \sum_{i\in[n]} \sum_{\ell=0}^{p-1} [\![\ell \equiv \sum_{j\in A} m_{ij} \pmod p]\!][\![p-\ell \equiv \sum_{j\in B} m_{ij} \pmod p]\!] np \\
&= \begin{cases} \sum_{i\in[n]} \operatorname{dlog}_{p,g} \sum_{j\in A\cup B} m_{ij} & \text{if } \prod_{i\in[n]} \sum_{j\in A\cup B} m_{ij} \not\equiv 0 \pmod p; \\ \ge np & \text{if } \prod_{i\in[n]} \sum_{j\in A\cup B} m_{ij} \equiv 0 \pmod p.\end{cases}
\end{aligned} \tag{35}$$

In particular, letting $f_{\sigma_L,\sigma_R,t} = \big|\big\{(A,B) \in \mathcal{L}_{\sigma_L}\times\mathcal{R}_{\sigma_R} : \langle\lambda(A),\rho(B)\rangle = t\big\}\big|$, it follows immediately from (35) that we have $w_{\sigma_1,\sigma_2}^{(e)} = \sum_{t=0,\ t\equiv e \pmod{p-1}}^{n(p-2)} f_{\sigma_L,\sigma_R,t}$, which enables us to recover $\operatorname{per} M$ from the counts of pairs in $\mathcal{L}_{\sigma_L}\times\mathcal{R}_{\sigma_R}$ by inner product.

Completing the Proof of Theorem 3. Suppose we have an algorithm for #InnerProduct that runs in $N^{2-\Omega(1/\log c)}$ time when given an input of N vectors from $\{0,1\}^{c\log N}$. Take $N = 2^{n/2-1}$ and observe that $\log N = n/2 - 1$. The reduction from previous section has $d \le n^4(\ln n)^3$ and thus we can take $c = (n\ln n)^3$ and thus solve $n\times n$ zero-one permanent in time $N^{2-\Omega(1/\log c)} = 2^{n-\Omega(n/\log n)}$. This completes the proof of Theorem 3. ◀

6 A Lower Bound for #OV via Zero-One Permanents

A Reduction from Zero-One Permanent to #OV. This section starts our work towards Theorem 4 without yet parameterizing the reduction in detail. As in Sect. 5, it suffices to describe how to compute $\operatorname{per} M$ modulo a given prime p with $2 \leq p \leq n \ln n$.

Let g be a positive integer parameter, which we assume divides n. For $h \in [g]$, let $V_h = \{i \in [n] : (h-1)n/g + 1 \leq i \leq hn/g\}$ be a partition of the rows of M into g groups, each of size n/g. Again from Ryser's formula, we have

$$\operatorname{per} M = (-1)^n \sum_{S \subseteq [n]} (-1)^{|S|} \prod_{h \in [g]} \prod_{i \in V_h} \sum_{j \in S} m_{ij} \,.$$

Grouping by sign $\sigma \in \{-1, 1\}$ and per-group residues $r \in \{0, 1, \ldots, p-1\}^g$, we thus have

$$\operatorname{per} M \equiv (-1)^n \sum_{r \in \{0,1,\ldots,p-1\}^g} (t_{1,r} - t_{-1,r}) \prod_{h=1}^{g} r_h \pmod{p}\,, \tag{36}$$

where $t_{\sigma,r} = \left|\left\{S \subseteq [n] : (-1)^{|S|} = \sigma \text{ and } \prod_{i \in V_h} \sum_{j \in S} m_{ij} \equiv r_h \pmod{p} \text{ for each } h \in [g]\right\}\right|$. Observe that given all the counts $t_{\sigma,r}$, it takes $O(p^g g)$ operations modulo p to compute the permanent modulo p via (36), which is less than $2^n n$ when $g < n/\log p$. We continue to describe how to get the counts $t_{\sigma,r}$ via orthogonal-vector counting.

Assuming that $n \geq 4$ is even, introduce again the split

$$L = \{1, 2, \ldots, n/2\} \qquad \text{and} \qquad R = \{n/2, n/2+1, \ldots, n\}\,.$$

Let the residue vector $r \in \{0, 1, \ldots, p-1\}^g$ be fixed. For $\sigma_L, \sigma_R \in \{1, -1\}$, let

$$t_{\sigma_L,\sigma_R,r} = \Big|\Big\{S \subseteq [n] : (-1)^{|S\cap L|} = \sigma_L \;,\; (-1)^{|S\cap R|} = \sigma_R\,, \\ \text{and } \prod_{i \in V_h} \sum_{j \in S} m_{ij} \equiv r_h \pmod{p} \text{ for each } h \in [g]\Big\}\Big|\,.$$

Clearly $t_{\sigma,r} = \sum_{\substack{\sigma_L,\sigma_R \in \{-1,1\} \\ \sigma_L\sigma_R = \sigma}} t_{\sigma_L,\sigma_R,r}$, so it suffices to focus on computing $t_{\sigma_L,\sigma_R,r}$ in what follows. We again work with the set families

$$\mathcal{L}_{\sigma_L} = \left\{A \subseteq L : (-1)^{|A|} = \sigma_L\right\} \qquad \text{and} \qquad \mathcal{R}_{\sigma_R} = \left\{B \subseteq R : (-1)^{|B|} = \sigma_R\right\}.$$

Let $D = [g] \times \{0, 1, \ldots, p-1\}^{n/g}$. We have $d = |D| = gp^{n/g}$.

For $A \in \mathcal{L}_{\sigma_L}$ and $B \in \mathcal{R}_{\sigma_R}$, define the vectors $\lambda(A) \in \{0,1\}^D$ and $\rho(B) \in \{0,1\}^D$ for all $(h, u) \in D$ by the rules

$$\lambda(A)_{hu} = \begin{cases} 1 & \text{if we have } \sum_{j \in A} m_{ij} \equiv u_{i-(h-1)n/g} \pmod{p} \text{ for all } i \in V_h; \\ 0 & \text{otherwise} \end{cases}$$

and

$$\rho(B)_{hu} = \begin{cases} 0 & \text{if } \prod_{i \in V_h} \left(u_{i-(h-1)n/g} + \sum_{j \in B} m_{ij}\right) \equiv r_h \pmod{p}; \\ 1 & \text{otherwise.} \end{cases} \tag{37}$$

Over the integers, from (37) we now have

$$\begin{aligned}
\langle \lambda(A), \rho(B) \rangle &= \sum_{(h,u)\in D} \lambda(A)_{hu}\rho(B)_{hu} \\
&= \sum_{h\in[g]} \sum_{u\in\{0,1,\ldots,p-1\}^{n/g}} \prod_{i\in V_h} \left[\!\!\left[\sum_{j\in A} m_{ij} \equiv u_{i-(h-1)n/g} \pmod p \right]\!\!\right] \\
&\qquad\qquad \left[\!\!\left[\prod_{i\in V_h} \left(u_{i-(h-1)n/g} + \sum_{j\in B} m_{ij}\right) \not\equiv r_h \pmod p \right]\!\!\right] \\
&= \sum_{h\in[g]} \left[\!\!\left[\prod_{i\in V_h} \left(\sum_{j\in A} m_{ij} + \sum_{j\in B} m_{ij} \right) \not\equiv r_h \pmod p \right]\!\!\right] \\
&= \begin{cases} 0 & \text{if we have } \prod_{i\in V_h} \sum_{j\in A\cup B} m_{ij} \equiv r_h \pmod p \text{ for each } h\in[g]; \\ \geq 1 & \text{otherwise.} \end{cases} \qquad (38)
\end{aligned}$$

In particular, we have $t_{\sigma_L,\sigma_R,r} = \left|\left\{(A,B) \in \mathcal{L}_{\sigma_L} \times \mathcal{R}_{\sigma_R} : \langle \lambda(A), \rho(B)\rangle = 0\right\}\right|$, which enables us to recover $\operatorname{per} M$ from the counts of orthogonal pairs in $\mathcal{L}_{\sigma_L} \times \mathcal{R}_{\sigma_R}$.

Completing the Proof of Theorem 4. Suppose now that we have an algorithm for #OV that runs in $N^{2-\Omega(1/\log^{1-\epsilon} c)}$ time for some $0 < \epsilon < 1$ when given an input of N vectors from $\{0,1\}^{c\log N}$. Take $N = 2^{n/2-1}$ and observe that $\log N = n/2 - 1$.

Let $K > 1$ be a constant that will depend on ϵ and the constant hidden by the $\Omega(\cdot)$ notation in the running time of the #OV algorithm. Take $g = \lfloor K^{-1/\epsilon} n (\log p)^{1-2/\epsilon} \rfloor$ and recall that the prime p is in the range $2 \leq p \leq n \ln n$. To compute the parameters $t_{\sigma,r}$ using the reduction in the previous section, for each prime p we need $4p^g$ invocations of the #OV algorithm on an input of N vectors of dimension $d = gp^{n/g}$. Thus, for all large enough n, since $\frac{1}{2}K^{-1/\epsilon} n (\log p)^{1-2/\epsilon} \leq g$, we have $d = gp^{n/g} \leq n 2^{2K^{1/\epsilon}(\log p)^{2/\epsilon}}$. Since clearly $d = c\log N = c(n/2-1)$ and $2/\epsilon > 2$, for all large enough n, we have $\log c \leq 1 + 2K^{1/\epsilon}(\log p)^{2/\epsilon} \leq 3K^{1/\epsilon}(\log p)^{2/\epsilon}$, where the last inequality depends on choosing a large enough K so that the inequality is true for $p = 2$. Thus, $-(\log c)^{\epsilon-1} \leq -3^{\epsilon-1}K^{1-1/\epsilon}(\log p)^{2-2/\epsilon}$. One invocation of the #OV algorithm thus runs in $N^{2-\Omega(\log^{\epsilon-1} c)} = 2^{n-\Omega(n3^{\epsilon-1}K^{1-1/\epsilon}(\log p)^{2-2/\epsilon})}$ time. For each prime $2 \leq p \leq n \ln n$, we need $4p^g \leq 2^{2+K^{-1/\epsilon} n (\log p)^{2-2/\epsilon}}$ invocations of the #OV algorithm. Thus, the running time of all the invocations for the prime p is bounded by $4p^g N^{2-\Omega(\log^{\epsilon-1} c)} \leq 2^{n-\Omega(n3^{\epsilon-1}K^{1-1/\epsilon}(\log p)^{2-2/\epsilon})+2+K^{-1/\epsilon} n (\log p)^{2-2/\epsilon}}$. By choosing a large enough K to dominate the constant hidden by the $\Omega(\cdot)$ notation in the running time of the #OV algorithm, we thus have, for all large enough n,

$$\begin{aligned}
4p^g N^{2-\Omega(\log^{\epsilon-1} c)} &\leq 2^{n-\Omega(n3^{\epsilon-1}K^{-1/\epsilon}(\log p)^{2-2/\epsilon})} \\
&\leq 2^{n-\Omega(n3^{\epsilon-1}K^{-1/\epsilon}(\log n + \log\ln n)^{2-2/\epsilon})} \\
&\leq 2^{n-\Omega(n(\log n)^{2-2/\epsilon})}.
\end{aligned}$$

Since there are at most $n \ln n$ primes p to consider, the total running time to compute $\operatorname{per} M$ is bounded by $2^{n-\Omega(n/\log^{2/\epsilon-2} n)}$. This completes the proof of Theorem 4. ◀

7 Further Applications

Counting Satisfying Assignments to a Sym∘And circuit via #InnerProduct. We describe how to embed a Sym∘And circuit, i.e., a circuit of s And gates working on n Boolean inputs, connected by a top gate that is an arbitrary symmetric gate, in a #InnerProduct instance

of size $N = 2^{n/2}$ and $d = s$. Assuming n even, we divide the n inputs in two equal halves L and R. We let $\mathcal{A}$ have one vector u for each assignment to the inputs in L, with one coordinate in u for each AND gate, representing the truth value of that gate restricted to the inputs in L. Likewise, we let $\mathcal{B}$ have one vector v for each assignment to the inputs in R, with each coordinate set to the truth value of the represented gate restricted to the inputs in R. It is readily verified that $\langle u, v\rangle$ counts the number of AND gates that are satisfied by the assignment represented by (u, v). Hence, knowing the number of assignments that satisfy exactly t of the AND gates, for $t = 0, 1, \ldots, s$, which is what the solution to the #INNERPRODUCT gives us, we can count the total number of assignments that also satisfies the top symmetric gate.

Variations where the circuit instead is a SYM∘OR or a SYM∘XOR, are also possible.

Computing the Weight Enumerator Polynomial via #InnerProduct. A binary linear code of length n and rank k is a linear subspace C with dimension k of the vector space $\mathbb{F}_2^n$. The *weight enumerator polynomial* is $W(C; x, y) = \sum_{w=0}^{n} A_w x^w y^{n-w}$, where $A_w = |\{c \in C : \langle c, c\rangle = w\}|$ for $w = 0, 1, \ldots, n$ is the *weight distribution*; that is, A_w equals the number of codewords of C having exactly w ones.

We will reduce the computation of the weight distribution, and hence the weight enumerator polynomial, to $(k/2+1)^2$ instances of #INNERPRODUCT with $N \leq 2^{k/2}$ and $d = 2(n-k)$ when k is even.

Let the $k \times n$ matrix G be the generating matrix of the code; that is, the codewords of C are exactly the row-span of G. We can assume without loss of generality that the generator matrix has the standard form $G = [I_k|P]$, where I_k is the $k \times k$ identity matrix. For each $s_A = 0, 1, \cdots, k/2$ and $s_B = 0, 1, \ldots, k/2$, we make one instance of #INNERPRODUCT.

We let the set $\mathcal{A}$ have one vector u for each code c obtained as the linear combination of exactly s_A of the first $k/2$ rows. Each of the $n-k$ last coordinates in the code word c is described by a block of two coordinates in u. If $c_i = 0$ we encode this as 01 in u, and if $c_i = 1$ we encode this as 10 in u. We concatenate all $n-k$ encoded blocks to obtain u. Likewise, we let the set $\mathcal{B}$ have one vector v for each code c obtained as a linear combination of s_B of the last $k/2$ rows. Again, each of the $n-k$ last coordinates in the code word c is described by a block of two coordinates in v, but the encoding is opposite the one for $\mathcal{A}$: If $c_i = 0$ we encode this as 10 in v, and if $c_i = 1$ we encode this as 01 in v. We again concatenate all $n-k$ encoded blocks to obtain v. With this design, it is readily verified that for $(u, v) \in \mathcal{A} \times \mathcal{B}$, the inner product $\langle u, v\rangle$ is equal to the number of ones in the last $n-k$ coordinates in the code word obtained as the sum of the code word represented by u and the code word represented by v. Also, by design the number of ones in the first k coordinates equals $s_A + s_B$. Hence, by summing over all pairs that have the same inner product t, and aggregating over all s_A and s_B, we can compute the weight distribution.

References

1 Amir Abboud, Richard Ryan Williams, and Huacheng Yu. More applications of the polynomial method to algorithm design. In Piotr Indyk, editor, *Proceedings of the Twenty-Sixth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2015, San Diego, CA, USA, January 4-6, 2015*, pages 218–230. SIAM, 2015. doi:10.1137/1.9781611973730.17.

2 Josh Alman. An illuminating algorithm for the light bulb problem. In Jeremy T. Fineman and Michael Mitzenmacher, editors, *2nd Symposium on Simplicity in Algorithms, SOSA@SODA 2019, January 8-9, 2019 - San Diego, CA, USA*, volume 69 of *OASICS*, pages 2:1–2:11. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2019. doi:10.4230/OASIcs.SOSA.2019.2.

3 Josh Alman, Timothy M. Chan, and R. Ryan Williams. Polynomial representations of threshold functions and algorithmic applications. In Irit Dinur, editor, *IEEE 57th Annual Symposium on Foundations of Computer Science, FOCS 2016, 9-11 October 2016, Hyatt Regency, New Brunswick, New Jersey, USA*, pages 467–476. IEEE Computer Society, 2016. doi:10.1109/FOCS.2016.57.

4 Josh Alman, Timothy M. Chan, and R. Ryan Williams. Faster deterministic and Las Vegas algorithms for offline approximate nearest neighbors in high dimensions. In Shuchi Chawla, editor, *Proceedings of the 2020 ACM-SIAM Symposium on Discrete Algorithms, SODA 2020, Salt Lake City, UT, USA, January 5-8, 2020*, pages 637–649. SIAM, 2020. doi:10.1137/1.9781611975994.39.

5 Josh Alman and Lijie Chen. Efficient construction of rigid matrices using an NP oracle. In David Zuckerman, editor, *60th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2019, Baltimore, Maryland, USA, November 9-12, 2019*, pages 1034–1055. IEEE Computer Society, 2019. doi:10.1109/FOCS.2019.00067.

6 Josh Alman and Ryan Williams. Probabilistic polynomials and hamming nearest neighbors. In Venkatesan Guruswami, editor, *IEEE 56th Annual Symposium on Foundations of Computer Science, FOCS 2015, Berkeley, CA, USA, 17-20 October, 2015*, pages 136–150. IEEE Computer Society, 2015. doi:10.1109/FOCS.2015.18.

7 Eric Bach and Jeffrey Shallit. *Algorithmic Number Theory. Vol. 1.* Foundations of Computing Series. MIT Press, Cambridge, MA, 1996. Efficient algorithms.

8 Eric T. Bax and Joel Franklin. A permanent algorithm with $\exp[\Omega(N^{1/3}/2 \ln N)]$ expected speedup for 0-1 matrices. *Algorithmica*, 32(1):157–162, 2002. doi:10.1007/s00453-001-0072-0.

9 Richard Beigel and Jun Tarui. On ACC. *Computational Complexity*, 4:350–366, 1994. doi:10.1007/BF01263423.

10 Andreas Björklund. Below all subsets for some permutational counting problems. In Rasmus Pagh, editor, *15th Scandinavian Symposium and Workshops on Algorithm Theory, SWAT 2016, June 22-24, 2016, Reykjavik, Iceland*, volume 53 of *LIPIcs*, pages 17:1–17:11. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2016. doi:10.4230/LIPIcs.SWAT.2016.17.

11 Andreas Björklund, Thore Husfeldt, and Isak Lyckberg. Computing the permanent modulo a prime power. *Inf. Process. Lett.*, 125:20–25, 2017. doi:10.1016/j.ipl.2017.04.015.

12 Andreas Björklund, Petteri Kaski, and Ryan Williams. Generalized Kakeya sets for polynomial evaluation and faster computation of fermionants. *Algorithmica*, 81(10):4010–4028, 2019. doi:10.1007/s00453-018-0513-7.

13 Andreas Björklund and Ryan Williams. Computing permanents and counting hamiltonian cycles by listing dissimilar vectors. In Christel Baier, Ioannis Chatzigiannakis, Paola Flocchini, and Stefano Leonardi, editors, *46th International Colloquium on Automata, Languages, and Programming, ICALP 2019, July 9-12, 2019, Patras, Greece.*, volume 132 of *LIPIcs*, pages 25:1–25:14. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2019. doi:10.4230/LIPIcs.ICALP.2019.25.

14 Timothy M. Chan and Ryan Williams. Deterministic APSP, orthogonal vectors, and more: Quickly derandomizing Razborov-Smolensky. In Robert Krauthgamer, editor, *Proceedings of the Twenty-Seventh Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2016, Arlington, VA, USA, January 10-12, 2016*, pages 1246–1255. SIAM, 2016. doi:10.1137/1.9781611974331.ch87.

15 Lijie Chen and Ryan Williams. An equivalence class for orthogonal vectors. In Timothy M. Chan, editor, *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2019, San Diego, California, USA, January 6-9, 2019*, pages 21–40. SIAM, 2019. doi:10.1137/1.9781611975482.2.

16 Don Coppersmith. Rapid multiplication of rectangular matrices. *SIAM J. Comput.*, 11(3):467–471, 1982. doi:10.1137/0211037.

17 Russell Impagliazzo, Ramamohan Paturi, and Francis Zane. Which problems have strongly exponential complexity? *J. Comput. Syst. Sci.*, 63(4):512–530, 2001. doi:10.1006/jcss.2001.1774.

18 Matti Karppa, Petteri Kaski, and Jukka Kohonen. A faster subquadratic algorithm for finding outlier correlations. *ACM Trans. Algorithms*, 14(3):31:1–31:26, 2018. doi:10.1145/3174804.

19 Donald E. Knuth. *The Art of Computer Programming, Volume 2: Seminumerical Algorithms*. Addison-Wesley, Reading, MA, 1998.

20 Herbert Robbins. A remark on Stirling's formula. *The American Mathematical Monthly*, 62(1):26–29, 1955.

21 J. Barkley Rosser and Lowell Schoenfeld. Approximate formulas for some functions of prime numbers. *Ill. J. Math.*, 6:64–94, 1962.

22 Herbert John Ryser. *Combinatorial Mathematics*. The Carus Mathematical Monographs, No. 14. Published by The Mathematical Association of America; distributed by John Wiley and Sons, Inc., New York, 1963.

23 Gregory Valiant. Finding correlations in subquadratic time, with applications to learning parities and the closest pair problem. *J. ACM*, 62(2):13:1–13:45, 2015. doi:10.1145/2728167.

24 Leslie G. Valiant. The complexity of computing the permanent. *Theor. Comput. Sci.*, 8:189–201, 1979. doi:10.1016/0304-3975(79)90044-6.

25 Joachim von zur Gathen and Jürgen Gerhard. *Modern Computer Algebra*. Cambridge University Press, third edition, 2013. doi:10.1017/CBO9781139856065.

26 Ryan Williams. A new algorithm for optimal 2-constraint satisfaction and its implications. *Theor. Comput. Sci.*, 348(2-3):357–365, 2005. doi:10.1016/j.tcs.2005.09.023.

Learning Stochastic Decision Trees

Guy Blanc
Stanford University, CA, USA

Jane Lange
MIT, Cambridge, MA, USA

Li-Yang Tan
Stanford University, CA, USA

Abstract

We give a quasipolynomial-time algorithm for learning *stochastic decision trees* that is optimally resilient to adversarial noise. Given an η-corrupted set of uniform random samples labeled by a size-s stochastic decision tree, our algorithm runs in time $n^{O(\log(s/\varepsilon)/\varepsilon^2)}$ and returns a hypothesis with error within an additive $2\eta+\varepsilon$ of the Bayes optimal. An additive 2η is the information-theoretic minimum.

Previously no non-trivial algorithm with a guarantee of $O(\eta)+\varepsilon$ was known, even for weaker noise models. Our algorithm is furthermore proper, returning a hypothesis that is itself a decision tree; previously no such algorithm was known even in the noiseless setting.

2012 ACM Subject Classification Theory of computation → Boolean function learning

Keywords and phrases Learning theory, decision trees, proper learning algorithms, adversarial noise

Digital Object Identifier 10.4230/LIPIcs.ICALP.2021.30

Category Track A: Algorithms, Complexity and Games

Related Version *Full Version*: `https://arxiv.org/pdf/2105.03594.pdf`

1 Introduction

Decision trees are a touchstone class in learning theory. There is by now a rich and vast literature on the problem of learning decision trees, spanning three decades and studying it in a variety of models and from a variety of perspectives [10, 32, 5, 16, 7, 27, 4, 17, 23, 30, 20, 31, 15, 28, 26, 22, 19, 9, 2, 3, 1, 6].

We consider the problem of learning *stochastic decision trees*, a generalization of standard deterministic decision trees that allows for stochastic nodes. This generalization broadens the expressive power of decision trees, enabling them to represent not just deterministic functions but also stochastic functions. Figure 1 depicts a stochastic decision tree with two stochastic nodes, labeled "$", one that branches on the outcome of a Bernoulli(0.8) random variable, and the other on the outcome of a Bernoulli(0.3) random variable.

Many real-world learning scenarios are inherently stochastic in nature, and relatedly, much of current research in learning theory focuses on the "probabilistic concept" generalization [24] of the standard PAC model of learning deterministic concepts (e.g. see [14, 13, 11, 12] for an ongoing line of work on learning neural networks in the probabilistic concept model). As discussed in [24], probabilistic concepts can also be viewed as latent variable models, where the uncertainty concerning latent variables is modeled as apparent probabilistic behavior.

Stochastic decision trees are a simple and natural way to represent stochastic functions. Despite compelling theoretical and practical motivations, there has thus far been considerably less attention on the problem learning stochastic decision trees as compared to deterministic decision trees. Many basic questions remain open; for example:

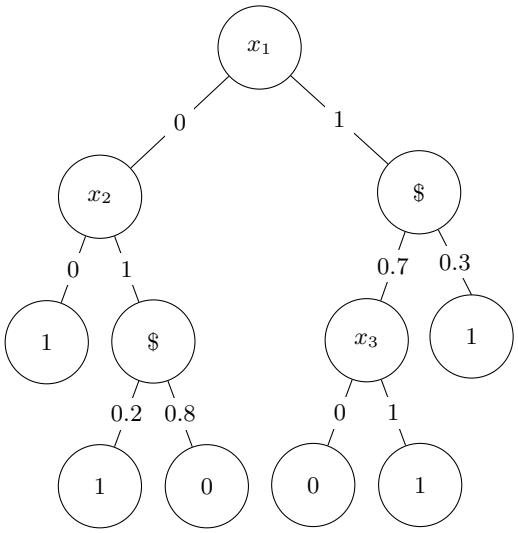


Figure 1 A stochastic decision tree with two stochastic nodes.

- Is there an algorithm for *properly* learning stochastic decision trees, one that returns a decision tree hypothesis?
- Is there an algorithm for learning stochastic decision trees that is resilient to *adversarial noise*?

These questions have been intensively studied in the case of deterministic decision trees, and the algorithms and techniques developed to answer them (e.g. [10, 21, 15]) have become foundational results in learning theory. A broad goal of our work is to help bring the state of our understanding of learning stochastic decision trees into closer alignment with that of deterministic decision trees.

1.1 Our results

We give new algorithms for learning stochastic decision trees under the uniform distribution. En route to our main result, we give the first algorithm for *properly* learning stochastic decision trees – our algorithm in fact returns a deterministic decision tree hypothesis:

▶ **Theorem 1** (Properly learning stochastic decision trees)**.** *There is an algorithm $\mathcal{A}$ with the following guarantee. For all $\varepsilon \in (0,1)$ and $s \in \mathbb{N}$, given access to labeled samples $(\boldsymbol{x}, \boldsymbol{T}(\boldsymbol{x}))$ where $\boldsymbol{T} : \{0,1\}^n \to \{0,1\}$ is a size-s stochastic decision tree and $\boldsymbol{x}$ is uniform random, $\mathcal{A}$ runs in $n^{O(\log(s/\varepsilon)/\varepsilon^2)}$ time and with high probability outputs a deterministic decision tree h such that $\Pr[h(\boldsymbol{x}) \neq \boldsymbol{T}(\boldsymbol{x})] \leq \mathrm{opt} + \varepsilon$, where opt denotes the Bayes optimal error for $\boldsymbol{T}$.*

Theorem 1 is a special case of our main result, which gives a generalization of the algorithm $\mathcal{A}$ of Theorem 1 that is optimally resilient to adversarial noise.

▶ **Definition 2** (η-corrupted samples; "nasty noise" [8])**.** *Let $\boldsymbol{f} : \{0,1\}^n \to \{0,1\}$ be a stochastic function. We say that $\mathcal{S}$ is an η-corrupted set of uniform random samples labeled by $\boldsymbol{f}$ if it is formed in the following fashion: draw a set of labeled samples $(\boldsymbol{x}, \boldsymbol{f}(\boldsymbol{x}))$ where $\boldsymbol{x}$ is uniform random, and modify any η fraction to form $\mathcal{S}$.*

We allow for corruptions of both the example (i.e. changing $\boldsymbol{x}$ to a different $\boldsymbol{x}'$) and its label (i.e. flipping $\boldsymbol{f}(\boldsymbol{x})$), and note that the adversarial choice of which η fraction of samples to corrupt can be adaptive, depending arbitrarily on the original uncorrupted set of samples. This is regarded as the most challenging noise model for classification problems; weaker noise models include random classification noise, Massart noise, and agnostic noise.

Our main result is as follows:

▶ **Theorem 3** (Our main result: Properly learning stochastic decision trees in the presence of adversarial noise)**.** *There is an algorithm $\mathcal{A}$ with the following guarantee. For all $\varepsilon, \eta \in (0,1)$ and $s \in \mathbb{N}$, given access to a sufficiently large η-corrupted set $\mathcal{S}$ of uniform random samples labeled by a size-s stochastic decision tree $\boldsymbol{T} : \{0,1\}^n \to \{0,1\}$, $\mathcal{A}$ runs in $n^{O(\log(s/\varepsilon)/\varepsilon^2)}$ time and with high probability outputs a decision tree hypothesis h such that $\Pr[h(\boldsymbol{x}) \neq \boldsymbol{T}(\boldsymbol{x})] \leq \mathrm{opt} + 2\eta + \varepsilon$, where opt denotes the Bayes optimal error for $\boldsymbol{T}$.*

An error of $\mathrm{opt} + 2\eta$ is the information-theoretic minimum (see e.g. [8]). Prior to our work there were (improper) algorithms that achieved either $\mathrm{opt} + O(\sqrt{\eta}) + \varepsilon$ or $2\mathrm{opt} + 2\eta + \varepsilon$, the low-degree algorithm of [29] and the L_1 polynomial regression algorithm of [21] respectively, but not the information-theoretically optimal $\mathrm{opt} + 2\eta + \varepsilon$. This was the case even for weaker noise models such as label-only noise (i.e. agnostic noise [18, 25]). In fact, the low-degree and L_1 polynomial regression algorithms are, in general, only known to be resilient to noise in the labels.

As our final contribution, we show that when applied in the context of decision tree learning, these algorithms are in fact resilient to noise in both the examples and their labels:

▶ **Theorem 4** (Noise-tolerant properties of the low-degree algorithm and L_1 polynomial regression)**.** *For all $\varepsilon, \eta \in (0,1)$ and $s \in \mathbb{N}$, given access to a sufficiently large η-corrupted set $\mathcal{S}$ of uniform random samples labeled by a size-s stochastic decision tree $\boldsymbol{T} : \{0,1\}^n \to \{0,1\}$,*

- *the low-degree algorithm runs in time $n^{O(\log(s/\varepsilon))}$ and with high probability outputs a hypothesis h satisfying $\Pr[h(\boldsymbol{x}) \neq \boldsymbol{T}(\boldsymbol{x})] \leq \mathrm{opt} + O(\sqrt{\eta}) + \varepsilon$.*
- *the L_1 polynomial regression algorithm runs in time $n^{O(\log(s/\varepsilon))}$ and with high probability outputs a stochastic hypothesis $\boldsymbol{h}$ satisfying $\Pr[\boldsymbol{h}(\boldsymbol{x}) \neq \boldsymbol{T}(\boldsymbol{x})] \leq 2\mathrm{opt} + 2\eta + \varepsilon$.*

1.1.1 Summary and comparison with existing algorithms

The low-degree algorithm of Linial, Mansour, and Nisan [29] and a recent algorithm of Chen and Moitra [9] for learning mixtures of subcubes can both be used to learn stochastic decision trees as a special case of their main results. The algorithm of [29] runs in time $n^{O(\log(s/\varepsilon))}$, whereas the algorithm of [9] runs in time $O_s(1) \cdot n^{O(\log s)} \cdot \mathrm{poly}(1/\varepsilon)$. However, neither of these algorithms returns a decision tree hypothesis, and hence both are improper when applied in this context. The classic algorithm of Ehrenfeucht and Haussler [10, 5] properly learns deterministic decision trees in time $n^{O(\log s)} \cdot \mathrm{poly}(1/\varepsilon)$. However, being an Occam algorithm, its analysis seems fundamentally unable to accommodate stochasticity of the target concept.

Table 1 summarizes our contributions and places them in the context of prior work.

1.2 Our techniques

Our approach to Theorems 1 and 3 is simple and has two main conceptual parts: a structural lemma concerning stochastic decision trees and a noise-tolerant algorithm for learning a special type of stochastic decision tree.

Table 1 Performance guarantees of our algorithm and existing algorithms for learning stochastic decision trees in the presence of adversarial noise. Among these algorithms, ours is the only one that returns a decision tree hypothesis. Prior to our work, the error guarantees for the low-degree algorithm and L_1 polynomial regression were only known for label noise; we show in the context of decision tree learning, these guarantees can be strengthened to allow for noise in both the examples and labels.

Reference	Technique	Running time	Error guarantee
[29]	Low-degree algorithm	$n^{O(\log(s/\varepsilon))}$	$\mathrm{opt} + O(\sqrt{\eta}) + \varepsilon$ (**This work**)
[21]	L_1 polynomial regression	$n^{O(\log(s/\varepsilon))}$	$2\mathrm{opt} + 2\eta + \varepsilon$ (**This work**)
[9]	Learning mixtures of subcubes	$O_s(1) \cdot n^{O(\log s)} \cdot \mathrm{poly}(\frac{1}{\varepsilon})$	$\mathrm{opt} + \varepsilon$ Noiseless setting ($\eta = 0$)
This work	Approximation by stochastic-leaf DTs; Noise-tolerant learning of stochastic-leaf DTs	$n^{O(\log(s/\varepsilon)/\varepsilon^2)}$	$\mathrm{opt} + 2\eta + \varepsilon$

- *Structural lemma:* We show that every size-s stochastic decision tree can be ε-approximated by a "stochastic-leaf decision tree" of size $s^{O(1/\varepsilon^2)}$. A stochastic-leaf decision tree is a very specific type of stochastic decision tree, one whose stochastic nodes only occur at its leaves.

This lemma reduces the task of learning stochastic decision trees to that of learning stochastic-leaf decision trees, with a catch: due to the approximation error incurred, the algorithm for learning stochastic-leaf decision trees has to be *noise-tolerant*.

- *Noise-tolerant learning stochastic-leaf decision trees:* Mehta and Raghavan [30] gave an algorithm for properly learning deterministic decision trees in the noiseless setting. We show that their algorithm can be generalized to handle stochastic-leaf decision trees, and furthermore, we show that our generalization is optimally resilient to adversarial noise. This stands in contrast to the algorithm of Ehrenfeucht and Haussler [10], which as mentioned above seems fundamentally unable to accommodate either stochasticity or noise.

We are hopeful that each of these two parts will see further utility in problems involving stochastic decision trees, beyond the learning-theoretic setting that is the focus of this work.

As for Theorem 4, the low-degree algorithm and L_1 polynomial regression are versatile and powerful "meta-algorithms" in learning, but they are not generally known to handle the challenging nasty noise. Our key observation here is that the mean functions of stochastic decision trees are well-approximated by low-degree polynomials *with bounded outputs*. We then show that when run on such polynomials, the low-degree algorithm and L_1 polynomial regression are in fact resilient to nasty noise. Given the broad applicability of both algorithms, we are similarly hopeful that this fact will be of independent interest beyond decision trees.

1.3 Preliminaries

Let $\boldsymbol{f} : \{0,1\}^n \to \{0,1\}$ be a stochastic function. We associate $\boldsymbol{f}$ with its *mean function* $\mu_{\boldsymbol{f}} : \{0,1\}^n \to [0,1]$, $\mu_{\boldsymbol{f}}(x) := \Pr_{\boldsymbol{f}}[\boldsymbol{f}(x) = 1]$. The *Bayes optimal classifier* for $\boldsymbol{f}$ is the (deterministic) function $x \mapsto \mathrm{round}(\mu_{\boldsymbol{f}}(x))$, where $\mathrm{round}(t) := \mathbb{1}[t \geq \frac{1}{2}]$. Given two stochastic functions $\boldsymbol{f}, \boldsymbol{h} : \{0,1\}^n \to \{0,1\}$, we define

$$\mathrm{error}_{\boldsymbol{f}}(\boldsymbol{h}) \coloneqq \mathop{\mathbb{E}}_{\boldsymbol{x}}\left[\Pr_{\boldsymbol{f},\boldsymbol{h}}[\boldsymbol{f}(\boldsymbol{x}) \neq \boldsymbol{h}(\boldsymbol{x})]\right],$$

where here and throughout this paper, $\boldsymbol{x}$ denotes a uniform random input from $\{0,1\}^n$. We define $\mathrm{opt}_{\boldsymbol{f}} \coloneqq \mathrm{error}_{\boldsymbol{f}}(\mathrm{round}(\mu_{\boldsymbol{f}}))$, and when $\boldsymbol{f}$ is clear from context, we simply write opt.

▶ **Fact 5** (Bayes optimal classifier minimizes classification error). *For all stochastic functions $\boldsymbol{f}, \boldsymbol{h} : \{0,1\}^n \to \{0,1\}$, we have $\mathrm{error}_{\boldsymbol{f}}(\boldsymbol{h}) \geq \mathrm{opt}_{\boldsymbol{f}}$.*

▶ **Fact 6** (L_1-error and Bayes optimality). *Let $\boldsymbol{f} : \{0,1\}^n \to \{0,1\}$ be a stochastic function. For any $h : \{0,1\}^n \to [0,1]$,*

$$\Pr[\mathrm{round}(h(\boldsymbol{x})) \neq \boldsymbol{f}(\boldsymbol{x})] \leq \mathrm{opt}_{\boldsymbol{f}} + 2\,\mathbb{E}[|\mu_{\boldsymbol{f}}(\boldsymbol{x}) - h(\boldsymbol{x})|].$$

Fact 6 states that if we have a function close to $\mu_{\boldsymbol{f}}$, we can convert it to a classifier with error close to $\mathrm{opt}_{\boldsymbol{f}}$.

Proof. We need to upper bound $\mathrm{error}_{\boldsymbol{f}}(\mathrm{round} \circ h) - \mathrm{opt}_{\boldsymbol{f}}$ at $2\,\mathbb{E}[|\mu_{\boldsymbol{f}}(\boldsymbol{x}) - h(\boldsymbol{x})|]$. We rewrite that quantity as

$$\begin{aligned}
\mathrm{error}_{\boldsymbol{f}}(\mathrm{round} \circ h) - \mathrm{opt}_{\boldsymbol{f}} &= \Pr_{\boldsymbol{x}\sim\{0,1\}^n}[\boldsymbol{f}(\boldsymbol{x}) \neq \mathrm{round}(h(\boldsymbol{x}))] - \Pr_{\boldsymbol{x}\sim\{0,1\}^n}[\boldsymbol{f}(x) \neq \mathrm{round}(\mu_{\boldsymbol{f}}(\boldsymbol{x}))] \\
&= \mathop{\mathbb{E}}_{\boldsymbol{x}\sim\{0,1\}^n}[|\,\mu_{\boldsymbol{f}}(\boldsymbol{x}) - \mathrm{round}(h(\boldsymbol{x}))\,| - |\,\mu_{\boldsymbol{f}}(\boldsymbol{x}) - \mathrm{round}(\mu_{\boldsymbol{f}}(\boldsymbol{x}))\,|] \\
&= \mathop{\mathbb{E}}_{\boldsymbol{x}\sim\{0,1\}^n}\left[\mathbb{1}(\mathrm{round}(h(\boldsymbol{x}))) \neq \mathrm{round}(\mu_{\boldsymbol{f}}(\boldsymbol{x}))) \cdot 2 \cdot \left|\,\mu_{\boldsymbol{f}}(\boldsymbol{x}) - \tfrac{1}{2}\,\right|\right]
\end{aligned}$$

It is only possible that $\mathrm{round}(h(x)) \neq \mathrm{round}(\mu_{\boldsymbol{f}}(x))$ if $|\,f(x) - \mu_{\boldsymbol{f}}(x)\,| \geq |\,\mu_{\boldsymbol{f}}(x) - \frac{1}{2}\,|$. Therefore,

$$\begin{aligned}
\mathrm{error}_{\boldsymbol{f}}(\mathrm{round} \circ h) - \mathrm{opt}_{\boldsymbol{f}} &\leq \mathop{\mathbb{E}}_{\boldsymbol{x}\sim\{0,1\}^n}\left[\mathbb{1}(|\,f(x) - \mu_{\boldsymbol{f}}(x)\,| \geq |\,\mu_{\boldsymbol{f}}(x) - \frac{1}{2}\,|) \cdot 2 \cdot \left|\,\mu_{\boldsymbol{f}}(\boldsymbol{x}) - \tfrac{1}{2}\,\right|\right] \\
&\leq 2 \mathop{\mathbb{E}}_{\boldsymbol{x}\sim\{0,1\}^n}[|\,f(x) - \mu_{\boldsymbol{f}}(x)\,|]\,. \qquad ◀
\end{aligned}$$

2 Approximating stochastic DTs with stochastic-leaf DTs

▶ **Definition 7** (Stochastic-leaf DT). *A stochastic-leaf DT is a stochastic DT for which all stochastic nodes have only leaves as their children.*

▶ **Lemma 8** (Approximating stochastic DTs with stochastic-leaf DTs). *Let $\boldsymbol{T}$ be a size-s stochastic DT. For every $\varepsilon \in (0, \frac{1}{2})$, there is a size-$S$* stochastic-leaf *DT $\overline{\boldsymbol{T}}$ such that $S \leq s^{O(1/\varepsilon^2)}$ and $\mathbb{E}_{\boldsymbol{x}}[|\mu_{\boldsymbol{T}}(\boldsymbol{x}) - \mu_{\overline{\boldsymbol{T}}}(\boldsymbol{x})|] \leq \varepsilon$.*

Proof. Let m denote the number of stochastic transitions in $\boldsymbol{T}$. For a fixed $r \in \{0,1\}^m$, let $\boldsymbol{T}(x, r)$ be the value of $\boldsymbol{T}$ evaluated on x with stochastic transitions determined by r. Suppose we pick random strings $\boldsymbol{r}_1, \ldots, \boldsymbol{r}_c \sim \{0,1\}^m$ independently and uniformly at random. For each $x \in \{0,1\}^n$, consider the following random variable:

$$\mathbf{est}(x) \coloneqq \mathop{\mathbb{E}}_{\boldsymbol{i}\in[c]}[\boldsymbol{T}(x, \boldsymbol{r_i})].$$

Note that

$$\begin{aligned}
\mathbb{E}_{\boldsymbol{r}_1,\ldots,\boldsymbol{r}_c\in\{0,1\}^m}[\mathbf{est}(x)] = \mu_{\boldsymbol{T}}(x) &= \mathop{\mathbb{E}}_{\boldsymbol{r}\sim\{0,1\}^m}[\boldsymbol{T}(x, \boldsymbol{r})] \\
\mathrm{Var}[\mathbf{est}(x)] &= \tfrac{1}{c} \cdot \mathbf{Var}_{\boldsymbol{r}\sim\{0,1\}^m}[\boldsymbol{T}(x, \boldsymbol{r})],
\end{aligned}$$

where in both cases above, $\boldsymbol{r} \sim \{0,1\}^m$ on the RHS denotes $\boldsymbol{r}$ chosen uniformly at random from $\{0,1\}^m$. Since $\boldsymbol{T}$ is $\{0,1\}$-valued, it has variance at most $\frac{1}{4}$. Hence, the variance of $\mathbf{est}(x)$ is at most $\frac{1}{4c}$. If we take $c = 1/\varepsilon^2$, the following holds for any $x \in \{0,1\}^n$:

$$\mathop{\mathbb{E}}_{\boldsymbol{r}_1,\ldots,\boldsymbol{r}_c \sim \{0,1\}^m} \left[\left(\mathbf{est}(x) - \mu_{\boldsymbol{T}}(x) \right)^2 \right] \leq \frac{\varepsilon^2}{4}, \quad \text{and therefore} \quad \mathop{\mathbb{E}}_{\boldsymbol{r}_1,\ldots,\boldsymbol{r}_c \sim \{0,1\}^m} [|\mathbf{est}(x) - \mu_{\boldsymbol{T}}(x)|] \leq \frac{\varepsilon}{2}.$$

Averaging over $\boldsymbol{x} \sim \{0,1\}^n$ and swapping expectations, we get:

$$\mathop{\mathbb{E}}_{\boldsymbol{r}_1,\ldots,\boldsymbol{r}_c \sim \{0,1\}^m} \left[\mathop{\mathbb{E}}_{\boldsymbol{x} \sim \{0,1\}^n} \left[|\mathbf{est}(x) - \mu_{\boldsymbol{T}}(x)| \right] \right] \leq \frac{\varepsilon}{2}.$$

Therefore, there must exist outcomes $r_1^\star, \ldots, r_c^\star \in \{0,1\}^m$ of $\boldsymbol{r}_1, \ldots, \boldsymbol{r}_c$ such that

$$\mathop{\mathbb{E}}_{\boldsymbol{x} \sim \{0,1\}^n} \left[\left| \mathop{\mathbb{E}}_{\boldsymbol{i} \in [c]} [\boldsymbol{T}(\boldsymbol{x}, r_{\boldsymbol{i}^\star})] - \mu_{\boldsymbol{T}}(\boldsymbol{x}) \right| \right] \leq \frac{\varepsilon}{2}. \tag{1}$$

For each $i \in [c]$, we define a size-s DT by fixing the stochastic nodes of $\boldsymbol{T}$ according to $r_i^\star \in \{0,1\}^m$. We define our stochastic-leaf DT $\overline{\boldsymbol{T}}$ by stacking these c many size-s DTs on top of one another: for each $i < n$, we replace each leaf of the i_{th} DT with a copy of the $(i+1)_{th}$ DT. Then for each leaf ℓ of this stacked tree, let x_ℓ be an input that is consistent with the root-to-ℓ path in $\overline{\boldsymbol{T}}$. We replace ℓ with a stochastic node which transitions to a 1-leaf with probability $p_\ell := \mathbb{E}_{\boldsymbol{i} \in [c]}[\boldsymbol{T}(x_\ell, r_{\boldsymbol{i}}^\star)]$, and to a 0-leaf with probability $1 - p_\ell$. Note that for each $i \in [c]$, the tree $\boldsymbol{T}(\cdot, r_i^\star)$ gives the same classification for all inputs reaching leaf ℓ of $\overline{\boldsymbol{T}}$, so p_ℓ does not depend on the choice of x_ℓ.

$\overline{\boldsymbol{T}}$ is a stochastic-leaf DT that computes $x \mapsto \mathbb{E}_{\boldsymbol{i} \in [c]}[\boldsymbol{T}(\boldsymbol{x}, r_{\boldsymbol{i}}^\star)]$, which by Equation (1), has sufficiently small error. Since this DT has size $s^c = s^{O(1/\varepsilon^2)}$, the proof of Lemma 8 is complete. ◀

3 A simple backtracking algorithm for finding the optimal small-depth tree

The algorithmic core of Theorems 1 and 3 is a recursive backtracking procedure FIND shown in Algorithm 1, which takes a labeled set of samples X and finds a depth-d decision tree that achieves minimal classification error. This algorithm is inspired by and simplifies the FIND algorithm given by Mehta and Raghavan [30] for building a minimum-error decision tree from any "sat-countable representation" of a function.

▶ **Lemma 9** (Correctness of FIND). *Consider any sample set X of labeled examples (x, y) and depth budget d. The algorithm FIND(X, d) (see Algorithm 1) returns a depth-d DT $T^\star$ that minimizes $\Pr_{(\boldsymbol{x},\boldsymbol{y}) \sim X}[T^\star(\boldsymbol{x}) \neq \boldsymbol{y}]$ among all depth-d DTs.*

Proof. We proceed by induction on d. If $d = 0$, then FIND returns at Step 1 and is clearly correct. For the inductive step, suppose that $d \geq 1$. For any $i \in [n]$, we first claim that the tree T_i defined in Step 2 is a depth d DT that minimizes classification error with respect to X among those that query x_i at the root. Let $(T_i)_{\text{left}}$ and $(T_i)_{\text{right}}$ be its left and right subtrees respectively. By the inductive hypothesis, the left and right subtrees $(T_i)_{\text{left}}$ and $(T_i)_{\text{right}}$ are depth $d-1$ DTs that minimize error with respect to $X_{x_i=0}$ and $X_{x_i=1}$ respectively. Hence, T_i is a depth d DT that achieves minimal error with respect to X among those that query x_i at the root.

Since FIND returns the $T_{i^\star}$ that minimizes $\Pr_{(\boldsymbol{x},\boldsymbol{y}) \sim X}[T_i(\boldsymbol{x}) \neq \boldsymbol{y}]$ among all $i \in [n]$ in Step 3, and each T_i is a minimal-error depth-d DT among those that query x_i at the root, we conclude that FIND returns a tree of minimal error with respect to X. ◀

Algorithm 1 A recursive backtracking algorithm for finding a depth-d DT of minimal classification error.

FIND(X, d):

Input: Set X of labeled examples (x, y) and depth budget d.

Output: A depth-d DT $T^\star$ that minimizes $\Pr_{(\boldsymbol{x},\boldsymbol{y})\sim X}[T^\star(\boldsymbol{x}) \neq \boldsymbol{y}]$ among all depth-d DTs.

1. If $d = 0$, return the constant $c \in \{0, 1\}$ that minimizes $\Pr_{(\boldsymbol{x},\boldsymbol{y})\sim X}[c \neq \boldsymbol{y}]$.
2. For every $i \in [n]$, let T_i be the DT defined as follows:
   - T_i queries x_i at the root;
   - Has FIND$(X_{x_i=0}, d-1)$ as its left subtree;
   - Has FIND$(X_{x_i=1}, d-1)$ as its right subtree.

   Here $X_{x_i=b}$ denotes the subset of X containing only examples where x_i is set to b.
3. Return the tree $T_{i^\star}$ that minimizes $\Pr_{(\boldsymbol{x},\boldsymbol{y})\in X}[T_i(\boldsymbol{x}) \neq \boldsymbol{y}]$ among all $i \in [n]$.

▶ **Lemma 10** (Efficiency of FIND). *Consider any sample set X of labeled examples and depth budget d. The algorithm FIND(X, d) (see Algorithm 1) takes time $n^{O(d)} \cdot O(|X|)$.*

Proof. Let $T(d)$ denote the running time of FIND when run with depth budget d. If $d = 0$ then the algorithm only executes Step 1, which can be done in $O(|X|)$ time by computing round$(\mathbb{E}_{(\boldsymbol{x},\boldsymbol{y})\sim X}[\boldsymbol{y}])$.

Next we consider the case of $d \geq 1$. In step 2, FIND recurses $2n$ times, each with d decremented by one. Each time it also partitions X into $X_{x_i=0}$ and $X_{x_i=1}$. All of these recursive calls and partitioning takes total time $2n \cdot T(d-1) + n|X|$. In step 3, FIND must compute $\Pr_{(\boldsymbol{x},\boldsymbol{y})\sim X}[T_i(\boldsymbol{x}) \neq \boldsymbol{y}]$ for up to n different coordinates i, where each T_i has depth at most d. This takes time $n \cdot d \cdot |X|$. We therefore have the recurrence relation:

$$T(d) \leq 2n \cdot T(d-1) + O(nd|X|).$$

Solving this recurrence relation gives us the bound $T(d) \leq (2n)^d \cdot O(nd|X|)$, which is $\leq n^{O(d)} \cdot O(|X|)$ as desired. ◀

4 Learning stochastic DTs: proofs of Theorems 1 and 3

4.1 Proof of Theorem 1

We recall Theorem 1, this time including the confidence parameter δ.

▶ **Theorem 1** (Properly learning stochastic decision trees). *There is an algorithm $\mathcal{A}$ with the following guarantee. For all $\varepsilon \in (0, 1)$ and $s \in \mathbb{N}$, given access to labeled samples $(\boldsymbol{x}, \boldsymbol{T}(\boldsymbol{x}))$ where $\boldsymbol{T} : \{0,1\}^n \to \{0,1\}$ is a size-s stochastic decision tree and $\boldsymbol{x}$ is uniform random, $\mathcal{A}$ runs in $n^{O(\log(s/\varepsilon)/\varepsilon^2)} \cdot \mathrm{poly}(\log(1/\delta))$ time and with probability $1 - \delta$ outputs a deterministic decision tree h such that $\Pr[h(\boldsymbol{x}) \neq \boldsymbol{T}(\boldsymbol{x})] \leq \mathrm{opt} + \varepsilon$, where opt denotes the Bayes optimal error for $\boldsymbol{T}$.*

Let $\boldsymbol{T}$ be a size-s stochastic decision tree. By Lemma 8, there is a stochastic-leaf decision tree $\overline{\boldsymbol{T}}$ of size $S \leq s^{O(1/\varepsilon^2)}$ such that $\mathbb{E}_{\boldsymbol{x}}[|\mu_{\boldsymbol{T}}(\boldsymbol{x}) - \mu_{\overline{\boldsymbol{T}}}(\boldsymbol{x})|] \leq \varepsilon$. Consider the Bayes optimal classifier $x \mapsto \mathrm{round}(\mu_{\overline{\boldsymbol{T}}}(x))$ for $\overline{\boldsymbol{T}}$. Since $\overline{\boldsymbol{T}}$ is a *stochastic-leaf* decision tree, we have that this function is computed by a size-S (deterministic) decision tree $T^\star$: to obtain $T^\star$ from $\overline{\boldsymbol{T}}$,

simply replace every stochastic node in $\overline{\boldsymbol{T}}$, all of which occur at the leaves of $\overline{\boldsymbol{T}}$, with a 1-leaf if it branches on Bernoulli(p) where $p \geq \frac{1}{2}$, and a 0-leaf otherwise. Applying Fact 6, we get that

$$\Pr_{\boldsymbol{x},\boldsymbol{T}}[T^\star(\boldsymbol{x}) \neq \boldsymbol{T}(\boldsymbol{x})] \leq \mathrm{opt}_{\boldsymbol{T}} + 2\varepsilon.$$

Next, consider the decision tree $T^\star_{\mathrm{trunc}}$ obtained by truncating $T^\star$ to depth $\log(S/\varepsilon)$ (and replacing all truncated branches with a leaf with an arbitrary value, say a 1-leaf). $T^\star_{\mathrm{trunc}}$ and $T^\star$ can only differ on inputs that reach a leaf in $T^\star$ of depth at least $\log(S/\varepsilon)$, and there are at most S such leaves. Therefore,

$$\Pr_{\boldsymbol{x}}[T^\star_{\mathrm{trunc}}(\boldsymbol{x}) \neq T^\star(\boldsymbol{x})] \leq 2^{-\log(S/\varepsilon)} \cdot S = \varepsilon.$$

Note that the depth of $T^\star_{\mathrm{trunc}}$ is $\leq \log(s/\varepsilon)/\varepsilon^2$. We have shown the following corollary of Lemma 8:

▶ **Corollary 11** (Approximating stochastic DTs with deterministic ones). *Let $\boldsymbol{T} : \{0,1\}^n \to \{0,1\}$ be a size-s stochastic DT. For every $\varepsilon \in (0, \frac{1}{2})$, there is a deterministic DT $T^\star_{\mathrm{trunc}} : \{0,1\}^n \to \{0,1\}$ such that*

1. $\mathrm{depth}(T^\star_{\mathrm{trunc}}) \leq \log(S/\varepsilon) \leq \log(s/\varepsilon)/\varepsilon^2$ *and*
2. $\Pr_{\boldsymbol{x},\boldsymbol{T}}[T^\star_{\mathrm{trunc}}(\boldsymbol{x}) \neq \boldsymbol{T}(\boldsymbol{x})] \leq \mathrm{opt}_{\boldsymbol{T}} + 3\varepsilon$.

To show that Find returns a tree of small error with respect to $\boldsymbol{T}$, we need the following generalization bound from [30]:

▶ **Lemma 12** (Generalization). *Let $\boldsymbol{T}$ be a stochastic tree of size s. For $S = s^{O(1/\varepsilon^2)}$ and a sample size of*

$$m \coloneqq \mathrm{poly}\left(n^{\log(S/\varepsilon)}, \frac{1}{\varepsilon}, \log\left(\frac{1}{\delta}\right)\right),$$

let X be a dataset of m i.i.d points of the form $(\boldsymbol{x}, \boldsymbol{T}(\boldsymbol{x}))$. Then $\textsc{Find}(X, \log(S/\varepsilon))$ outputs $T^\star$ such that

$$\Pr_{\text{draw of } \boldsymbol{X}}\left[\Pr_{\boldsymbol{x}\sim\{0,1\}^n}[T^\star(\boldsymbol{x}) \neq \boldsymbol{T}(\boldsymbol{x})] \leq \mathrm{opt}_{\boldsymbol{T}} + 3\varepsilon\right] \geq 1 - \delta.$$

Proof. The proof is given in the proof of Theorem 2 in [30]. Lemma 9 gives us that Find outputs a tree of minimal error with respect to X. They apply Chernoff bounds to bound the probability that a fixed tree T' of depth $\log(S/\varepsilon)$ and error $> \mathrm{opt}_{\boldsymbol{T}} + 3\varepsilon$ with respect to $\boldsymbol{T}$ has smaller error with respect to X than $T^\star_{\mathrm{trunc}}$ as described in Corollary 11. More specifically, the probability over draws of X that $\Pr_{(\boldsymbol{x},\boldsymbol{y})\sim X}[T^\star_{\mathrm{trunc}}(\boldsymbol{x}) \neq \boldsymbol{y}] > \mathrm{opt}_{\boldsymbol{T}} + 3\varepsilon$ or $\Pr_{(\boldsymbol{x},\boldsymbol{y})\sim X}[T'(\boldsymbol{x}) \neq \boldsymbol{y}] \leq \mathrm{opt}_{\boldsymbol{T}} + 3\varepsilon$ is exponentially small in $|X|$. This is a bound on the probability that Find outputs a particular tree of error greater than $\mathrm{opt}_{\boldsymbol{T}} + 3\varepsilon$; the lemma follows from a union bound over all trees of depth at most $\log(S/\varepsilon)$. ◀

Lemma 10 gives us that $\textsc{Find}(X, \log(S/\varepsilon))$ runs in time $n^{O(\log(S/\varepsilon))} \cdot O(|X|) = n^{O(\log(s/\varepsilon)/\varepsilon^2)} \cdot O(|X|)$. For confidence parameter δ, $|X|$ is polynomial in $n^{\log(S/\varepsilon)}$, $\log(1/\varepsilon)$, and $\log(1/\delta)$. Thus, the total runtime of Find is $n^{O(\log(s/\varepsilon)/\varepsilon^2)} \cdot \mathrm{poly}\log(1/\delta))$. The desired result holds by renaming $\varepsilon' = \varepsilon/3$. ◀

4.2 Proof of Theorem 3

We recall Theorem 3, this time including the confidence parameter δ.

▶ **Theorem 3** (Our main result). *There is an algorithm $\mathcal{A}$ with the following guarantee. For all $\varepsilon, \eta \in (0,1)$ and $s \in \mathbb{N}$, given access to a sufficiently large η-corrupted set $\mathcal{S}$ of uniform random samples labeled by a size-s stochastic decision tree $\boldsymbol{T} : \{0,1\}^n \to \{0,1\}$, $\mathcal{A}$ runs in $n^{O(\log(s/\varepsilon)/\varepsilon^2)} \cdot \mathrm{poly}(\log(1/\delta))$ time and with probability $1-\delta$ outputs a decision tree hypothesis h such that $\Pr[h(\boldsymbol{x}) \neq \boldsymbol{T}(\boldsymbol{x})] \leq \mathrm{opt} + 2\eta + \varepsilon$, where opt denotes the Bayes optimal error for $\boldsymbol{T}$.*

The proof requires the following fact.

▶ **Fact 13** (Error from sample corruption). *For any bounded function $p : \{0,1\}^n \to [0,1]$ and sample $\mathcal{S}^\circ$ of points $(x_1, y_1), \ldots, (x_m, y_m)$ with $0 \leq y_i \leq 1$. Let $\mathcal{S}$ be a corrupted sample formed by picking an arbitrary η-fraction of points $\mathcal{S}^\circ$ and replacing each with an arbitrary (also bounded) point. Then for any $\mathrm{err} : [0,1] \times [0,1] \to [0,1]$*

$$\left| \mathop{\mathbb{E}}_{(\boldsymbol{x},\boldsymbol{y})\sim\mathcal{S}^\circ} [\mathrm{err}(p(\boldsymbol{x}), \boldsymbol{y})] - \mathop{\mathbb{E}}_{(\boldsymbol{x},\boldsymbol{y})\sim\mathcal{S}} [\mathrm{err}(p(\boldsymbol{x}), \boldsymbol{y})] \right| < \eta.$$

Recall $T^\star_{\mathrm{trunc}}$ as described in Corollary 11, which has error $\leq \mathrm{opt}_{\boldsymbol{T}} + O(\varepsilon)$ with respect to $\boldsymbol{T}$. Let $\mathcal{S}^\circ$ be the uncorrupted set of examples of $\boldsymbol{T}$, and $\mathcal{S}$ be an η-corruption of $\mathcal{S}^\circ$. Then with probability $1-\delta$ over draws of $\mathcal{S}^\circ$,

$$\Pr_{(\boldsymbol{x},\boldsymbol{y})\sim\mathcal{S}^\circ}[T^\star_{\mathrm{trunc}}(\boldsymbol{x}) \neq \boldsymbol{y}] \leq \mathrm{opt}_{\boldsymbol{T}} + 3\varepsilon \qquad \text{(Lemma 12)}$$

$$\Pr_{(\boldsymbol{x},\boldsymbol{y})\sim\mathcal{S}}[T^\star_{\mathrm{trunc}}(\boldsymbol{x}) \neq \boldsymbol{y}] \leq \mathrm{opt}_{\boldsymbol{T}} + \eta + 3\varepsilon. \qquad \text{(Fact 13)}$$

Let $T^\star$ be the output of $\textsc{Find}(\mathcal{S}, \log(S/\varepsilon))$. Then,

$$\Pr_{(\boldsymbol{x},\boldsymbol{y})\sim\mathcal{S}}[T^\star(\boldsymbol{x}) \neq \boldsymbol{y}] \leq \mathrm{opt}_{\boldsymbol{T}} + \eta + 3\varepsilon \qquad \text{(Lemma 9)}$$

$$\Pr_{(\boldsymbol{x},\boldsymbol{y})\sim\mathcal{S}^\circ}[T^\star(\boldsymbol{x}) \neq \boldsymbol{y}] \leq \mathrm{opt}_{\boldsymbol{T}} + 2\eta + 3\varepsilon. \qquad \text{(Fact 13)}$$

$$\Pr_{\text{draw of } \mathcal{S}^\circ}\left[\Pr_{x\sim\{0,1\}^n}[T^\star(x) \neq \boldsymbol{T}(x)] \leq \mathrm{opt}_{\boldsymbol{T}} + 2\eta + 3\varepsilon \right] > 1-\delta \qquad \text{(Lemma 12)}$$

The desired result holds by renaming $\varepsilon' = \varepsilon/3$. ◀

5 Noise-tolerant properties of L_1 and L_2 regression

In this section, we prove Theorem 4, showing that the low-degree algorithm of [29] (also known as L_2 regression) and L_1 regression algorithm of [21] both learn stochastic-leaf DTs with adversarial corruption, albeit with worse parameters than our method. Throughout this section, we use the following function.

▶ **Definition 14** (The trunc function). *The function,* $\mathrm{trunc} : \mathbb{R} \to [0,1]$, *is defined as*

$$\mathrm{trunc}(x) = \begin{cases} 0 & \text{if } x < 0 \\ 1 & \text{if } x > 1 \\ x & \text{otherwise.} \end{cases}$$

The basis of the results in this section is Proposition 15, that if $\boldsymbol{T}$ is a size-s stochastic DT, there is a degree $\log(s/\varepsilon)$ bounded polynomial $p : \{0,1\}^n \to [0,1]$ which is ε close to $\mu_{\boldsymbol{T}}$:

▶ **Proposition 15** ($\mu_{\boldsymbol{T}}$ is morally low degree). *Let $\boldsymbol{T}$ be a size-s stochastic DT. There is a polynomial $p : \{0,1\}^n \to [0,1]$ such that $\Pr_{\boldsymbol{x}}[p(\boldsymbol{x}) \neq \mu_{\boldsymbol{T}}(\boldsymbol{x})] \leq \varepsilon$, where $\deg(p) \leq \log(s/\varepsilon)$.*

In order to handle our challenging noise model, it is important that we can guarantee the p in Proposition 15 is bounded. Without that guarantee, L_1 and L_2 regression are not known to handle noise in both the examples and the labels.

Proof. For any leaf ℓ of $\boldsymbol{T}$, let $\text{depth}(\ell)$ be the number of *deterministic* nodes on the root-to-leaf path to ℓ, not counting ℓ itself. The fraction of inputs in $\{0,1\}^n$ that have a nonzero chance of reaching ℓ is $2^{-\text{depth}(\ell)}$. Now, let $\boldsymbol{T}'$ be the stochastic decision tree that is nearly equivalent to $\boldsymbol{T}$ except if an input reaches a leaf with deterministic depth more than $\log(s/\varepsilon)$, $\boldsymbol{T}'$ returns 0. We claim that $p \coloneqq \mu_{\boldsymbol{T}'}$ satisfies Proposition 15. For that, we need to verify three things about $\mu_{\boldsymbol{T}'}$:

1. $\mu_{\boldsymbol{T}'}$ and $\mu_{\boldsymbol{T}}$ are close: $\Pr_{\boldsymbol{x}}[\mu_{\boldsymbol{T}}(\boldsymbol{x}) \neq \mu_{\boldsymbol{T}'}(\boldsymbol{x})] \leq \varepsilon$. This is true because $\boldsymbol{T}$ and $\boldsymbol{T}'$ can differ only on inputs which reach a leaf with deterministic depth at least $\log(s/\varepsilon)$. At most $2^{-\log(s/\varepsilon)} = \varepsilon/s$ fraction of inputs reach each such leaf, and there are at most s of them.
2. $\mu_{\boldsymbol{T}'}$ is a degree $\log(s/\varepsilon)$ polynomial. We can write $\mu_{\boldsymbol{T}'}(x)$ as

$$\begin{aligned}\mu_{\boldsymbol{T}'}(x) &= \sum_{\text{leaves } \ell \in \boldsymbol{T}'} \Pr[x \text{ reaches } \ell] \cdot (\text{label of } \ell) \\ &= \sum_{\text{leaves } \ell \in \boldsymbol{T}} \Pr[x \text{ reaches } \ell] \cdot \mathbb{1}[\text{depth}(\ell) \leq \log(s/\varepsilon)] \cdot (\text{label of } \ell).\end{aligned}$$

   The expression $\Pr[x \text{ reaches } \ell]$ is a degree $\text{depth}(\ell)$ polynomial. Therefore, $\mu_{\boldsymbol{T}'}(x)$ is a degree $\log(s/\varepsilon)$ polynomial.
3. The output of $\mu_{\boldsymbol{T}'}$ is bounded on $[0,1]$. This is true since $\boldsymbol{T}'$ always returns a value in $\{0,1\}$. ◀

5.1 L_2 Regression

Given corrupted samples from some stochastic DT $\boldsymbol{T}$, we will apply Lemma 16, given below, to show that L_2 regression can find a function f that is close to $\mu_{\boldsymbol{T}}$. Then, we will apply Fact 6 to generate a hypothesis with error close to the Bayes optimal error.

▶ **Lemma 16** (L_2 error to mean error). *Fix any stochastic DT $\boldsymbol{T} : \{0,1\}^n \to \{0,1\}$, degree $d \in \mathbb{N}$, and $\varepsilon, \delta > 0$. For a sample size of*

$$m \coloneqq \text{poly}\left(n^d, 1/\varepsilon, \log(1/\delta)\right),$$

let $\mathcal{S}^\circ$ be a dataset of m i.i.d points of the form $(\boldsymbol{x}, \boldsymbol{T}(\boldsymbol{x}))$. With probability at least $1-\delta$, there exists a constant $C \in \mathbb{R}$ for which the following holds for all degree d polynomials $p : \{0,1\}^n \to \mathbb{R}$.

$$\left| \mathop{\mathbb{E}}_{(\boldsymbol{x},\boldsymbol{y}) \sim \mathcal{S}^\circ} \left[(\text{trunc}(p(\boldsymbol{x})) - \boldsymbol{y})^2\right] - \left(\mathop{\mathbb{E}}_{\boldsymbol{x} \sim \{0,1\}^n} \left[(\text{trunc}(p(\boldsymbol{x})) - \mu_{\boldsymbol{T}}(\boldsymbol{x}))^2\right] + C \right) \right| \leq \varepsilon. \tag{2}$$

Proof. We prove Lemma 16 in two steps: First, we argue that there is a C for which Equation (2) holds for any fixed polynomial with extremely high probability. Then, we discretize the set of all truncated degree d polynomials into a finite set $\mathcal{P}$. By union bound, we can show that Equation (2) applies to all functions in $\mathcal{P}$, and since every truncated degree d polynomial is sufficiently close to a function in $\mathcal{P}$, this is enough to guarantee that Equation (2) applies to all degree d polynomials.

We use the following identity: For any constant $a \in \mathbb{R}$ and random variable $\boldsymbol{z} \in \mathbb{R}$,

$$\mathop{\mathbb{E}}_{\boldsymbol{z}}\left[(a-\boldsymbol{z})^2\right] = (a - \mathbb{E}[\boldsymbol{z}])^2 + \mathrm{Var}[\boldsymbol{z}].$$

Fix any $p : \{0,1\}^n \to \mathbb{R}$. For any $x \in \{0,1\}^n$, $\boldsymbol{T}(x)$ is a random variable with mean $\mu_{\boldsymbol{T}}(x)$. Therefore,

$$\mathop{\mathbb{E}}_{\boldsymbol{x}\sim\{0,1\}^n}\left[(\mathrm{trunc}(p(\boldsymbol{x})) - \boldsymbol{T}(x))^2\right] = \mathop{\mathbb{E}}_{\boldsymbol{x}\sim\{0,1\}^n}\left[(\mathrm{trunc}(p(\boldsymbol{x})) - \mu_{\boldsymbol{T}}(x))^2\right] + \mathop{\mathbb{E}}_{\boldsymbol{x}\sim\{0,1\}^n}\left[\mathrm{Var}[\boldsymbol{T}(x)]\right]$$

For $C = \mathbb{E}_{\boldsymbol{x}\sim\{0,1\}^n}\left[\mathrm{Var}[\boldsymbol{T}(x)]\right]$, Equation (2) holds in expectation over $\mathcal{S}$ with $\varepsilon = 0$. Since $(\mathrm{trunc}(p(x)) - y)^2$ is bounded on $[0,1]$, we can apply Hoeffdings inequality: For any fixed p, Equation (2) holds with probability at least $1 - 2\exp_e(-2m^2\varepsilon^2)$.

We next discretize the set of all truncated degree d polynomials. Let $\mathcal{P}$ be the following finite set of functions,

$$\mathcal{P} \coloneqq \{\mathrm{trunc} \circ p \mid p \text{ is degree-}d \text{ polynomial with coefficients that are all a multiple of } \varepsilon/n^d\}$$

Degree d polynomials have at most n^d coefficients. Therefore,

$$\log(|\mathcal{P}|) \le \log\left(\left(\frac{n^d}{\varepsilon}\right)^{n^d}\right) = \mathrm{poly}\left(n^{O(d)}, \log(1/\varepsilon)\right).$$

This means that for the sample size in Lemma 16, Equation (2) holds for all functions in $\mathcal{P}$ with probability at least $1-\delta$. We show that Equation (2) holding for function in $\mathcal{P}$ implies the desired result.

Every degree d truncated polynomial is *pointwise* close to a function in $\mathcal{P}$: Fix any degree d polynomial p. There is some $f \in \mathcal{P}$, for which

$$|\mathrm{trunc}(p(x)) - f(x)| < \varepsilon \quad \text{for all } x \in \{0,1\}^n.$$

This f is easy to specify: It's the truncation of p', where p' is p with all of its coefficients rounded to the nearest ε/n^d. In order to expand Equation (2) to p, we use the following inequality for all $a, \varepsilon \in [0,1]$:

$$|\,(a+\varepsilon)^2 - a^2\,| = |2a\varepsilon| + \varepsilon^2 \le 3|\varepsilon|.$$

Therefore,

$$\begin{aligned}\left|\mathop{\mathbb{E}}_{(\boldsymbol{x},\boldsymbol{y})\sim\mathcal{S}}\left[(\mathrm{trunc}(p(\boldsymbol{x})) - \boldsymbol{y})^2\right] - \mathop{\mathbb{E}}_{(\boldsymbol{x},\boldsymbol{y})\sim\mathcal{S}}\left[f(\boldsymbol{x}) - \boldsymbol{y})^2\right]\right| &\le 3 \max_{x\in\{0,1\}^n} |\,\mathrm{trunc}(p(x)) - f(x)\,| \\ &\le 3\,\varepsilon.\end{aligned}$$

Similarly, $\mathbb{E}_{\boldsymbol{x}\sim\{0,1\}^n}\left[(\mathrm{trunc}(p(\boldsymbol{x})) - \mu_{\boldsymbol{T}}(\boldsymbol{x}))^2\right]$ and $\mathbb{E}_{\boldsymbol{x}\sim\{0,1\}^n}\left[(f(\boldsymbol{x}) - \mu_{\boldsymbol{T}}(\boldsymbol{x}))^2\right]$ are within 3ε of one another. Finally, by triangle inequality,

$$\left|\mathop{\mathbb{E}}_{(\boldsymbol{x},\boldsymbol{y})\sim\mathcal{S}}\left[(\mathrm{trunc}(p(\boldsymbol{x})) - \boldsymbol{y})^2\right] - \left(\mathop{\mathbb{E}}_{\boldsymbol{x}\sim\{0,1\}^n}\left[(\mathrm{trunc}(p(\boldsymbol{x})) - \mu_{\boldsymbol{T}}(\boldsymbol{x}))^2\right] + C\right)\right| \le 7\varepsilon.$$

The desired result holds if we rename $\varepsilon' = \frac{\varepsilon}{7}$. ◀

We are now ready to prove the low-degree algorithm (i.e. L_2 regression) part of Theorem 4.

▶ **Lemma 17** (L_2 regression part of Theorem 4). *Choose any $\varepsilon, \eta, \delta \in (0,1)$, $s \in \mathbb{N}$, and size-s stochastic decision tree $\boldsymbol{T} : \{0,1\}^n \to \{0,1\}$. For a sample of size*

$$m \coloneqq \text{poly}\left(n^{O(d)}, \frac{1}{\varepsilon}, \log\left(\frac{1}{\delta}\right)\right),$$

let $\mathcal{S}$ be an η-corrupted set of m uniform random samples from $\boldsymbol{T}$. If

$$p^* = \underset{\text{Degree } \log(s/\varepsilon) \text{ polynomials } p}{\arg\min} \left(\mathop{\mathbb{E}}_{(\boldsymbol{x},\boldsymbol{y})\sim\mathcal{S}} \left[(p(\boldsymbol{x}) - \boldsymbol{y})^2\right] \right),$$

and $h : \{0,1\}^n \to \{0,1\}$ is the hypothesis $h(x) = \text{round}(\text{trunc}(p^*(x)))$. Then with probability at least $1-\delta$ over the randomness of the sample,

$$\text{error}_{\boldsymbol{T}}(h) \le \text{opt} + O(\sqrt{\eta}) + \varepsilon.$$

Proof. Let $\mathcal{S}^\circ$ be the original uncorrupted (i.i.d) set of samples, from which $\mathcal{S}$ differs on at most η fraction of points. By Lemma 16, Equation (2) holds, with respect to $\mathcal{S}^\circ$, for all degree $\log(s/\varepsilon)$ polynomials with probability at least $1-\delta$. We show that if it holds, then $\text{error}_{\boldsymbol{T}}(h) \le \text{opt} + O(\sqrt{\eta}) + \varepsilon$.

Proposition 15 guarantees there exists $p : \{0,1\}^n \to [0,1]$, a degree $\log(s/\varepsilon)$ bounded polynomial, satisfying

$$\mathop{\mathbb{E}}_{\boldsymbol{x}\sim\{0,1\}^n} \left[(p(\boldsymbol{x}) - \mu_{\boldsymbol{T}}(\boldsymbol{x}))^2\right] \le \varepsilon.$$

Fix C as in Lemma 16. Combining Equation (2) and Fact 13, we have that

$$\mathop{\mathbb{E}}_{(\boldsymbol{x},\boldsymbol{y})\sim\mathcal{S}} \left[(p(\boldsymbol{x}) - \boldsymbol{y})^2\right] \le C + 2\varepsilon + \eta.$$

Since p^* has the minimum L_2 error of all degree $\log(s/\varepsilon)$ polynomials on $\mathcal{S}$,

$$\mathop{\mathbb{E}}_{(\boldsymbol{x},\boldsymbol{y})\sim\mathcal{S}} \left[(p^*(\boldsymbol{x}) - \boldsymbol{y})^2\right] \le C + 2\varepsilon + \eta.$$

Truncating p^* can only decrease its L_2 error. Combining that with a second application of Fact 13,

$$\mathop{\mathbb{E}}_{(\boldsymbol{x},\boldsymbol{y})\sim\mathcal{S}^\circ} \left[(\text{trunc}(p^*(\boldsymbol{x})) - \boldsymbol{y})^2\right] \le C + 2\varepsilon + 2\eta.$$

Then, by Equation (2),

$$\mathop{\mathbb{E}}_{\boldsymbol{x}\sim\{0,1\}^n} \left[(\text{trunc}(p^*(\boldsymbol{x})) - \mu_{\boldsymbol{T}}(\boldsymbol{x}))^2\right] \le 3\varepsilon + 2\eta. \tag{3}$$

Finally,

$$\begin{aligned}
\text{error}_{\boldsymbol{T}}(h) &\le \text{opt}_{\boldsymbol{T}} + 2 \mathop{\mathbb{E}}_{\boldsymbol{x}\sim\{0,1\}^n} [|\, \mu_{\boldsymbol{T}}(\boldsymbol{x}) - \text{trunc}(p^*(\boldsymbol{x}))\,|] && \text{Fact 6} \\
&\le \text{opt}_{\boldsymbol{T}} + 2\sqrt{\mathop{\mathbb{E}}_{\boldsymbol{x}\sim\{0,1\}^n} [(\mu_{\boldsymbol{T}}(\boldsymbol{x}) - \text{trunc}(p^*(\boldsymbol{x})))^2]} && \text{Jensen's inequality} \\
&\le \text{opt}_{\boldsymbol{T}} + 2\sqrt{3\varepsilon + 2\eta} && \text{Equation (3)} \\
&\le \text{opt}_{\boldsymbol{T}} + O(\sqrt{\varepsilon}) + O(\sqrt{\eta}).
\end{aligned}$$

The desired result then holds by renaming $\varepsilon' = \Omega(\varepsilon^2)$. ◀

5.2 L_1 regression

We will need the following generalization bound:

▶ **Lemma 18** (L_1 error generalization). *Fix any stochastic DT* $\boldsymbol{T} : \{0,1\}^n \to \{0,1\}$*, degree* $d \in \mathbb{N}$*, and* $\varepsilon, \delta > 0$*. For a sample size of*

$$m := \text{poly}\left(n^{O(d)}, \frac{1}{\varepsilon}, \log\left(\frac{1}{\delta}\right)\right),$$

let $\mathcal{S}^\circ$ *be a dataset of* m *i.i.d points of the form* $(\boldsymbol{x}, \boldsymbol{T}(\boldsymbol{x}))$*. With probability at least* $1-\delta$*, the following holds for all degree* d *polynomials* $p : \{0,1\}^n \to \mathbb{R}$*.*

$$\left| \mathop{\mathbb{E}}_{(\boldsymbol{x},\boldsymbol{y})\sim\mathcal{S}^\circ} \left[|\text{trunc}(p(\boldsymbol{x})) - \boldsymbol{y}|^2 \right] - \mathop{\mathbb{E}}_{\boldsymbol{x}\sim\{0,1\}^n, \boldsymbol{T}} \left[|\text{trunc}(p(\boldsymbol{x})) - \boldsymbol{T}(\boldsymbol{x})| \right] \right| \leq \varepsilon. \tag{4}$$

Lemma 18 can be proven using the same discretization argument as Lemma 16. We omit the proof for brevity.

▶ **Lemma 19** (L_1 regression part of Theorem 4). *Choose any* $\varepsilon, \eta, \delta \in (0,1)$*,* $s \in \mathbb{N}$*, and size-*s *stochastic decision tree* $\boldsymbol{T} : \{0,1\}^n \to \{0,1\}$*. For a sample of size*

$$m := \text{poly}\left(n^{O(d)}, \frac{1}{\varepsilon}, \log\left(\frac{1}{\delta}\right)\right),$$

let $\mathcal{S}$ *be an* η*-corrupted set of* m *uniform random samples from* $\boldsymbol{T}$*. If*

$$p^* = \mathop{\arg\min}_{\text{Degree } \log(s/\varepsilon) \text{ polynomials } p} \left(\mathop{\mathbb{E}}_{(\boldsymbol{x},\boldsymbol{y})\sim\mathcal{S}} [|\, p(\boldsymbol{x}) - \boldsymbol{y})\,|] \right),$$

and $\boldsymbol{h} : \{0,1\}^n \to \{0,1\}$ *is the randomized hypothesis where* $\boldsymbol{h}(x)$ *is* 1 *with probability* $\text{trunc}(p^*(x))$ *and* 0 *otherwise. Then with probability at least* $1-\delta$ *over the randomness of the sample,*

$$\text{error}_{\boldsymbol{T}}(\boldsymbol{h}) \leq 2\text{opt} + 2\eta + \varepsilon.$$

Proof. Let $\mathcal{S}^\circ$ be the original uncorrupted (i.i.d) set of samples from which $\mathcal{S}$ differs on at most η fraction of points. By Lemma 18, Equation (4) holds, with respect to $\mathcal{S}^\circ$ for all degree $\log(s/\varepsilon)$ polynomials with probability at least $1-\delta$. We show that if it holds, then $\text{error}_{\boldsymbol{T}}(h) \leq 2\text{opt} + 2\eta + \varepsilon$.

Proposition 15 guarantees there exists $p : \{0,1\}^n \to [0,1]$, a degree $\log(s/\varepsilon)$ polynomial, satisfying

$$\mathop{\mathbb{E}}_{\boldsymbol{x}} [|p(\boldsymbol{x}) - \mu_{\boldsymbol{T}}(\boldsymbol{x})|] \leq \varepsilon.$$

We first bound the expected error of $\mu_{\boldsymbol{T}}(\boldsymbol{x})$ relative to $\boldsymbol{T}(\boldsymbol{x})$.

$$\begin{aligned}
\mathop{\mathbb{E}}_{\boldsymbol{x}} [|\mu_{\boldsymbol{T}}(\boldsymbol{x}) - \boldsymbol{T}(\boldsymbol{x})|] &= \mathop{\mathbb{E}}_{\boldsymbol{x}} [\Pr[\boldsymbol{T}(\boldsymbol{x}) = 1](1 - \mu_{\boldsymbol{T}}(\boldsymbol{x})) + \Pr[\boldsymbol{T}(\boldsymbol{x}) = 0](\mu_{\boldsymbol{T}}(\boldsymbol{x}))] \\
&= \mathop{\mathbb{E}}_{\boldsymbol{x}} [2\mu_{\boldsymbol{T}}(\boldsymbol{x})(1 - \mu_{\boldsymbol{T}}(\boldsymbol{x}))] \\
&\leq 2 \cdot \mathop{\mathbb{E}}_{\boldsymbol{x}} [\min(\mu_{\boldsymbol{T}}(\boldsymbol{x}), 1 - \mu_{\boldsymbol{T}}(\boldsymbol{x}))] \\
&= 2 \cdot \text{opt}_{\boldsymbol{T}}
\end{aligned}$$

By triangle inequality, we have that $\mathbb{E}_{\boldsymbol{x}}[|p(\boldsymbol{x}) - \boldsymbol{T}(\boldsymbol{x})|] \leq 2 \cdot \text{opt}_{\boldsymbol{T}} + \varepsilon$. By Equation (4)

$$\mathop{\mathbb{E}}_{\boldsymbol{x},\boldsymbol{y}\sim\mathcal{S}^\circ}[|p(\boldsymbol{x}) - \boldsymbol{y}|] \leq 2 \cdot \text{opt}_{\boldsymbol{T}} + 2\varepsilon.$$

By Fact 13 initialized with $\text{err}(x, y) = |x - y|$,

$$\mathop{\mathbb{E}}_{\boldsymbol{x},\boldsymbol{y}\sim\mathcal{S}}[|p(\boldsymbol{x}) - \boldsymbol{y}|] \leq 2 \cdot \text{opt}_{\boldsymbol{T}} + 2\varepsilon + \eta.$$

Since p^* has minimum L_1 error among all degree $\log(s/\varepsilon)$ polynomials,

$$\mathop{\mathbb{E}}_{\boldsymbol{x},\boldsymbol{y}\sim\mathcal{S}}[|p^*(\boldsymbol{x}) - \boldsymbol{y}|] \leq 2 \cdot \text{opt}_{\boldsymbol{T}} + 2\varepsilon + \eta.$$

Reapplying Fact 13, combined with the fact that truncating p^* can only decrease its error,

$$\begin{aligned}\mathop{\mathbb{E}}_{\boldsymbol{x},\boldsymbol{y}\sim\mathcal{S}^\circ}[|\text{trunc}(p^*(\boldsymbol{x})) - \boldsymbol{y}|] &\leq \mathop{\mathbb{E}}_{\boldsymbol{x},\boldsymbol{y}\sim\mathcal{S}}[|\text{trunc}(p^*(\boldsymbol{x})) - \boldsymbol{y}|] + \eta \\ &\leq 2 \cdot \text{opt}_{\boldsymbol{T}} + 2\varepsilon + 2\eta.\end{aligned}$$

Applying Equation (4) again.

$$\begin{aligned}\mathop{\mathbb{E}}_{\boldsymbol{x}\sim\{0,1\}^n}[|p(\boldsymbol{x}) - \boldsymbol{T}(\boldsymbol{x})|] &\leq \mathop{\mathbb{E}}_{\boldsymbol{x},\boldsymbol{y}\sim\mathcal{S}^\circ}[|p(\boldsymbol{x}) - \boldsymbol{T}(\boldsymbol{x})|] + \varepsilon \\ &\leq 2 \cdot \text{opt}_{\boldsymbol{T}} + 3\varepsilon + 2\eta.\end{aligned}$$

Finally, since $\boldsymbol{h}(x)$ returns 1 with probability $\text{trunc}(p^*(x))$, and $\boldsymbol{T}(x)$ is always in $\{0, 1\}$,

$$\begin{aligned}\Pr_{\boldsymbol{x},\boldsymbol{h},\boldsymbol{T}}[\boldsymbol{h}(\boldsymbol{x}) \neq \boldsymbol{T}(\boldsymbol{x})] &= \mathop{\mathbb{E}}_{\boldsymbol{x}\sim\{0,1\}^n}[|p(\boldsymbol{x}) - \boldsymbol{T}(\boldsymbol{x})|] \\ &\leq 2 \cdot \text{opt}_{\boldsymbol{T}} + 3\varepsilon + 2\eta.\end{aligned}$$

The desired result holds with the renaming $\varepsilon' = \frac{\varepsilon}{3}$. ◀

References

1. Guy Blanc, Neha Gupta, Jane Lange, and Li-Yang Tan. Universal guarantees for decision tree induction via a higher-order splitting criterion. In *Proceedings of the 34th Conference on Neural Information Processing Systems (NeurIPS)*, 2020.
2. Guy Blanc, Jane Lange, and Li-Yang Tan. Provable guarantees for decision tree induction: the agnostic setting. In *Proceedings of the 37th International Conference on Machine Learning (ICML)*, 2020. Available at `arXiv:2006.00743`.
3. Guy Blanc, Jane Lange, and Li-Yang Tan. Top-down induction of decision trees: rigorous guarantees and inherent limitations. In *Proceedings of the 11th Innovations in Theoretical Computer Science Conference (ITCS)*, volume 151, pages 1–44, 2020.
4. Avirm Blum, Merrick Furst, Jeffrey Jackson, Michael Kearns, Yishay Mansour, and Steven Rudich. Weakly learning DNF and characterizing statistical query learning using Fourier analysis. In *Proceedings of the 26th Annual ACM Symposium on Theory of Computing (STOC)*, pages 253–262, 1994.
5. Avrim Blum. Rank-r decision trees are a subclass of r-decision lists. *Inform. Process. Lett.*, 42(4):183–185, 1992. `doi:10.1016/0020-0190(92)90237-P`.
6. Alon Brutzkus, Amit Daniely, and Eran Malach. ID3 learns juntas for smoothed product distributions. In *Proceedings of the 33rd Annual Conference on Learning Theory (COLT)*, pages 902–915, 2020.
7. Nader Bshouty. Exact learning via the monotone theory. In *Proceedings of 34th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 302–311, 1993.

8 Nader H Bshouty, Nadav Eiron, and Eyal Kushilevitz. Pac learning with nasty noise. *Theoretical Computer Science*, 288(2):255–275, 2002.

9 Sitan Chen and Ankur Moitra. Beyond the low-degree algorithm: mixtures of subcubes and their applications. In *Proceedings of the 51st Annual ACM Symposium on Theory of Computing (STOC)*, pages 869–880, 2019.

10 Andrzej Ehrenfeucht and David Haussler. Learning decision trees from random examples. *Information and Computation*, 82(3):231–246, 1989.

11 Surbhi Goel, Aravind Gollakota, Zhihan Jin, Sushrut Karmalkar, and Adam Klivans. Superpolynomial lower bounds for learning one-layer neural networks using gradient descent. In *Proceedings of the 37th International Conference on Machine Learning (ICML)*, volume 119, pages 3587–3596, 2020.

12 Surbhi Goel, Aravind Gollakota, and Adam R. Klivans. Statistical-query lower bounds via functional gradients. In *Proceedings of the Annual Conference on Neural Information Processing Systems (NeurIPS)*, 2020.

13 Surbhi Goel and Adam Klivans. Learning neural networks with two nonlinear layers in polynomial time. In *Proceedings of the 32nd Conference on Learning Theory (COLT)*, volume 99, pages 1470–1499, 2019.

14 Surbhi Goel, Adam Klivans, and Raghu Meka. Learning one convolutional layer with overlapping patches. In *Proceedings of the 35th International Conference on Machine Learning (ICML)*, volume 80, pages 1783–1791, 2018.

15 Parikshit Gopalan, Adam Kalai, and Adam Klivans. Agnostically learning decision trees. In *Proceedings of the 40th ACM Symposium on Theory of Computing (STOC)*, pages 527–536, 2008.

16 Thomas Hancock. Learning $k\mu$ decision trees on the uniform distribution. In *Proceedings of the 6th Annual Conference on Computational Learning Theory (COT)*, pages 352–360, 1993.

17 Thomas Hancock, Tao Jiang, Ming Li, and John Tromp. Lower bounds on learning decision lists and trees. *Information and Computation*, 126(2):114–122, 1996.

18 David Haussler. Decision theoretic generalizations of the pac model for neural net and other learning applications. *Information and computation*, 100(1):78–150, 1992.

19 Elad Hazan, Adam Klivans, and Yang Yuan. Hyperparameter optimization: A spectral approach. *Proceedings of the 6th International Conference on Learning Representations (ICLR)*, 2018.

20 Jeffrey C. Jackson and Rocco A. Servedio. On learning random dnf formulas under the uniform distribution. *Theory of Computing*, 2(8):147–172, 2006. doi:10.4086/toc.2006.v002a008.

21 Adam Kalai, Adam Klivans, Yishay Mansour, and Rocco A. Servedio. Agnostically learning halfspaces. *SIAM Journal on Computing*, 37(6):1777–1805, 2008.

22 Adam Kalai, Alex Samorodnitsky, and Shang-Hua Teng. Learning and smoothed analysis. In *Proceedings of the 50th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 395–404, 2009.

23 Michael Kearns and Yishay Mansour. On the boosting ability of top-down decision tree learning algorithms. *Journal of Computer and System Sciences*, 58(1):109–128, 1999.

24 Michael Kearns and Robert Schapire. Efficient distribution-free learning of probabilistic concepts. *Journal of Computer and System Sciences*, 48(3):464–497, 1994.

25 Michael Kearns, Robert Schapire, and Linda Sellie. Toward efficient agnostic learning. *Machine Learning*, 17(2/3):115–141, 1994.

26 Adam Klivans and Rocco Servedio. Toward attribute efficient learning of decision lists and parities. *Journal of Machine Learning Research*, 7(Apr):587–602, 2006.

27 Eyal Kushilevitz and Yishay Mansour. Learning decision trees using the fourier spectrum. *SIAM Journal on Computing*, 22(6):1331–1348, 1993.

28 Homin Lee. *On the learnability of monotone functions.* PhD thesis, Columbia University, 2009.

29 Nathan Linial, Yishay Mansour, and Noam Nisan. Constant depth circuits, Fourier transform and learnability. *Journal of the ACM*, 40(3):607–620, 1993.

30 Dinesh Mehta and Vijay Raghavan. Decision tree approximations of boolean functions. *Theoretical Computer Science*, 270(1-2):609–623, 2002.

31 Ryan O'Donnell and Rocco Servedio. Learning monotone decision trees in polynomial time. *SIAM Journal on Computing*, 37(3):827–844, 2007.

32 Ronald Rivest. Learning decision lists. *Machine learning*, 2(3):229–246, 1987.

Breaking $O(nr)$ for Matroid Intersection

Joakim Blikstad ✉
KTH Royal Institute of Technology, Stockholm, Sweden

Abstract

We present algorithms that break the $\tilde{O}(nr)$-independence-query bound for the Matroid Intersection problem *for the full range of* r; where n is the size of the ground set and $r \leq n$ is the size of the largest common independent set. The $\tilde{O}(nr)$ bound was due to the efficient implementations [CLSSW FOCS'19; Nguyễn 2019] of the classic algorithm of Cunningham [SICOMP'86]. It was recently broken for large r ($r = \omega(\sqrt{n})$), first by the $\tilde{O}(n^{1.5}/\varepsilon^{1.5})$-query $(1-\varepsilon)$-approximation algorithm of CLSSW [FOCS'19], and subsequently by the $\tilde{O}(n^{6/5}r^{3/5})$-query exact algorithm of BvdBMN [STOC'21]. No algorithm – even an approximation one – was known to break the $\tilde{O}(nr)$ bound for the full range of r. We present an $\tilde{O}(n\sqrt{r}/\varepsilon)$-query $(1-\varepsilon)$-approximation algorithm and an $\tilde{O}(nr^{3/4})$-query exact algorithm. Our algorithms improve the $\tilde{O}(nr)$ bound and also the bounds by CLSSW and BvdBMN for the full range of r.

2012 ACM Subject Classification Theory of computation → Discrete optimization; Theory of computation → Approximation algorithms analysis; Mathematics of computing → Matroids and greedoids

Keywords and phrases Matroid Intersection, Combinatorial Optimization, Approximation Algorithms

Digital Object Identifier 10.4230/LIPIcs.ICALP.2021.31

Category Track A: Algorithms, Complexity and Games

Funding This project has received funding from the European Research Council (ERC) under the European Unions Horizon 2020 research and innovation programme under grant agreement No 71567.

Acknowledgements I thank Danupon Nanongkai and Sagnik Mukhopadhyay for insightful discussions and their valuable comments throughout the development of this work.

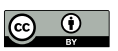
1 Introduction

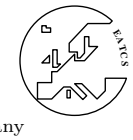
Matroid Intersection is a fundamental problem in combinatorial optimization that has been studied for more than half a century. The classic version of this problem is as follows: *Given two matroids* $\mathcal{M}_1 = (V, \mathcal{I}_1)$ *and* $\mathcal{M}_2 = (V, \mathcal{I}_2)$ *over a common ground set* V *of* n *elements, find the largest common independent set* $S^* \in \mathcal{I}_1 \cap \mathcal{I}_2$ *by making independence oracle queries*[1] *of the form "Is* $S \in \mathcal{I}_1$*?" or "Is* $S \in \mathcal{I}_2$*?" for* $S \subseteq V$*.* The size of the largest common independent set is usually denoted by r.

Matroid intersection can be used to model many other combinatorial optimization problems, such as bipartite matching, arborescences, spanning tree packing, etc. As such, designing algorithms for matroid intersection is an interesting problem to study.

In this paper, we consider the task of finding a $(1-\varepsilon)$-approximate solution to the matroid intersection problem, that is finding some common independent set S of size at least $(1-\varepsilon)r$. We show an improvement of approximation algorithms for matroid intersection, and as a consequence also obtain an improvement for the *exact* matroid intersection problem.

[1] There are also other oracle models considered in the literature (e.g. rank-oracles), but in this paper we focus on the independence query model. Whenever we say *query* in this paper, we thus mean *independence query*.





Previous work. Polynomial algorithms for matroid intersection started with the work of Edmond's $O(n^2r)$-query algorithms [6, 7, 8] in the 1960s. Since then, there has been a long line of research e.g. [1, 2, 3, 4, 5, 9, 10]. Cunningham [5] designed a $O(nr^{1.5})$-query blocking-flow algorithm in 1986, similar to that of Hopcroft-Karp's bipartite-matching or Dinic's maximum-flow algorithms. Chekuri and Quanrud [4] pointed out that Cunningham's classic algorithm [5] from 1986 is already a $O(nr/\varepsilon)$-query $(1-\epsilon)$-approximation algorithm. Recently, Chakrabarty-Lee-Sidford-Singla-Wong [3] and Nguyễn [11] independently showed how to implement Cunningham's classic algorithm using only $\tilde{O}(nr)$ independence queries. This is akin to spending $\tilde{O}(n)$ queries to find each of the so-called *augmenting paths*. A fundamental question is whether several augmenting paths can be found simultaneously to break the $\tilde{O}(nr)$ bound.

This question has been answered for large r ($r = \omega(\sqrt{n})$), first by the $\tilde{O}(n^{1.5}/\varepsilon^{1.5})$-query $(1-\epsilon)$-approximation algorithm of Chakrabarty-Lee-Sidford-Singla-Wong[2] [3], and very recently by the randomized $\tilde{O}(n^{6/5}r^{3/5})$-query exact algorithm of Blikstad-v.d.Brand-Mukhopadhyay-Nanongkai [2]. Whether we can break the $O(nr)$-query bound *for the full range of* r remained open *even for approximation algorithms.*

Our results. We break the $O(nr)$-query bound for both *approximation* and *exact* algorithms. We first state our results for approximate matroid intersection.[3]

▶ **Theorem 1** (Approximation algorithm)**.** *There is a deterministic algorithm which given two matroids $\mathcal{M}_1 = (V, \mathcal{I}_1)$ and $\mathcal{M}_2 = (V, \mathcal{I}_2)$ on the same ground set V, finds a common independent set $S \in \mathcal{I}_1 \cap \mathcal{I}_2$ with $|S| \geq (1-\varepsilon)r$, using $O\left(\frac{n\sqrt{r \log r}}{\varepsilon}\right)$ independence queries.*

Plugging Theorem 1 in the framework of [2], we get an improved algorithm – more efficient than the previous state-of-the-art – for exact matroid intersection which we state next.

▶ **Theorem 2** (Exact algorithm)**.** *There is a randomized algorithm which given two matroids $\mathcal{M}_1 = (V, \mathcal{I}_1)$ and $\mathcal{M}_2 = (V, \mathcal{I}_2)$ on the same ground set V, finds a common independent set $S \in \mathcal{I}_1 \cap \mathcal{I}_2$ of maximum cardinality r, and w.h.p.[4] uses $O(nr^{3/4} \log n)$ independence queries. There is also a deterministic exact algorithm using $O(nr^{5/6} \log n)$ queries.*

▶ Remark 3. Although we only focus on the query-complexity in this paper, we note that the time-complexity of the algorithms are dominated by query-oracle calls. That is, our approximation algorithm runs in $\tilde{O}(n\sqrt{r}\mathcal{T}_{\text{ind}}/\varepsilon)$ time, and the exact algorithms in $\tilde{O}(nr^{3/4}\mathcal{T}_{\text{ind}})$ (randomized) respectively $\tilde{O}(nr^{5/6}\mathcal{T}_{\text{ind}})$ time (deterministic), where $\mathcal{T}_{\text{ind}}$ denotes the time-complexity of the independence-oracle.

1.1 Technical Overview

Approximation algorithm. Our approximation algorithm (Theorem 1) is a modified version of Chakrabarty-Lee-Sidford-Singla-Wong's $\tilde{O}(n^{1.5}/\varepsilon^{1.5})$-query approximation algorithm [3, Section 6]. The algorithm is based on the ideas of Cunningham's classic blocking-flow

[2] In the same paper they also show a $\tilde{O}(n^2r^{-1}\varepsilon^{-2} + r^{1.5}\varepsilon^{-4.5})$-query algorithm.

[3] The $\tilde{O}(n^2r^{-1}\varepsilon^{-2} + r^{1.5}\varepsilon^{-4.5})$-query algorithm of [3] is the only previous algorithm which is more efficient than our algorithm in some range of r and ε. Actually, since the $\tilde{O}(n^2r^{-1}\varepsilon^{-2} + r^{1.5}\varepsilon^{-4.5})$-query algorithm use the $\tilde{O}(n^{1.5}/\varepsilon^{1.5})$ algorithm as a subroutine, we do get a slightly improved version by using our $\tilde{O}(n\sqrt{r}/\varepsilon)$ algorithm as the subroutine instead: $\tilde{O}(n^2r^{-1}\varepsilon^{-2} + r^{1.5}\varepsilon^{-4})$.

[4] *w.h.p. = with high probability* meaning with probability $1 - n^{-c}$ for some arbitrarily large constant c.

algorithm [5] and runs in $O(1/\varepsilon)$ phases, where in each phase the algorithm seeks to find a *maximal* set of *augmentations* in the *exchange graph*. Given a common independent set $S \in \mathcal{I}_1 \cap \mathcal{I}_2$, the *exchange graph* $G(S)$ is a directed bipartite graph (with bipartition $(S + \{s, t\}, V \setminus S)$). Finding a shortest (s, t)-path, called an *augmenting path*, in $G(S)$ means one can increase the size of the common independent set S by 1. Since the exchange graph changes after each augmentation,[5] and we do not know how to find a single augmenting path faster than $\Omega(n)$ queries, the need to find several augmentations in parallel arises. [3, Section 6] introduces the notion of *augmenting sets*: a generalization of the classical *augmenting paths* but where one can perform many augmentations in parallel.

So the revised goal of the algorithm is to, in each phase, efficiently find a *maximal augmenting set* (akin to a *blocking-flow* in bipartite matching or flow algorithms). Towards this goal, the algorithm maintains a relaxed version of augmenting set – called a *partial* augmenting set – and keeps *refining* it to make it "better" (i.e. closer to a maximal augmenting set). Here we give two independent improvements on top of the algorithm of [3]:

1. The algorithm of [3] refines the partial augmenting set by a sequence of operations on two adjacent distance layers in the exchange graph. In our algorithm, we instead consider *three* consecutive layers for our basic refinement procedures. This lets us focus our analysis on what happens in S – the "left" side of the bipartite exchange graph – which contains at most r elements in total (in contrast to [3] where the performance analysis is dependent on all n elements). The number of times we need to run the refinement procedures thus depends on r, instead of n, which makes the algorithm faster when $r = o(n)$.
2. When the partial augmenting set is "close enough" to a maximal augmenting set, [3] falls back to finding the remaining augmenting paths one at a time. In our algorithm, we also change to a different procedure when the partial augmenting set is close enough to maximal. The difference is that, instead of finding arbitrary augmenting paths, we find a special type of *valid paths* with respect to the partial augmenting set, so that these paths can be used to further improve (refine) the partial augmenting set. The number of valid paths we need to find is less than the number of augmenting paths [3] needs to find. This decreases the dependency on ε in the final algorithm.

The first improvement (Item 1) replaces the $\sqrt{n}$ term with a $\sqrt{r}$ term in the query complexity of the algorithm. The second improvement (Item 2) shaves off a $1/\sqrt{\varepsilon}$ term from the query complexity. Together they thus bring down the query complexity from $\tilde{O}(\frac{n\sqrt{n}}{\varepsilon\sqrt{\varepsilon}})$ in [3] to $\tilde{O}(\frac{n\sqrt{r}}{\varepsilon})$ as in our Theorem 1. Note that these two improvements are independent of each other, and can be applied individually.

Exact algorithm. To obtain the exact algorithm (Theorem 2), we use the framework of Blikstad-v.d.Brand-Mukhopadhyay-Nanongkai's $\tilde{O}(n^{6/5}r^{3/5})$-query exact algorithm [2]. The main idea of this algorithm is to combine approximation algorithms – which can efficiently find a common independent set only εr away from the optimal – with a randomized $\tilde{O}(n\sqrt{r})$-query subroutine to find each of the remaining *few, very long* augmenting paths. The $\tilde{O}(n^{6/5}r^{3/5})$-query exact algorithm [2] currently uses Chakrabarty-Lee-Sidford-Singla-Wong's $\tilde{O}(n^{1.5}/\varepsilon^{1.5})$ approximation algorithm [3] as a subroutine. Simply replacing it with our improved approximation algorithm (Theorem 1) yields our $\tilde{O}(nr^{3/4})$-query exact algorithm.

[5] Unlike what happens in augmenting path algorithms for flow and bipartite matching, where the underlying graphs remain the same.

2 Preliminaries

We use the standard definitions of *matroid* $\mathcal{M} = (V, \mathcal{I})$; *rank* $\text{rk}(X)$ for any $X \subseteq V$; *exchange graph* $G(S)$ for a common independent set $S \in \mathcal{I}_1 \cap \mathcal{I}_2$; and *augmenting paths* in $G(S)$ throughout this paper. For completeness, we define them below. We also need the notions of *augmenting sets* introduced by [3], which we also define in later this section.

Matroids

▶ **Definition 4** (Matroid). A *matroid* is a tuple $\mathcal{M} = (V, \mathcal{I})$ of a *ground set* V of n elements, and non-empty family $\mathcal{I} \subseteq 2^V$ of *independent sets* satisfying
Downward closure: if $S \in \mathcal{I}$, then $S' \in \mathcal{I}$ for all $S' \subseteq S$.
Exchange property: if $S, S' \in \mathcal{I}$, $|S| > |S'|$, then there exists $x \in S \setminus S'$ such that $S' \cup \{x\} \in \mathcal{I}$.

▶ **Definition 5** (Set notation). We will use $A + x$ and $A - x$ to denote $A \cup \{x\}$ respectively $A \setminus \{x\}$, as is usual in matroid intersection literature. We will also use $\bar{A} := V \setminus A$, $A + B := A \cup B$, and $A - B := A \setminus B$.

▶ **Definition 6** (Matroid rank). The *rank* of $A \subseteq V$, denoted by $\text{rk}(A)$, is the size of the largest (or, equivalently, any maximal) independent set contained in A. It is well-known that the rank-function is submodular, i.e. $\text{rk}(A + x) - \text{rk}(A) \geq \text{rk}(B + x) - \text{rk}(B)$ whenever $A \subseteq B \subseteq V$ and $x \in V \setminus B$.[6] Note that $\text{rk}(A) = |A|$ if and only if $A \subseteq \mathcal{I}$.

▶ **Definition 7** (Matroid Intersection). Given two matroids $\mathcal{M}_1 = (V, \mathcal{I}_1)$ and $\mathcal{M}_2 = (V, \mathcal{I}_2)$ over the same ground set V, a *common independent set* S is a set in $\mathcal{I}_1 \cap \mathcal{I}_2$. The *matroid intersection problem* asks us to find the largest common independent set – whose cardinality we denote by r. We use rk_1 and rk_2 to be the rank functions of the corresponding matroids.

The Exchange Graph

Many matroid intersection algorithms, e.g. those in [1, 2, 5, 7, 9, 11], are based on iteratively finding *augmenting paths* in the *exchange graph*.

▶ **Definition 8** (Exchange graph). Given two matroids $\mathcal{M}_1 = (V, \mathcal{I}_1)$ and $\mathcal{M}_2 = (V, \mathcal{I}_2)$ over the same ground set, and a common independent set $S \in \mathcal{I}_1 \cap \mathcal{I}_2$, the *exchange graph* $G(S)$ is a directed bipartite graph on vertex set $V \cup \{s, t\}$ with the following arcs (or directed edges):
1. (s, b) for $b \in \bar{S}$ when $S + b \in \mathcal{I}_1$.
2. (b, t) for $b \in \bar{S}$ when $S + b \in \mathcal{I}_2$.
3. (a, b) for $b \in \bar{S}, a \in S$ when $S + b - a \in \mathcal{I}_1$.
4. (b, a) for $b \in \bar{S}, a \in S$ when $S + b - a \in \mathcal{I}_2$.

We will denote the set of elements at distance k from s by the distance-layer D_k.

▶ **Definition 9** (Shortest augmenting path). A shortest (s, t)-path $p = (s, b_1, a_1, b_2, a_2, \ldots, a_\ell, b_{\ell+1}, t)$ (with $b_i \in \bar{S}$ and $a_i \in S$) in $G(S)$ is called a *shortest augmenting path*. We can *augment* S along the path p to obtain $S' = S \oplus p = S + b_1 - a_1 + b_2 - a_2 \ldots + b_{\ell+1}$, which is well-known to also be a common independent set (with $|S'| = |S| + 1$) [5]. Conversely, there must exist a shortest augmenting path whenever $|S| < r$.

The following lemma is very useful for $(1-\varepsilon)$-approximation algorithms since it essentially says that one needs only to consider paths up to length $O(\frac{1}{\varepsilon})$.

[6] Usually denoted as the *diminishing returns* property of submodular functions.

▶ **Lemma 10** (Cunningham [5]). *If the length of the shortest (s,t)-path in $G(S)$ is at least $2\ell+2$, then $|S| \geq (1-O(1/\ell))r$.*

▶ **Lemma 11** (Exchange discovery by binary search [3, 11]). *Suppose $\mathcal{M}=(V,\mathcal{I})$ is a matroid, $Y \subseteq X \in \mathcal{I}$, and $b \notin X$ such that $X+b \notin \mathcal{I}$. Then, using $O(\log|Y|)$ independence queries one can find some $a \in Y$ such that $X+b-a \in \mathcal{I}$ or determine that none exist.*[7]

Augmenting Sets

A generalization of the classical *augmenting paths* – called *augmenting sets* – play a key role in the approximation algorithm of [3], and therefore also in the modified version of this algorithm presented in this paper. In order to efficiently find "good" augmenting sets, the algorithm works with a relaxed form of them instead: *partial* augmenting sets. The following definitions and key properties of (partial) augmenting sets are copied from [3] where one can find the corresponding proofs.

▶ **Definition 12** (Augmenting Sets, from [3, Definition 24]). Let $S \in \mathcal{I}_1 \cap \mathcal{I}_2$ and $G(S)$ be the corresponding exchange graph with shortest (s,t)-path of length $2(\ell+1)$ and distance layers $D_1, D_2, \ldots, D_{2\ell+1}$. A collection of sets $\Pi_\ell := (B_1, A_1, B_2, A_2, \ldots, A_\ell, B_{\ell+1})$ form an *augmenting set* (of *width* w) in $G(S)$ if the following conditions are satisfied:
(a) For $1 \leq k \leq \ell+1$, we have $A_k \subseteq D_{2k}$ and $B_k \subseteq D_{2k-1}$.
(b) $|B_1| = |A_1| = |B_2| = \cdots = |B_{\ell+1}| = w$
(c) $S + B_1 \in \mathcal{I}_1$
(d) $S + B_{\ell+1} \in \mathcal{I}_2$
(e) For all $1 \leq k \leq \ell$, we have $S - A_k + B_{k+1} \in \mathcal{I}_1$
(f) For all $1 \leq k \leq \ell$, we have $S - A_k + B_k \in \mathcal{I}_2$

▶ **Definition 13** (Partial Augmenting Sets, from [3, Definition 37]). We say that $\Phi_\ell := (B_1, A_1, B_2, A_2, \ldots, A_\ell, B_{\ell+1})$ forms a *partial augmenting set* if it satisfies the conditions (a), (c), (d), and (e) of an *augmenting set*, plus the following two relaxed conditions:
(b) $|B_1| \geq |A_1| \geq |B_2| \geq \cdots \geq |B_{\ell+1}|$.
(f) For all $1 \leq k \leq \ell$, we have $\mathrm{rk}_2(S - A_k + B_k) = \mathrm{rk}_2(S)$.

▶ **Theorem 14** (from [3, Theorem 25]). *Let $\Pi_\ell := (B_1, A_1, B_2, A_2, \cdots, B_\ell, A_\ell, B_{\ell+1})$ be the an augmenting set in the exchange graph $G(S)$. Then the set $S' := S \oplus \Pi_\ell := S + B_1 - A_1 + B_2 - \cdots + B_\ell - A_\ell + B_{\ell+1}$ is a common independent set.*[8]

We also need the notion of *maximal* augmenting sets, which naturally correspond to a maximal ordered collection of shortest augmenting paths, where, after augmentation, the (s,t)-distance must have increased. The following are due to [3].

▶ **Definition 15** (Maximal Augmenting Sets, from [3, Definition 35]). Let $\Pi_\ell = (B_1, A_1, B_2, \cdots, B_\ell, A_\ell, B_{\ell+1})$ and $\tilde{\Pi}_\ell = (\tilde{B}_1, \tilde{A}_1, \tilde{B}_2, \cdots, \tilde{B}_\ell, \tilde{A}_\ell, \tilde{B}_{\ell+1})$ be two augmenting sets in $G(S)$. We say $\tilde{\Pi}_\ell$ *contains* Π_ℓ if $B_k \subseteq \tilde{B}_k$ and $A_k \subseteq \tilde{A}_k$, for all k. An augmenting set Π_ℓ is called *maximal* if there exists no other augmenting set $\tilde{\Pi}_\ell$ containing Π_ℓ.

▶ **Theorem 16** (from [3, Theorem 36]). *An augmenting set Π_ℓ is maximal if and only if there is no augmenting path of length at most $2(\ell+1)$ in $G(S \oplus \Pi_\ell)$.*

[7] When $X = S$, we can use this to find edges of type 3 and 4 in the exchange graph.
[8] Note that $|S'| = |S| + w$, where w is the width of Π_ℓ. In particular, an augmenting set with width $w = 1$ is exactly an augmenting path.

3 Improved Approximation Algorithm

Our algorithm closely follows the algorithm of Chakrabarty-Lee-Sidford-Singla-Wong [3, Section 6]. The algorithm runs in phases, where in each phase the algorithm finds a maximal set of augmentations to perform, so that the (s,t)-distance in the exchange graph increases between phases. By Lemma 10, only $O(1/\varepsilon)$ phases are necessary.

In the beginning of a phase, the algorithm runs a breadth-first-search to compute the distance layers $D_1, D_2, \ldots D_{2\ell+1}$ in the exchange graph $G(S)$, where S is the current common independent set. The total number of independence queries, across all phases, for these BFS's can be bounded by $O(n\log(r)/\varepsilon)$. We refer to [3, Algorithm 4, Lemma 19, and Proof of Theorem 21] for how to implement such a BFS efficiently.

After the distance layers have been found, the search for a maximal augmenting set begins. We start by summarizing on a high level how the algorithm of [3] does this in two stages:

1. The first stage keeps track of a *partial* augmenting set which it keeps *refining* by a series of operations on adjacent distance layers in the exchange graph, to make it closer to a *maximal* augmenting set.
2. When we are "close enough" to a *maximum* augmenting set, the second stage handles the last few augmenting paths – for which the first stage slows down – by finding the remaining augmenting paths individually one at a time.

Here we give two independent improvements over the algorithm of [3], one for each stage. The first improvement is to replace the refine operations in the first stage by a new subroutine `RefineABA` (Section 3.1.2) working on *three* consecutive layers instead of two. This allows us to measure progress in terms of r instead of n. The second improvement is for the second stage where we, instead of finding arbitrary augmenting paths, work directly on top of the output of the first stage and find a specific type of *valid paths* with respect to the partial augmenting set, using a new a subroutine `RefinePath` (Section 3.2).

3.1 Implementing a Phase: Refining

The basic refining ideas and procedures in this section are the same as in [3]. The goal is to keep track of a partial augmenting set $\Phi_\ell = (B_1, A_1, B_2, \ldots, A_\ell, B_{\ell+1})$ which is iteratively made "better" through some *refine procedures*. Eventually, the partial augmenting set will become a maximal augmenting set, which concludes the phase. Towards this goal, we maintain three types of elements in each layer:

Selected. Denoted by A_k or B_k. These form the partial augmenting set $\Phi_\ell = (B_1, A_1, B_2, \ldots, A_\ell, B_{\ell+1})$.

Removed. Denoted by R_k. These elements are safe to disregard from further computation (i.e. deemed useless) when refining Φ_ℓ towards a maximal augmenting set.

Fresh. Denoted by F_k. These are the elements that are neither *selected* nor *removed*.

Elements can change their types from *fresh* $\to$ *selected* $\to$ *removed*, but never in the other direction. Initially, we start with all elements being fresh.[9] For convenience, we also define "imaginary" layers D_0 and $D_{2\ell+2}$ with $A_0 = R_0 = F_0 = D_0 = A_{\ell+1} = R_{2\ell+2} = F_{2\ell+2} = D_{2\ell+2} = \emptyset$. The algorithm maintains the following *phase invariants* (which are initially satisfied) during the refinement process:

[9] This differs slightly from [3], where the initially B_1 is greedily picked to be maximal so that $S + B_1 \in \mathcal{I}_1$, while the rest of the elements are fresh.

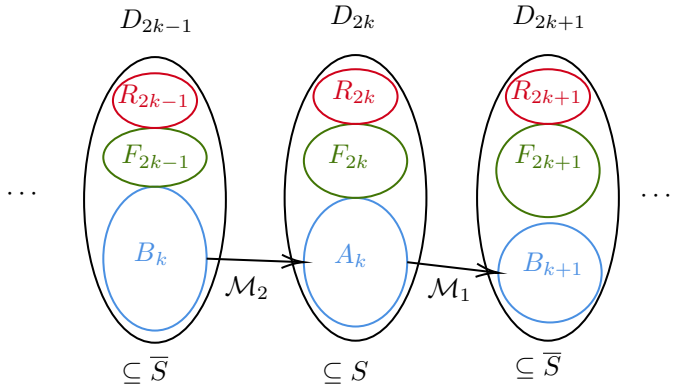


Figure 1 An illustration of a few neighboring layers. Note that $(B_k, R_{2k-1}, F_{2k-1})$ form a partition of odd layer $D_{2k-1} \subseteq \bar{S}$, and (A_k, R_{2k}, F_{2k}) form a partition of even layer $D_{2k} \subseteq S$.

▶ **Definition 17** (Phase Invariants, from [3, Section 6.3.2])**.** The *phase invariants* are:

(a-b) $\Phi_\ell = (B_1, A_1, B_2, \ldots, A_\ell, B_{\ell+1})$ forms a partial augmenting set.[10]

(c) For $1 \leq k \leq \ell$, for any $X \subseteq B_{k+1} + F_{2k+1} = D_{2k+1} - R_{2k+1}$, if $S - (A_k + R_{2k}) + X \in \mathcal{I}_1$ then $S - A_k + X \in \mathcal{I}_1$. [11]

(d) $\mathrm{rk}_2(W + R_{2k-1}) = \mathrm{rk}_2(W)$ where $W = S - (D_{2k} - R_{2k}) + B_k$.

▶ Remark 18. Invariant (c) essentially says that if R_{2k+1} is "useless", then so is R_{2k}. Similarly, Invariant (d) says that if R_{2k} is "useless", then so is R_{2k-1}. Together they imply that we can safely ignore all the removed elements.

▶ **Lemma 19.** *Suppose that (i) the phase invariants hold; (ii)* $|B_1| = |A_1| = \cdots = |B_{\ell+1}|$*; and (iii)* B_1 *is a maximal subset of* $D_1 \setminus R_1$ *satisfying* $S + B_1 \in \mathcal{I}_1$*. Then* $(B_1, A_1, \ldots, B_{\ell+1})$ *is a maximal augmenting set.*

Proof idea. (See [3, Proof of Lemma 44] for a complete proof). If it was not maximal, there exists an augmenting path $(b_1, a_1, \ldots, b_{\ell+1})$ in the exchange graph after augmenting along $(B_1, A_1, \ldots, B_{\ell+1})$. However, (iii) then says that b_1 must have been removed since it cannot be fresh. But if b_1 is removed, then so was a_1, then so was b_2 etc., by invariants (c) and (d) (this requires a technical, but straightforward, argument). However, $b_{\ell+1}$ cannot have been removed (by invariant (d)), which gives the desired contradiction. ◀

3.1.1 Refining Two Adjacent Layers

We now present the basic refinement procedures from [3], which are operations on neighboring layers. There is some asymmetry in how (odd, even) and (even, odd) layer-pairs are handled, arising from the inherent asymmetry of the independence query between S and $\bar{S}$, but the ideas are the same.

`RefineAB(`k`)` extends B_{k+1} as much as possible while respecting invariant (a-b) (Lines 1-2). Then a maximal collection of element in A_k which can be "matched" to B_{k+1} is found, and the others elements in A_k are removed (Lines 3-4).

[10] The naming of this invariant as (a-b) is to be consistent with [3] where this condition is split up into two separate items (a) and (b).

[11] An equivalent condition for (c) is: $\mathrm{rk}_1(W - R_{2k}) = \mathrm{rk}_1(W) - |R_{2k}|$, where $W = S - A_k + (D_{2k+1} - R_{2k+1})$.

`RefineBA(k)` finds a maximal subset B_k that can be "matched" to $A_k + F_{2k}$, and removes the other elements of B_k (Lines 1-2). Then A_k is extended with elements from F_{2k} which are the endpoints of the above "matching" (Lines 3-4).

Algorithm 1 `RefineAB`(k). (called `Refine1` in [3, Algorithm 9])

1: Find maximal $B \subseteq F_{2k+1}$ s.t. $S - A_k + B_{k+1} + B \in \mathcal{I}_1$
2: $B_{k+1} \longleftarrow B_{k+1} + B, F_{2k+1} \longleftarrow F_{2k+1} - B$
3: Find maximal $A \subseteq A_k$ s.t. $S - A_k + B_{k+1} + A \in \mathcal{I}_1$
4: $A_k \longleftarrow A_k - A, R_{2k} \longleftarrow R_{2k} + A$

Algorithm 2 `RefineBA`(k). (called `Refine2` in [3, Algorithm 10])

1: Find maximal $B \subseteq B_k$ s.t. $S - (D_{2k} - R_{2k}) + B \in \mathcal{I}_2$
2: $R_{2k-1} \longleftarrow R_{2k-1} + B_k \backslash B, B_k \longleftarrow B$
3: Find maximal $A \subseteq F_{2k}$ s.t. $S - (D_{2k} - R_{2k}) + B_k + A \in \mathcal{I}_2$
4: $A_k \longleftarrow A_k + F_{2k} \backslash A, F_{2k} \longleftarrow A$

The following properties of the `RefineAB` and `RefineBA` methods are proven in [3].

▶ **Lemma 20** (from [3, Lemmas 40-42]). *Both `RefineAB` and `RefineBA` preserve the invariants. Also: after `RefineAB`(k) is run, we have $|A_k| = |B_{k+1}|$ (unless $k = 0$). After `RefineBA`(k) is run, we have $|B_k| = |A_k|$ (unless $k = \ell + 1$).*

▶ **Lemma 21** (from [3, Lemma 45]). *`RefineAB` can be implemented with $O(|D_{2k}| + |D_{2k+1}|)$ queries. `RefineBA` can be implemented with $O(|D_{2k-1}| + |D_{2k}|)$ queries.*

▶ **Observation 22.** *Lemma 20 is particularly interesting. It says that at least $|A_k^{old}| - |B_{k+1}^{old}|$ (respectively $|B_k^{old}| - |A_k^{old}|$) elements change type when running `RefineAB` (respectively `RefineAB`).*

▶ Remark 23. Observation 22 is used in [3] to bound the number of times one needs to refine the partial augmenting set. Indeed, every element can only change its type a constant number of times. In a single refinement pass, procedures `RefineAB`(k) and `RefineBA`(k) are called for all k, and we obtain a telescoping sum guaranteeing us that $|B_1^{old}| - |B_{\ell+1}^{old}|$ elements have changed their types. Hence, after $O(\sqrt{n})$ refinement passes we have $|B_1| - |B_{\ell+1}| \leq \sqrt{n}$, and we are "close" to having a maximal augmenting set – only around $\sqrt{n}$ many augmenting paths away. This is essentially what lets [3] obtain their subquadratic $\tilde{O}(n^{1.5}/\text{poly}(\varepsilon))$ algorithm.

3.1.2 Refining Three Adjacent Layers

We are now ready to present the new `RefineABA` method (Algorithm 3), which is **not** present in [3]. This method works similarly to `RefineAB` and `RefineBA`, but on **three** (instead of two) consecutive layers $(D_{2k}, D_{2k+1}, D_{2k+2})$ with the corresponding sets (A_k, B_{k+1}, A_{k+1}).

The motivation for this new procedure is that we can get a stronger version of Observation 22: after running `RefineABA`(k) we want that at least $|A_k^{old}| - |A_{k+1}^{old}|$ element in **even** layers have changed types. Note that there are at most $|S| \leq r$ elements in the even layers (as opposed to n elements in total, which can be much larger), so this means we need to refine the partial augmenting set fewer times when using `RefineABA` compared to when just using `RefineAB` and `RefineBA`. In particular, we will get that after $O(\sqrt{r})$ refinement passes, $|B_1| - |B_{\ell+1}| \leq \sqrt{r}$.

▶ Remark 24. A natural question to ask is if it actually could be the case that only elements in odd layers (i.e. those in $\bar{S}$ which there are up to n many of) change their type (while elements in even layers do not) during the refinement passes in the algorithm of [3] (which only uses the two-layer refinement procedures)? That is, is the new three-layer refinement procedure necessary? The answer is yes. Consider for example the case with 5 layers $B_1 \subseteq D_1; A_1 \subseteq D_2; B_2 \subseteq D_3; A_2 \subseteq D_4; B_3 \subseteq D_5$ where $q := |B_1| = |A_1|$ and $|A_2| = |B_3| = 0$. Refining the consecutive pair (B_1, A_1) or (A_2, B_3) will not do anything. When refining (A_1, B_2) it could be the case that only B_2 increases (say any q-size subset in D_3 can be "matched" with A_1). Similarly, when refining (B_2, A_2) it could be the case that only B_2 decreases (say there is only a single element in D_3 which could be "matched" with anything in the next layer D_4, then it is unlikely that this specific element is already selected in B_2). In this case, we would need to run the two-layer refinement procedures around $|D_3|/q \approx n/q$ times before anything other than B_2 changes. In contrast, the new `RefineABA` method would, when run on (A_1, B_2, A_2), terminate with $|A_1| = |B_2| = |A_2|$ (that is it would have found the "special" element in D_3 the first time it is run).

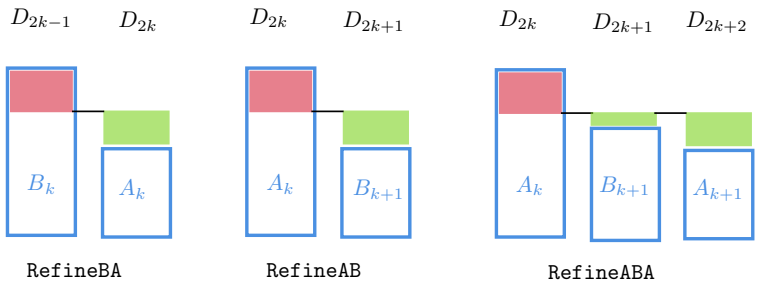


Figure 2 An illustration how the different refine methods change the partial augmenting sets. Newly selected elements are marked in green, while newly removed elements are marked in red.

To explain how `RefineABA` works, let us start with a simple case, namely when $S = \emptyset$, i.e. there is only one layer between s and t in the exchange graph. Here, finding a maximal augmenting set is the same as finding some maximal set B which is independent in both matroids. Running `RefineAB` would extend this B with elements as long as it is independent in the first matroid (ignoring the second matroid), while `RefineBA` would throw away elements from B until it is independent in the second matroid (now ignoring the first matroid). If we just alternate running `RefineAB` and `RefineBA` we would in the worst case need to do this up to n times (which is too expensive). Instead, there is a very simple greedy algorithm that efficiently finds a maximal set B independent in both of the matroids[12]: *for each element, include it in B if this does not break independence for either matroid.* This is akin to how our `RefineABA` method works: it looks at the constraints from both matroids simultaneously (both neighboring layers) and greedily selects B.

In the general case, `RefineABA` can be seen as running `RefineAB` and `RefineBA` simultaneously. The algorithm starts by asserting $|B_{k+1}| = |A_{k+1}|$ (so that $S + B_{k+1} - A_{k+1} \in \mathcal{I}_2$) by running `RefineBA`. So now we have both $S + B_{k+1} - A_k \in \mathcal{I}_1$ and $S + B_{k+1} - A_{k+1} \in \mathcal{I}_2$, and the algorithm proceeds to greedily extend B_{k+1} while it is still consistent with both the previous layer A_k and the next layer $A_{k+1} + F_{2k+2}$. Some care has to be taken here to also mark elements as removed to preserve the phase invariants. Finally, the algorithm decreases the size of A_k, respectively increases the size of A_{k+1}, to both match $|B_{k+1}|$.

[12] This algorithm on its own is a well-known $\frac{1}{2}$-approximation algorithm for matroid intersection.

Algorithm 3 `RefineABA`(k).

```
1: RefineBA(k + 1)
2: for x ∈ F_{2k+1} do
3:     if S − A_k + B_{k+1} + x ∈ I_1 then
4:         if S − A_{k+1} − F_{2k+2} + B_{k+1} + x ∈ I_2 then
5:             B_{k+1} ← B_{k+1} + x,   F_{2k+1} ← F_{2k+1} − x        ▷ Select x
6:         else
7:             R_{2k+1} ← R_{2k+1} + x,   F_{2k+1} ← F_{2k+1} − x      ▷ Remove x
8: RefineBA(k + 1)
9: RefineAB(k)
```

We now state some properties of `RefineABA`. These properties are relatively straightforward, although technical and notation-heavy, to prove.

▶ **Lemma 25.** *`RefineABA`(k) preserves the phase invariants.*

▶ **Lemma 26.** *After `RefineABA`(k) is run, we have $|A_k| = |B_{k+1}| = |A_{k+1}|$ (unless $k = 0$ or $k = \ell$, where the sets $A_0 = A_{\ell+1} = \emptyset$ are "imaginiary").*

▶ **Lemma 27.** *`RefineABA`(k) uses $O(|D_{2k}| + |D_{2k+1}| + |D_{2k+2}|)$ independence queries.*

Proof of Lemma 25. Intuitively, the only tricky part is showing that invariant (c) is preserved when some x is removed in line 7. We can pretend that we add x to B_{k+1} temporarily, and then run `RefineBA`$(k+1)$ in a way which would remove this x immediately (and thus removing x did indeed preserve the invariants). We present a formal proof below.

We already know that `RefineAB` and `RefineBA` preserve the invariants by Lemma 20, so it suffices to check that the for-loop starting in line 2 preserves the invariants. We verify that this is the case after processing each $x \in F_{2k+1}$ in the for-loop:

Invariant (a-b) holds by design: when x is added to B_{k+1} we know both that $S - A_k + B_{k+1} + x \in \mathcal{I}_1$ and $\mathrm{rk}_2(S - A_{k+1} + B_{k+1})$ cannot decrease. Note also that $\mathrm{rk}_2(S - A_{k+1} + B_{k+1}) \leq \mathrm{rk}_2(S)$ when $k + 1 \leq \ell$ too (so it cannot increase either), since otherwise there must exist some $b \in B_{k+1}$ so that $S + b \in \mathcal{I}_2$ (by the matroid exchange property) which is impossible since we are not in the last layer (the layer preceding t in $G(S)$).

Invariant (c) trivially holds, since the set $B_{k+1} + F_{2k+1}$ will only decrease, which only restricts the choice of $X \subseteq B_{k+1} + F_{2k+1}$.

Invariant (d) will also be preserved. We need to argue that this is the case when x is removed in line 7. Let $W := S - A_{k+1} - F_{2k+2} + B_{k+1} = S - (D_{2k+2} - R_{2k+2}) + B_{k+1}$, and R^{old}_{2k+1} be the set R_{2k+1} before x was added to it. First note that $W \in \mathcal{I}_2$, since this holds after the `RefineBA` call in line 1, (since $|A_{k+1}| = |B_{k+1}|$ after this call) and B_{k+1} is only extended with elements which preserve this property. This means that $\mathrm{rk}_2(W + x) = \mathrm{rk}_2(W) = |W|$, since $W + x = S - A_{k+1} - F_{2k+2} + B_{k+1} + x \notin \mathcal{I}_2$. Since the invariant held before, we also know that $\mathrm{rk}_2(W + R^{old}_{2k+1}) = \mathrm{rk}_2(W) = |W|$. Hence W is a maximal independent (in $\mathcal{M}_2$) subset of $W + R^{old}_{2k+1} + x$, as neither x nor elements from R^{old}_{2k+1} can be used to extend it. Hence $\mathrm{rk}_2(W + R^{old}_{2k+1} + x) = |W| = \mathrm{rk}_2(W)$; that is invariant (d) is preserved. ◀

Proof of Lemma 26. We focus our attention on the `RefineBA` and `RefineAB` calls in lines 8-9, and argue that they do not change B_{k+1}. This would prove the lemma, since by Lemma 20 we would then have $|A_k| = |B_{k+1}|$ and $|B_{k+1}| = |A_{k+1}|$.

Indeed, `RefineBA`$(k+1)$ finds a maximal $B \subseteq B_{k+1}$ such that $S-(D_{2k+2}-R_{2k+2})+B \subseteq \mathcal{I}_2$, and remove all elements not in B from B_{k+1}. Here, $B = B_{k+1}$ will be found, since $S-(D_{2k+2}-R_{2k+2})+B_{k+1} \in \mathcal{I}_2$ after the for-loop in line 2 of `RefineABA`.

Similarly, we see that `RefineAB`(k) finds a maximal $B \subseteq F_{2k+1}$ such that $S-A_k+B_{k+1}+B \in \mathcal{I}_1$, and extend B_{k+1} with this B. However, only $B = \emptyset$ works, since each $x \in F_{2k+1}$ for which $S-A_k+B_{k+1}+x \in \mathcal{I}_1$ was either selected or removed in lines 5 or 7. ◀

Proof of Lemma 27. `RefineAB`(k) uses $O(|D_{2k}|+|D_{2k+1}|)$ queries, and `RefineBA`$(k+1)$ uses $O(|D_{2k+1}|+|D_{2k+2}|)$ queries. The for-loop in line 2 will use $O(|D_{2k+1}|)$ queries. ◀

3.1.3 Refinement Pass

We can now present the full `Refine` method (Algorithm 4), which simply scans over the layers and calls `RefineABA` on them. Our `Refine` is a modified version of `Refine` from [3, Algorithm 11] using our new `RefineABA` method instead of just `RefineAB` and `RefineBA`. Just replacing the `Refine` method in the final algorithm of [3] with our modified `Refine` below leads to an $\tilde{O}(n\sqrt{r}/\varepsilon^{1.5})$-query algorithm (compared to their $\tilde{O}(n^{1.5}/\varepsilon^{1.5})$), and concludes our first improvement (as discussed in Item 1 in Section 1.1).

Algorithm 4 `Refine`(k).

1: **for** $k = \ell,\ \ell-1,\ \ell-2,\ \ldots,\ 1,\ 0$ **do**
2: `RefineABA`(k)

The following Lemma 28 will be useful to bound the number of `Refine` calls needed in our final algorithm, and closely corresponds to [3, Corollary 43]. Our `Refine` implementation has the advantage that it only counts the elements in the **even** layers, of which there are at most r.

▶ **Lemma 28.** *Let $(B_1^{old}, A_1^{old}, \ldots)$ and $(B_1^{new}, A_1^{new}, \ldots)$ be the sets before and after `Refine` is run. Then at least $|B_1^{new}| - |B_{\ell+1}^{new}|$ elements in even layers have changed types.*

Proof. Note that whenever A_k changes, it is because some elements changed it types in D_{2k}. In particular, if the size of A_k increases (respectively decreases) by z, at least z elements will change types from fresh to selected (respectively from selected to removed) in D_{2k}.

After the first iteration $|A_\ell| = |B_{\ell+1}^{new}|$, so at least $|A_\ell^{old}| - |B_{\ell+1}^{new}|$ elements in $D_{2\ell}$ changed types. Similarly, after the iteration when $k = i$ (for $1 \le i \le \ell-1$), $|A_i| = |A_{i+1}|$, and hence at least $|A_i^{old}| - |A_i|$ elements in D_{2i} changed types plus at least $|A_{i+1}| - |A_{i+1}^{old}|$ elements in D_{2i+2} changed types.[13] Finally, after the last iteration $|A_1| = |B_1^{new}|$, and hence at least $|B_1^{new}| - |A_1^{old}|$ elements in D_2 changed types.

The above terms telescope, and we conclude that at least $|B_1^{new}| - |B_{\ell+1}^{new}|$ elements in the even layers changed its types when `Refine` was run. ◀

▶ **Lemma 29.** *`Refine` uses $O(n)$ independence queries.*

Proof. This follows directly by Lemma 27. ◀

[13] $|A_{i+1}| \le |A_{i+1}^{old}|$ just before the `RefineABA`(i) call, since earlier iterations can only have decreased the size of $|A_{i+1}|$.

3.2 Refining Along a Path

If we just run `Refine` until we get a maximal augment set (i.e. until $|B_1| = |B_{\ell+1}|$) we need to potentially run `Refine` as many as $\Theta(r)$ times, which needs too many independence queries. Lemma 28 tells us that `Refine` makes the most "progress" while $|B_1| - |B_{\ell+1}|$ is large: in fact, only $O(r/p)$ calls to `Refine` is needed until $|B_1| - |B_{\ell+1}| \leq p$. The idea in [3] is thus to stop refining when $|B_1| - |B_{\ell+1}|$ is small enough and fall back to finding augmenting paths one at a time (they prove that one needs to find at most $O((|B_1| - |B_{\ell+1}|)\ell)$ many). We use a similar idea in that we swap to a different procedure when $|B_1| - |B_{\ell+1}|$ is small enough, the difference being that we still work with the partial augmenting set. This will let us show that only $O(|B_1| - |B_{\ell+1}|)$ many "paths" need to be found, saving a factor $\ell \approx \frac{1}{\varepsilon}$ compared to [3].

This section thus describes the second improvement (as discussed in Item 2 in Section 1.1). Note that this improvement is independent of the first improvement (i.e. the three-layer refine). We aim to prove the following lemma.

▶ **Lemma 30.** *There exists a procedure (`RefinePath`, Algorithm 5), which uses $O(n \log r)$ independence queries, preserves the invariants, and either:*

(i) *Increases the size of $B_{\ell+1}$ by at least 1.*

(ii) *Terminates with $(B_1, A_1, \ldots, B_{\ell+1})$ being a maximal augmenting set.*

`RefinePath` attempts to find what we call a *valid path*. What we want is a sequence of elements which we can add to the partial augmenting set without violating the invariants and the properties of the partial augmenting set. It turns out (not very surprisingly) that such sequences of elements can be characterized by a notion of *paths* in something which resembles the *exchange graph with respect to our partial augmenting set*. This is what motivates the definition of *valid paths* below.

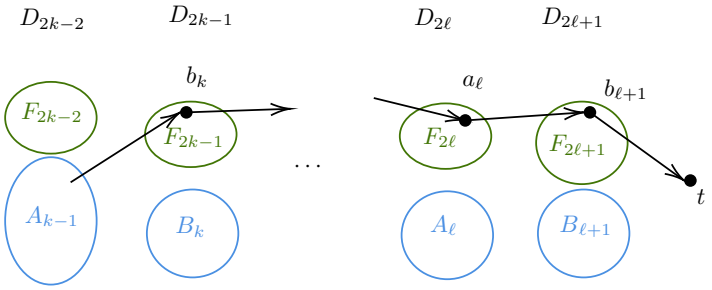


Figure 3 A valid path $(b_k, \ldots, a_\ell, b_{\ell+1}, t)$ "starting" from the partial augmenting set at A_{k-1}, so that we can use Lemma 33 and augment along it.

▶ **Definition 31** (Valid path)**.** A sequence $(b_i, a_i, b_{i+1}, \ldots, b_{\ell+1}, t)$ (or $(a_i, b_{i+1}, \ldots, b_{\ell+1}, t)$) is called a *valid path* (with respect to the partial augmenting set) if for all $k \geq i$:

(a) $a_k \in F_{2k}$ and $b_k \in F_{2k-1}$.

(b) $S + B_{\ell+1} + b_{\ell+1} \in \mathcal{I}_2$.

(c) $S - A_k + B_k - a_k + b_k \in \mathcal{I}_2$.

(d) $S - A_k + B_{k+1} - a_k + b_{k+1} \in \mathcal{I}_1$.

▶ Remark 32. Compare the properties of valid paths with the edges in the exchange graph from Definition 8. A valid path is essentially a path in the exchange graph *after* we have already augmented S by our partial augmenting set (even though this exchange graph is not exactly defined, since it is not guaranteed that S remains a common independent set when augmented by a *partial* augmenting set).

▶ **Lemma 33.** *If $p = (b_i, a_i, b_{i+1}, \ldots, b_{\ell+1}, t)$ is a valid path starting at b_i, such that $S - A_{i-1} + B_i + b_i \in \mathcal{I}_1$, then $(B_1, A_1, \ldots, B_{i-1}, A_{i-1}, B_i + b_i, A_i + a_i, \ldots, B_{\ell+1} + b_\ell)$ is a partial augmenting set satisfying the invariants.*

Proof. That it forms a partial augmenting set is true by the definition of valid paths, and the fact that $S - A_{i-1} + B_i + b_i \in \mathcal{I}_1$. Indeed, it cannot be the case that $|A_{i-1}| < |B_i + b_i|$ when $i > 1$, since then $\text{rk}_1(S - A_{i-1} + B_i + b_i) > |S| = \text{rk}_1(S)$ implies that some element $x \in (B_i + b_i)$ satisfies $S + x \in \mathcal{I}_1$ (i.e. it is in the first layer D_1) by the exchange property of matroids. Invariants (c) and (d) are trivially true since the sets A_k and B_k are only extended. ◀

The goal of `RefinePath` (Algorithm 5) is thus to find a valid path satisfying the conditions in Lemma 33. Towards this goal, `RefinePath` will start from the last layer $D_{2\ell+1}$ and "scan left" in a breadth-first-search manner while keeping track of valid paths starting at each fresh vertex x (the next element on such a path will be stored as $\texttt{next}[x]$). If at some point one valid path can "enter" the partial augmenting set in a layer, we are done and can use Lemma 33. We also show that it is safe (i.e. preserves the invariants) to remove all the fresh elements x for which we cannot find a valid path starting at x.

To efficiently find the "edges" during our breadth-first-search using only independence-queries, we use the binary-search trick from Lemma 11. However, this relies on the partial augmenting set being locally "flat" in the layers we are currently exploring, i.e. $|B_k| = |A_k|$ respectively $|B_k| = |A_{k+1}|$. We can ensure this by running `RefineAB` respectively `RefineBA` while performing the scan.

Now we are ready to present the pseudo-code of the `RefinePath` method (Algorithm 5). Due to the asymmetry between even/odd layers and independence queries, we need to handle moving from layer B to A and from A to B a bit differently, but the ideas are similar.

▶ **Lemma 34.** *`RefinePath` preserves the invariants.*

Proof. The proof is relatively straightforward, but technical. The only non-trivial part is showing that invariants (c) and (d) are preserved after we remove something in line 8 or line 20. Intuitively, if we remove b in line 8, we can instead think of temporarily adding b to B_k and running $\texttt{RefineBA}(k)$ in such a way so that b is immediately removed. A similar intuitive argument works for line 20. We next present a formal proof.

We know that `RefineAB` and `RefineBA` preserve the invariants, by Lemma 20. We also know by Lemma 33 that adding a valid path to the partial augmenting set also preserves the invariants. So what remains is to show that the invariants are preserved after:

Line 8. We only need to check invariant (d), the other ones trivially hold. Let $W = S - A_k - F_{2k} + B_k = S - (D_{2k} - R_{2k}) + B_k$ and R^{old}_{2k-1} be R_{2k-1} before b was added to it. Note that b is such that $W + b \notin \mathcal{I}_2$, and we know that $W \subseteq S - A_k + B_k \in \mathcal{I}_2$ and hence $\text{rk}_2(W + R^{old}_{2k-1}) = \text{rk}_2(W) = |W|$ and $\text{rk}_2(W + b) = \text{rk}_2(W) = |W|$. We thus need to show that $\text{rk}_2(W + R^{old}_{2k-1} + b) = |W|$ too, which is clear since W is a maximal independent subset of $W + R^{old}_{2k-1} + b$ (it can neither be extended with elements from R^{old}_{2k-1} nor with b).

Line 20. We only need to check invariant (c), the other ones trivially hold. We imagine we add the $a \in Q$ to R_{2k-2} one-by-one, and show that the invariant (c) is preserved after each such addition. So consider some $a \in Q$ which will be removed, and let R^{old}_{2k-2} be the set R_{2k-2} just before we added a to it. First note that $\text{rk}_1(S - A_{k-1} + B_k + F_{2k-1} - a) = \text{rk}_1(S - A_{k-1} + B_k + F_{2k-1}) - 1 = |S - A_{k-1} + B_k| - 1$, as otherwise there must exist some $b \in F_{2k-1}$ such that $S - A_{k-1} + B_k + b - a \in \mathcal{I}_1$ (by the matroid exchange property),

Algorithm 5 `RefinePath`.

1: **for** $k = \ell + 1,\ \ell,\ \ldots,\ 2,\ 1$ **do**
▷ Process (B_k, A_k)
2: `RefineBA`(k)
3: **if** some element a was added to A_k in the above refine-call **then**
4: Add the valid path starting at `next`$[a]$ to the partial augmenting set
5: **return**
6: **for** each element $b \in F_{2k-1}$ **do**
7: **if** $S - A_k - F_{2k} + B_k + b \notin \mathcal{I}_2$ **then**
8: Remove b, that is: $F_{2k-1} \leftarrow F_{2k-1} - b, \quad R_{2k-1} \leftarrow R_{2k-1} + b$
9: **else**
10: Find an $a \in F_{2k}$ such that $S - A_k + B_k + b - a \in \mathcal{I}_2$. Let `next`$[b] = a$.
11: (Or, if $k = \ell + 1$, just let `next[b]` $= t$)
▷ Process (A_{k-1}, B_k)
12: **if** some element $b \in F_{2k-1}$ satisfies $S - A_{k-1} + B_k + b \in \mathcal{I}_1$ **then**
13: Add the valid path starting at b to the partial augmenting set.
14: **return**
15: `RefineAB`$(k-1)$
16: $Q \leftarrow F_{2k-2}$.
17: **for** each element $b \in F_{2k-1}$ **do**
18: **while** can find $a \in Q$ such that $S - A_{k-1} + B_k + b - a \in \mathcal{I}_1$ **do**
19: $Q \leftarrow Q - a$. Let `next`$[a] = b$.
20: Remove all elements in Q, that is: $F_{2k-2} \leftarrow F_{2k-2} - Q, \quad R_{2k-2} \leftarrow R_{2k-2} + Q$.

21: If we reached here, $(B_1, A_1, \ldots, B_{\ell+1})$ is a maximal augmenting set.

and a would have been discovered in line 18 and therefore been removed from Q. So the "return" of adding a to $S - A_{k-1} + B_k + F_{2k-1} - a$ is increasing the rank by 1. Now consider some arbitrary $X \subseteq B_k + F_{2k-1}$ such that $S - A_{k-1} + X - R^{old}_{2k-2} - a \in \mathcal{I}_1$. We need to show that $S - A_{k-1} + X \in \mathcal{I}_1$. Note that $S - A_{k-1} + X - R^{old}_{2k-2} - a \subseteq S - A_{k-1} + B_k + F_{2k-1} - a$. Hence, by the diminishing returns (of adding a) we know $\mathrm{rk}_1(S - A_{k-1} + X - R^{old}_{2k-2}) \geq \mathrm{rk}_1(S - A_{k-1} + X - R^{old}_{2k-2} - a) + 1 = |S - A_{k-1} + X - R^{old}_{2k-2}|$, or equivalently that $S - A_{k-1} + X - R^{old}_{2k-2} \in \mathcal{I}_1$. Since the invariant held before, we conclude that $S - A_{k-1} + X \in \mathcal{I}_1$ too, which finishes the proof. ◀

Valid paths. The algorithm keeps track of a valid path starting at each fresh vertex it has processed. That is, after processing layer D_k, all elements in F_k must be the beginning of a valid path, else they were removed. In particular, the algorithm remembers the valid path starting at x as $(x, \texttt{next}[x], \texttt{next}[\texttt{next}[x]], \ldots)$. It is easy to verify that this sequence does indeed satisfy the conditions of *valid paths* by inspecting lines 10 and 18.

We also discuss what happens when the algorithm chooses to add a valid path to the partial augmenting set (i.e. in line 4 or 13). If we are in Line 13, we can directly apply Lemma 33. Say we instead are in Line 4, and some a which was previously fresh has been added to A_k. The `RefineBA` call can only have increased A_k (that is $A_k \supseteq A^{old}_k + a$), so $S - A_k + B_{k+1} + b \in \mathcal{I}_1$ will holds for $b = \texttt{next}[a]$ and we can apply Lemma 33 here too.

When no path is found. In the case when no valid path to add to the partial augmenting set is found, `RefinePath` must terminate with $|B_1| = |A_1| = \cdots = |B_{\ell+1}|$. This is because the `RefineAB` and `RefineBA` will never select any new elements. That is `RefineBA` will not change A_k (as otherwise we enter the if-statement at line 4), and `RefineAB` will not change B_k (since if $b \in F_{2k-1}$ with $S - A_{k-1} + B_k + b \in \mathcal{I}_1$ existed we would have entered the if-statement at line 13). We also remark that `RefinePath` ends with B_1 being a maximal subset of $D_1 \setminus R_1$, as otherwise some b would have been found in line 12. Hence Lemma 19 implies that $(B_1, A_1, \ldots, B_{\ell+1})$ now forms a *maximal* augmenting set.

Query complexity. The `RefineAB` and `RefineBA` calls will in total use $O(n)$ queries. The independence checks at Lines 7 and 12 happens at most once for each element, and thus use $O(n)$ queries in total. Lines 10 and 18 can be implemented using the binary-search-exchange-discovery Lemma 11. Hence Line 10 will use, in total, $O(n \log r)$ queries and Line 18 will use, in total, $O(n \log r)$ queries (since each $a \in Q$ will be discovered at most once). So we conclude that Algorithm 5 uses $O(n \log r)$ independence queries.

3.3 Hybrid Algorithm

Now we are finally ready to present the full algorithm of a phase, which is parameterized by a variable p. The following algorithm is similar to that of [3, Algorithm 12] but uses our improved `Refine` method and finds individual paths using the `RefinePath` method.

Algorithm 6 Phase ℓ.

1: Calculate the distance layers by a BFS.
2: Run `Refine` (Algorithm 4) until $|B_1| - |B_{\ell+1}| \leq p$, but at least once.
3: Run `RefinePath` (Algorithm 5) until $(B_1, A_1, \ldots B_{\ell+1})$ is maximal. Augment along it.

▶ **Lemma 35.** *Except for line 1, Algorithm 6 uses $O(nr/p + np \log r)$ queries.*[14]

Proof. Lemma 28 tells us that `Refine` changes types of at least p elements in even layers (i.e. elements in S) every time it is run, except maybe the last time. Thus we only run `Refine` $O(|S|/p + 1)$ times. Each call takes $O(n)$ queries (Lemma 29), for a total of $O(nr/p)$ queries in line 2 of the algorithm.

Now we argue that B_1 can never become larger than what it was just after line 2 was run. This is because `Refine` will run at least once, and ends with a `RefineABA`(0) call which in turn ends with a `RefineAB(0)` call – which extends B_1 to be a maximal set in $D_1 \setminus R_1$ for which $S + B_1 \subseteq \mathcal{I}_1$ holds.[15]

Lemma 30 tells us that each (except the last) time `RefinePath` is run, $B_{\ell+1}$ increases by 1. This can happen at most p times, so line 3 uses a total of $O(np \log r)$ queries. ◀

Now it is easy to prove Theorem 1, which we restate below.

[14] Compare this to $O(n^2/p + np\ell \log r)$ in [3]. The improvement from n^2/p to nr/p comes from the use of the new three-layer `RefineABA` method, and the (independent) improvement from $np\ell \log r$ to $np \log r$ comes from the use of the new `RefinePath` method.

[15] Indeed, since $\mathcal{M}_1$ is a matroid, all such maximal sets have the same size, so we can never obtain something larger later.

▶ **Theorem 1** (Approximation algorithm). *There is a deterministic algorithm which given two matroids $\mathcal{M}_1 = (V, \mathcal{I}_1)$ and $\mathcal{M}_2 = (V, \mathcal{I}_2)$ on the same ground set V, finds a common independent set $S \in \mathcal{I}_1 \cap \mathcal{I}_2$ with $|S| \geq (1-\varepsilon)r$, using $O\left(\frac{n\sqrt{r \log r}}{\varepsilon}\right)$ independence queries.*

Proof. Pick $p = \sqrt{r/\log r}$.[16] Then each phase will use $O(n\sqrt{r \log r})$ independence queries (by Lemma 35), plus a total of $O(\frac{1}{\varepsilon} n \log r)$ to run the BFS's across all phases (see [3] for details on the BFS implementation). Since we need only run $O(\frac{1}{\varepsilon})$ phases (by Lemma 10 and Theorem 16), in total the algorithm will use $O(\frac{1}{\varepsilon} n\sqrt{r \log r})$ queries. ◀

4 Exact Matroid Intersection

In this section, we prove Theorem 2 (restated below) by showing how our improved approximation algorithm leads to an improved exact algorithm when combined with the algorithms of [2].

▶ **Theorem 2** (Exact algorithm). *There is a randomized algorithm which given two matroids $\mathcal{M}_1 = (V, \mathcal{I}_1)$ and $\mathcal{M}_2 = (V, \mathcal{I}_2)$ on the same ground set V, finds a common independent set $S \in \mathcal{I}_1 \cap \mathcal{I}_2$ of maximum cardinality r, and w.h.p.[17] uses $O(nr^{3/4} \log n)$ independence queries. There is also a deterministic exact algorithm using $O(nr^{5/6} \log n)$ queries.*

Approximation algorithms are great at finding the *many, very short* augmenting paths efficiently. Blikstad-v.d.Brand-Mukhopadhyay-Nanongkai [2, Algorithm 2] very recently showed how to efficiently find the remaining *few, very long* augmenting paths, with a randomized algorithm using $\tilde{O}(n\sqrt{r})$ queries per augmentation (or, with a slightly less efficient deterministic algorithm using $\tilde{O}(nr^{2/3})$ queries). In the randomized $\tilde{O}(n^{6/5}r^{3/5})$-query exact algorithm of [2, Algorithm 3], the current bottleneck is the approximation algorithm used. Replacing the use of the $\tilde{O}(n^{1.5}/\varepsilon^{1.5})$-query approximation algorithm from [3] with our improved version we obtain the more efficient randomized[18] $\tilde{O}(nr^{3/4})$-query Algorithm 7.

Algorithm 7 Exact Matroid Intersection. (Modified version of [2, Algorithm 3])

1: Run the approximation algorithm (Theorem 1) with $\varepsilon = r^{-1/4}$ to obtain a common independent set S of size at least $(1-\varepsilon)r = r - r^{3/4}$.
2: Starting with S, run Cunningham's algorithm (as implemented by [3]), until the distance between s and t becomes larger than $r^{3/4}$.
3: Keep finding augmenting paths – one at a time – to augment along, using the randomized $O(n\sqrt{r} \log n)$-query algorithm of [2, Algorithm 2]. When no (s,t)-path can be found in the exchange graph, S is a largest common independent set.

Query complexity. We analyse the individual lines of Algorithm 7.

Line 1. We see that the approximation algorithm uses $O(nr^{3/4} \log n)$ queries in line 1.

Line 2. One need to (i) compute distances up to $d = r^{3/4}$, and (ii) perform at most $O(r^{3/4})$ augmentations. [2, 3, 11] show how to do (i) in $O(nd \log n) = O(nr^{3/4} \log n)$ queries in total over all phases of Cunningham's algorithm, and how to do (ii) using $O(n \log n)$ queries per augmentation (for a total of $O(nr^{3/4} \log n)$ queries).

[16] Compare this to $p = \sqrt{n\varepsilon/\log r}$ in [3].
[17] *w.h.p. = with high probability* meaning with probability $1 - n^{-c}$ for some arbitrarily large constant c.
[18] The deterministic algorithm of Theorem 2 is obtained in the same fashion but by using the deterministic version of the augmenting path finding algorithm [2, Algorithm 2].

Line 3. By Lemma 10, line 3 runs $O(r^{1/4})$ times – each using $O(n\sqrt{r}\log n)$ queries – for a total of $O(nr^{3/4}\log n)$ queries.

▶ Remark 36. In Algorithm 7, the bottleneck between line 1-2 and line 2-3 now matches (which was not the case in [2]). This means that if one wants to improve the algorithm by replacing the subroutines in line 1 and 3, one need to **both** improve the approximation algorithm (line 1) and the method to find a single augmenting-path (line 3).

References

1 Martin Aigner and Thomas A. Dowling. Matching theory for combinatorial geometries. *Transactions of the American Mathematical Society*, 158(1):231–245, 1971.

2 Joakim Blikstad, Jan van den Brand, Sagnik Mukhopadhyay, and Danupon Nanongkai. Breaking the quadratic barrier for matroid intersection. In *STOC*. ACM, 2021.

3 Deeparnab Chakrabarty, Yin Tat Lee, Aaron Sidford, Sahil Singla, and Sam Chiu-wai Wong. Faster matroid intersection. In *FOCS*, pages 1146–1168. IEEE Computer Society, 2019.

4 Chandra Chekuri and Kent Quanrud. A fast approximation for maximum weight matroid intersection. In *SODA*, pages 445–457. SIAM, 2016.

5 William H. Cunningham. Improved bounds for matroid partition and intersection algorithms. *SIAM J. Comput.*, 15(4):948–957, 1986.

6 Jack Edmonds. Submodular functions, matroids, and certain polyhedra. In *Combinatorial structures and their applications*, pages 69–87. Gordon and Breach, 1970.

7 Jack Edmonds. Matroid intersection. In *Annals of discrete Mathematics*, volume 4, pages 39–49. Elsevier, 1979.

8 Jack Edmonds, GB Dantzig, AF Veinott, and M Jünger. Matroid partition. *50 Years of Integer Programming 1958–2008*, page 199, 1968.

9 Eugene L. Lawler. Matroid intersection algorithms. *Math. Program.*, 9(1):31–56, 1975.

10 Yin Tat Lee, Aaron Sidford, and Sam Chiu-wai Wong. A faster cutting plane method and its implications for combinatorial and convex optimization. In *FOCS*, pages 1049–1065. IEEE Computer Society, 2015.

11 Huy L. Nguyen. A note on cunningham's algorithm for matroid intersection. *CoRR*, abs/1904.04129, 2019. `arXiv:1904.04129`.

Graph Similarity and Homomorphism Densities

Jan Böker
RWTH Aachen University, Germany

Abstract

We introduce the tree distance, a new distance measure on graphs. The tree distance can be computed in polynomial time with standard methods from convex optimization. It is based on the notion of fractional isomorphism, a characterization based on a natural system of linear equations whose integer solutions correspond to graph isomorphism. By results of Tinhofer (1986, 1991) and Dvořák (2010), two graphs G and H are fractionally isomorphic if and only if, for every tree T, the number of homomorphisms from T to G equals the corresponding number from T to H, which means that the tree distance of G and H is zero. Our main result is that this correspondence between the equivalence relations "fractional isomorphism" and "equal tree homomorphism densities" can be extended to a correspondence between the associated distance measures. Our result is inspired by a similar result due to Lovász and Szegedy (2006) and Borgs, Chayes, Lovász, Sós, and Vesztergombi (2008) that connects the cut distance of graphs to their homomorphism densities (over all graphs), which is a fundamental theorem in the theory of graph limits. We also introduce the path distance of graphs and take the corresponding result of Dell, Grohe, and Rattan (2018) for exact path homomorphism counts to an approximate level. Our results answer an open question of Grohe (2020) and help to build a theoretical understanding of vector embeddings of graphs.

The distance measures we define turn out be closely related to the cut distance. We establish our main results by generalizing our definitions to graphons, which are limit objects of sequences of graphs, as this allows us to apply techniques from functional analysis. We prove the fairly general statement that, for every "reasonably" defined graphon pseudometric, an exact correspondence to homomorphism densities can be turned into an approximate one. We also provide an example of a distance measure that violates this reasonableness condition. This incidentally answers an open question of Grebík and Rocha (2021).

2012 ACM Subject Classification Mathematics of computing → Graph theory

Keywords and phrases graph similarity, homomorphism densities, cut distance

Digital Object Identifier 10.4230/LIPIcs.ICALP.2021.32

Category Track A: Algorithms, Complexity and Games

Related Version *Full Version*: `https://arxiv.org/abs/2104.14213` [4]

Acknowledgements The author would like to thank Yufei Zhao for pointing out how a non-quantitative inverse counting lemma for the cut distance follows from the compactness of the graphon space and the anonymous reviewers for their helpful comments.

1 Introduction

Vector representations of graphs allow to apply standard machine learning techniques to graphs, and a variety of methods to generate such embeddings has been studied in the machine learning literature. However, from a theoretical point of view, these embeddings have not received much attention and are not well understood. Some machine learning methods only implicitly operate on such vector representations as they only access the inner products of these vectors. These methods are known as *kernel methods* and most graph kernels are based on counting occurrences of certain substructures, e.g., walks or trees. See [15] for a recent survey on vector embeddings.

Many kinds of substructure counts in a graph such as graph motifs are actually just *homomorphism* counts "in disguise", and hence, homomorphisms provide a formal and flexible framework for counting all kinds of substructures in graphs [5]; a homomorphism from a graph F to a graph G is a mapping from the vertices of F to the vertices of G such that every edge of F is mapped to an edge of G. A theorem of Lovász from 1967 [18], which states that two graphs G and H are isomorphic if and only if, for every graph F, the number $\mathsf{hom}(F,G)$ of homomorphisms from F to G equals the corresponding number $\mathsf{hom}(F,H)$ from F to H, led to the development of the theory of graph limits [2, 20], where one considers convergent sequences of graphs and their limit objects, graphons. In terms of the *homomorphism vector* $\mathsf{Hom}(G) \coloneqq (\mathsf{hom}(F,G))_{F \text{ graph}}$ of a graph G, the result of Lovász states that graphs are mapped to the same vector if and only if they are isomorphic.

Computing an entry of $\mathsf{Hom}(G)$ is $\#P$-complete and recent results have mostly focused on restrictions $\mathsf{Hom}_{\mathcal{F}}(G) \coloneqq (\mathsf{hom}(F,G))_{F \in \mathcal{F}}$ of these vectors to classes $\mathcal{F}$ for which computing these entries is actually tractable. Under a natural assumption from parameterized complexity theory, this is the case for precisely the classes $\mathcal{F}$ of bounded tree width [6]. This has led to various surprisingly clean results, e.g., for trees and, more general, graphs of bounded treewidth [10], cycles and paths [7], planar graphs [22], and, most recently, graphs of bounded tree-depth [14]. These results only show what it means for graphs to be mapped to the same homomorphism vector; they do not say anything about the similarity of two graphs if the homomorphism vectors are not exactly the same but *close*. Grohe formulated the vague hypothesis that, for suitable classes $\mathcal{F}$, the embedding $\mathsf{Hom}_{\mathcal{F}}$ combined with a suitable inner product on the latent space induces a natural similarity measure on graphs [15]. This is supported by initial experiments, which show that homomorphism vectors in combination with support vector machines perform well on standard graph classification. Our results further support this hypothesis from a theoretical standpoint by showing that tree homomorphism counts provide a robust similarity measure.

For the class $\mathcal{T}$ of trees and two graphs G and H, we have $\mathsf{Hom}_{\mathcal{T}}(G) = \mathsf{Hom}_{\mathcal{T}}(H)$ if and only if G and H are not distinguished by *color refinement* (also known as the 1-*dimensional Weisfeiler-Leman algorithm*) [10], a popular heuristic for graph isomorphism. Another characterization of this equivalence due to Tinhofer is that of *fractional isomorphism* [26, 27]. Let $A \in \mathbb{R}^{V(G)\times V(G)}$ and $B \in \mathbb{R}^{V(H)\times V(H)}$ be the adjacency matrices of G and H, respectively, and consider the following system $\mathsf{F}_{\text{iso}}(G,H)$ of linear equations:

$$\mathsf{F}_{\text{iso}}(G,H): \begin{cases} AX = XB \\ X\mathbf{1}_{V(H)} = \mathbf{1}_{V(G)} \\ \mathbf{1}_{V(G)}^T X = \mathbf{1}_{V(H)}^T \end{cases}$$

Here, X denotes a $(V(G) \times V(H))$-matrix of variables, and $\mathbf{1}_U$ denotes the all-1 vector over the index set U. The non-negative integer solutions to $\mathsf{F}_{\text{iso}}(G,H)$ are precisely the permutation matrices that describe isomorphisms between G and H. The non-negative real solutions are called *fractional isomorphisms* of G and H. Tinhofer proved that G and H are not distinguished by the color refinement algorithm if and only if there is a fractional isomorphism of G and H. Grohe proposed to define a similarity measure based on this characterization [15]: For a matrix norm $\|\cdot\|$ that is invariant under permutations of the rows and columns, consider

$$\operatorname{dist}_{\|\cdot\|}(G,H) \coloneqq \min_{\substack{X \in [0,1]^{V(G)\times V(H)}, \\ X \text{ doubly stochastic}}} \|AX - XB\|.$$

Most graph distance measures based on matrix norms are highly intractable as the problem of their computation is related to notoriously hard maximum quadratic assignment problem [23]. This hardness, which stems from the minimization over the set of all permutation matrices, motivated Grohe to propose $\mathrm{dist}_{\|\cdot\|}$, where the set of all permutation matrices is relaxed to the the convex set of doubly stochastic matrices, yielding a convex optimization problem. With the results of Tinhofer and Dvořák, we know that the graphs of distance zero w.r.t. $\mathrm{dist}_{\|\cdot\|}$ are precisely those that cannot be distinguished by tree homomorphism counts.

So far, the only known connection between a graph distance measure based on matrix norms and graph homomorphisms is between the *cut distance* and normalized homomorphism numbers (called *homomorphism densities*) [2]. Grohe asks whether a similar correspondence between $\mathrm{dist}_{\|\cdot\|}$ and restricted homomorphism vectors can be established, and we give a positive answer to this question. We introduce the *tree distance* $\delta^{\mathcal{T}}$ of graphs, which is a normalized variant of $\mathrm{dist}_{\|\cdot\|}$ and show the following theorem, which is stated here only informally. We also introduce the *path distance* $\delta^{\mathcal{P}}$ of graphs and prove the analogous theorem to Theorem 1 for $\delta^{\mathcal{P}}$ and normalized path homomorphism counts.

▶ **Theorem 1** (Informal Theorem 6 and Theorem 7). *Two graphs G and H are similar w.r.t. $\delta^{\mathcal{T}}$ if and only if the homomorphism densities $t(T,G)$ and $t(T,H)$ are close for trees T.*

In the theory of graph limits, *graphons* serve as limit objects for sequences of graphs. By defining distance measures on the more general graphons, we are able to use techniques from functional analysis to show that any "reasonably" defined pseudometric on graphons satisfying an exact correspondence to homomorphism densities also has to satisfy an approximate one. As an application, we get that both the tree and the path distance satisfy this correspondence to tree and path homomorphism densities, respectively. For the case of trees, we rely on a generalization of the notion of fractional isomorphism to graphons by Grebík and Rocha [13]. For the case of paths, we prove this generalization of the result of Dell, Grohe, and Rattan [7] by ourselves.

This paper is organized as follows. In the preliminaries, Section 2, we collect the definitions of graphs, the space $L_2[0,1]$, graphons, and the cut distance. In Section 3, we define the tree distance and the path distance for graphs and formally state Theorem 1 and its path counterpart. In Section 4, we state and prove the theorems that allow us to show these correspondences for graphon pseudometrics. Section 5 provides the first application of these tools for the tree distance: we first state the needed result of fractional isomorphism of graphons due to Grebík and Rocha and then use this to define the tree distance of graphons. These definitions and results specialize to the ones presented in Section 3 for graphs. The treatment of the path distance for graphons is similar to the one of the tree distance, except for the fact that we prove a characterization of graphons with the same path homomorphism densities ourselves, and can be found in Section 6. In Section 7, we define another distance measure on graphs based on the invariant computed by the color refinement algorithm and show that it only satisfies one direction of the approximate correspondence to tree homomorphism densities. Our counterexample incidentally answers an open question of Grebík and Rocha [13]. Section 8 poses some interesting open questions that come up during the study of these distance measures. All missing proofs can be found in the full version of the paper.

2 Preliminaries

2.1 Graphs

By the term graph, we refer to a simple, undirected, and finite graph. For a graph G, we denote its vertex set by $V(G)$ and its edge set by $E(G)$, and we let $\mathsf{v}(G) \coloneqq |V(G)|$ and $\mathsf{e}(G) \coloneqq |E(G)|$. We usually view the adjacency matrix A of a graph G as a matrix $A \in \mathbb{R}^{V(G)\times V(G)}$, i.e., it is indexed by the vertices of G. Sometimes, we assume without loss of generality that the vertex set of a graph is $[n] \coloneqq \{1, \dots, n\}$, where $n \in \mathbb{N}$ is a natural number. A homomorphism from a graph F to a graph G is a mapping $\varphi\colon V(F) \to V(G)$ such that $\varphi(u)\varphi(v) \in E(G)$ for every $uv \in E(F)$. We denote the number of homomorphisms from F to G by $\mathsf{hom}(F, G)$. The homomorphism density from F to G is given by $t(F, G) \coloneqq \mathsf{hom}(F, G)/\mathsf{v}(G)^{\mathsf{v}(F)}$.

A *weighted graph* $G = (V, a, B)$ consists of a vertex set V, a positive real vector $a = (\alpha_v)_{v\in V} \in \mathbb{R}^V$ of vertex weights and a real symmetric matrix $B = (\beta_{uv}) \in [0, 1]^{V\times V}$ of edge weights; that is, we restrict ourselves to edge weights from $[0, 1]$. We write $\mathsf{v}(G) = |V|$, $V(G) = V$, $\alpha_v(G) = \alpha_v$, $\alpha_G = \sum_{v\in V(G)} \alpha_v(G)$ and $\beta_{uv}(G) = \beta_{uv}$. A weighted graph is called *normalized* if $\alpha_G = 1$. For a simple graph F and a weighted graph G, we define the *homomorphism number*

$$\mathsf{hom}(F, G) = \sum_{\varphi\colon V(F)\to V(G)} \prod_{v\in V(F)} \alpha_{\varphi(v)}(G) \prod_{uv\in E(F)} \beta_{\varphi(u)\varphi(v)}(G)$$

and the *homomorphism density* $t(F, G) = \mathsf{hom}(F, G)/\alpha_G^{\mathsf{v}(F)}$. When viewing a graph as a weighted graph in the obvious way, these notions coincide with the ones for graphs.

2.2 The Space $L_2[0, 1]$ and Graphons

A detailed introduction to functional analysis can be found in [8]; here, we only repeat some notions we use throughout the main body of the paper. Let $L_2[0, 1]$ denote the space of $\mathbb{R}$-valued 2-integrable functions on $[0, 1]$ (modulo equality almost anywhere). We could consider consider a standard Borel space with a Borel probability measure, cf. [13], but for the sake of convenience, we stick to $[0, 1]$ with the Lebesgue measure just as [20]. The space $L_2[0, 1]$ is a Hilbert space with the inner product defined by $\langle f, g\rangle \coloneqq \int_{[0,1]} f(x)g(x)\,dx$ for functions $f, g \in L_2[0, 1]$. Let $T\colon L_2[0, 1] \to L_2[0, 1]$ be a bounded linear operator, or operator for short. We write $\|T\|_{2\to 2}$ for its operator norm, i.e., $\|T\|_{2\to 2} = \sup_{\|g\|_2\le 1}\|Tg\|_2$. The *Hilbert adjoint* of T is the unique operator $T^*\colon L_2[0, 1] \to L_2[0, 1]$ such that $\langle Tf, g\rangle = \langle f, T^*g\rangle$ for all $f, g \in L_2[0, 1]$, and T is called self-adjoint if $T^* = T$.

Let $\mathcal{W}$ denote the set of all bounded symmetric measurable functions $W\colon [0, 1]^2 \to \mathbb{R}$, called *kernels*. Let $\mathcal{W}_0 \subseteq \mathcal{W}$ denote all such W that satisfy $0 \le W \le 1$; such a W is called a *graphon*. Every kernel $W \in \mathcal{W}$ defines a self-adjoint operator $T_W\colon L_2[0, 1] \to L_2[0, 1]$ by setting $(T_W f)(x) = \int_{[0,1]} W(x, y)f(y)\,dy$ for every $x \in [0, 1]$, which then is a Hilbert-Schmidt operator, and in particular, compact [20].

A kernel $W \in \mathcal{W}$ is called a *step function* if there is a partition $S_1 \cup \dots \cup S_k$ of $[0, 1]$ such that W is constant on $S_i \times S_j$ for all $i, j \in [k]$. For a weighted graph H on $[n]$, one can define a step function $W_H \in \mathcal{W}$ by splitting $[0, 1]$ into n intervals $I_1, \dots, I_n$, where I_i has length $\lambda(I_i) = \alpha_i(H)/\alpha(H)$ for every $i \in [n]$, and letting $W_H(x, y) \coloneqq \beta_{ij}(H)$ for all $x \in I_i, y \in I_j$ and $i, j \in [n]$. Of course, W_H depends on the labeling of the vertices of H. Note that W_H is a graphon, and in particular, W_G is a graphon for every graph G.

2.3 The Cut Distance

See [20] for a thorough introduction to the cut distance. The usual definition of the cut distance involves the *blow-up* $G(k)$ of a graph G by $k \geq 0$, where every vertex of G is replaced by k identical copies, to get graphs on the same number of vertices. Going this route is rather cumbersome, and we directly define the cut distance for weighted graphs via *fractional overlays*; this definition also applies to graphs in the straightforward way. A *fractional overlay* of weighted graphs G and H is a matrix $X \in \mathbb{R}^{V(G)\times V(H)}$ such that $X_{uv} \geq 0$ for all $u \in V(G)$, $v \in V(H)$, $\sum_{v\in V(H)} X_{uv} = \alpha_u(G)/\alpha_G$ for every $u \in V(G)$, and $\sum_{u\in V(G)} X_{uv} = \alpha_v(H)/\alpha_H$ for every $v \in V(H)$. Let $\mathcal{X}(G,H)$ denote the set of all fractional overlays of G and H. Note that, for graphs G and H, the second and third condition just say that the row and column sums of X are $1/\mathsf{v}(G)$ and $1/\mathsf{v}(H)$, respectively. For weighted graphs G and H and a fractional overlay $X \in \mathcal{X}(G,H)$, let

$$d_\square(G,H,X) \coloneqq \max_{Q,R\subseteq V(G)\times V(H)} \Big| \sum_{\substack{iu\in Q,\\ jv\in R}} X_{iu}X_{jv}(\beta_{ij}(G) - \beta_{uv}(H))\Big|.$$

Then, define the *cut distance* $\delta_\square(G,H) \coloneqq \min_{X\in\mathcal{X}(G,H)} d_\square(G,H,X)$.

Defining the cut distance of graphons is actually much simpler. Define the *cut norm* on the linear space $\mathcal{W}$ of kernels by $\|W\|_\square \coloneqq \sup_{S,T\subseteq[0,1]} \left|\int_{S\times T} W(x,y)\,dx\,dy\right|$ for $W \in \mathcal{W}$; here, as in the whole of the paper, we tacitly assume sets (and functions) we take an infimum or supremum over to be measurable. Let $S_{[0,1]}$ denote the group of all invertible measure-preserving maps $\varphi\colon [0,1] \to [0,1]$. For a kernel $W \in \mathcal{W}$ and a $\varphi \in S_{[0,1]}$, let W^φ be the kernel defined by $W^\varphi(x,y) \coloneqq W(\varphi(x),\varphi(y))$. For kernels $U, W \in \mathcal{W}$, define their *cut distance* by setting $\delta_\square(U,W) \coloneqq \inf_{\varphi\in S_{[0,1]}} \|U - W^\varphi\|_\square$. This coincides with the previous definition when viewing weighted graphs as graphons [20, Lemma 8.9]. We can also express $\delta_\square(U,W)$ via the kernel operator as $\delta_\square(U,W) = \inf_{\varphi\in S_{[0,1]}} \sup_{f,g\colon [0,1]\to[0,1]} \left|\langle f, T_{U-W^\varphi} g\rangle\right|$ [20, Lemma 8.10]. The definition of the cut distance is quite robust. For example, allowing f and g in the previous definition to be complex-valued or choosing a different operator norm does not make a difference in most cases [16, Appendix E].

For a graph F and a kernel $W \in \mathcal{W}$, define the *homomorphism density*

$$t(F,W) \coloneqq \int_{[0,1]^{V(F)}} \prod_{ij\in E(F)} W(x_i,x_j) \prod_{i\in V(F)} dx_i,$$

which coincides with the previous definition when viewing weighted graphs as graphons [20, Equation (7.2)]. Lemma 2 and Lemma 3 state the connection between the cut distance and homomorphism densities: Informally, the Lemma 2 states that graphons that are close in the cut distance have similar homomorphism densities, while Lemma 3 states that graphs that have similar homomorphism densities are close in the cut distance. We refer to such statements as a *counting lemma* and an *inverse counting lemma*, respectively.

▶ **Lemma 2** (Counting Lemma [21]). *Let F be a simple graph, and let $U, W \in \mathcal{W}_0$ be graphons. Then, $|t(F,U) - t(F,W)| \leq \mathsf{e}(F)\cdot\delta_\square(U,W)$.*

▶ **Lemma 3** (Inverse Counting Lemma [3, 20]). *Let $k > 0$, let $U, W \in \mathcal{W}_0$ be graphons, and assume that, for every graph F on k vertices, we have $|t(F,U) - t(F,W)| \leq 2^{-k^2}$. Then, $\delta_\square(U,W) \leq 50/\sqrt{\log k}$.*

In particular, graphons U and W have cut distance zero if and only if, for every graph F, we have $t(F,U) = t(F,W)$. Call a sequence $(W_n)_{n\in\mathbb{N}}$ of graphons *convergent* if, for every

graph F, the sequence $(t(F, W_n))_{n\in\mathbb{N}}$ is Cauchy. The two theorems above yield that $(W_n)_{n\in\mathbb{N}}$ is convergent if and only if $(W_n)_{n\in\mathbb{N}}$ is Cauchy in $\delta_\square$. Let $\widetilde{\mathcal{W}_0}$ be obtained from $\mathcal{W}_0$ by identifying graphons with cut distance zero; such graphons are called *weakly isomorphic*. One of the main results from graph limit theory is the compactness of the space $(\widetilde{\mathcal{W}_0}, \delta_\square)$.

▶ **Theorem 4** ([19]). *The space $(\widetilde{\mathcal{W}_0}, \delta_\square)$ is compact.*

3 Similarity Measures of Graphs

In this section, we define the tree and path distances of graphs and formally state the correspondences to tree and path homomorphism densities, respectively. All presented results are specializations of the results for graphons proven in Section 5 and Section 6.

3.1 The Tree Distance of Graphs

Recall that two graphs G and H have the same tree homomorphism counts if and only if the system $\mathsf{F}_{\text{iso}}(G, H)$ of linear equations has a non-negative solution. Based on this, Grohe proposed $\text{dist}_{\|\cdot\|}$ as a similarity measure of graphs. This is nearly what we define as the tree distance of graphs. What is missing is, first, a more general definition for graphs with different numbers of vertices and, second, an appropriate choice of a matrix norm with an appropriate normalization factor; analogously to the cut distance, we normalize the tree distance to values in $[0, 1]$. As in the definition of the cut distance in the preliminaries, we handle graphs on different numbers of vertices by considering fractional overlays instead of blow-ups (and doubly stochastic matrices). Recall that a fractional overlay of graphs G and H is a matrix $X \in \mathbb{R}^{V(G)\times V(H)}$ such that $X_{uv} \geq 0$ for all $u \in V(G)$, $v \in V(H)$, $\sum_{v\in V(H)} X_{uv} = 1/\mathsf{v}(G)$ for every $u \in V(G)$, and $\sum_{u\in V(G)} X_{uv} = 1/\mathsf{v}(H)$ for every $v \in V(H)$. If $\mathsf{v}(G) = \mathsf{v}(H)$, then the difference between a fractional overlay and a doubly stochastic matrix is just a factor of $\mathsf{v}(G)$. Also recall that $\mathcal{X}(G, H)$ denotes the set of all fractional overlays of G and H.

We consider two matrix norms for the tree distance: First, just like in the definition of the cut distance, we use the *cut norm* for matrices, introduced by Frieze and Kannan [12], defined as $\|A\|_\square \coloneqq \max_{S\subseteq[m],T\subseteq[n]} |\sum_{i\in S, j\in T} A_{ij}|$ for $A \in \mathbb{R}^{m\times n}$. Second, we also consider the more standard spectral norm $\|A\|_2 \coloneqq \sup_{x\in\mathbb{R}^n, \|x\|_2\leq 1} \|Ax\|_2$ of a matrix $A \in \mathbb{R}^{m\times n}$. From a computational point of view, the Frobenius norm might also be appealing, but this would lead to a different topology, cf. [16, Appendix E].

▶ **Definition 5** (Tree Distance of Graphs). *Let G and H be graphs with adjacency matrices $A \in \mathbb{R}^{V(G)\times V(G)}$ and $B \in \mathbb{R}^{V(H)\times V(H)}$, respectively. Then, define*

$$\delta_\square^{\mathcal{T}}(G, H) \coloneqq \inf_{X\in\mathcal{X}(G,H)} \frac{1}{\mathsf{v}(G)\cdot\mathsf{v}(H)} \|\mathsf{v}(H)\cdot AX - \mathsf{v}(G)\cdot XB\|_\square \text{ and}$$

$$\delta_2^{\mathcal{T}}(G, H) \coloneqq \inf_{X\in\mathcal{X}(G,H)} \frac{1}{\sqrt{\mathsf{v}(G)\mathsf{v}(H)}} \|\mathsf{v}(H)\cdot AX - \mathsf{v}(G)\cdot XB\|_2.$$

Note that the spectral norm requires an adapted normalization factor in Definition 5. The advantage of $\delta_\square^{\mathcal{T}}$ is the close connection to the cut distance, which also utilizes the cut norm. However, the crucial advantage of the spectral norm is that minimization of the spectral norm of a matrix is a standard application of interior-point methods in convex optimization. In particular, an ε-solution to $\delta_2^{\mathcal{T}}$ can be computed in polynomial time [24, Section 6.3.3]. For $\delta_\square^{\mathcal{T}}$, it is not clear whether this is possible.

From the results of Section 5, we get that $\delta_\square^{\mathcal{T}}$ and $\delta_2^{\mathcal{T}}$ are pseudometrics (Lemma 16) and that two graphs have distance zero if and only if their tree homomorphism densities are the

same (Lemma 18). Moreover, we have $\delta_{\square}^{\mathcal{T}} \leq \delta_{\square}$ (Lemma 19), and these pseudometrics are invariant under blow-ups. Finally, we get the following counting lemma (Corollary 20) and inverse counting lemma (Corollary 21).

▶ **Theorem 6** (Counting Lemma for $\delta^{\mathcal{T}}$, Graphs). *Let $\delta^{\mathcal{T}} \in \{\delta_{\square}^{\mathcal{T}}, \delta_2^{\mathcal{T}}\}$. For every tree T and every $\varepsilon > 0$, there is an $\eta > 0$ such that, for all graphs G and H, if $\delta^{\mathcal{T}}(G,H) \leq \eta$, then $|t(T,G) - t(T,H)| \leq \varepsilon$.*

▶ **Theorem 7** (Inverse Counting Lemma for $\delta^{\mathcal{T}}$, Graphs). *Let $\delta^{\mathcal{T}} \in \{\delta_{\square}^{\mathcal{T}}, \delta_2^{\mathcal{T}}\}$. For every $\varepsilon > 0$, there are $k > 0$ and $\eta > 0$ such that, for all graphs G and H, if $|t(T,G) - t(T,H)| \leq \eta$ for every tree T on at most k vertices, then $\delta^{\mathcal{T}}(G,H) \leq \varepsilon$.*

3.2 The Path Distance of Graphs

Dell, Grohe, and Rattan proved that two graphs G and H have the same path homomorphism counts if and only if the system $\mathsf{F}_{\text{iso}}(G,H)$ of linear equations has a real solution [7]. This transfers to the definition of the path distance, i.e., we define the path distance analogously to the tree distance but relax the non-negativity condition of fractional overlays. For graphs G and H, we call a matrix $X \in \mathbb{R}^{V(G)\times V(H)}$ a *signed fractional overlay* of G and H if $\|Xy\|_2 \leq \|y\|_2/\sqrt{\mathsf{v}(G)\mathsf{v}(H)}$ for every $y \in \mathbb{R}^{V(H)}$, $\sum_{v\in V(H)} X_{uv} = 1/\mathsf{v}(G)$ for every $u \in V(G)$, and $\sum_{u\in V(G)} X_{uv} = 1/\mathsf{v}(H)$ for every $v \in V(H)$. Let $\mathcal{S}(G,H)$ denote the set of all signed fractional overlays of G and H. The first condition requires that X is a contraction (up to a scaling factor) in the spectral norm; we need this to guarantee that our definition of the path distance actually yields a pseudometric. This restriction to the spectral norm stems from the fact that the proof of Dell, Grohe, and Rattan [7] (and our generalization thereof to graphons) only guarantees that the constructed solution is a contraction in the spectral norm, cf. Section 6 for the details.

▶ **Definition 8** (Path Distance of Graphs). *Let G and H be graphs with adjacency matrices $A \in \mathbb{R}^{V(G)\times V(G)}$ and $B \in \mathbb{R}^{V(H)\times V(H)}$, respectively. Then, define*

$$\delta_2^{\mathcal{P}}(G,H) \coloneqq \inf_{X\in\mathcal{S}(G,H)} \frac{1}{\sqrt{\mathsf{v}(G)\mathsf{v}(H)}} \|\mathsf{v}(H)\cdot AX - \mathsf{v}(G)\cdot XB\|_2.$$

From Section 6, we get that $\delta_2^{\mathcal{P}}$ is a pseudometric (Lemma 25) that is invariant under blow-ups and that has as graphs of distance zero precisely these with the same path homomorphism densities. Moreover, we get the following (quantitative) counting lemma (Corollary 27) and inverse counting lemma (Corollary 30).

▶ **Theorem 9** (Counting Lemma for $\delta_2^{\mathcal{P}}$, Graphs). *Let P be a path, and let G and H be graphs. Then, $|t(P,G) - t(P,H)| \leq \mathsf{e}(P)\cdot\delta_2^{\mathcal{P}}(G,H)$.*

▶ **Theorem 10** (Inverse Counting Lemma for $\delta_2^{\mathcal{P}}$, Graphs). *For every $\varepsilon > 0$, there are $k > 0$ and $\eta > 0$ such that, for all graphs G and H, if $|t(P,G) - t(P,H)| \leq \eta$ for every path P on at most k vertices, then $\delta_2^{\mathcal{P}}(G,H) \leq \varepsilon$.*

4 Graphon Pseudometrics and Homomorphism Densities

In this section, we provide the main tools we need to prove the correspondences between the tree and path distances and tree and path homomorphism densities, respectively. Consider a pseudometric δ on graphons. We say that δ *is compatible with* $\delta_{\square}$ if, for every sequence of graphons $(U_n)_n$, $U_n \in \mathcal{W}_0$, and every graphon $\widetilde{U} \in \mathcal{W}_0$, $\delta_{\square}(U_n,\widetilde{U}) \xrightarrow{n\to\infty} 0$ implies

$\delta(U_n, \widetilde{U}) \xrightarrow{n\to\infty} 0$. For example, this is the case if $\delta \leq \delta_\square$, i.e., graphons only get closer if we consider δ instead of $\delta_\square$. We anticipate that the pseudometrics we are interested in, the tree distance and the path distance, are compatible with $\delta_\square$.

Together, the next two theorems state that every pseudometric that is compatible with $\delta_\square$ and whose graphons of distance zero can be characterized by homomorphism densities from a class of graphs $\mathcal{F}$ already has to satisfy both a counting lemma and an inverse counting lemma for this class $\mathcal{F}$. The proof of these theorems is a simple compactness argument, utilizing the compactness of the graphon space, Theorem 4, and the counting lemma for $\delta_\square$, Lemma 2. Therefore, it is absolutely crucial that we consider a pseudometric defined on graphons as the limit of a sequence of graphs may not be a graph.

▶ **Theorem 11** (Counting Lemma for $\mathcal{F}$)**.** *Let $\mathcal{F}$ be a class of graphs, and let $\delta^{\mathcal{F}}$ be a pseudometric on graphons such that (1) $\delta^{\mathcal{F}}$ is compatible with $\delta_\square$ and (2), for all graphons $U, W \in \mathcal{W}_0$, $\delta^{\mathcal{F}}(U, W) = 0$ implies $t(F, U) = t(F, W)$ for every graph $F \in \mathcal{F}$. Then, for every graph $F \in \mathcal{F}$ and every $\varepsilon > 0$, there is an $\eta > 0$ such that, for all graphons $U, W \in \mathcal{W}_0$, if $\delta^{\mathcal{F}}(U, W) \leq \eta$, then $|t(F, U) - t(F, W)| \leq \varepsilon$.*

Proof of Theorem 11. We proceed by contradiction and assume that the statement does not hold. Then, there is a graph $F \in \mathcal{F}$ and an $\varepsilon > 0$ such that, for every $\eta > 0$, there are graphons $U, W \in \mathcal{W}_0$ such that $\delta^{\mathcal{F}}(U, W) \leq \eta$ and $|t(F, U) - t(F, W)| > \varepsilon$.

Let $k > 0$. Then, by choosing $\eta = \frac{1}{k}$, we get that there are graphons $U_k, W_k \in \mathcal{W}_0$ such that $\delta^{\mathcal{F}}(U_k, W_k) \leq \frac{1}{k}$ and $|t(F, U_k) - t(F, W_k)| > \varepsilon$. By the compactness theorem, Theorem 4, we get that the sequence $(U_k)_k$ has a convergent subsequence $(U_{k_i})_i$ converging to a graphon $\widetilde{U}$ in the metric $\delta_\square$. By another application of that theorem, we get that $(W_{k_i})_i$ has a convergent subsequence $(W_{\ell_i})_i$ converging to a graphon $\widetilde{W}$ in the metric $\delta_\square$. Then, $(U_{\ell_i})_i$ and $(W_{\ell_i})_i$ are sequences converging to $\widetilde{U}$ and $\widetilde{W}$ in the metric $\delta_\square$, respectively.

Now, for every $i > 0$, we have

$$\delta^{\mathcal{F}}(\widetilde{U}, \widetilde{W}) \leq \delta^{\mathcal{F}}(\widetilde{U}, U_{\ell_i}) + \delta^{\mathcal{F}}(U_{\ell_i}, W_{\ell_i}) + \delta^{\mathcal{F}}(W_{\ell_i}, \widetilde{W}).$$

By assumption, we have $\delta^{\mathcal{F}}(U_{\ell_i}, W_{\ell_i}) \leq \frac{1}{\ell_i}$, which means that $\delta^{\mathcal{F}}(U_{\ell_i}, W_{\ell_i}) \xrightarrow{i\to\infty} 0$. Since $\delta_\square(U_{\ell_i}, \widetilde{U}) \xrightarrow{i\to\infty} 0$ and $\delta_\square(W_{\ell_i}, \widetilde{W}) \xrightarrow{i\to\infty} 0$, the first assumption about $\delta^{\mathcal{F}}$ yields that we also have $\delta^{\mathcal{F}}(U_{\ell_i}, \widetilde{U}) \xrightarrow{i\to\infty} 0$ and $\delta^{\mathcal{F}}(W_{\ell_i}, \widetilde{W}) \xrightarrow{i\to\infty} 0$. Hence, we must have $\delta^{\mathcal{F}}(\widetilde{U}, \widetilde{W}) = 0$.

Since $\delta^{\mathcal{F}}(\widetilde{U}, \widetilde{W}) = 0$, we have $t(F, \widetilde{U}) = t(F, \widetilde{W})$ by the second assumption about $\delta^{\mathcal{F}}$. By the Counting Lemma, Lemma 2, we get that $|t(F, U_{\ell_i}) - t(F, \widetilde{U})| \xrightarrow{i\to\infty} 0$ and $|t(F, \widetilde{W}) - t(F, W_{\ell_i})| \xrightarrow{i\to\infty} 0$. Now, for every $i > 0$, we have

$$|t(F, U_{\ell_i}) - t(F, W_{\ell_i})| \leq |t(F, U_{\ell_i}) - t(F, \widetilde{U})| + |t(F, \widetilde{U}) - t(F, \widetilde{W})| + |t(F, \widetilde{W}) - t(F, W_{\ell_i})|$$

Hence, $|t(F, U_{\ell_i}) - t(F, W_{\ell_i})| \xrightarrow{i\to\infty} 0$. This contradicts the fact that $|t(F, U_{\ell_i}) - t(F, W_{\ell_i})| > \varepsilon$ for every i. ◀

Just as the proof of Theorem 11, the proof of Theorem 12 only relies on the compactness of the graphon space and the counting lemma for $\delta_\square$, and not on a counting lemma for a specific class of graphs or the inverse counting lemma for $\delta_\square$.

▶ **Theorem 12** (Inverse Counting Lemma for $\mathcal{F}$)**.** *Let $\mathcal{F}$ be a class of graphs, and let $\delta^{\mathcal{F}}$ be a pseudometric on graphons such that (1) $\delta^{\mathcal{F}}$ is compatible with $\delta_\square$ and (2), for all graphons $U, W \in \mathcal{W}_0$, $t(F, U) = t(F, W)$ for every graph $F \in \mathcal{F}$ implies $\delta^{\mathcal{F}}(U, W) = 0$. Then, for every $\varepsilon > 0$, there are $k > 0$ and $\eta > 0$ such that, for all graphons $U, W \in \mathcal{W}_0$, if $|t(F, U) - t(F, W)| \leq \eta$ for every graph $F \in \mathcal{F}$ on at most k vertices, then $\delta^{\mathcal{F}}(U, W) \leq \varepsilon$.*

Proof. We proceed by contradiction and assume that the statement does not hold. Then, there is an $\varepsilon > 0$ such that, for every $k > 0$ and every $\eta > 0$, there are graphons $U, W \in \mathcal{W}_0$ such that $|t(F,U) - t(F,W)| \leq \eta$ for every graph $F \in \mathcal{F}$ on at most k vertices but $\delta^{\mathcal{F}}(U,W) > \varepsilon$.

Let $k > 0$. Then, by choosing $\eta = \frac{1}{k}$, we get that there are graphons $U_k, W_k \in \mathcal{W}_0$ such that $|t(F,U_k) - t(F,W_k)| \leq \frac{1}{k}$ for every graph $F \in \mathcal{F}$ on at most k vertices and $\delta^{\mathcal{F}}(U_k,W_k) > \varepsilon$. By the compactness theorem, Theorem 4, we get that the sequence $(U_k)_k$ has a convergent subsequence $(U_{k_i})_i$ converging to a graphon $\widetilde{U}$ in the metric $\delta_\square$. By another application of that theorem, we get that $(W_{k_i})_i$ has a convergent subsequence $(W_{\ell_i})_i$ converging to a graphon $\widetilde{W}$ in the metric $\delta_\square$. Then, $(U_{\ell_i})_i$ and $(W_{\ell_i})_i$ are sequences converging to $\widetilde{U}$ and $\widetilde{W}$ in the metric $\delta_\square$, respectively.

Let $F \in \mathcal{F}$ be a graph. Now, for every $i > 0$, we have

$$|t(F,\widetilde{U}) - t(F,\widetilde{W})| \leq |t(F,\widetilde{U}) - t(F,U_{\ell_i})| + |t(F,U_{\ell_i}) - t(F,W_{\ell_i})| + |t(F,W_{\ell_i}) - t(F,\widetilde{W})|$$

By the counting lemma for $\delta_\square$, Lemma 2, we get that $|t(F,\widetilde{U}) - t(F,U_{\ell_i})| \xrightarrow{i\to\infty} 0$ and $|t(F,W_{\ell_i}) - t(F,\widetilde{W})| \xrightarrow{i\to\infty} 0$. Moreover, by assumption, we have $|t(F,U_{\ell_i}) - t(F,W_{\ell_i})| \leq \frac{1}{\ell_i}$ for large enough i, which means that also $|t(F,U_{\ell_i}) - t(F,W_{\ell_i})| \xrightarrow{i\to\infty} 0$. Hence, we must have $t(F,\widetilde{U}) = t(F,\widetilde{W})$.

As we have $t(F,\widetilde{U}) = t(F,\widetilde{W})$ for every graph $F \in \mathcal{F}$, the second assumption about $\delta^{\mathcal{F}}$ yields that $\delta^{\mathcal{F}}(\widetilde{U},\widetilde{W}) = 0$. Since $\delta_\square(U_{\ell_i},\widetilde{U}) \xrightarrow{i\to\infty} 0$ and $\delta_\square(W_{\ell_i},\widetilde{W}) \xrightarrow{i\to\infty} 0$, we also have $\delta^{\mathcal{F}}(U_{\ell_i},\widetilde{U}) \xrightarrow{i\to\infty} 0$ and $\delta^{\mathcal{F}}(W_{\ell_i},\widetilde{W}) \xrightarrow{i\to\infty} 0$ by the first assumption about $\delta^{\mathcal{F}}$. Now, for every $i > 0$, we have

$$\delta^{\mathcal{F}}(U_{\ell_i},W_{\ell_i}) \leq \delta^{\mathcal{F}}(U_{\ell_i},\widetilde{U}) + \delta^{\mathcal{F}}(\widetilde{U},\widetilde{W}) + \delta^{\mathcal{F}}(\widetilde{W},W_{\ell_i}).$$

Hence, $\delta^{\mathcal{F}}(U_{\ell_i},W_{\ell_i}) \xrightarrow{i\to\infty} 0$. This contradicts the fact that $\delta^{\mathcal{F}}(U_{\ell_i},W_{\ell_i}) > \varepsilon$ for every i. ◀

5 Homomorphisms from Trees

In this section, we define the tree distance of graphons. To use the results from Section 4, we prove that the graphons of distance zero are precisely those with the same tree homomorphism densities (Lemma 18) and that the tree distance is compatible with the cut distance (Lemma 19). As for graphs, we define two variants of the tree distance, which yield the same topology (Lemma 17): one using the analogue of the cut norm and one using the analogue of the spectral norm.

5.1 Fractional Isomorphism of Graphons

Recall that two graphs G and H with adjacency matrices $A \in \mathbb{R}^{V(G)\times V(G)}$ and $B \in \mathbb{R}^{V(H)\times V(H)}$, respectively, are called fractionally isomorphic if there is a doubly stochastic matrix $X \in \mathbb{R}^{V(G)\times V(H)}$ such that $AX = XB$. Grebík and Rocha proved Theorem 13, which generalizes this to graphons [13]; doubly stochastic matrices become *Markov operators* [11]. An operator $S\colon L_2[0,1] \to L_2[0,1]$ is called a Markov operator if $S \geq 0$, i.e., $f \geq 0$ implies $S(f) \geq 0$, $S(\mathbf{1}) = \mathbf{1}$, and $S^*(\mathbf{1}) = \mathbf{1}$, where $\mathbf{1}$ is the all-one function on $[0,1]$. We denote the set of all Markov operators $S\colon L_2[0,1] \to L_2[0,1]$ by $\mathcal{M}$.

▶ **Theorem 13** ([13], Part of Theorem 1.2). *Let $U, W \in \mathcal{W}_0$ be graphons. There is a Markov operator $S\colon L_2[0,1] \to L_2[0,1]$ such that $T_U \circ S = S \circ T_W$ if and only if $t(T,U) = t(T,W)$ for every tree T.*

5.2 The Tree Distance

Recall that, for graphons $U, W \in \mathcal{W}_0$, the cut distance of U and W can be written as $\delta_\square(U,W) = \inf_{\varphi \in S_{[0,1]}} \sup_{f,g\colon [0,1]\to[0,1]} \left|\langle f, T_{U-W^\varphi} g\rangle\right|$. We obtain the tree distance of U and W by relaxing measure-preserving maps to Markov operators.

▶ **Definition 14** (Tree Distance). *Let $U, W \in \mathcal{W}_0$ be graphons. Then, define*

$$\delta^{\mathcal{T}}_\square(U,W) \coloneqq \inf_{S\in\mathcal{M}} \sup_{f,g\colon [0,1]\to[0,1]} |\langle f, (T_U \circ S - S \circ T_W) g\rangle| \text{ and}$$
$$\delta^{\mathcal{T}}_{2\to 2}(U,W) \coloneqq \inf_{S\in\mathcal{M}} \|T_U \circ S - S \circ T_W\|_{2\to 2}.$$

As the notation $\delta^{\mathcal{T}}_\square$ indicates, the definition of $\delta^{\mathcal{T}}_\square$ is based (although not explicitly) on the cut norm, while $\delta^{\mathcal{T}}_{2\to 2}$ is defined via the operator norm $\|\cdot\|_{2\to 2}$, which corresponds to the spectral norm for matrices. One can verify that these definitions specialize to the ones for graphs from Section 3.1.

▶ **Lemma 15.** *Let G and H be graphs. Then, $\delta^{\mathcal{T}}_\square(G,H) = \delta^{\mathcal{T}}_\square(W_G, W_H)$ and $\delta^{\mathcal{T}}_2(G,H) = \delta^{\mathcal{T}}_{2\to 2}(W_G, W_H)$.*

We verify that the tree distance actually is a pseudometric. To prove the triangle inequality for $\delta^{\mathcal{T}}_\square$ and $\delta^{\mathcal{T}}_{2\to 2}$, we use that a Markov operator is a contraction on $L_\infty[0,1]$ and $L_2[0,1]$, respectively [11, Theorem 13.2 b)].

▶ **Lemma 16.** *$\delta^{\mathcal{T}}_\square$ and $\delta^{\mathcal{T}}_{2\to 2}$ are pseudometrics on $\mathcal{W}_0$.*

The Riesz-Thorin Interpolation Theorem (see, e.g., [1, Theorem 1.1.1]) allows to prove that both variants of the tree distance define the same topology.

▶ **Lemma 17.** *Let $U, W \in \mathcal{W}_0$ be graphons. Then, $\delta^{\mathcal{T}}_\square(U,W) \le \delta^{\mathcal{T}}_{2\to 2}(U,W) \le 4\delta^{\mathcal{T}}_\square(U,W)^{1/2}$.*

To be able to apply the results from Section 4, we need that the tree distance of two graphons is zero if and only if their tree homomorphism densities are the same. Let $U, W \in \mathcal{W}_0$ be graphons. From the respective definitions, it is not immediately clear that $\delta^{\mathcal{T}}_\square(U,W) = 0$ or $\delta^{\mathcal{T}}_{2\to 2}(U,W) = 0$ implies $t(T,U) = t(T,W)$ for every tree T since the infimum over all Markov operators might not be attained. Here, we can use a continuity argument as the set of Markov operators is compact in the weak operator topology [11, Theorem 13.8]. However, we have to take a detour via a third variant of the tree distance where compactness in the weak operator topology suffices. All the details can be found in the full version of the paper.

▶ **Lemma 18.** *Let $U, W \in \mathcal{W}_0$ be graphons. Then, $\delta^{\mathcal{T}}_\square(U,W) = 0$ if and only if $t(T,U) = t(T,W)$ for every tree T.*

The *Koopman operator* $T_\varphi\colon f \mapsto f \circ \varphi$ of a measure-preserving map $\varphi\colon [0,1] \to [0,1]$ is a Markov operator [11, Example 13.1, 3)]. Hence, the tree distance can be seen as the relaxation of the cut distance obtained by relaxing measure-preserving maps to Markov operators. In particular, this means that the tree distance is compatible with the cut distance.

▶ **Lemma 19.** *Let $U, W \in \mathcal{W}_0$ be graphons. Then, $\delta^{\mathcal{T}}_\square(U,W) \le \delta_\square(U,W)$.*

With Lemma 18 and Lemma 19 we can apply the theorems of Section 4 and get both a counting lemma and an inverse counting lemma for the tree distance.

▶ **Corollary 20** (Counting Lemma for $\delta^{\mathcal{T}}$). *Let $\delta^{\mathcal{T}} \in \{\delta^{\mathcal{T}}_\square, \delta^{\mathcal{T}}_{2\to 2}\}$. For every tree T and every $\varepsilon > 0$, there is an $\eta > 0$ such that, for all graphons $U, W \in \mathcal{W}_0$, if $\delta^{\mathcal{T}}(U,W) \le \eta$, then $|t(T,U) - t(T,W)| \le \varepsilon$.*

▶ **Corollary 21** (Inverse Counting Lemma for $\delta^{\mathcal{T}}$). *Let $\delta^{\mathcal{T}} \in \{\delta^{\mathcal{T}}_{\square}, \delta^{\mathcal{T}}_{2\to 2}\}$. For every $\varepsilon > 0$, there are $k > 0$ and $\eta > 0$ such that, for all graphons $U, W \in \mathcal{W}_0$, if $|t(T,U) - t(T,W)| \leq \eta$ for every tree T on k vertices, then $\delta^{\mathcal{T}}(U,W) \leq \varepsilon$.*

6 Homomorphisms from Paths

In this section, we define the path distance of graphons. We prove a quantitative counting lemma for it (Corollary 27) and only rely on the results from Section 4 to obtain an inverse counting lemma. To this end, we we prove that the graphons of distance zero are precisely those with the same path homomorphism densities (Lemma 28) and that the path distance is compatible with the cut distance (Lemma 29). Since there is no existing characterization of graphons with the same path homomorphism densities that we can rely on, we first generalize the result of Dell, Grohe, and Rattan to graphons (Theorem 22).

6.1 Path Densities and Graphons

Dell, Grohe, and Rattan have shown the surprising fact that G and H have the same path homomorphism counts if and only if the system $\mathsf{F}_{\text{iso}}(G,H)$ has a real solution [7]. We need a generalization of their characterization to graphons in order to define the path distance of graphons and apply the results from Section 4. If two graphons $U, W \in \mathcal{W}_0$ have the same path homomorphism densities, the proof of Theorem 22 yields an operator $S\colon L_2[0,1] \to L_2[0,1]$ such that $S(\mathbf{1}) = \mathbf{1}$ and $S^*(\mathbf{1}) = \mathbf{1}$, which generalizes the result of [7] in a straight-forward fashion. An important detail is that the proof also yields that S is an L_2-contraction; this guarantees that the path distance satisfies the triangle inequality, i.e., that it is a pseudometric in the first place. For the sake of brevity, we call an operator $S\colon L_2[0,1] \to L_2[0,1]$ a *signed Markov operator* if S is an L_2-contraction, i.e., $\|Sf\|_2 \leq \|f\|_2$ for every $f \in L_2[0,1]$, $S(\mathbf{1}) = \mathbf{1}$, and $S^*(\mathbf{1}) = \mathbf{1}$. Let $\mathcal{S}$ denote the set of all signed Markov operators. It is easy to see that $\mathcal{S}$ is closed under composition and Hilbert adjoints.

▶ **Theorem 22.** *Let $U, W \in \mathcal{W}_0$. There is a signed Markov operator $S\colon L_2[0,1] \to L_2[0,1]$ such that $T_U \circ S = S \circ T_W$ if and only if $t(P,U) = t(P,W)$ for every path P.*

Homomorphism densities from paths can be expressed in terms of operator powers. For $\ell \geq 0$, let P_ℓ denote the path of length ℓ. Then, for a graphon U, we have

$$t(P_\ell, U) = \int_{[0,1]^{\ell+1}} \prod_{i\in[\ell]} U(x_i, x_{i+1}) \prod_{i\in[\ell+1]} dx_i = \langle \mathbf{1}, T_U^\ell \mathbf{1} \rangle$$

for every $\ell \geq 0$. The proof of Theorem 22 utilizes the Spectral Theorem for compact operators on Hilbert spaces to express $\mathbf{1}$ as a sum of orthogonal eigenfunctions. For a kernel $W \in \mathcal{W}$, $T_W\colon L_2[0,1] \to L_2[0,1]$ is a Hilbert-Schmidt operator and, hence, compact [20]. Since $L_2[0,1]$ is separable and T_W is compact and self-adjoint, the Spectral Theorem yields that there is a countably infinite orthonormal basis $\{f'_i\}$ of $L_2[0,1]$ consisting of eigenfunctions of T_W with the corresponding multiset of eigenvalues $\{\lambda_n\} \subseteq \mathbb{R}$ such that $\lambda_n \xrightarrow{n\to\infty} 0$ (see, e.g., [9]). If graphons U and W have the same path homomorphism densities, an interpolation argument yields that the lengths of the eigenvectors in the decomposition of $\mathbf{1}$ and their eigenvalues have to be the same. Then, one can define the operator S from these eigenfunctions of U and W. The detailed proof can be found in the full version of the paper.

6.2 The Path Distance

We define the *path distance* of graphons can analogously to the tree distance. However, as the proof Theorem 22 does not yield that the resulting operator is an L_∞-contraction, we are limited in our choice of norms.

▶ **Definition 23** (Path Distance). *Let $U, W \in \mathcal{W}_0$ be graphons. Then, define*

$$\delta^{\mathcal{P}}_{2\to 2}(U,W) \coloneqq \inf_{S\in\mathcal{S}} \|T_U \circ S - S \circ T_W\|_{2\to 2}.$$

One can verify that this defines a pseudometric that specializes to the one for graphs from Section 3.2.

▶ **Lemma 24.** *Let G and H be graphs. Then, $\delta^{\mathcal{P}}_2(G,H) = \delta^{\mathcal{P}}_{2\to 2}(W_G, W_H)$.*

▶ **Lemma 25.** *$\delta^{\mathcal{P}}_{2\to 2}$ is a pseudometric on $\mathcal{W}_0$.*

To apply the theorems of Section 4, we need that two graphons have distance zero in the path distance if and only if their path homomorphism densities are the same and that $\delta^{\mathcal{P}}_{2\to 2}$ is compatible with $\delta_\square$. For the former, we deviate from the way we proceeded for the tree distance as we actually can prove a quantitative counting lemma.

▶ **Theorem 26** (Counting Lemma for Paths). *Let P be a path, and let $U, W \in \mathcal{W}_0$ be graphons. Then, for every operator $S\colon L_2[0,1] \to L_2[0,1]$ with $S(\mathbf{1}) = \mathbf{1}$ and $S^*(\mathbf{1}) = \mathbf{1}$,*

$$|t(P,U) - t(P,W)| \leq \mathsf{e}(P) \cdot \sup_{f,g\colon [0,1]\to[0,1]} |\langle f, (T_U \circ S - S \circ T_W) g\rangle|.$$

Proof. Let $\ell \in \mathbb{N}$ and $S \in \mathcal{S}$. Then,

$$\begin{aligned}
|t(P_\ell,U) - t(P_\ell,W)| &= |\langle \mathbf{1}, T^\ell_U(S\mathbf{1})\rangle - \langle (S^*\mathbf{1}), T^\ell_W \mathbf{1}\rangle| \\
&= \Big| \sum_{i\in[\ell]} \big(\langle \mathbf{1}, (T^{\ell-i+1}_U \circ S \circ T^{i-1}_W)\mathbf{1}\rangle - \langle \mathbf{1}, (T^{\ell-i}_U \circ S \circ T^{i}_W)\mathbf{1}\rangle \big) \Big| \\
&= \Big| \sum_{i\in[\ell]} \langle T^{\ell-i}_U \mathbf{1}, (T_U \circ S - S \circ T_W)(T^{i-1}_W \mathbf{1})\rangle \Big| \\
&\leq \ell \cdot \sup_{f,g\colon [0,1]\to[0,1]} |\langle f, (T_U \circ S - S \circ T_W) g\rangle|. \qquad \blacktriangleleft
\end{aligned}$$

Theorem 26 suggests that, for graphons $U, W \in \mathcal{W}_0$, one should define

$$\delta^{\mathcal{P}}_\square(U,W) \coloneqq \inf_{S\in\mathcal{S}} \sup_{f,g\colon [0,1]\to[0,1]} |\langle f, (T_U \circ S - S \circ T_W) g\rangle|.$$

Then, we have $|t(P,U) - t(P,W)| \leq \mathsf{e}(P) \cdot \delta^{\mathcal{P}}_\square(U,W)$ for every path P. However, as mentioned before, we cannot verify that $\delta^{\mathcal{P}}_\square$ is a pseudometric as the operator S might not be an L_∞-contraction.

▶ **Corollary 27** (Counting Lemma for $\delta^{\mathcal{P}}_{2\to 2}$). *Let P be a path, and let $U, W \in \mathcal{W}_0$ be graphons. Then, $|t(P,U) - t(P,W)| \leq \mathsf{e}(P) \cdot \delta^{\mathcal{P}}_{2\to 2}(U,W)$.*

Proof. By the Cauchy-Schwarz inequality, we have

$$\begin{aligned}
\sup_{f,g\colon [0,1]\to[0,1]} |\langle f, (T_U \circ S - S \circ T_W) g\rangle| &\leq \sup_{f,g\colon [0,1]\to[0,1]} \|f\|_2 \|(T_U \circ S - S \circ T_W) g\|_2 \\
&\leq \sup_{g\colon [0,1]\to[0,1]} \|T_U \circ S - S \circ T_W\|_{2\to 2} \|g\|_2 \\
&\leq \|T_U \circ S - S \circ T_W\|_{2\to 2}
\end{aligned}$$

for every operator $S\colon L_2[0,1] \to L_2[0,1]$. Hence, the statement follows from Theorem 26. $\blacktriangleleft$

With this explicit counting lemma, we obtain that two graphons have distance zero in the path distance if and only if their path homomorphism densities are the same.

▶ **Lemma 28.** *Let $U, W \in \mathcal{W}_0$ be graphons. Then, $\delta^{\mathcal{P}}_{2\to 2}(U,W) = 0$ if and only if $t(P,U) = t(P,W)$ for every path P.*

Proof. If $\delta^{\mathcal{P}}_{2\to 2}(U,W) = 0$, then Corollary 27 yields that $t(P,U) = t(P,W)$ for every path P. On the other hand, if $t(P,U) = t(P,W)$ for every path P, then there is a signed Markov operator $S \in \mathcal{S}$ with $T_U \circ S = S \circ T_W$ by Theorem 22. Then, $\delta^{\mathcal{P}}_{2\to 2}(U,W) = 0$ follows immediately from the definition. ◀

By definition, the path distance is bounded from above by the tree distance (with the appropriate norm), which means that it also is compatible with the cut distance.

▶ **Lemma 29.** *Let $U, W \in \mathcal{W}_0$ be graphons. Then, $\delta^{\mathcal{P}}_{2\to 2}(U,W) \leq \delta^{\mathcal{T}}_{2\to 2}(U,W)$.*

With these lemmas, we can apply Theorem 12 and obtain the following inverse counting lemma for the path distance.

▶ **Corollary 30** (Inverse Counting Lemma for $\delta^{\mathcal{P}}_{2\to 2}$)**.** *For every $\varepsilon > 0$, there are $k > 0$ and $\eta > 0$ such that, for all graphons $U, W \in \mathcal{W}_0$, if $|t(P,U) - t(P,W)| \leq \eta$ for every path P on at most k vertices, then $\delta^{\mathcal{P}}_{2\to 2}(U,W) \leq \varepsilon$.*

7 The Color Distance

Color Refinement, also known as the *1-dimensional Weisfeiler-Leman algorithm*, is a heuristic graph isomorphism test. It computes a coloring of the vertices of a graph in a sequence of *refinement rounds*; we say that color refinement *distinguishes* two graphs if the computed color patterns differ. Formally, for a graph G, we let $C^G_0(u) = 1$ for every $u \in V(G)$ and $C^G_{i+1}(u) = \{\!\{C^G_i(v) \mid uv \in E(G)\}\!\}$ for every $i \geq 0$. Let $C^G_\infty = C^G_i$ for the smallest i such that $C^G_i(u) = C^G_i(v) \iff C^G_{i+1}(u) = C^G_{i+1}(v)$ for all $u, v \in G$ ("C_i is stable"). Then, color refinement distinguishes two graphs G and H if there is an $i \geq 0$ such that $\{\!\{C^G_i(v) \mid v \in V(G)\}\!\} \neq \{\!\{C^H_i(v) \mid v \in V(H)\}\!\}$. It is well-known that the partition $\{C^{-1}_\infty(i) \mid i \in C_\infty(V(G))\}$ is the *coarsest equitable partition* of $V(G)$, where a partition Π of $V(G)$ is called *equitable* if for all $P, Q \in \Pi$ and $u, v \in P$, the vertices u and v have the same number of neighbors in Q.

For a graph G, we can define a weighted graph G/C^G_∞ by letting $V(G/C^G_\infty) \coloneqq \{C^{-1}_\infty(i) \mid i \in C_\infty(V(G))\}$, $\alpha_C(G/C^G_\infty) \coloneqq |C|$ for $C \in V(G/C^G_\infty)$, and $\beta_{CD}(G/C^G_\infty) \coloneqq M^G_{CD}/|D|$ for all $C, D \in V(G/C^G_\infty)$, where M^G_{CD} is the number of neighbors a vertex from C has in D, which is the same for all vertices in C as the partition induced by the colors of C^G_∞ is equitable. Note that we have $|C|M^G_{CD} = |D|M^G_{DC}$ as both products describe the number of edges between C and D, i.e., G/C^G_∞ is well-defined. Usually, when talking about the invariant $\mathcal{I}^2_C$ computed by color refinement (see, e.g., [17]), one does not normalize M^G_{CD} by $|D|$. However, by doing so, we do not only get a weighted graph (with symmetric edge weights), but the graphs G and G/C^G_∞ actually have the same tree homomorphism counts. Grebík and Rocha already introduced the graphon analogue $U/\mathcal{C}(U)$ of G/C^G_∞ and proved the same fact for it [13, Corollary 4.3]; hence, we omit the proof.

▶ **Lemma 31.** *Let T be a tree, and let G be a graph. Then, $\mathsf{hom}(T,G) = \mathsf{hom}(T, G/C^G_\infty)$.*

By the result of Dvořák [10], G/C^G_∞ and H/C^H_∞ are isomorphic if and only if G and H have the same tree homomorphism counts. Hence, it is tempting to define a tree distance-like similarity measure on graphs by simply considering the cut distance of G/C^G_∞ and H/C^H_∞.

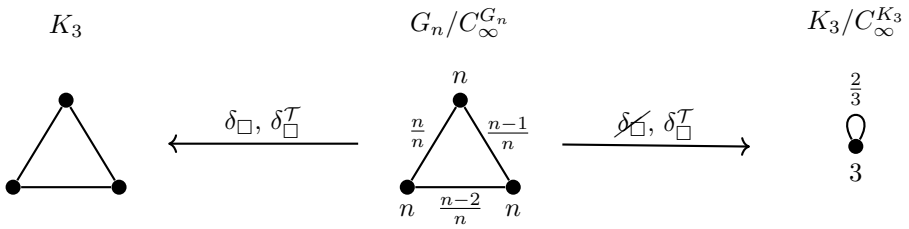


Figure 1 An example separating the color distance from the tree distance.

For graphs G and H, we call $\delta^{\mathcal{C}}_\square(G,H) \coloneqq \delta_\square(G/C^G_\infty, H/C^H_\infty)$ the *color distance* of G and H. As the cut distance $\delta_\square$ is a pseudometric on graphs, so is $\delta^{\mathcal{C}}_\square$. For $\delta^{\mathcal{C}}_\square$, we immediately obtain a quantitative counting lemma from Lemma 2 and Lemma 31.

▶ **Corollary 32** (Counting Lemma for $\delta^{\mathcal{C}}_\square$)**.** *Let T be a tree, and let G and H be graphs. Then, $|t(T,G) - t(T,H)| \leq |E(T)| \cdot \delta^{\mathcal{C}}_\square(G,H)$.*

Clearly, $\delta^{\mathcal{T}}_\square$ and $\delta^{\mathcal{C}}_\square$ have the same graphs of distance zero. Moreover, one can easily verify that the tree distance is bounded from above by the color distance.

▶ **Lemma 33.** *Let G and H be graphs. Then, $\delta^{\mathcal{T}}_\square(G,H) \leq \delta^{\mathcal{C}}_\square(G,H)$.*

Proof. We have $\delta^{\mathcal{T}}_\square(G,H) = \delta^{\mathcal{T}}_\square(G/C^G_\infty, H/C^H_\infty) \leq \delta_\square(G/C^G_\infty, H/C^H_\infty) = \delta^{\mathcal{C}}_\square(G,H)$ by Lemma 31 and Lemma 19. ◀

Now, the obvious question is whether these pseudometrics are the same or, at least, define the same topology. But it is not hard to find a counterexample; the color distance sees differences between graphs that the tree distance and tree homomorphisms do not see. In particular, an inverse counting lemma cannot hold for the color distance. See Figure 1, and for the moment, assume that we can construct a sequence $(G_n)_n$ of graphs such that $G_n/C^{G_n}_\infty$ is as depicted. It is easy to verify that $\delta_\square(G_n/C^{G_n}_\infty, K_3) \xrightarrow{n\to\infty} 0$, and thus, both $\delta^{\mathcal{T}}_\square(G_n,K_3) \xrightarrow{n\to\infty} 0$ and $|t(T,G_n) - t(T,K_3)| \xrightarrow{n\to\infty} 0$ for every tree T. But, $\delta^{\mathcal{C}}_\square(G_n,K_3) \geq \frac{1}{3}\cdot\frac{1}{3}\cdot\frac{2}{3}$ for every n since $G_n/C^{G_n}_\infty$ has a vertex without a loop.

The existence of graphs G_n such that $G_n/C^{G_n}_\infty$ is as depicted in Figure 1 follows easily from inversion results for the color refinement invariant $\mathcal{I}^2_C$. Otto first proved that $\mathcal{I}^2_C$ admits polynomial time inversion on structures [25], and Kiefer, Schweitzer, and Selman gave a simple construction to show that $\mathcal{I}^2_C$ admits linear-time inversion on the class of graphs [17]. Basically, we partition $3n$ vertices into three sets of size n and add edges between these partitions such that they induce n-, $(n-1)$-, and $(n-2)$-regular bipartite graphs.

The example in Figure 1 actually answers an open question of Grebík and Rocha [13, Question 3.1]. They ask whether the set $\{W/\mathcal{C}(W) \mid W \in \widetilde{\mathcal{W}_0}\}$ is closed in $\widetilde{\mathcal{W}_0}$: it is not. With a more refined argument, we can actually show that $\{W_{G/C^G_\infty} \mid G \text{ graph}\}$ is already dense in $\widetilde{\mathcal{W}_0}$. By properly rounding the weights of a given weighted graph, we can turn the inversion result of [17] into a statement about approximate inversion.

▶ **Theorem 34.** *Let H be a weighted graph. For every $n \geq 2\cdot \mathsf{v}(H)$, there is a graph G on n^2 vertices such that $\delta_\square(G/C^G_\infty, H) \leq 3\cdot\mathsf{v}(H)/n + \frac{1}{4}\cdot(\mathsf{v}(H)/n)^2$.*

In Theorem 34, the size of the resulting graph depends on how close we want it to be to the input graph. A simple consequence of the compactness of the graphon space is that, for $\varepsilon > 0$, we can approximate any graphon with an error of ε in $\delta_\square$ by a graph on $N(\varepsilon)$

vertices, where $N(\varepsilon)$ is independent of the graphon [20, Corollary 9.25]. With Theorem 34, this implies that the same is possible with the weighted graphs G/C^G_∞. This also means that the closure of the set $\{W_{G/C^G_\infty} \mid G \text{ graph}\}$ is already $\widetilde{\mathcal{W}_0}$.

8 Conclusions

We have introduced similarity measures for graphs that can be formulated as convex optimization problems and shown surprising correspondences to tree and path homomorphism densities. This takes previous results on the "expressiveness" of homomorphism counts from an exact to an approximate level. Moreover, it helps to give a theoretical understanding of kernel methods in machine learning, which are often based on counting certain substructures in graphs. Proving the correspondences to homomorphism densities was made possible by introducing our similarity measures for the more general case of graphons, where tools from functional analysis let us prove the general statement that every "reasonably defined" pseudometric has to satisfy a correspondence to homomorphism densities.

Various open questions remain. The compactness argument used in Section 4 only yields non-quantitative statements. Hence, we do not know how close the graphs have to be in the pseudometric for their homomorphism densities to be close and vice versa. Only for paths we were able to prove a quantitative counting lemma, which uses the same factor $\mathtt{e}(F)$ as the counting lemma for general graphs. It seems conceivable that a quantitative counting lemma for trees that uses the same factor $\mathtt{e}(T)$ also holds. As the proof of the quantitative inverse counting lemma is quite involved [3, 20], proving such statements for trees and paths should not be easy.

More in reach seems to be the question of how the tree distance generalizes to the class $\mathcal{T}_k$ of graphs of treewidth at most k. Homomorphism counts from graphs in $\mathcal{T}_k$ can also be characterized in terms of linear equations in the case of graphs [10] (see also [7]). How does such a characterization for graphons look like? And how does one define a distance measure from this?

Another open question concerns further characterizations of fractional isomorphism, e.g., the color refinement algorithm, which gives a characterization based on equitable partitions. Can one prove a correspondence between the tree distance and, say, ε-equitable partitions? It is not hard to come up with a definition for such partitions; the hard part is to prove that graphs that are similar in the tree distance possess such a partition.

References

1 Jöran Bergh and Jörgen Löfström. *Interpolation Spaces: An Introduction*. Die Grundlehren Der Mathematischen Wissenschaften in Einzeldarstellungen. Springer-Verlag, 1976. doi:10.1007/978-3-642-66451-9.

2 Christian Borgs, Jennifer Chayes, László Lovász, Vera T. Sós, Balázs Szegedy, and Katalin Vesztergombi. Graph limits and parameter testing. In *Proceedings of the Thirty-Eighth Annual ACM Symposium on Theory of Computing*, STOC '06, page 261–270, New York, NY, USA, 2006. Association for Computing Machinery. doi:10.1145/1132516.1132556.

3 Christian Borgs, Jennifer T. Chayes, László Lovász, Vera T. Sós, and Katalin Vesztergombi. Convergent sequences of dense graphs i: Subgraph frequencies, metric properties and testing. *Advances in Mathematics*, 219(6):1801–1851, 2008. doi:10.1016/j.aim.2008.07.008.

4 Jan Böker. Graph similarity and homomorphism densities, 2021. arXiv:2104.14213.

5 Radu Curticapean, Holger Dell, and Dániel Marx. Homomorphisms are a good basis for counting small subgraphs. In *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing*, STOC 2017, page 210–223, New York, NY, USA, 2017. Association for Computing Machinery. doi:10.1145/3055399.3055502.

6 Víctor Dalmau and Peter Jonsson. The complexity of counting homomorphisms seen from the other side. *Theoretical Computer Science*, 329(1):315–323, 2004. doi:10.1016/j.tcs.2004.08.008.

7 Holger Dell, Martin Grohe, and Gaurav Rattan. Lovász Meets Weisfeiler and Leman. In Ioannis Chatzigiannakis, Christos Kaklamanis, Dániel Marx, and Donald Sannella, editors, *45th International Colloquium on Automata, Languages, and Programming (ICALP 2018)*, volume 107 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 40:1–40:14, Dagstuhl, Germany, 2018. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. doi:10.4230/LIPIcs.ICALP.2018.40.

8 Richard M. Dudley. *Real Analysis and Probability*. Cambridge Studies in Advanced Mathematics. Cambridge University Press, 2 edition, 2002. doi:10.1017/CBO9780511755347.

9 Nelson Dunford and Jacob T. Schwartz. *Linear Operators, Part 2: Spectral Theory, Self Adjoint Operators in Hilbert Space*. Linear Operators. Interscience Publishers, 1963.

10 Zdeněk Dvořák. On recognizing graphs by numbers of homomorphisms. *Journal of Graph Theory*, 64(4):330–342, 2010. doi:10.1002/jgt.20461.

11 Tanja Eisner, Bálint Farkas, Markus Haase, and Rainer Nagel. *Operator Theoretic Aspects of Ergodic Theory*. Graduate Texts in Mathematics. Springer International Publishing, 2015. doi:10.1007/978-3-319-16898-2.

12 Alan Frieze and Ravindran Kannan. Quick approximation to matrices and applications. *Combinatorica*, 19:175–220, February 1999. doi:10.1007/s004930050052.

13 Jan Grebík and Israel Rocha. Fractional isomorphism of graphons, 2021. Accepted to Combinatorica. arXiv:1909.04122.

14 Martin Grohe. Counting bounded tree depth homomorphisms. In *Proceedings of the 35th Annual ACM/IEEE Symposium on Logic in Computer Science*, LICS '20, page 507–520, New York, NY, USA, 2020. Association for Computing Machinery. doi:10.1145/3373718.3394739.

15 Martin Grohe. Word2vec, node2vec, graph2vec, x2vec: Towards a theory of vector embeddings of structured data. In *Proceedings of the 39th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems*, PODS'20, page 1–16, New York, NY, USA, 2020. Association for Computing Machinery. doi:10.1145/3375395.3387641.

16 Svante Janson. Graphons, cut norm and distance, couplings and rearrangements. *New York Journal of Mathematics*, 4, 2013.

17 Sandra Kiefer, Pascal Schweitzer, and Erkal Selman. Graphs identified by logics with counting. In Giuseppe F Italiano, Giovanni Pighizzini, and Donald T. Sannella, editors, *Mathematical Foundations of Computer Science 2015*, pages 319–330, Berlin, Heidelberg, 2015. Springer Berlin Heidelberg. doi:10.1007/978-3-662-48057-1_25.

18 László Lovász. Operations with structures. *Acta Mathematica Hungarica*, 18(3-4):321–328, 1967. doi:10.1007/BF02280291.

19 László Lovász and Balázs Szegedy. Szemerédi's lemma for the analyst. *GAFA Geometric And Functional Analysis*, 17:252–270, 2007. doi:10.1007/s00039-007-0599-6.

20 László Lovász. *Large Networks and Graph Limits*, volume 60 of *Colloquium Publications*. American Mathematical Society, 2012. doi:10.1090/coll/060.

21 László Lovász and Balázs Szegedy. Limits of dense graph sequences. *Journal of Combinatorial Theory, Series B*, 96(6):933–957, 2006. doi:10.1016/j.jctb.2006.05.002.

22 Laura Mančinska and David E. Roberson. Quantum isomorphism is equivalent to equality of homomorphism counts from planar graphs. In *2020 IEEE 61st Annual Symposium on Foundations of Computer Science (FOCS)*, pages 661–672, 2020. doi:10.1109/FOCS46700.2020.00067.

23 Viswanath Nagarajan and Maxim Sviridenko. On the maximum quadratic assignment problem. *Mathematics of Operations Research*, 34(4):859–868, 2009. doi:10.1287/moor.1090.0418.

24 Yurii Nesterov and Arkadii Nemirovskii. *Interior-point Polynomial Algorithms in Convex Programming*. Studies in Applied Mathematics. Society for Industrial and Applied Mathematics, 1994. doi:10.1137/1.9781611970791.

25 Martin Otto. Canonization for two variables and puzzles on the square. *Annals of Pure and Applied Logic*, 85(3):243–282, 1997. doi:10.1016/S0168-0072(96)00047-4.

26 Gottfried Tinhofer. Graph isomorphism and theorems of birkhoff-type. *Computing*, 36(4):285–300, 1986. doi:10.1007/BF02240204.

27 Gottfried Tinhofer. A note on compact graphs. *Discrete Applied Mathematics*, 30(2):253–264, 1991. doi:10.1016/0166-218X(91)90049-3.

Direct Sum and Partitionability Testing over General Groups

Andrej Bogdanov
Department of Computer Science and Engineering and Institute of Theoretical Computer Science and Communications, Chinese University of Hong Kong, China

Gautam Prakriya
Institute of Theoretical Computer Science and Communications,
Chinese University of Hong Kong, China

Abstract

A function $f(x_1, \ldots, x_n)$ from a product domain $\mathcal{D}_1 \times \cdots \times \mathcal{D}_n$ to an abelian group $\mathcal{G}$ is a *direct sum* if it is of the form $f_1(x_1) + \cdots + f_n(x_n)$. We present a new 4-query direct sum test with optimal (up to constant factors) soundness error. This generalizes a result of Dinur and Golubev (RANDOM 2019) which is tailored to the target group $\mathcal{G} = \mathbb{Z}_2$. As a special case, we obtain an optimal *affinity test* for $\mathcal{G}$-valued functions on domain $\{0,1\}^n$ under product measure. Our analysis relies on the hypercontractivity of the binary erasure channel.

We also study the testability of *function partitionability* over product domains into disjoint components. A $\mathcal{G}$-valued $f(x_1, \ldots, x_n)$ is *k-direct sum partitionable* if it can be written as a sum of functions over k nonempty disjoint sets of inputs. A function $f(x_1, \ldots, x_n)$ with unstructured product range $\mathcal{R}^k$ is *direct product partitionable* if its outputs depend on disjoint sets of inputs.

We show that direct sum partitionability and direct product partitionability are one-sided error testable with $O((n-k)(\log n + 1/\epsilon) + 1/\epsilon)$ adaptive queries and $O((n/\epsilon)\log^2(n/\epsilon))$ nonadaptive queries, respectively. Both bounds are tight up to the logarithmic factors for constant ϵ even with respect to adaptive, two-sided error testers. We also give a non-adaptive one-sided error tester for direct sum partitionability with query complexity $O(kn^2(\log n)^2/\epsilon)$.

2012 ACM Subject Classification Theory of computation → Streaming, sublinear and near linear time algorithms; Theory of computation → Randomness, geometry and discrete structures

Keywords and phrases Direct Sum Test, Function Partitionability, Hypercontractive Inequality, Property Testing

Digital Object Identifier 10.4230/LIPIcs.ICALP.2021.33

Category Track A: Algorithms, Complexity and Games

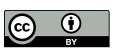
Related Version *Full Version*: `https://eccc.weizmann.ac.il/report/2020/164/` [5]

Funding *Andrej Bogdanov*: Work funded by Hong Kong RGC GRF grants CUHK14207618 and CUHK14209920.

Acknowledgements We thank Chandra Nair and Yan Nan Wang for valuable discussions on the hypercontractivity of the binary erasure channel and an anonymous reviewer for bringing to our attention the works [2] and [14].

1 Introduction

In their seminal result, Blum, Luby and Rubinfeld [4] gave a four query test to determine whether a function $f : \mathbb{F}_2^n \to \mathbb{F}_2$ is affine. We consider a natural generalization of the notion of affinity to functions $f(x_1, \cdots, x_n)$ from $\{0,1\}^n$ to an arbitrary abelian group $\mathcal{G}$: Is f of the form $x_1 \cdot g_1 + \cdots + x_n \cdot g_n + g_0$ for some group elements $g_0, \ldots, g_n \in \mathcal{G}$? The analysis of Blum, Luby and Rubinfeld does not apply unless there is a group homomorphism from the domain to the range.



In this work we give an optimal four query affinity test for functions from $\{0,1\}^n$ to an arbitrary abelian group $\mathcal{G}$.

More generally, our test can be used to determine if a function $f(x_1,\ldots,x_n)$ from a finite product domain $\mathcal{D}_1 \times \cdots \times \mathcal{D}_n$ to an abelian group $\mathcal{G}$ is a *direct sum*, i.e., whether f is of the form $\sum f_i(x_i)$. This resolves a conjecture of Dinur and Golubev [9].

In contrast to the work of Blum, Luby, and Rubinfeld, which was primarily motivated by applications to probabilistically checkable proofs, direct sum testing over general groups arises in the context of testing *function partionability*: Can a multivariate function be decomposed into independent or loosely related components? Bogdanov and Wang [6] discuss the relevance of this question for real-valued functions to the problem of identifying decompositions of control variables in high-dimensional reinforcement learning. In that setting a direct sum decomposition of the advantage function f describes a system that can be partitioned into independent components, which are lower-dimensional and therefore typically easier to learn. An efficient testing algorithm can be used to probe the existence of such a decomposition before any effort is expended into learning it.

In this work we consider the following two natural partitioning problems for discrete functions over product domains $\mathcal{D}_1 \times \cdots \times \mathcal{D}_n$ (endowed with a product distribution):

- A *direct sum partition* ($\oplus$-partition) of f into k components is a representation of the form $f(x_1,\ldots,x_n) = f_1(x_{S_1}) + \cdots + f_k(x_{S_k})$, where $S_1,\ldots,S_k$ are disjoint nonempty sets of variables. Here, the range of f is an abelian group $(\mathcal{G},+)$.
- A *direct product partition* ($\otimes$-partition) of f is a representation of the form $f(x_1,\ldots,x_n) = (f_1(x_{S_1}),\ldots,f_k(x_{S_k}))$, where $S_1,\ldots,S_k$ are disjoint nonempty sets of variables. Here, the range of f is a k-product set $\mathcal{R}^k$.

We are interested in the query complexity of testing partitionability: Given oracle access to f and parameters k,ϵ, how many queries does it take to tell whether f is partitionable or ϵ-far from partitionable?

The related tasks of *direct product testing* and *direct sum testing* ask for the existence of such representations under a known (fixed) partition of inputs. Motivated by applications to probabilistically checkable proofs, Dinur and Steurer [10] and Dickstein and Dinur [8] analyze a 2-query direct product test of essentially optimal soundness.

The query complexity of direct sum testing for $\mathbb{Z}_2$-valued functions, that is of testing whether a function $f\colon \mathcal{D}_1 \times \cdots \times \mathcal{D}_n \to \mathbb{Z}_2$ is of the form $f(x_1,\ldots,x_n) = f_1(x_1)+\cdots+f_n(x_n)$, was recently resolved by Dinur and Golubev [9]. They proposed and analysed a 4-query test of optimal (up to constant factors) soundness error. Their tester does not naturally extend to functions valued in arbitrary abelian groups.

Bogdanov and Wang [6] proposed an agnostic learning algorithm for unknown direct sum partitions. As a consequence of their analysis they concluded that $\oplus$-partitionability is testable with $O(kn^3/\epsilon)$ non-adaptive queries. They also showed that $\Omega(n-k+1)$ queries are necessary for constant ϵ. To the best of our knowledge $\otimes$-partitionability has not been studied before.

Our Results

We analyze a new 4-query direct sum test for functions valued over arbitrary abelian groups. The test is based on the following dual characterization: $f\colon \mathcal{D}_1 \times \cdots \times \mathcal{D}_n \to \mathcal{G}$ is a direct sum $f_1(x_1) + \cdots + f_n(x_n)$ if and only if $D_f(S,\overline{S};x,y) = 0$ for all pairs of inputs x,y and partitions $(S,\overline{S})$ of $[n]$, where

$$D_f(S,\overline{S};x,y) = f(x) - f(y_S x) - f(y_{\overline{S}} x) + f(y).$$

Here and in the rest of the manuscript, $y_S x$ is the string in $\mathcal{D}_1 \times \cdots \times \mathcal{D}_n$ that matches y in the S-coordinates and x in the other coordinates. We assume that the domain $\mathcal{D}_1 \times \cdots \times \mathcal{D}_n$ is furnished with a product distribution: For x chosen at random from $\mathcal{D}_1 \times \cdots, \times \mathcal{D}_n$, the coordinates $x_1, \ldots, x_n$ are independent.

The tester accepts if $D_f(S, \overline{S}; x, y) = 0$ for random independent inputs $x, y \in \mathcal{D}_1 \times \cdots \times \mathcal{D}_n$ and a uniformly random partition $(S, \overline{S})$ of $[n]$. Our main result is an optimal (up to constant factor) bound on the soundness error $\rho(f) = \Pr[D_f(S, \overline{S}; x, y) \neq 0]$ of this test in terms of the distance $\delta(f) = \min_g\{\Pr_x[f(x) \neq g(x)] \colon g \text{ is a direct sum}\}$.

▶ **Theorem 1.** *There is an absolute constant $c > 0$ such that for every collection of finite sets $\mathcal{D}_1, \ldots, \mathcal{D}_n$, every abelian group $\mathcal{G}$, and every $f \colon \mathcal{D}_1 \times \cdots \times \mathcal{D}_n \to \mathcal{G}$, $\rho(f) \geq c \cdot \delta(f)$.*

An important special case of the theorem concerns the Boolean domain $\mathcal{D}_1 = \cdots = \mathcal{D}_n = \{0, 1\}$ under the uniform distribution (Proposition 5). The class of direct sums from $\{0, 1\}^n$ to $\mathcal{G}$ is then precisely the class of affine functions $f(x) = x_1 g_1 + \cdots + x_n g_n + g_0$ for some group elements $g_0, g_1, \ldots, g_n \in \mathcal{G}$.

Using Theorem 1, we obtain the following upper bound on the query complexity of $\oplus$-partitionability.

▶ **Theorem 2.** *Direct sum partitionability over any abelian group is one-sided testable with $O((n-k)(\log n + 1/\epsilon) + 1/\epsilon)$-queries.*

We also prove an upper bound on the query complexity of $\otimes$-partitionability:

▶ **Theorem 3.** *Direct product partitionability is one-sided testable with $O((n/\epsilon)\log^2(n/\epsilon))$ non-adaptive queries.*

The testers in Theorem 2 and 3 are time-efficient.

By the lower bound of Bogdanov and Wang [6], the $\oplus$-partitionability tester is tight up to the $\log n$ factor for constant ϵ. In the special case when $k = n$, direct sum partitionability reduces to direct sum testing and the query complexity is the same as that of Theorem 1.

Our tester for $\oplus$-partitionability is adaptive. We also give a one-sided non-adaptive tester of query complexity $O(kn^2(\log n)^2/\epsilon)$. A non-adaptive lower bound of $\Omega((n-k+1)\,(\log(n-k+1)/\epsilon^c)\log(\log(n-k+1)/\epsilon^c))$ for any $c > 1$ follows from the work of Servedio et al. [13] on junta testing. As in the case of juntas, it follows that adaptivity helps in testing $\oplus$-partitionability for some settings of parameters.

The $\otimes$-partitionability tester is also nearly tight: We show that direct product partitionability requires $\Omega(n)$ queries for $\epsilon = 1/2$ for adaptive testers, and $\Omega(\frac{n}{\epsilon \log(1/\epsilon)})$ queries for non-adaptive testers, for every $k \geq 2$.

The non-adaptive test for $\oplus$-partitionability, and the lower bounds for $\oplus$-partitionability and $\otimes$-partitionability are deferred to the full version of this paper [5].

Ideas and Techniques

Direct sum testing over general groups

The main ingredient of Dinur and Golubev's direct sum tester for $\mathbb{Z}_2$-valued functions is an implicit reduction from general product domains to the Boolean domain $\{0, 1\}^n$ under the uniform distribution. We abstract and generalize their reduction. In interest of space we defer the details of this reduction to full version of this paper [5]. To complete their proof, Dinur and Golubev instantiate the reduction with the $\mathbb{Z}_2$-affinity test of Blum, Luby, and Rubinfeld [4].

Table 1 Summary of algorithmic results. All tests have one-sided error.

Property	Our Results	Prior Work
Direct Sum: $f : \mathcal{D}_1 \times \cdots \times \mathcal{D}_n \to \mathcal{G}$ s.t. $f(x) = f(x_1) + \cdots + f(x_n)$	4-query test for arbitrary abelian group $\mathcal{G}$ (Theorem 1)	4-query test for $\mathcal{G} = \mathbb{Z}_2$ [9]
k-$\oplus$-partitionability: $f : \mathcal{D}_1 \times \cdots \times \mathcal{D}_n \to \mathcal{G}$ s.t. $\exists S_1, \ldots, S_k \subseteq [n]$, $f(x) = f(x_{S_1}) + \cdots + f(x_{S_k})$	$O((n-k)(\log n + 1/\epsilon) + 1/\epsilon)$-query adaptive test (Theorem 2) $O(kn^2(\log n)^2/\epsilon)$-query non-adaptive test (See full version [5])	$O(kn^3/\epsilon)$-query non-adaptive test [6]
k-$\otimes$-partitionability: $f : \mathcal{D}_1 \times \cdots \times \mathcal{D}_n \to \mathcal{R}^k$ s.t. $\exists S_1, \ldots, S_k \subseteq [n]$, $f(x) = (f(x_{S_1}), \ldots, f(x_{S_k}))$	$O((n/\epsilon)\log^2(n/\epsilon))$-query non-adaptive test (Theorem 3)	

Our main technical contribution is a tight analysis of the affinity test D_f applied to functions $f\colon \{0,1\}^n \to \mathcal{G}$ valued in an arbitrary abelian group $\mathcal{G}$. To give a sense why the test is sound, let us argue that $\rho(f) = \Omega(\delta(f))$ under the additional assumption that f is close to a direct sum, say if $\delta = \delta(f) \leq 1/27$.

Let B be the set of measure at most $1/27$ on which f differs from its closest direct sum. We claim that conditioned on $x \in B$, the probability that any of the other test queries y, $y_S x$, $y_{\overline{S}} x$ land in B is at most $\delta + 2\delta^{1/3}$. By independence, the probability that $y \in B$ conditioned on $x \in B$ is exactly δ. In contrast, $y_S x$ and $y_{\overline{S}} x$ are not independent of x, but can be sampled by processing x through a binary symmetric channel with crossover probability $1/4$. The bound $\Pr[y_S x \in B | x \in B] \leq \delta^{1/3}$ follows from the small-set expansion of this channel [1], which is equivalent to the hypercontractivity of the corresponding Markov operator [7]. Since the event "$x \in B$ and $y_S x \notin B$ and $y_{\overline{S}} x \notin B$ and $y \notin B$" results in rejection, it follows that $\rho(f) \geq \delta \cdot (1 - \delta - 2\delta^{1/3})$, which is at least $\frac{8}{27}\delta$ by the closeness assumption on f.

For larger values of $\delta(f)$, our proof strategy is to argue that f can be *decoded* to a direct sum function by making at most $O(\rho(f))$ "changes" to the truth-table of f. The decoding algorithm we analyze in Lemma 6 is *iterative plurality* (i.e., iterative maximum likelihood). We show that the function

$$\phi(x) = \text{plurality}_{S,y}\ f(y_S x) + f(y_{\overline{S}} x) - f(y) \tag{1}$$

is, on the one hand, $2\delta(f)$-close to f, and on the other hand, has substantially smaller rejection probability of $\rho(\phi) \leq \rho(f)/2$. By iterating the decoding, i.e. applying the plurality to ϕ again, we arrive at a function that is $4\delta(f)$ close to f and passes the test with probability one, thus must equal a direct product.

This argument is inspired by the linearity test analysis of Blum, Ruby, and Rubinfeld (BLR), who also decode f to a function that is, on the one hand, close to f and, on the other hand, passes their test with probability 1. However, unlike the BLR decoder which yields a linear function after a single round of self-correction, ours inherently requires multiple iterations. For example, if f is a direct sum corrupted on all inputs with relative hamming weight around $1/4$, then $\phi(0)$ is unlikely to be correctly decoded (as $y_S 0$ and $y_{\overline{S}} 0$ will typically be corrupted) and so will typically be inconsistent with a direct sum.

Nevertheless, the high-level structure of our argument closely parallels the BLR analysis. First, in Claim 10 we show that for all but an $o(\rho)$-fraction of inputs x, the plurality in (1) is a strong majority consistent with 99% of the choices of $(S, \overline{S})$ and y. Second, we use the algebraic structure of our test (Claim 9) to show that if $D_\phi(S, \overline{S}; x, y) \neq 0$ then

$D_f(U, V; w, z) \neq 0$ for a substantially larger fraction of query sequences $(w, z_U w, z_{\overline{U}} w, z)$ that can be sampled by applying suitable "noise" to $(S, \overline{S}; x, y)$. If we represent the partition $(S, \overline{S})$ by a binary string $\sigma \in \{0,1\}^n$ (with 1 and 0 indicating memberships in S and $\overline{S}$, respectively), we show that the relevant noise can be modeled by independent fixed-probability *erasures* applied to the symbols of σ, x, and y. Using hypercontractivity bounds for the binary erasure channel [11], we conclude that ϕ fails the test on a significantly smaller fraction of queries than f does.

In the special case when the target group is $\mathbb{Z}_2$, the soundness error of D_f can be directly shown to be within a constant factor of the soundness error of the Dinur-Golubev tester (even though the two tests are different). The main motivating applications for function partitionability, however, concern real-valued functions [6]. The analysis of our $\oplus$-partionability testers for such functions relies on Theorem 1.

The idea of soundness analysis by iterative plurality decoding was introduced by Ben-Sasson et al. [2] and used by Shpilka and Wigderson [14] in the context of randomness-efficient linearity testing.

Testing partitionability

The main ingredient in our $\oplus$-partitionability algorithms is the direct 2-sum test D_f. The structure of this test allows us to efficiently detect a pair of variables x_s, x_t that must fall in the same component of the partition in any far from $\oplus$-partitionable function, effectively reducing the instance size by one variable.

Our $\otimes$-partitionability test looks for an input variable that is influential in at least two of the output coordinates of f. The analysis of this test is based on Lemma 24, which states that such a variable must exist in any far from partitionable function.

Organization

Section 2 outlines the proof of Theorem 1 in the case when the domain is the Boolean hypercube. The analysis is based on the convergence of the iterative decoder (Lemma 6), which is proved in Section 3. To prove Theorem 1 we use a reduction from testing functions over arbitrary product domains to testing functions on the hypercube (See the full version [5] for details of this reduction). Sections 4 and 5 describe and analyze the partitionability testers for direct sum and direct product, respectively.

Definitions and Notation

Let $\mathcal{D} \doteq \mathcal{D}_1 \times \ldots \times \mathcal{D}_n$ be a finite set. For strings $x, y \in \mathcal{D}$ and a set of indices $S \subseteq [n]$, let x_S to refer to the projection of x onto the coordinates in S. For strings $x^{(1)}, \ldots x^{(k)} \in \mathcal{D}$, and a partition $S_1, \ldots, S_k$ of $[n]$, let $x^{(1)}_{S_1} \ldots x^{(k)}_{S_k}$ be the string in $\mathcal{D}$ that is identical to $x^{(i)}$ on indices in S_i. For a bipartition $(S, \overline{S})$, we often write $x_S y$ instead of $x_S y_{\overline{S}}$.

In sections 2 and 3 we identify a bipartition $(S, \overline{S})$ of $[n]$ with its indicator vector $\sigma \in \{0,1\}^n$, and write $D_f(\sigma; x, y)$ instead of $D_f(S, \overline{S}; x, y)$, and x_σ instead of x_S.

We extend the definition of D_f to pairs of disjoint sets (S, T) that do not necessarily partition $[n]$ as

$$D_f(S, T; x, y) \doteq f(x) - f(y_S x) - f(y_T x) + f(y_{S \cup T} x).$$

2 Direct Sum Test for Functions on the Boolean Hypercube

The following dual characterization of direct sums motivates our test.

▶ **Fact 4.** *A function $f : \{0,1\}^n \to \mathcal{G}$ is a direct sum if and only if $D_f(\pi; x, y) = 0$ for every choice of $x, y, \pi \in \{0,1\}^n$.*

Proof of Fact 4. The "only if" direction is immediate from the definition of a direct sum. We prove the "if" direction. Let f be such that $D_f(\pi; x, y) = 0$ for every choice of $x, y, \pi \in \{0,1\}^n$. Fix $y \in \{0,1\}^n$. For every $x \in \{0,1\}^n$ we can write $f(x)$ as

$$\begin{aligned} f(x) &= f(x_{\{1\}}y) + f(y_{\{1\}}x) - f(y) \\ &= f(x_{\{1\}}y) + f(x_{\{2\}}y) + f(y_{\{1,2\}}x) - 2f(y) \\ &\vdots \\ &= f(x_{\{1\}}y) + f(x_{\{2\}}y) + \ldots + f(x_{\{n\}}y) - (n-1)f(y). \end{aligned}$$

Therefore, f is a direct sum. ◀

Algorithm 1 Direct sum test for functions over $\{0,1\}^n$.

Oracle : $f\colon \{0,1\}^n \to \mathcal{G}$

1 Sample $x, y, \pi \in \{0,1\}^n$ independently and uniformly at random.
2 If $f(x) + f(y) - f(x_\pi y) - f(y_\pi x) = 0$, **accept**.
3 Else, **reject**.

By Fact 4, the test accepts every direct sum with probability 1. The following proposition establishes soundness of the test. Let $\rho(f)$ denote the probability that Algorithm 1 rejects the function f. That is, $\rho(f) \doteq \Pr_{x,y,\pi}[D_f(\pi; x, y) \neq 0]$.

▶ **Proposition 5** (Soundness). *There exist a universal constant $\eta \in [0,1]$ such that for every function $f : \{0,1\}^n \to \mathcal{G}$,*

$$\rho(f) \geq \min(\delta/4, \eta),$$

where δ is the distance between f and the set of direct sums.

▶ **Lemma 6** (Iterative decoding). *There exists a universal constant $\eta \in [0,1]$ such that for every function $f : \{0,1\}^n \to \mathcal{G}$ with $\rho(f) < \eta$, there exists a function $\phi : \{0,1\}^n \to \mathcal{G}$ such that:*

(i) *the function ϕ is $2\rho(f)$-close to f, and*
(ii) $\rho(\phi) \leq \rho(f)/2$.

Proof of Proposition 5. Iteratively applying Lemma 6 results in a sequence of functions $f = f_0, f_1, \ldots$, such that for all $t \geq 1$, (i) the distance between f_t and f_{t-1} is at most $2\rho(f_{t-1})$, and (ii) $\rho(f_t) \leq \rho(f_{t-1})/2$. The probability that the test rejects a function is a discrete quantity. So, by (ii), there must exist an integer t such that $\rho(f_t) = 0$. That is, $D_{f_t}(\pi; x, y) = 0$ for every choice of $x, y, \pi \in \{0,1\}^n$. By Fact 4 this means f_t is a direct sum. The distance between f and the direct sum f_t at most

$$\sum_{i=0}^{t-1} 2\rho(f_i) \leq 2\sum_{i=0}^{t-1} \rho(f)/2^i \leq 4\rho(f).$$ ◀

3 Analysis of Iterative Decoding

We begin with a sketch of the proof of Lemma 6. As mentioned in the introduction, the proof follows in the footsteps of the analysis of the BLR linearity test. We define $\phi(x)$ to be $\text{plurality}_{y,\pi} f(x_\pi y) + f(y_\pi x) - f(y)$. Markov's inequality allows us to bound the distance between ϕ and f by $2\rho(f)$.

To show that Test 1 rejects ϕ with probability at most $\rho(f)/2$, we first show that for all but a $o(\rho(f))$-fraction of choices of x, $\phi(x)$ is defined by a strict majority that makes up at least 6/7-th of the plurality vote (See Claim 10). The fraction of x's that contribute to the plurality is at least the probability of a collision, i.e., $\Pr_{y,z,\sigma,\pi}[f(x_\pi y) + f(y_\pi x) - f(y) = f(x_\sigma z) + f(z_\sigma x) - f(z)]$. Using the algebraic identity in Claim 8, we can express this probability as

$$\Pr_{y,z,\pi,\sigma}[D_f(\pi; x_\pi z_{\overline{\pi}}, y) - D_f(\pi; y_\pi x_{\overline{\pi}}, z) + D_f(\pi \oplus \sigma; x_\sigma z_{\overline{\sigma}}, z_\sigma x_{\overline{\sigma}}) = 0].$$

The analysis of the BLR test also uses an analogous algebraic identity to bound the collision probability. The difference is that the resulting expression in the BLR analysis is made up of evaluations of the BLR test at points independent of x. This allows one to argue that the plurality vote is made up of a strict majority at all points x. In our setting, the arguments of D_f in the expression above are correlated with x. However, we can view these arguments as the result of passing x through a noisy binary erasure channel. This allows for the application of the hypercontractive inequality to bound the fraction of x for which the collision probability is less than 6/7.

We then show that for all but $o(\rho(f))$ choices of $x, y, \pi \in \{0,1\}^n$ there exist $z, w, \sigma \in \{0,1\}^n$ such that, (A) the value of $D_\phi(\pi; x, y) = \phi(x) - \phi(x_\pi y) - \phi(y_\pi x) + \phi(y)$ does not change after the following substitutions, and (B) the resulting expression post substitution evaluates to zero.

$$\begin{aligned}
\phi(x) &\leftarrow f(x_\sigma z) + f(z_\sigma x) - f(z) \\
\phi(x_\pi y) &\leftarrow f((x_\pi y)_\sigma (z_\pi w)) + f((z_\pi w)_\sigma (x_\pi y)) - f(z_\pi w) \\
\phi(y_\pi x) &\leftarrow f((y_\pi x)_\sigma (w_\pi z)) + f((w_\pi z)_\sigma (y_\pi x)) - f(w_\pi z) \\
\phi(y) &\leftarrow f(y_\sigma w) + f(w_\sigma y) - f(y).
\end{aligned} \tag{2}$$

It follows that the probability that ϕ is rejected by the test is $o(\rho(f))$.

By Claim 10, for all but $o(\rho(f))$ choices of x, y, π the substitutions do not change the value of $D_\phi(\pi; x, y)$ with probability at least 4/7. For (B), we use the algebraic identity in Claim 9 to rewrite the expression after substitution as

$$D_f(\pi \oplus \sigma; x_\sigma z, y_\sigma w) - D_f(\pi \oplus \overline{\sigma}; x_{\overline{\sigma}} z, y_{\overline{\sigma}} w) + D_f(\pi; z, w).$$

Again we show that the arguments of the D_f terms can be viewed as the result of passing x, y and π through independent binary erasure channels. Using the hypercontractive inequality, we conclude that for all but $o(\rho(f))$ choices of x, y and π the expression after substitution evaluates to zero for most choices of z, w and σ. By a union bound we can ensure that (A) and (B) hold simultaneously for the same z, w and σ.

The following technical lemma establishes the bounds we prove using the hypercontractivity of the binary erasure channel. The proof is presented in Section 3.1.

Let $\text{QUERIES}(\pi; x, y)$ denote the vector in $(\{0,1\}^n)^4$ whose entries are the four queries that Algorithm 1 makes when π, x, y is sampled. That is, $\text{QUERIES}(\pi; x, y) = (x, x_\pi y, y_\pi x, y)$.

▶ **Lemma 7.** *Let* $B\textsc{ad} \subset (\{0,1\}^n)^4$ *be a set such that the probability that* $\textsc{queries}(\pi, x, y)$ *lands in* $B\textsc{ad}$*, when* π, x, y *are chosen independently and uniformly at random, is* ρ*.*

(i) $\mu_x(A_1) \le 21^2\rho^{4/3}$*, where*

$$A_1 = \{x \mid \Pr_{\pi,y,z}[\textsc{queries}(\pi; x_\pi z_{\overline{\pi}}, y) \in B\textsc{ad}] \ge 1/21\}.$$

(ii) $\mu_x(A_2) \le 21^2\rho^{4/3}$*, where*

$$A_2 = \{x \mid \Pr_{\pi,\sigma,z}[\textsc{queries}(\pi \oplus \sigma; x_\sigma z_{\overline{\sigma}}, z_\sigma x_{\overline{\sigma}}) \in B\textsc{ad}] \ge 1/21\}.$$

(iii) $\mu_{\pi,x,y}(A_3) \le 7^2\rho^{2/(1+\sqrt{2/3})} \le 7^2\rho^{1.1}$*, where*

$$A_3 = \{(\pi, x, y) \mid \Pr_{\sigma,z,w}[\textsc{queries}(\pi \oplus \sigma; x_\sigma z, y_\sigma w) \in B\textsc{ad}] \ge 1/7\}.$$

(iv) $\mu_\pi(A_4) \le 7^2\rho^{4/3}$*, where*

$$A_4 = \{\pi \mid \Pr_{z,w}[\textsc{queries}(\pi; z, w) \in B\textsc{ad}] \ge 1/7\}.$$

We will also need the following algebraic identities.

▷ Claim 8. The following identity holds:

$$D_f(\pi; x, y) - D_f(\sigma; x, z) = D_f(\pi; x_\pi z_{\overline{\pi}}, y) - D_f(\pi; y_\pi x_{\overline{\pi}}, z) + D_f(\pi \oplus \sigma; x_\sigma z_{\overline{\sigma}}, z_\sigma x_{\overline{\sigma}}).$$

Proof of Claim 8. The claim follows by adding the following two identities:

$$\begin{aligned} D_f(\pi; x, y) - D_f(\pi; x, z) &= -f(x_\pi y_{\overline{\pi}}) - f(y_\pi x_{\overline{\pi}}) + f(y) + f(x_\pi z_{\overline{\pi}}) + f(z_\pi x_{\overline{\pi}}) - f(z) \\ &= f(x_\pi z_{\overline{\pi}}) + f(y) - f(x_\pi y_{\overline{\pi}}) - f(y_\pi z_{\overline{\pi}}) \\ &\quad - f(y_\pi x_{\overline{\pi}}) - f(z) + f(y_\pi z_{\overline{\pi}}) + f(z_\pi x_{\overline{\pi}}) \\ &= D_f(\pi; x_\pi z_{\overline{\pi}}, y) - D_f(\pi; y_\pi x_{\overline{\pi}}, z) \end{aligned}$$

$$\begin{aligned} D_f(\pi; x, z) - D_f(\sigma; x, z) &= -f(x_{\pi\sigma} x_{\pi\overline{\sigma}} z_{\overline{\pi}\sigma} z_{\overline{\pi\sigma}}) - f(z_{\pi\sigma} z_{\pi\overline{\sigma}} x_{\overline{\pi}\sigma} x_{\overline{\pi\sigma}}) + f(x) + f(z) \\ &\quad + f(x_{\pi\sigma} z_{\pi\overline{\sigma}} x_{\overline{\pi}\sigma} z_{\overline{\pi\sigma}}) + f(z_{\pi\sigma} x_{\pi\overline{\sigma}} z_{\overline{\pi}\sigma} x_{\overline{\pi\sigma}}) - f(x) - f(z) \\ &= D_f(\pi \oplus \sigma; x_\sigma z_{\overline{\sigma}}, z_\sigma x_{\overline{\sigma}}) \end{aligned}$$

◀

To analyze the substitutions (2) we set $D_{\phi,f}(\pi; x, y) = \phi(x) - f(x_\pi y) - f(y_\pi x) + f(x)$. In particular, $D_{f,f} = D_f$.

▷ Claim 9 (16-point identity). The following identity holds:

$$\begin{aligned} &D_{\phi,f}(\sigma; x, z) - D_{\phi,f}(\sigma; x_\pi y, w_\pi z) - D_{\phi,f}(\sigma; y_\pi x, z_\pi w) + D_{\phi,f}(\sigma; y, w) \\ &\qquad = D_\phi(\pi; x, y) - D_f(\pi \oplus \sigma; x_\sigma z, y_\sigma w) - D_f(\pi \oplus \overline{\sigma}; x_{\overline{\sigma}} z, y_{\overline{\sigma}} w) + D_f(\pi; z, w). \end{aligned}$$

Proof of Claim 9. We write $xyzw$ to denote the string $x_{\pi\sigma} y_{\pi\overline{\sigma}} z_{\overline{\pi}\sigma} w_{\overline{\pi\sigma}}$. With this notation,

$$\begin{array}{lllll} +D_{\phi,f}(\sigma; x, z) & = +\phi(xxxx) & -f(xzxz) & -f(zxzx) & +f(zzzz) \\ -D_{\phi,f}(\sigma; x_\pi y, w_\pi z) & = -\phi(xxyy) & +f(xwyz) & +f(wxzy) & -f(wwzz) \\ -D_{\phi,f}(\sigma; y_\pi x, z_\pi w) & = -\phi(yyxx) & +f(yzxw) & +f(zywx) & -f(zzww) \\ +D_{\phi,f}(\sigma; y, w) & = +\phi(yyyy) & -f(ywyw) & -f(wywy) & +f(wwww) \\ & \quad \| & \| & \| & \| \\ & +D_\phi(\pi; x, y) & -D_f(\pi \oplus \sigma; x_\sigma z, y_\sigma w) & -D_f(\pi \oplus \overline{\sigma}; x_{\overline{\sigma}} z, y_{\overline{\sigma}} w) & +D_f(\pi; z, w) \end{array}$$

The identity states that the column sums and the row sums add up. ◁

Proof of Lemma 6. Let ϕ be a function defined as $\phi(x) = \text{plurality}_{y,\pi}\, f(x_\pi y) + f(y_\pi x) - f(y)$. That is, $\phi(x)$ is the most frequent value of $f(x_\pi y) + f(y_\pi x) - f(y)$, where $y, \pi \in \{0,1\}^n$. Ties are broken arbitrarily. We show that ϕ satisfies the hypothesis of the lemma.

(i) ϕ is $2\rho(f)$-close to f: For $x \in \{0,1\}^n$, let $\rho_x \doteq \Pr_{y,\pi}[f(x) \neq f(x_\pi y) + f(y_\pi x) - f(y)]$. Note that $\mathbb{E}_x[\rho_x] = \rho(f)$, and that if $\rho_x < 1/2$ then $f(x) = \phi(x)$. Thus, by Markov's inequality,

$$\Pr_x[f(x) \neq \phi(x)] \leq \Pr_x[\rho_x \geq 1/2] \leq 2\rho(f).$$

(ii) $\rho(\phi) \leq \rho(f)/2$: We begin by showing that with probability $\rho(f)/12$ over the choice of x, the plurality that defines $\phi(x)$ is a majority made up of 6/7-th of the votes. Then $\Pr_{\pi,y}[D_{\phi,f}(\pi; x, y) = 0]$ is the fraction of votes that constitute the plurality defining $\phi(x)$. Let

$$\textsc{Weak-Maj} = \left\{x \;\middle|\; \Pr_{y,\pi}[D_{\phi,f}(\pi; x, y) \neq 0] \geq 1/7\right\}.$$

▷ Claim 10 (Strong Majority). $\mu_x(\textsc{Weak-Maj}) \leq \rho(f)/12$.

Proof. The fraction of votes that contribute to the plurality $\Pr_{y,\pi}[D_{\phi,f}(\pi; x, y) = 0]$ is an upper bound on the collision probability $\Pr_{y,\pi,z,\sigma\in\{0,1\}^n}[D_{\phi,f}(\pi; x, y) = D_{\phi,f}(\sigma; x, z)]$. This is because

$$\begin{aligned}\Pr_{y,\pi,z,\sigma}[D_{\phi,f}(\pi; x, y) = D_{\phi,f}(\sigma; x, z)] &= \sum_{\gamma\in\mathcal{G}} \Pr_{y,\pi}[D_{\phi,f}(\pi; x, y) = \gamma]^2 \\ &\leq \max_{\gamma\in\mathcal{G}} \Pr_{y,\pi}[D_{\phi,f}(\pi; x, y) = \gamma] \\ &= \Pr_{y,\pi}[D_{\phi,f}(\pi; x, y) = 0].\end{aligned}$$

The final equality holds because $\phi(x) = \arg\max_{\beta\in\mathcal{G}} \Pr_{y,\pi}[\beta - f(x_\pi y) - f(y_\pi x) + f(y) = 0]$. We showed that

$$\mu_x(\textsc{Weak-Maj}) \leq \mu_x\{x \mid \Pr_{y,\pi,z,\sigma}[D_{\phi,f}(\pi; x, y) \neq D_{\phi,f}(\sigma; x, z)] \geq 1/7\}.$$

We now use Lemma 7 to bound the right hand side. Let $\textsc{Bad}_f \subset (\{0,1\}^n)^4$ be the set of queries on which D_f fails, namely

$$\textsc{Bad}_f = \{\textsc{queries}(\pi; x, y) \mid D_f(\pi; x, y) \neq 0\}.$$

This is a set of measure $\mu_{\pi,x,y}(\textsc{Bad}) = \rho(f)$. Since $D_{\phi,f}(\pi; x, y) - D_{\phi,f}(\sigma; x, z) = D_f(\pi; x, y) - D_f(\sigma; x, z)$, by the algebraic identity in Claim 8 and a union bound, we have

$$\begin{aligned}\mu_x(\textsc{Weak-Maj}) &\leq \mu_x\{x \mid \Pr_{\pi,\sigma,y,z}[D_{\phi,f}(\pi; x, y) - D_{\phi,f}(\sigma; x, z) \neq 0] \geq 1/7\} \\ &\leq \mu_x\{x \mid \Pr_{\pi,y,z}[D_f(\pi; x_\pi z_{\overline{\pi}}, y) \neq 0] \geq 1/21\} \\ &\quad + \mu_x\{x \mid \Pr_{\pi,y,z}[D_f(\pi; y_\pi x_{\overline{\pi}}, z) \neq 0] \geq 1/21\} \\ &\quad + \mu_x\{x \mid \Pr_{\pi,\sigma,z}[D_f(\pi \oplus \sigma; x_\sigma z_{\overline{\sigma}}, z_\sigma x_{\overline{\sigma}}) \neq 0] \geq 1/21\} \\ &= \mu_x(A_1) + \mu_x(A_1) + \mu_x(A_2),\end{aligned}$$

where A_1 and A_2 are the sets

$$A_1 = \{x \mid \Pr_{\pi,y,z}[\textsc{queries}(\pi; x_\pi z_{\overline{\pi}}, y) \in \textsc{Bad}_f] \geq 1/21\},$$

$$A_2 = \{x \mid \Pr_{\pi,\sigma,z}[\textsc{queries}(\pi \oplus \sigma; x_\sigma z_{\overline{\sigma}}, z_\sigma x_{\overline{\sigma}}) \in \textsc{Bad}_f] \geq 1/21\}.$$

By Lemma 7 we get that $\mu_x(\textsc{Weak-Maj}) \leq \rho(f)/12$, for small enough η. ◁

We are now ready to prove that $\rho(\phi) = \Pr_{\pi,x,y}[D_\phi(\pi;x,y) \neq 0] \leq \rho(f)/2$. In order to do so we define a set $\textsc{Bad}_\phi$ of triples (π,x,y) such that $\mu_{\pi,x,y}(\textsc{Bad}_\phi) \leq \rho(f)/2$, and if $(\pi,x,y) \notin \textsc{Bad}_\phi$ then $D_\phi(\pi,x,y) = 0$.

Let A_3 and A_4 denote the sets

$$A_3 = \{(\pi,x,y) \mid \Pr_{\sigma,z,w}[\textsc{queries}(\pi \oplus \sigma; x_\sigma z, y_\sigma w) \in \textsc{Bad}_f] \geq 1/7\},$$

$$A_4 = \{\pi \mid \Pr_{z,w}[\textsc{queries}(\pi; z, w) \in \textsc{Bad}_f] \geq 1/7\}.$$

Let $\textsc{Bad}_\phi$ be the set

$$\{(\pi,x,y) \mid (\text{One of } x, y, x_\pi y, y_\pi x \text{ lies in } \textsc{Weak-Maj}) \text{ or } ((\pi,x,y) \in A_3) \text{ or } (\pi \in A_4)\}.$$

By Lemma 7, $\mu_{\pi,x,y}(A_3) \leq \rho(f)/12$, and $\mu_\pi(A_4) \leq \rho(f)/12$, for small enough η. As $x, y, x_\pi y, y_\pi x$ are all random, by a union bound we have

$$\mu_{\pi,x,y}(\textsc{Bad}_\phi) \leq 4\mu_x(\textsc{Weak-Maj}) + \mu_{\pi,x,y}(A_3) + \mu_\pi(A_4) \leq \rho(f)/2.$$

All that remains to show is that if $(\pi,x,y) \notin \textsc{Bad}_\phi$, $D_\phi(\pi;x,y) = 0$. On rearranging the terms in the algebraic identity of Claim 9, we get

$$\begin{aligned} D_\phi(\pi;x,y) = \; & D_{\phi,f}(\sigma;x,z) - D_{\phi,f}(\sigma;x_\pi y, w_\pi z) - D_{\phi,f}(\sigma;y_\pi x; z_\pi w) + D_{\phi,f}(\sigma;y,w) \\ & + D_f(\pi \oplus \sigma; x_\sigma z, y_\sigma w) + D_f(\pi \oplus \overline{\sigma}; x_{\overline{\sigma}} z, y_{\overline{\sigma}} w) - D_f(\pi;z,w). \end{aligned} \tag{3}$$

Fix a triple $(\pi,x,y) \notin \textsc{Bad}_\phi$. We show that $D_\phi(\pi,x,y) = 0$, by showing that there exists a choice of σ, z and w for which the right hand side of Equation (3) evaluates to zero. By Equation (3) and a union bound,

$$\begin{aligned} \Pr_{\sigma,z,w}[D_\phi(\pi;x,y) \neq 0] &= \Pr_{\sigma,z,w}\begin{bmatrix} D_{\phi,f}(\sigma;x,z) \neq 0 \\ \text{or } D_{\phi,f}(\sigma;x_\pi y, w_\pi z) \neq 0 \\ \text{or } D_{\phi,f}(\sigma;y_\pi x; z_\pi w) \neq 0 \\ \text{or } D_{\phi,f}(\sigma;y,w) \neq 0 \\ \text{or } D_f(\pi \oplus \sigma; x_\sigma z, y_\sigma w) \neq 0 \\ \text{or } D_f(\pi \oplus \overline{\sigma}; x_{\overline{\sigma}} z, y_{\overline{\sigma}} w) \neq 0 \\ \text{or } D_f(\pi;z,w) \neq 0 \end{bmatrix} \\ &< 4/7 + \Pr_{\sigma,z,w}\begin{bmatrix} \textsc{queries}(\pi \oplus \sigma; x_\sigma z, y_\sigma w) \in \textsc{Bad}_f \\ \text{or } \textsc{queries}(\pi \oplus \overline{\sigma}; x_{\overline{\sigma}} z, y_{\overline{\sigma}} w) \in \textsc{Bad}_f \\ \text{or } \textsc{queries}(\pi;z,w) \in \textsc{Bad}_f \end{bmatrix} \\ &< 4/7 + 3/7 = 1. \end{aligned}$$

The first inequality holds because $x, x_\pi y, y_\pi x, y \notin \textsc{Weak-Maj}$, and the second inequality holds because $(\pi,x,y) \notin A_3$ and $\pi \notin A_4$. Since the probability $\Pr_{\sigma,z,w}[D_\phi(\pi;x,y) \neq 0]$ is either 0 or 1, it must be that $D_\phi(\pi;x,y) = 0$. Therefore,

$$\rho(\phi) = \Pr_{\pi,x,y}[D_\phi(\pi;x,y) \neq 0] \leq \mu_{\pi,x,y}(\textsc{Bad}_\phi) \leq \rho(f)/2. \qquad \blacktriangleleft$$

3.1 Proof of Lemma 7

We begin with some preliminaries on discrete channels and hypercontractivity. For a motivating discussion on hypercontractivity and a proof of Fact 14 below see Chapter 9 of [12].

▶ **Definition 11** (Discrete channels). *A* discrete channel *is a triple* $(\mathcal{U}, P, \mathcal{V})$*, where* $\mathcal{U}$ *and* $\mathcal{V}$ *are finite sets representing the* input alphabet *and* output alphabet*, and* P *is a* $\mathcal{U} \times \mathcal{V}$ *probability* transition matrix *that describes the distribution of the output conditioned on the input. The* composition *of two channels* $(\mathcal{U}, P_1, \mathcal{V})$ *and* $(\mathcal{V}, P_2, \mathcal{W})$ *is the channel* $(\mathcal{U}, P_1 \cdot P_2, \mathcal{W})$*, where* $\cdot$ *is matrix multiplication.*

The binary erasure channel will play an important role in the proof of Lemma 7.

▶ **Definition 12** (Binary Erasure Channel). *The* binary erasure channel $BEC(e)$ *with erasure probability* e *has input alphabet* $\{0,1\}$ *and output alphabet* $\{0,1,\perp\}$*, and probability transition matrix* $P(x|x) = 1-e, P(\perp|x) = e$*.*

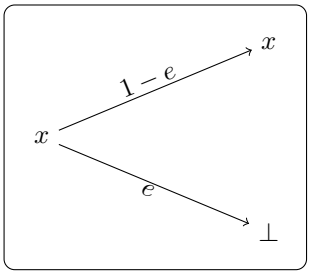


Figure 1 The binary erasure channel $BEC(e)$.

For a real valued random variable U and $p \geq 1$, we denote the p-norm of U by $\|U\|_p \doteq \mathbb{E}_U[|U|^p]^{1/p}$.

▶ **Definition 13** (Hypercontractivity). *For* $1 \leq q \leq p$*, A pair of random variables* (U, V) *is* (p,q)*-hypercontractive if for every pair of real valued functions* f, g*,*

$$\mathbb{E}[f(U)g(V)] \leq \|f(U)\|_{p'} \|\|g(V)\|_q,$$

where $p' = p/(p-1)$ *is the Hölder conjugate of* p*.*

▶ **Fact 14** (Tensorisation [7]). *If* (U_1, V_1) *and* (U_2, V_2) *are independent random variables that are* (p,q)*-hypercontractive, then* $((U_1, U_2), (V_1, V_2))$ *is* (p,q) *hypercontractive.*

▶ **Theorem 15** (Hypercontractivity of $BEC(e)$ [11]). *Let* U *be distributed uniformly over* $\{0,1\}$ *and let* $V \in \{0,1,\perp\}$ *denote the output of* $BEC(e)$ *on input* U*. Then* (U, V) *is* (p,q)*-hypercontractive for all* $1 \leq q \leq p$ *such that*

$$\frac{q-1}{p-1} \geq 1 - e.$$

▶ **Fact 16** (Composition). *Let* $(\mathcal{U}, P_1, \mathcal{V})$ *and* $(\mathcal{V}, P_2, \mathcal{W})$ *be two channels. Let* U *be a random variable over* $\mathcal{U}$*. Let* V *be the random variable that represents the output of the first channel on input* U*, and* W *the random variable that represents the output of the second channel on input* V*. If* (U, V) *is* (p,q)*-hypercontractive then so is* (U, W)*.*

Proof of Fact 16. Let $f : \mathcal{U} \to \mathbb{R}$ and $g : \mathcal{W} \to \mathbb{R}$ be arbitrary functions. Since $U \to V \to W$ is a markov chain, we have

$$\mathbb{E}_{U,W}[f(U)g(W)] = \mathbb{E}_{U,V}[f(U)E_W[g(W) \mid V]] \leq \|f(U)\|_{p'} \|\mathbb{E}_W[g(W) \mid V]\|_q,$$

where the inequality holds because (U, V) is (p,q) hypercontractive.

Now, by Jensen's inequality,

$$\mathbb{E}_V[\mathbb{E}_W[g(W) \mid V]^q]^{1/q} \leq \mathbb{E}_V[\mathbb{E}_W[g(W)^q \mid V]]^{1/q} = \mathbb{E}_W[g(W)^q]^{1/q} = \|g(W)\|_q$$

Therefore, (U, W) is (p, q) hypercontractive. ◀

The following claim captures the small-set expansion interpretation of hypercontractivity [1] in the form used in the proof of Lemma 7.

▷ Claim 17. Let U, V be random variables that take values in $\mathcal{U}$ and $\mathcal{V}$ respectively. Let $B \subset \mathcal{V}$ be a set such that $\Pr[V \in B] = \rho$. Let $A \subset \mathcal{U}$ denote the set $\{u \mid \Pr[V \in B \mid U = u] \geq \theta\}$. If (U, V) is (p, q) hypercontractive, then $\Pr[U \in A] \leq \rho^{p/q}/\theta^p$.

Proof. Let 1_A and 1_B denote the indicator functions of the sets A and B. Since (U, V) are (p, q) hypercontractive,

$$\theta \cdot \Pr[U \in A] \leq \Pr[V \in B \mid U \in A] \Pr[U \in A] = \mathbb{E}[1_A(U)1_B(V)] \leq \|1_A(U)\|_{p'} \|1_B(V)\|_q,$$

where $p' = p/(p-1)$. Note that $\|1_B(V)\| = \rho^{1/q}$, and $\|1_A(U)\|_{p'} = \Pr[U \in A]^{1/p'}$. Therefore, $\Pr[U \in A]^{1/p} \leq \rho^{1/q}/\theta$, that is, $\Pr[U \in A] \leq \rho^{p/q}/\theta^p$. ◁

Proof of Lemma 7. We need to bound the probabilities of four sets of the form

$$\{u \in \Sigma^n \mid \Pr[\textsc{queries}(\psi(u)) \in \textsc{bad} | U = u]\} \geq \theta,$$

where ψ is some (randomized) function. All bounds of the form $\theta^{-2}\rho^{2/q}$ will follow from Claim 17 by showing that the channel $U \to \textsc{queries}(\psi(u))$ is $(2, q)$-hypercontractive for a suitable choice of q ($q = 3/2$ for parts (i), (ii), (iv) and $q = 1 + \sqrt{2/3}$ for part (iii)).

The *channel* $(\Sigma^n, P_n, \{0,1\}^{n\times 4})$ that maps $u \in \Sigma^n$ to $\textsc{queries}(\psi(u)) \in \{0,1\}^{n\times 4}$ acts independently on the symbols $u_1, \ldots, u_n$. In all cases, the i-th bits of the four queries (q_1, q_2, q_3, q_4) are obtained by applying the one-dimensional channel P_1 to u_i. Therefore, P_n tensorizes as $P_n = P_1^{\otimes n}$. By Fact 14, it is sufficient to show that the channel P_1 is hypercontractive. We may and will therefore assume, without loss of generality, that $n = 1$.

We now demonstrate how each of the four channels of interest can be decomposed into a binary erasure channel with constant erasure probability ($e = 1/2$ in parts (i), (ii), (iv) and $e = 1 - \sqrt{2/3}$ in part (iii)) and some other fixed channel. The Lemma then follows from Fact 16 and Theorem 15 with $q = 2 - e$.

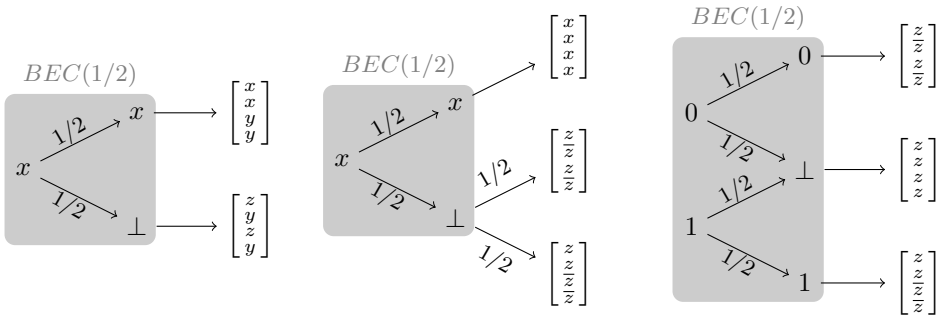


Figure 2 Channels (i) $x \to \textsc{queries}(\pi, x_\pi z_{\overline{\pi}}, y)$; (ii) $x \to \textsc{queries}(\pi \oplus \sigma, x_\sigma z_{\overline{\sigma}}, z_\sigma x_{\overline{\sigma}})$; (iv) $\pi \to \textsc{queries}(\pi, z, w)$. y and z are random bits.

(i) The channel $x \to \text{QUERIES}(\pi; x_\pi z_{\overline{\pi}}, y) = (x_\pi z_{\overline{\pi}}, x_\pi y_{\overline{\pi}}, y_\pi z_{\overline{\pi}}, y)$ from $\Sigma = \{0,1\}$ to $\{0,1\}^4$ can be decomposed in the following way: On input x the channel samples a random bit π and outputs $xxyy$ if $\pi = 1$, and $zyzy$ if $\pi = 0$ for random y and z. This channel can be alternatively described as $BEC(1/2)$ composed with a second channel that outputs $xxyy$ if there is no erasure and the independent symbol $zyzy$ otherwise. See Figure 2 (i).

(ii) The channel from $\Sigma = \{0,1\}$ to $\{0,1\}^4$ is of the form

$$x \to \text{QUERIES}(\pi \oplus \sigma, x_\sigma z_{\overline{\sigma}}, z_\sigma x_{\overline{\sigma}}) = \begin{pmatrix} x_{\pi\sigma} z_{\pi\overline{\sigma}} x_{\overline{\pi}\sigma} z_{\overline{\pi\sigma}} \\ z_{\pi\sigma} z_{\pi\overline{\sigma}} x_{\overline{\pi}\sigma} x_{\overline{\pi\sigma}} \\ x_{\pi\sigma} x_{\pi\overline{\sigma}} z_{\overline{\pi}\sigma} z_{\overline{\pi\sigma}} \\ z_{\pi\sigma} x_{\pi\overline{\sigma}} z_{\overline{\pi}\sigma} x_{\overline{\pi\sigma}} \end{pmatrix} = \begin{cases} xzxz, & \text{if } \pi\sigma = 1, \\ zzxx, & \text{if } \pi\overline{\sigma} = 1, \\ xxzz, & \text{if } \overline{\pi}\sigma = 1, \\ zxzx, & \text{if } \overline{\pi\sigma} = 1, \end{cases}$$

where π, σ, z are random bits. We can alternatively describe it like this: If $z = x$, then output $xxxx$. If $z \neq x$ and $\pi \oplus \sigma = 1$, then output $zz\overline{z}\overline{z}$. If $z \neq x$ and $\pi \oplus \sigma = 0$, then output $zzzz$.

This channel can be factored through $BEC(1/2)$ as in Figure 2 (ii). If there is no erasure, the second channel outputs $xxxx$. If there is an erasure, then the second channel outputs $zz\overline{z}\overline{z}$ with probability $1/2$ and $z\overline{z}z\overline{z}$ with probability $1/2$.

(iii) The channel from $\Sigma = \{0,1\}^3$ to $\{0,1\}^4$ is of the form

$$\begin{pmatrix} \pi \\ x \\ y \end{pmatrix} \to \text{QUERIES}(\pi \oplus \sigma; x_\sigma z, y_\sigma w) = \begin{pmatrix} x_{\pi\sigma} z_{\pi\overline{\sigma}} x_{\overline{\pi}\sigma} z_{\overline{\pi\sigma}} \\ y_{\pi\sigma} z_{\pi\overline{\sigma}} x_{\overline{\pi}\sigma} w_{\overline{\pi\sigma}} \\ x_{\pi\sigma} w_{\pi\overline{\sigma}} y_{\overline{\pi}\sigma} z_{\overline{\pi\sigma}} \\ y_{\pi\sigma} w_{\pi\overline{\sigma}} y_{\overline{\pi}\sigma} w_{\overline{\pi\sigma}} \end{pmatrix} = \begin{cases} xyxy, & \text{if } \pi\sigma = 1, \\ zzww, & \text{if } \pi\overline{\sigma} = 1, \\ xxyy, & \text{if } \overline{\pi}\sigma = 1, \\ zwzw, & \text{if } \overline{\pi\sigma} = 1. \end{cases}$$

Consider the composition of the following two channels. The first channel views the symbol πxy as three bits and independently applies $BEC(1/4)$ to π and $BEC(1 - \sqrt{2/3})$ to x and y. The second channel is described in Figure 3.

- If π is erased, the second channel outputs $zzzz$, for a uniform bit z. This corresponds to the event $z = w$ and $\sigma = 0$.
- If π is not erased but one of x, y is erased, the second channel samples a uniform bit $z \in \{0,1\}$ and outputs $z\overline{z}z\overline{z}$ if $\pi = 0$ and $zz\overline{z}\overline{z}$ if $\pi = 1$. This corresponds to the event $z \neq w$ and $\sigma = 0$.
- If there are no erasures, then the second channel outputs $xyxy$ if $\pi = 1$, and $xxyy$ if $\pi = 0$. This corresponds to the event $\sigma = 1$.

The first channel is $BEC(\frac{1}{4}) \otimes BEC(1 - \sqrt{2/3}) \otimes BEC(1 - \sqrt{2/3})$. Since $1 - \sqrt{2/3} \leq 1/4$, by Fact 14 it inherits the hypercontractivity parameters of $BEC(1 - \sqrt{2/3})$.

(iv) The channel $\pi \to \text{QUERIES}(\pi, z, w)$ from $\Sigma = \{0,1\}$ to $\{0,1\}^4$ outputs $zzww$ if $\pi = 1$ and $zwzw$ if $\pi = 0$ for random bits z and w. Alternatively, the channel can be described as a uniform choice between $zzzz$ and $zz\overline{z}\overline{z}$ when $\pi = 1$ and a uniform choice between $zzzz$ and $z\overline{z}z\overline{z}$ when $\pi = 0$. This can be modeled as the composition of $BEC(1/2)$ and a second channel that outputs $zzzz$ if there is an erasure, and either $zz\overline{z}\overline{z}$ or $z\overline{z}z\overline{z}$ depending on the value of π_i otherwise. See Figure 2 (iv). ◀

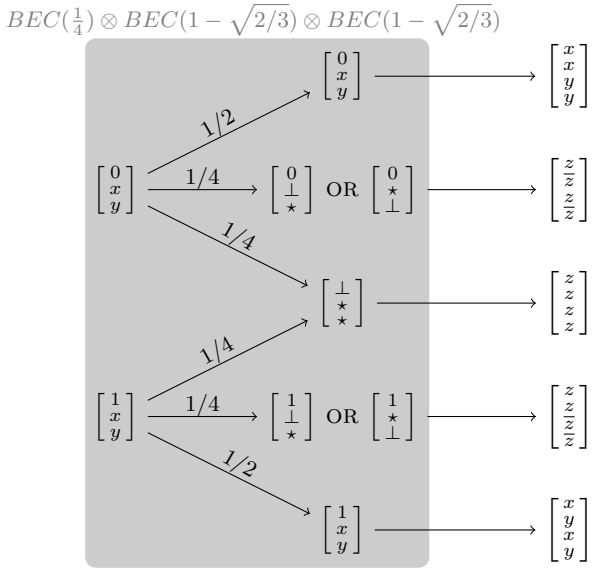


Figure 3 (iii) Channel $(\pi, x, y) \to \text{QUERIES}(\pi \oplus \sigma, x_\sigma z, y_\sigma w)$. A $\star$ represents any of $\{0, 1, \perp\}$ and z is a random bit.

4 Testing $\oplus$-Partitionability

Recall that a function $f : \mathcal{D} = \mathcal{D}_1 \times \cdots \times \mathcal{D}_n \to \mathcal{G}$ is *k-$\oplus$-partitionable* if there exists a k-partition $S_1, \ldots, S_k$ of $[n]$, and functions $f_1, \ldots, f_k$ such that $f(x) = f_1(x_{S_1}) + \ldots + f_k(x_{S_k})$, for all $x \in \mathcal{D}$.

Recall that for disjoint sets $S, T \subseteq [n]$,

$$D_f(S, T; x, y) \doteq f(x) - f(y_S x) - f(y_T x) + f(y_{S \cup T} x)$$

The following claim is an immediate consequence of Theorem 1, and allows us to determine whether a function f is $\oplus$-partitionable with respect to a fixed partition $S_1, \ldots, S_k$.

▷ Claim 18. Let $(S, \overline{S})$ be a random coarsening of a k-partition $(S_1, \ldots, S_k)$ obtained by adding each S_i to S with probability $1/2$. If f is ϵ-far from $\oplus$-partitionable with respect to $S_1, \ldots, S_k$, then $D_f(S, \overline{S}; x, y)$ is nonzero with probability $\Omega(\epsilon)$.

To determine whether a function is k-$\oplus$-partitionable, our testers use the 4-query test D_f to group together variables that cannot occur in different partition components. If the tester finds fewer than k groups, it rejects, otherwise it accepts.

4.1 Adaptive Test for $\oplus$-Partitionability

Our test for k-$\oplus$-partitionability (Algorithm 3) seeks to identify a pair of *contractable* variables s, t that must fall in the same component of a partition. Variables s and t are then contracted and the test is repeated until either fewer than k variables are left (giving a certificate of non-partionability) or no contractable candidates can be found.

A sufficient condition for contractability is that $D_f(\{s\}, \{t\}; x, y)$ is nonzero for some assignment x, y. We start by splitting the variables into k components $S_1, \ldots, S_k$ arbitrarily and zero-testing $D_f(S, \bar{S}; x, y)$ for a random coarsening of the components into $S, \bar{S}$. By Claim 18, the zero-test fails with probability at least $\Omega(\epsilon)$, where ϵ is the distance between f and the set of functions that are $\oplus$-partitionable with respect to $S_1, \ldots, S_k$.

Once such a bipartition $S, \bar{S}$ is identified, s and t can be identified via binary search using Algorithm 2 below. The same idea was used by Blais [3] to identify an influential variable in his junta test. Our t is in fact the influential variable in the function $g(x_{[n]\setminus S}) = f(x) - f(y_S x)$, for fixed x_S, y_S, returned by Blais' test.

Algorithm 2 Violating pair adaptive search.

Oracle : $f\colon \mathcal{D}_1 \times \cdots \times \mathcal{D}_n \to \mathcal{G}$
Input : $(S, T; x, y)$ such that $D_f(S, T; x, y) \neq 0$.
Output : $(\{s\}, \{t\})$ such that $D_f(\{s\}, \{t\}; x', y') \neq 0$ for some x', y'.

If $|S| = |T| = 1$, output S, T.
If $|T| = 1$, swap S and T.
do
  Split T into two subsets T' and T'' of (almost) equal size.
  If $D_f(S, T'; x, y) \neq 0$, recursively run on input $(S, T'; x, y)$.
  Otherwise, recursively run on input $(S, T''; y, x)$.

The correctness of Algorithm 2 is based on the following identity.

▷ Claim 19. $D_f(S, T \cup T'; x, y) = D_f(S, T; x, y) + D_f(S, T'; y, x)$ for disjoint sets S, T, T'.

Proof. Without loss of generality take $S = \{1\}$, $T = \{2\}$, $T' = \{3\}$ and assume there are no other inputs (they are all fixed). By the definition of D_f,

$$\begin{aligned} D_f(\{1\}, \{2\}; x, y) &= f(x_1x_2x_3) + f(y_1y_2x_3) - f(x_1y_2x_3) - f(y_1x_2x_3) \\ D_f(\{1\}, \{3\}; y, x) &= f(y_1y_2y_3) + f(x_1y_2x_3) - f(y_1y_2x_3) - f(x_1y_2y_3) \\ -D_f(\{1\}, \{2,3\}; x, y) &= -f(x_1x_2x_3) - f(y_1y_2y_3) + f(x_1y_2y_3) + f(y_1x_2x_3). \end{aligned}$$

The terms on the right hand side cancel out. ◁

▶ **Lemma 20.** *Algorithm 2 is correct and has query complexity at most* $4(\lceil \log |S| \rceil + \lceil \log |T| \rceil)$.

Proof. The correctness follows from Claim 19 and from the symmetry of D_f in the S, T inputs. As for the query complexity, the algorithm makes four queries (in fact at most two additional queries) in each iteration, and each iteration shrinks one of the original inputs S, T by half. ◀

In the following algorithm we let $\mathcal{P}(S_1, \ldots, S_k)$ be the distribution on disjoint pairs of sets $(S, \bar{S})$ from Claim 18.

Algorithm 3 Adaptive tester for k-$\oplus$-paritionability.

Oracle : $f\colon \mathcal{D}_1 \times \ldots \times \mathcal{D}_n \to \mathcal{G}$
Input : Size k of partition

If f has fewer than k variables, output "not partitionable".
Otherwise, partition variables arbitrarily into k sets $S_1, \ldots, S_k$.
repeat
  Choose sets $(S, \bar{S})$ at random from $\mathcal{P}(S_1, \ldots, S_k)$.
  Choose random inputs x, y.
until $D_f(S, T; x, y) \neq 0$;
Run violating pair adaptive search on input $(S, \bar{S}; x, y)$ to obtain outputs $\{s\}, \{t\}$.
Contract variables s and t in the oracle and repeat.

Proof of Theorem 2. We analyze Algorithm 3. First assume f is k-$\oplus$-partitionable. By Lemma 20, f only contracts variables s, t that are not split by the partition (otherwise $D_f(\{s\}, \{t\}; x', y')$ always vanishes). Therefore f cannot be contracted down to $k-1$ inputs and the tester accepts with probability one.

Now assume f is ϵ-far from partitionable. We will argue that Algorithm 3 outputs "not partitionable" after $O((n-k+1)(\log n + 1/\epsilon))$ queries in expectation by induction on n. Assume $n \geq k$. By Claim 18, Loop 6 takes $O(1/\epsilon)$ iterations to complete in expectation, and each iteration costs four queries to f. By Lemma 20, line 7 takes another $O(\log n)$ queries. After merging s and t the resulting function on $n-1$ inputs can only be farther from partitionable, so by inductive assumption the expected query complexity $Q(n)$ is at most $Q(n-1) + O(\log n + 1/\epsilon)$. This gives the desired bound. By Markov's inequality, Algorithm 3 makes at most twice this number of queries with probability at least half.

The query complexity can be improved slightly to the stated bound $O((n-k)(\log n + 1/\epsilon) + 1/\epsilon)$ by observing that the violating pair search in line 7 can be bypassed when $n = k$ since a proof of non-partitionability has already been discovered in line 6. ◀

5 Testing $\otimes$-Partitionability

Recall that a function $f : \mathcal{D} = \mathcal{D}_1 \times \cdots \times \mathcal{D}_n \to \mathcal{R}^k$ is *k-$\otimes$-partitionable* if there exists a k-partition $S_1, \ldots, S_k$ of $[n]$, and functions $f_1, \ldots, f_k$ such that $f(x) = (f_1(x_{S_1}), \ldots, f_k(x_{S_k}))$ for all $x \in \mathcal{D}$.

In this section we present a $O((n/\epsilon)\log^2(n/\epsilon))$-query non-adaptive one-sided error test for $\otimes$-partitionability (see Theorem 3). We begin with an overview of the construction.

First, some notation. For a function $f : \mathcal{D} \to \mathcal{R}^k$ and a subset $T \subseteq [k]$ we write f_T to refer to the function obtained by projecting the output of f onto the coordinates in T. We often write x_i instead of $x_{\{i\}}$ and f_j instead of $f_{\{j\}}$.

For simplicity, suppose that $k = 2$. Then f is 2-$\otimes$-partitionable if and only if every variable has non-zero influence on at most one of the two coordinates of f. So our task boils down to determining whether there is a variable that is influential in both coordinates of the output of f. The key observation that allows us to find such a coordinate with a small number of queries is that if f is ϵ-far from $\otimes$-partitionable, then

$$\sum_{i \in [n]} \min(\mathrm{Inf}(i; f_1), \mathrm{Inf}(i; f_2)) \geq \epsilon. \tag{4}$$

Therefore, if f is ϵ-far from $\otimes$-partitionable, there must be a coordinate that has influence at least ϵ/n in both coordinates. This immediately suggests an $O(n^2/\epsilon)$ query test: for each variable use $O(n/\epsilon)$ queries to determine whether it is influential in both coordinates.

We obtain an improvement in the query complexity by exploiting a trade-off between the number of samples, $i \in [n]$, required to find a variable that is influential in both coordinates, and the number of samples required to certify that a variable is indeed influential in both coordinates.

Given subsets $S_1, \ldots, S_k$ of $[n]$, let $\Delta_f(S_1, \ldots, S_k)$ be the distance from $f = (f_1, \ldots, f_k)$ to the closest function $g = (g_1, \ldots, g_k)$ in which g_j does not depend on the inputs in S_j, and define the *influence* of $(S_1, \ldots, S_k)$ on f as

$$\mathrm{Inf}(S_1, \ldots, S_k; f) = \Pr[f_j(x) \neq f_j(y_{S_j} x) \text{ for some } j],$$

where x, y is an independent pair of inputs.

▶ **Proposition 21.** $\mathrm{Inf}(S_1,\dots,S_k;f) \le \sum_{i=1}^n \mathrm{Inf}(i;f_{J(i)})$, *where* $J(i)$ *is the set of output coordinates* $j \in [k]$ *for which* S_j *contains* i.

Proof. Let E be the event "$f_j(x) \neq f_j(y_{S_j}x)$ for some j". Let h^i be the hybrid input in which $h^i_t = y_t$ for $t \le i$ and x_t for $t > i$. Then $h^0 = x$ and $h^n = y$. If the event "$f_j(x) \neq f_j(y_{S_j}x)$" occurs, then one of the events "$f_j(h^{i-1}_{S_j}x) \neq f_j(h^i_{S_j}x)$" must occur for some i between 1 and n. By the union bound, $\Pr(E) \le \sum_{i=1}^n \Pr(E_i)$, where E_i is the event "$f_j(h^{i-1}_{S_j}x) \neq f_j(h^i_{S_j}x)$ for some j." The inputs $h^{i-1}_{S_j}x$ and $h^i_{S_j}x$ are identical unless $i \in S_j$, in which case they differ only in the i-th coordinate where they are independent. Therefore

$$\Pr(E_i) = \Pr[f_j(x) \neq f_j(x^i) \text{ for some } j \in J(i)],$$

where x^i is x with its i-th input resampled independently. The right hand side is precisely the influence of i in $f_{J(i)}$. ◀

▷ Claim 22. $\Delta_f(S_1,\dots,S_k) \le \mathrm{Inf}(S_1,\dots,S_k;f)$.

Proof. By averaging, there must exist an assignment a to y such that

$$\mathrm{Inf}(S_1,\dots,S_k;f) \ge \Pr[f_j(x) \neq f_j(a_{S_j}x) \text{ for some } j].$$

Define $g_j(x) = f_j(a_{S_j}x)$ (on all inputs). Then g_j does not depend on the inputs in S_j, so

$$\Delta_f(S_1,\dots,S_k) \le \Pr[f_j(x) \neq g_j(x) \text{ for some } j] = \Pr[f_j(x) \neq f_j(a_{S_j}x) \text{ for some } j] \le \Pr(E).$$

◁

▷ Claim 23. Let $k \ge 2$ and $f(x_i) = (f_1(x_i),\dots,f_k(x_i))$ be a possibly randomized univariate function. Let d an the output coordinate that maximizes $\mathrm{Inf}(i;f_d)$. There exists a partition $(P,\overline{P})$ of the output coordinates such that $\mathrm{Inf}(i;f_P)$ and $\mathrm{Inf}(i;f_{\overline{P}})$ are both at least $\mathrm{Inf}(i;f_{[n]\setminus\{d\}})/3$.

Proof. Let I_j be the event $f_j(x) \neq f_j(y)$ for random independent x and y. Then $\mathrm{Inf}(i;f_T) = \Pr(\cup_{j\in T} I_j)$. Let $\delta_i = \Pr(\cup_{j\neq d} I_j)$. If $\Pr(I_d) \ge \delta_i/3$ then the partition $(\{d\},[n]\setminus\{d\})$ satisfies the conclusion. Otherwise, $\Pr(I_j) \le \delta_i/3$ for all j. Then some partition of type $(I_1 \cup \cdots \cup I_j, I_{j+1} \cup \cdots \cup I_k)$ works: If j is the first set for which $\Pr(I_1 \cup \cdots \cup I_j)$ exceeds $\delta_i/3$, then

$$\Pr(I_1 \cup \cdots \cup I_j) \le \Pr(I_1 \cup \cdots \cup I_{j-1}) + \Pr(I_j) \le 2\delta_i/3.$$

Since

$$\Pr(I_1 \cup \cdots \cup I_j) + \Pr(I_{j+1} \cup \cdots \cup I_k) \ge \Pr(\cup_j I_j) \ge \delta_i,$$

the event $I_{j+1} \cup \cdots \cup I_k$ also has probability at least $\delta_i/3$. ◁

▶ **Lemma 24.** *If* f *is* δ*-far from* $\otimes$*-partitionable then there exist partitions* $(P(1),\overline{P(1)}),\dots,(P(n),\overline{P(n)})$ *of* $[k]$ *such that*

$$\sum_{i=1}^n \min\{\mathrm{Inf}(i;f_{P(i)}),\mathrm{Inf}(i;f_{\overline{P(i)}})\} \ge \frac{\delta}{3}.$$

Proof. Let $j^*(i)$ be the maximizer of $\mathrm{Inf}(i; f_j)$ (breaking ties arbitrarily), and S_j be the set of all i such that $j^*(i) \neq j$. Then $J(i) = \{j \colon i \in S_j\} = [n] \setminus \{j^*(i)\}$. By Proposition 21 and Claim 22, $\delta \leq \Delta_f(S_1, \ldots, S_k) \leq \sum \mathrm{Inf}(i; f_{[n]\setminus\{j^*(i)\}})$. By Claim 23 applied to f as a function of x_i only (randomized over the other inputs), $\mathrm{Inf}(i; f_{[n]\setminus\{j^*(i)\}})/3 \leq \min\{\mathrm{Inf}(i; f_{P(i)}), \mathrm{Inf}(i; f_{\overline{P(i)}})\}$. ◀

Algorithm 4 Non-adaptive tester for $\otimes$-partitionability.

Oracle: $f : \mathcal{D} = \mathcal{D}_1 \times \cdots \times \mathcal{D}_n \to \mathcal{R}^k$
Input : Proximity parameter ϵ
1 **foreach** $r \in \{0, \ldots, \lceil \log(3n/\epsilon) \rceil\}$ **do**
2 Let $S \subseteq [n]$ be a set of $3 \cdot \lceil \frac{6n \log(3n/\epsilon)}{2^r \epsilon} \rceil$ indices sampled uniformly at random from $[n]$.
3 **foreach** $i \in S$ **do**
4 Sample $3 \cdot 2^{r+1}$ independent pairs of inputs from $\mathcal{D}$.
5 **if** $\exists$ *samples* $(x, y), (x', y')$, *and* $j \neq j' \in [k]$ *such that* $f_j(x) \neq f_j(y_{\{i\}}x)$ *and* $f_{j'}(x') \neq f_{j'}(y'_{\{i\}}x')$ **then**
6 Reject.
7 Accept.

Proof of Theorem 3. We show that Algorithm 4 satisfies the statement of the theorem. In each iteration of the outer loop, $O(n/\epsilon \log(n/\epsilon))$ queries are made to f. Thus, in total the algorithm makes $O((n/\epsilon)\log^2(n/\epsilon))$ queries.

The test has perfect completeness because the condition on Line 5 is never triggered if f is a direct product.

We now argue soundness. If f is ϵ-far from being a direct product, then by Lemma 24, for every $i \in [n]$ there exist partitions $(P(i), \overline{P(i)})$ such that

$$\sum_{i \in [n]} M_i \geq \epsilon/3, \tag{5}$$

where $M_i = \min\{\mathrm{Inf}(i; f_{P(i)}), \mathrm{Inf}(i; f_{\overline{P(i)}})\}$.

For $r \in \{0, \ldots, \lceil \log(3n/\epsilon) \rceil\}$, let A_r denote the set $\{i \mid M_i \in [1/2^r, 1/2^{r+1})\}$. By (5) and an averaging argument, we know that there exists an ℓ such that $|A_\ell| \geq \lceil \frac{2^\ell \epsilon}{6 \log(3n/\epsilon)} \rceil$. For such an ℓ, we show that the probability that the algorithm rejects in the ℓ-th iteration is at least $2/3$.

Consider the ℓ-th iteration of the outer loop. The probability that no index in A_ℓ is picked at Line 2 is at most $(1 - \frac{|A_\ell|}{n})^{3 \cdot \frac{n}{|A_\ell|}} \leq 1/e^3$.

In an iteration of the inner loop corresponding to an index $i \in A_\ell$, the probability that either $f_{P(i)}(x) = f_{P(i)}(y)$ for all sampled pairs (x, y), or $f_{\overline{P(i)}}(x) = f_{\overline{P(i)}}(y)$ for all sampled pairs (x, y) is at most $2 \cdot (1 - 1/2^{\ell+1})^{3 \cdot 2^{\ell+1}} \leq 2/e^3$. This tells us that the probability that the algorithm rejects in the ℓ-th iteration of the outer loop conditioned on $A_\ell \cap S \neq \emptyset$ is at least $(1 - 2/e^3)$. Since $A_\ell \cap S$ is empty with probability at most $1/e^3$, the probability that the algorithm rejects in the ℓ-th iteration is at least $(1 - 3/e^3) \geq 2/3$. ◀

References

1. Rudolf Ahlswede and Peter Gacs. Spreading of sets in product spaces and hypercontraction of the markov operator. *Ann. Probab.*, 4(6):925–939, December 1976. doi:10.1214/aop/1176995937.
2. Eli Ben-Sasson, Madhu Sudan, Salil Vadhan, and Avi Wigderson. Randomness-efficient low degree tests and short PCPs via epsilon-biased sets. In *Proceedings of the Thirty-Fifth Annual ACM Symposium on Theory of Computing*, STOC '03, pages 612–621, New York, NY, USA, 2003. Association for Computing Machinery. doi:10.1145/780542.780631.
3. Eric Blais. Testing juntas nearly optimally. In *Proceedings of the 41st Annual ACM Symposium on Theory of Computing, STOC 2009, Bethesda, MD, USA, May 31 - June 2, 2009*, pages 151–158, 2009. doi:10.1145/1536414.1536437.
4. M. Blum, M. Luby, and R. Rubinfeld. Self-testing/correcting with applications to numerical problems. In *Proceedings of the Twenty-Second Annual ACM Symposium on Theory of Computing*, pages 73–83, New York, NY, USA, 1990. Association for Computing Machinery. doi:10.1145/100216.100225.
5. Andrej Bogdanov and Gautam Prakriya. Direct sum and partitionability testing over general groups. *Electronic Colloquium on Computational Complexity*, 2020. URL: https://eccc.weizmann.ac.il/report/2020/164.
6. Andrej Bogdanov and Baoxiang Wang. Learning and testing variable partitions. In Thomas Vidick, editor, *11th Innovations in Theoretical Computer Science Conference, ITCS 2020, January 12-14, 2020, Seattle, Washington, USA*, volume 151 of *LIPIcs*, pages 37:1–37:22. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPIcs.ITCS.2020.37.
7. Aline Bonami. Étude des coefficients de Fourier des fonctions de $l^p(g)$. *Annales de l'Institut Fourier*, 20(2):335–402, 1970. doi:10.5802/aif.357.
8. Yotam Dikstein and Irit Dinur. Agreement testing theorems on layered set systems. In David Zuckerman, editor, *60th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2019, Baltimore, Maryland, USA, November 9-12, 2019*, pages 1495–1524. IEEE Computer Society, 2019. doi:10.1109/FOCS.2019.00088.
9. Irit Dinur and Konstantin Golubev. Direct sum testing: The general case. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques, APPROX/RANDOM 2019, September 20-22, 2019, Massachusetts Institute of Technology, Cambridge, MA, USA*, pages 40:1–40:11, 2019. doi:10.4230/LIPIcs.APPROX-RANDOM.2019.40.
10. Irit Dinur and David Steurer. Direct product testing. In *IEEE 29th Conference on Computational Complexity, CCC 2014, Vancouver, BC, Canada, June 11-13, 2014*, pages 188–196. IEEE Computer Society, 2014. doi:10.1109/CCC.2014.27.
11. Chandra Nair and Yan Nan Wang. Evaluating hypercontractivity parameters using information measures. In *2016 IEEE International Symposium on Information Theory (ISIT)*, pages 570–574, 2016.
12. Ryan O'Donnell. *Analysis of Boolean Functions*. Cambridge University Press, USA, 2014.
13. Rocco A. Servedio, Li-Yang Tan, and John Wright. Adaptivity Helps for Testing Juntas. In David Zuckerman, editor, *30th Conference on Computational Complexity (CCC 2015)*, volume 33 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 264–279, Dagstuhl, Germany, 2015. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. doi:10.4230/LIPIcs.CCC.2015.264.
14. Amir Shpilka and Avi Wigderson. Derandomizing homomorphism testing in general groups. *SIAM J. Comput.*, 36(4):1215–1230, 2006. doi:10.1137/S009753970444658X.

4 vs 7 Sparse Undirected Unweighted Diameter is SETH-Hard at Time $n^{4/3}$

Édouard Bonnet
Univ Lyon, CNRS, ENS de Lyon, Université Claude Bernard Lyon 1, LIP UMR5668, France

Abstract

We show, assuming the Strong Exponential Time Hypothesis, that for every $\varepsilon > 0$, approximating undirected unweighted Diameter on n-vertex m-edge graphs within ratio $7/4 - \varepsilon$ requires $m^{4/3-o(1)}$ time, even when $m = \tilde{O}(n)$. This is the first result that conditionally rules out a near-linear time 5/3-approximation for undirected Diameter.

2012 ACM Subject Classification Theory of computation → Graph algorithms analysis

Keywords and phrases Diameter, inapproximability, SETH lower bounds, k-Orthogonal Vectors

Digital Object Identifier 10.4230/LIPIcs.ICALP.2021.34

Category Track A: Algorithms, Complexity and Games

Related Version *Full Version*: https://arxiv.org/abs/2101.02312

Funding This work was supported by the grant from French National Agency under PRC program (project Digraphs, ANR-19-CE48-0013-01).

1 Introduction

The diameter of a graph is the length of a longest shortest path between two of its vertices. We write Diameter for the algorithmic task of computing the diameter of an input graph. Throughout the paper, n implicitly denotes the number of vertices of a graph, and m, its number of edges. We will often prefix Diameter with *undirected/directed* to indicate whether or not edges may be oriented[1], and *unweighted/weighted* to indicate whether or not non-negative edge weights are allowed.

A fairly recent and active line of work aims to determine the best runtime for an algorithm approximating Diameter within a given ratio. First, there is an exact algorithm running in time[2] $\tilde{O}(mn)$, which computes n shortest-path trees from every vertex of the graph. Secondly, there is a 2-approximation running in time $\tilde{O}(m)$, which computes a shortest-path tree from an arbitrary vertex and outputs the largest distance found. There are an $\tilde{O}(m^{3/2})$ time 3/2-approximation for directed weighted Diameter [1, 15, 6], and for every non-negative integer k, an $\tilde{O}(mn^{\frac{1}{k+1}})$ time $(2-2^{-k})$-approximation[3] for *undirected* weighted Diameter [4]. We refer the interested reader to the survey of Rubinstein and Vassilevska Williams [16].

We will now focus on sparse graphs, for which $m = \tilde{O}(n)$. This is because the current paper deals with conditional lower bounds on approximating Diameter, and all such results even work with that restriction. Observe that, on sparse graphs, the first result of the previous paragraph is a near-quadratic 1-approximation, while the second result is a near-linear 2-approximation. One can represent these ratio-runtime trade-offs in the two-dimensional plane. The ultimate goal of fine-grained complexity, in that particular context, is to obtain a

[1] In directed Diameter, we are to compute the length of a longest shortest path taken from any vertex to any vertex.
[2] where $\tilde{O}(\cdot)$ suppresses the polylogarithmic factors
[3] with an extra additive factor depending on the weights

complete curve of algorithms linking these two extreme points, matched by tight conditional lower bounds. We now present one way of deriving conditional lower bounds for polytime problems.

Lower bounds based on the Strong Exponential Time Hypothesis

The Strong Exponential Time Hypothesis (SETH, for short) asserts that for every $\varepsilon > 0$, there is an integer k such that k-SAT cannot be solved in time $(2-\varepsilon)^n$ on n-variable instances [11]. At first glance, this assumption should only be useful to rule out some specific running time for NP-hard problems which, like the satisfiability problem, seems to require superpolynomial time. Such conditional lower bounds to classical [7] or parameterized algorithms [8] are overviewed in a survey [14] on the consequences of the SETH (as well as the weaker assumption ETH) on solving computationally hard problems.

Interestingly, using the SETH to rule out a given running time for a polynomial-time solvable problem took more time. In a survey of fine-grained complexity [18], Vassilevska Williams dates the first reduction (albeit used positively) from SAT to a problem in P back to 2005 [17]. We will see that this reduction to 2-Orthogonal Vectors, where one wants to find two orthogonal 0, 1-vectors within a given list, is very relevant to the fine-grained complexity of Diameter. As it turns out, the first SETH-based lower bound for a polytime graph problem occurred almost a decade later, on the very unweighted undirected Diameter [15].

There might have been a psychological barrier in reducing a "hard" problem to an "easy" one, in order to derive a conditional lower bound. However this makes perfect sense. Let us give an apropos example. Suppose (as it is actually the case) that one can create in time $O(n)$ a list of n 0,1-vectors with $n = O(2^{N/2})$, from an N-variable SAT formula, such that there is pair of orthogonal vectors in the list if and only if the formula is satisfiable. Now a truly subquadratic algorithm, that is in time $n^{2-\varepsilon}$ for some $\varepsilon > 0$, for 2-Orthogonal Vectors would enable to solve SAT in time $O(2^{(1-\varepsilon/2)N}) = O((2-\delta)^N)$ for some $\delta > 0$, contradicting the SETH. We thus say that 2-Orthogonal Vectors is *SETH-hard* at time n^2, and more generally a problem Π is *SETH-hard* at time T if it requires time $T^{1-o(1)}$ under the SETH.

SETH lower bounds for Diameter

There is a handful of SETH-hardness results on approximating Diameter [15, 2, 3, 9, 12, 13]. Unless the SETH fails, any $3/2-\varepsilon$-approximation for sparse undirected unweighted Diameter, with $\varepsilon > 0$, requires time $n^{2-o(1)}$ [15] (this is the above-mentioned seminal result to the fine-grained complexity within P), whereas any $5/3-\varepsilon$-approximation requires time $n^{3/2-o(1)}$ [12] (an early version of [13]). Since a 5/3-approximation of Diameter running in near-linear time was consistent with the then knowledge (up until mid-August 2020, even in weighted directed graphs) Rubinstein and Vassilevska Williams [16] and Li [12] ask for such an algorithm or some lower bounds with a ratio closer to 2.

In the last few months, there were several developments on directed graphs. The author showed that, under the SETH, $7/4-\varepsilon$-approximating sparse directed weighted Diameter requires time $n^{4/3-o(1)}$ [3]. Then Wein and Dalirrooyfard [9], and independently, Li [13] (an updated version of [12]) both show that not only this result holds on directed *unweighted* graphs but they generalize it in the following way: Unless the SETH fails, for every $\varepsilon > 0$ and every integer $k \geqslant 4$, $\frac{2k-1}{k} - \varepsilon$-approximating directed unweighted Diameter requires time $n^{\frac{k}{k-1}-o(1)}$.

Despite these advances, a near-linear time 5/3-approximation for the *undirected* Diameter may still have existed. In this paper, we rule out this possibility by showing the following (see Figure 1 for a visual summary of what is now known on approximating undirected Diameter).

▶ **Theorem 1.** *Unless the SETH fails, for any $\varepsilon > 0$, $7/4 - \varepsilon$-approximating Diameter on undirected unweighted n-vertex $\tilde{O}(n)$-edge graphs requires $n^{4/3-o(1)}$ time.*

In particular we resolve [16, Open Question 2.2.], on the existence of a near-linear time 5/3-approximation for *undirected* Diameter, by the negative.

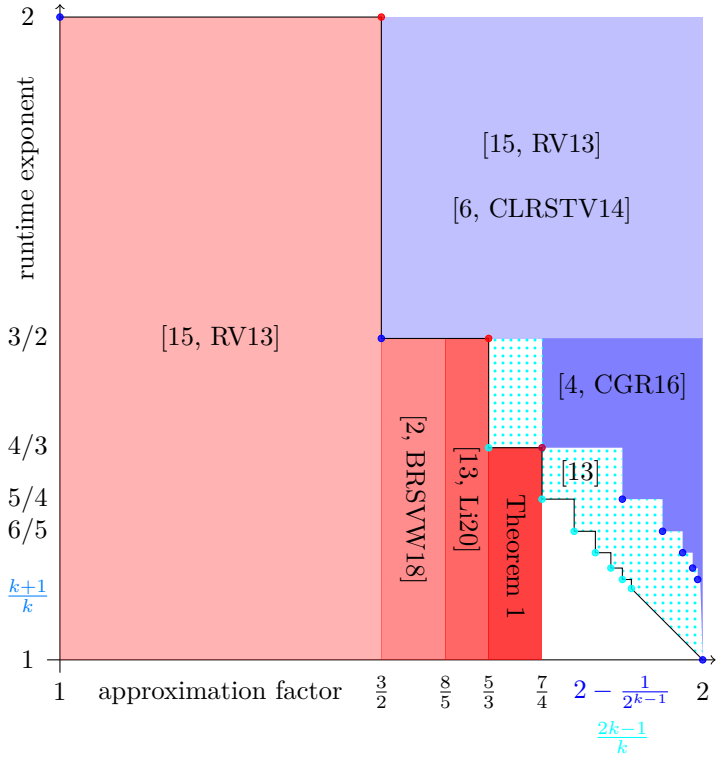


Figure 1 Approximability of sparse undirected unweighted Diameter. Blue areas are feasible, as witnessed by algorithms at bottom-left corners (blue dots). The red regions are SETH-hard, as witnessed by reductions at top-right corners (red dots). Dotted cyan areas are not SETH-hard, unless the NSETH fails. The current landscape for the sparse undirected *weighted* Diameter is the same, except the middle red region is entirely due to Backurs et al. [2] instead of [13]. The axis-parallel black curve represents the tractability frontier as foreseen by Conjecture 2.

In light of the recent results (see in particular the paragraph on barriers to SETH-hardness), it is reasonable to conjecture that the four variants of sparse Diameter (undirected/directed unweighted/weighted) are equally approximable. More precisely, we venture the following optimistic prediction.

▶ **Conjecture 2.** *Sparse (un)directed (un)weighted Diameter is $2 - \frac{1}{k}$-approximable in time $\tilde{O}(n^{\frac{k+1}{k}})$ for every $k \in \mathbb{N}^+ \cup \{\infty\}$. Furthermore unless the SETH fails, approximating sparse (un)directed (un)weighted Diameter within ratio better than $2 - \frac{1}{k+1}$ requires time $n^{\frac{k+1}{k}-o(1)}$ for every $k \in \mathbb{N}^+$.*

Conjecture 2 is naturally equivalent to obtaining the algorithms for the directed weighted Diameter and the SETH-hardness for the undirected unweighted Diameter. Settling the conjecture would give a complete landscape of the approximability of Diameter, where if one represents the results in the two-dimensional space of approximation factor vs runtime exponent, the feasible and infeasible regions are separated by a rectilinear curve with infinitely many corners (the black curve drawn in Figure 1). In that respect, our contribution is to give the third lower bound on the curve (i.e., North-East corner) after Roditty and Vassilevska Williams gave the first [15], and Li, the second [12]. Perhaps our new ideas (together with the recent constructions in the directed case of Wein and Dalirrooyfard [9], and Li [13]) will also help in generalizing the lower bound predicted by Conjecture 2 to every positive integer k.

Barriers to SETH lower bounds

Conjecture 2 is partly prompted by intriguing results due to Li [13]. To state them, we need to recall the definition of a strengthening of SETH introduced by Carmosino et al. [5]. It is called NSETH for Nondeterministic SETH. NSETH asserts that for every $\varepsilon > 0$, there is an integer k such that the k-Taut problem cannot be solved in *non-deterministic* time $(2-\varepsilon)^n$, where k-Taut asks, given a k-DNF formula whether every truth assignment satisfies it (in other words, if it is a tautology). Li shows, for all four variants of Diameter *but* the directed weighted one, that no point positioned strictly above the rectilinear black curve of Figure 1 can be shown SETH-hard, under the NSETH (and, if randomized reductions are permitted, under a stronger assumption, called NUNSETH for Non-Uniform NSETH).

Conjecture 2 is very optimistic since it predicts that every such point will be explained by an algorithm. There are many alternatives to that event. For instance NSETH could be false[4], or the intractability region could extend further North via a non SETH-based reduction, or via a deterministic SETH-based reduction in the directed weighted case. Besides it would require significant progress in approximating the sparse directed Diameter, when currently no algorithm running in time $n^{\frac{3}{2}-\varepsilon}$ achieves approximation factor better than 2.

The second half of Conjecture 2 shown for every $k \geqslant 4$ on directed graphs [9, 13], and for $k = 1, 2, 3$ on undirected graphs, is much easier to believe in.

Techniques

Like every mentioned Diameter lower bound (for more details, see the paragraphs on k-OV and Diameter in the surveys [18, 16]), we reduce from k-Orthogonal Vectors, where one seeks, in a given set of N 0, 1-vectors of dimension ℓ, k vectors such that at every index, at least one of these k vectors has a 0 entry. Under the SETH, k-Orthogonal Vectors requires time $N^{k-o(1)}$ [17], even when ℓ is polylogarithmic in N.

Here we will reduce from 4-Orthogonal Vectors. We thus wish to build a graph on $\tilde{O}(N^3)$ vertices and edges with diameter 7 if there is an orthogonal quadruple (i.e., a solution to the 4-Orthogonal Vectors instance), and diameter 4 otherwise. Following a reduction to ST-Diameter[5] by Backurs et al. [2] (arguably also following [15]) most of the reductions (as in [3, 9, 13]) feature layers $L_0, L_1, \ldots, L_{k-1}, L_k$, with only (forward) edges between two consecutive L_i. The vertices within the same layer share the same number of "vector attributes" and "index attributes". The interplay between vector and index attributes

[4] If we are totally honest, even the weaker SETH does not gather such a wide consensus, and is false if quantum computation is allowed.

[5] where one seeks the length of a longest shortest path from a vertex of S to a vertex of T

in defining the vertices and edges is made so that if there are no k orthogonal vectors, then there are paths of "optimal" length k between every pair in $L_0 \times L_k$, whereas if there is set X of k orthogonal vectors, a pair in $L_0 \times L_k$ jointly encoding X is suddenly very far apart (usually and ideally at distance $2k-1$).

Then the challenge is to make sure that, on NO-instances, the other pairs (not in $L_0 \times L_k$) are at distance at most k, without destroying the previous property. The core of our reduction is similar to our previous construction for directed weighted Diameter [3]. However we simplify and streamline it in the following way. As in the first construction of Li [12], we collapse some layers into one. We will have $L_0 = L_4 (= T)$ and $L_1 = L_3 (= C)$, while L_2 is called P. This makes the case analyses simpler (fewer kinds of pairs to consider).

At this point, we face the same issue as in [3]: There are pairs in $T \times P$ that are too far apart. On directed graphs, this can be fixed by adding parallel layers and appropriate "back" edges [3, 9] or simply "back" edges [13]. This is no longer an option. Instead we add a set I of vertices with only index attributes. These vertices link the right pairs of $T \times P$ with path of length 4 (we are back to the first variation on the theme [15]). To emphasize that the situation is somewhat delicate, we observe that not all the pairs of $T \times P$ can be at distance 4, since otherwise every pair in $T \times T$ is at distance at most 6. We set I at distance 3 of T (by initially putting edges of weight 3). This permits to cliquify I without creating TT-paths of length at most 6. In turn, this puts every pair involving I at distance at most 4, as well as pairs of $(C \cup P) \times P$. Note that as long as $d(T,X) + d(T,Y) \geqslant 3$ (or $k-1$), one can have *all* the pairs of $X \times Y$ at distance 4 (or k), without creating undesired TT-paths of length at most 6 (or $2k-2$).

We then remove the weight-3 edges between T and I. This involves some vertex splits transforming T into T, T', T'', and a simpler echo of the idea of having the clique I, with a clique I' connecting appropriately the pairs in $T \times T''$.

2 Preliminaries

We use standard graph-theoretic notations. If G is a graph, $V(G)$ denotes its vertex set, and $E(G)$, its edge set. We denote the edge set between $X \subseteq V(G)$ and $Y \subseteq V(G)$ by $E(X,Y)$. If $S \subseteq V(G)$, $G[S]$ denotes the subgraph of G induced by S. *Weighted graphs* have positive edge weights. (Throughout the paper, we will only need edges of weight 1 and 3.) We exclusively deal with undirected graphs (for which the distance function is symmetric). For $u, v \in V(G)$, $d_G(u,v)$ denotes the distance between u and v in G, that is, the number of edges in a shortest path between u and v. For every positive integer r and every vertex $u \in V(G)$, $N^r_G[u]$ denotes the set of vertices v such that $d_G(u,v) \leqslant r$. In unweighted graphs, the closed neighborhood of u, denoted $N_G[u]$, coincides with $N^1_G[u]$. However in a weighted graph $N^1_G[u]$ would for instance not contain the neighbors of u via an edge of weight greater than 1. This subtlety will arise only once, and we will remind the reader in due time. For every positive integer r and every vertex $S \subseteq V(G)$, $N^r_G[S]$ denotes the set of vertices $v \in V(G)$ such that $d_G(u,v) \leqslant r$ for some $u \in S$. We observe that, in unweighted graphs, $N^r_G[S]$ coincides with $N_G[N_G[\cdots N_G[N_G[S]]\cdots]]$ where $N_G[\cdot]$ is applied r times and $N_G[S]$ is the closed neighborhood of S. We drop the subscript in the above notations, if the graph G is clear from the context.

We denote by $\mathrm{diam}(G)$ the diameter of G, that is, $\max_{u,v \in V(G)} d(u,v)$. The Diameter problem asks, given a graph G, for the value of $\mathrm{diam}(G)$. We call uv-path, a path going from vertex u to vertex v, and ST-path (with possibly $S = T$), any path going from some vertex $u \in S$ to some vertex $v \in T$.

If ℓ is a positive integer, $[\ell]$ denotes the set $\{1, 2, \ldots, \ell\}$. If v is a vector and i is a positive integer, then $v[i]$ denotes the i-th coordinate of v. We use $\mathrm{maj}(a_1, \ldots, a_h)$ to denote the value with the largest number of occurrences in the tuple $(a_1, \ldots, a_h)$.

For every fixed positive integer k, the k-Orthogonal Vectors (k-OV for short) problem is as follows. It asks, given a set S of 0,1-vectors in $\{0,1\}^\ell$, if there are k vectors $v_1, \ldots, v_k \in S$ such that for every $i \in [\ell]$, $\Pi_{h\in[k]} v_h[i] = 0$, or equivalently, $v_1[i] = v_2[i] = \cdots = v_k[i] = 1$ does not hold. Williams [17] showed that, assuming the SETH, k-OV requires $N^{k-o(1)}$ time with $N := |S|$. Furthermore, using the Sparsification Lemma [10], this lower bound holds even when, say, $\ell = \lceil \log^2 N \rceil$. Here we will leverage this lower bound for $k = 4$. This is, in the context of the SETH-hardness of approximating Diameter, a usual opening step: For example, Roditty and Vassilevska Williams [15] uses this lower bound for $k = 2$, Li [12], for $k = 3$ and general $k \geqslant 3$, the author [3], for $k = 4$, Wein and Dalirrooyfard [9], for general $k \geqslant 5$ and $k = 4$.

3 A simpler reduction with edge weights

From any set S of N vectors in $\{0,1\}^\ell$, we build an undirected weighted graph $G = \rho(S)$ (with edge weights 1 and 3, only) with $O(N^3 + N^2\ell^3 + \ell^5)$ vertices and $O(N^3\ell^5 + N^2\ell^6 + \ell^{10})$ edges such that if S admits an orthogonal quadruple then the diameter of G is (at least) 7, whereas if S has no orthogonal quadruple then the diameter of G is (at most) 4. We recall that 4-OV requires $N^{4-o(1)}$ time, unless the SETH fails, even when $\ell = \lceil \log^2 N \rceil$ [17]. In that case, the graph G has $O(N^3)$ vertices and $\tilde{O}(N^3)$ edges. Hence any algorithm approximating sparse undirected weighted Diameter within ratio better than $7/4$ in time $n^{4/3-\delta}$, with $\delta > 0$, would refute the SETH.

3.1 Construction

We first describe the vertex set of G, then its edge set, and finally check that the number of vertices and edges are as announced.

Vertex set

Every vertex of G is the concatenation of a possibly empty tuple of vectors of S, called *vector tuple*, followed by a possibly empty tuple of possibly equal indices of $[\ell]$, called *index tuple*. Each coordinate of the vector tuple is called a *vector field*, while each coordinate of the index tuple is called an *index field*. The set $V(G)$ is partitioned into four sets: T (for **t**riples), C (for **c**ouples), P (for **p**airs), and I (for **i**ndices). The names behind T, C, P reflect the number and the nature (ordered or unordered) of the vector fields. Each of these sets comprise vertices with up to three vector fields and five index fields. They are defined in the following way.

- T: for every ordered triple $a, b, c \in S$ with a, b, c pairwise distinct, we add vertex (a, b, c) to T. Thus vertices of T have three vector fields and no index field.
- C: for every $a \neq b \in S$ and $i, j, k \in [\ell]$ such that $a[i] = a[j] = a[k] = 1$ and $\mathrm{maj}(b[i], b[j], b[k]) = 1$, we add vertex (a, b, i, j, k) to C. Thus vertices of C have two vector fields and three index fields.
- P: for every $a, b \in S$ and $i, j, k \in [\ell]$ such that $a[i] = a[j] = a[k] = 1$ and $b[i] = b[j] = b[k] = 1$, we add vertex $(\{a, b\}, i, j, k)$ to P. We will still see a and b as filling the *two* vector fields of the vertex, without a *first* vector field and a *second* vector field. Contrary to vertices of C, $(\{a, b\}, i, j, k)$ and $(\{b, a\}, i, j, k)$ are two names for the same vertex

(whereas (a, b, i, j, k) and (b, a, i, j, k) are two distinct vertices, whose existence implies slightly different properties). Thus vertices of P also have two vector fields and three index fields. Note also that $\{a, b\}$ is a multiset, since a may be equal to b.

- I: for every $p_1, p_2, i, j, k \in [\ell]$, we add vertex (p_1, p_2, i, j, k) to I. The chosen labels for the five index fields anticipate that, to build the edge set, it is convenient to imagine a separation after the first two index fields of the tuple. The vertices of I have no vector field and five index fields.

Edge set

We will put some edges between T and C, C and P, P and I, and I and T. In addition, we put *index-switching edges* within I and within C. An index-switching edge is between two vertices of the same set (I or C) with the same vector tuple (which is always the case in I) and distinct index tuples. The only edges with a weight different than 1 are the edges between I and T, which all have weight 3. Thus, unless specified otherwise, an edge has weight 1.

The total list of edges is as follows.

- We add all the index-switching edges within I and C. Thus $G[I]$ is a clique and $G[C]$ is a disjoint union of cliques (while $G[T]$ and $G[P]$ remain independent sets). More explicitly, we have an edge between every pair of distinct vertices $(p_1, p_2, i, j, k) \in I$ and $(p'_1, p'_2, i', j', k') \in I$, and for every $a \neq b \in S$ between every pair of distinct vertices $(a, b, i, j, k) \in C$ and $(a, b, i', j', k') \in C$.
- $E(T, C)$: We add an edge between every $(a, b, c) \in T$ and $(a, b, i, j, k) \in C$ provided that there is an $h \in \{i, j, k\}$ such that $b[h] = c[h] = 1$.
- $E(C, P)$: We add an edge between every $(a, b, i, j, k) \in C$ and $(\{c, d\}, i, j, k) \in P$ whenever $a \in \{c, d\}$.
- $E(T, I)$: We add an edge of weight 3 between every $(a, b, c) \in T$ and $(p_1, p_2, i, j, k) \in I$ whenever $a[p_1] = b[p_1] = c[p_1] = a[p_2] = b[p_2] = c[p_2] = 1$.
- $E(I, P)$: We add an edge between every $(p_1, p_2, i, j, k) \in I$ and $(\{a, b\}, i, j, k) \in P$ whenever $a[p_1] = b[p_2] = 1$ or $a[p_2] = b[p_1] = 1$.

This ends the construction. See Figure 2 for an illustration.

Vertex and edge count

There are $O(N^3)$ vertices in T, $O(N^2\ell^3)$, in $C \cup P$, and ℓ^5, in I, hence $O(N^3 + N^2\ell^3 + \ell^5) = O(N^3)$ in total. There are $O(N^3\ell^3)$ edges in $E(T, C) \cup E(C, P)$, $O(N^2\ell^6)$, in $E(C)$, $O(N^3\ell^5)$, in $E(T, I)$, $O(N^2\ell^5)$, in $E(I, P)$, and $O(\ell^{10})$ in $E(I)$, hence $O(N^3\ell^5 + N^2\ell^6 + \ell^{10}) = \tilde{O}(N^3)$ edges in total. Furthermore G can be built in time $\tilde{O}(N^3)$.

3.2 The absence of orthogonal quadruple implies diameter at most 4

Assuming that there is no orthogonal quadruple, we show that every pair of vertices of G is at distance at most 4. For that we repeatedly use that, for every $a, b, c, d \in S$, $\mathrm{ind}(a, b, c, d) := \min\{i \in [\ell] \mid a[i] = b[i] = c[i] = d[i] = 1\}$ is a well-defined index in $[\ell]$. We only take the minimum index to have a deterministic notation, but there is nothing particular with it, and any index of the non-empty $\{i \in [\ell] \mid a[i] = b[i] = c[i] = d[i] = 1\}$ would work all the same.

We first observe that every vertex is at distance at most 3 from I.

▶ **Lemma 3.** *$N^1[I] \supseteq I \cup P$, $N^2[I] \supseteq I \cup P \cup C$, and $N^3[I] = V(G)$.*

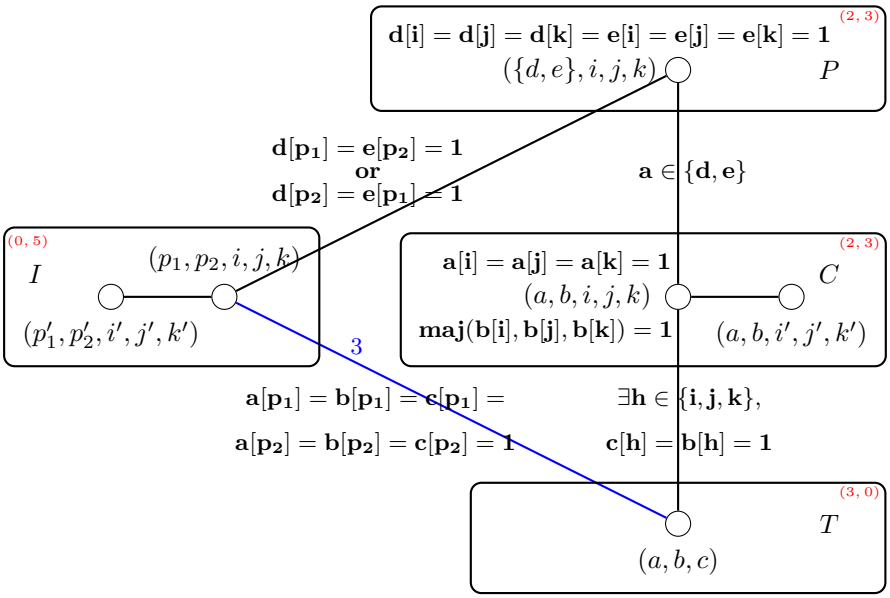


Figure 2 The weighted construction G. In bold, the conditions for the existence of a vertex or of an edge. The edge in blue, and more generally every edge of $E(T, I)$, has weight 3, while all other edges have weight 1. The pairs in red recall, for vertices of the corresponding set, the length of their vector tuple followed by the length of their index tuple.

Proof. The first and second inclusions are actually equalities but we will not need those facts. $N^1[I] \supseteq I \cup P$ since every $(\{a, b\}, i, j, k) \in P$ is adjacent (with an edge of weight 1) to $(i, i, i, j, k) \in I$. Then, $N^2[I] \supseteq N^1[I \cup P] \supseteq I \cup P \cup C$ since every $(a, b, i, j, k) \in C$ is adjacent to $(\{a, a\}, i, j, k) \in P$. Finally, $N^3[I] \supseteq N^1[I \cup P \cup C] = V(G)$ since every $(a, b, c) \in T$ is adjacent to $(a, b, i, i, i) \in C$ for some $i \in [\ell]$, for otherwise a, b, c is an orthogonal triple. ◀

We now exhibit paths of length at most 4 between every pair of vertices of G. For the case disjunction, initially imagine the K_4 with loops on vertices T, C, P, I, where edges correspond to kinds of pairs that are left to check. The following paragraphs remove all its edges in the order: all edges incident to I, all remaining edges incident to P but TP, all remaining edges incident to C, the loop on T, and finally the edge TP.

Between $u \in I$ and $v \in V(G)$

As $G[I]$ is a clique and, by Lemma 3, $N^3[I] = V(G)$, every vertex $u \in I$ is at distance at most 4 from every vertex $v \in V(G)$.

Between $u \in P$ and $v \in P \cup C$

For every $u \in P$, $N^2[u] \supset I$ and so $N^4[u] \supset P \cup C$, by Lemma 3. In particular there is a path of length at most 4 between u and any vertex $v \in P \cup C$.

Between $u \in C$ and $v \in T \cup C$

Let (a, b) be the two vector fields of $u \in C$, (c, d) be the first two vector fields of $v \in T \cup C$, and e be the third vector field of v if $v \in T$. Let $i = \text{ind}(a, b, c, d)$, $j = \text{ind}(a, c, d, e)$ if $v \in T$, and $j = i$ if $v \in C$. We observe that $(a, b, i, i, j), (\{a, c\}, i, i, j), (c, d, i, i, j)$ are (existing) vertices

of C, P, and C, respectively, and that $u-(a,b,i,i,j)-(\{a,c\},i,i,j)-(c,d,i,i,j)$ is a path of length 3 in G. The existence of these vertices is implied by $a[i]=b[i]=c[i]=d[i]=1$, $a[j]=c[j]=d[j]=1$. The first edge of the path is an index-switching edge within C. The existence of the other edges is implied by $a\in\{a,c\}$, $c\in\{a,c\}$, and the fact that the index tuple (i,i,j) does not change.

Finally if $v\in C$, then the index-switching edge $(c,d,i,i,j)-v$ completes the uv-path of length 4. If instead $v\in T$, then the edge $(c,d,i,i,j)-(c,d,e)=v$ completes the uv-path of length 4. This edge exists since $d[j]=e[j]=1$.

Between $u\in T$ and $v\in T$

Let $u=(a,b,c),v=(d,e,f)\in T$, $i=\mathrm{ind}(a,b,c,d)$, $j=\mathrm{ind}(a,b,d,e)$ and $k=\mathrm{ind}(a,d,e,f)$. Then $u=(a,b,c)-(a,b,i,j,k)-(\{a,d\},i,j,k)-(d,e,i,j,k)-(d,e,f)=v$ is a path of length 4 in G. These vertices exist since a and d have value 1 on indices i,j,k, b, on indices i,j, and e, on indices j,k. The first edge exists since $b[i]=c[i]=1$, the next two edges exist for similar reasons as invoked in the previous paragraph, and the fourth edge exists since $e[k]=f[k]=1$.

Between $u\in T$ and $v\in P$

Let $u=(a,b,c)\in T$ and $v=(\{d,e\},i,j,k)\in P$. We set $p_1=\mathrm{ind}(a,b,c,d)$, $p_2=\mathrm{ind}(a,b,c,e)$, and exhibit a uv-path of length 4 via I. Indeed $u=(a,b,c)-(p_1,p_2,i,j,k)-(\{d,e\},i,j,k)=v$ is a path of length 4 in G (recall that the first edge of the path has weight 3). Edge $(a,b,c)-(p_1,p_2,i,j,k)\in E(T,I)$ exists since $a[p_1]=b[p_1]=c[p_1]=a[p_2]=b[p_2]=c[p_2]=1$. Edge $(p_1,p_2,i,j,k)-(\{d,e\},i,j,k)\in E(I,P)$ exists since $d[p_1]=e[p_2]=1$ and the three last indices (i,j,k) remain unchanged.

3.3 The presence of orthogonal quadruple implies diameter at least 7

Let $a,b,c,d\in S$ be an orthogonal quadruple, that is, such that there is *no* index $i\in[\ell]$ satisfying $a[i]=b[i]=c[i]=d[i]=1$. We may further assume that a,b,c,d are all distinct since checking for an orthogonal triple can be done in time $\tilde{O}(N^3)$. We will now show that there is no path $\mathcal{P}$ of length at most 6 between $u=(a,b,c)\in T$ and $v=(d,c,b)\in T$.

Since the distance between every pair of vertices in $T\times I$ is at least 3, a TT-path of length at most 6 cannot contain an edge of the clique $G[I]$, nor more generally intersects I at least twice. We thus distinguish two cases: (case A) $\mathcal{P}$ visits I exactly once, and (case B) $\mathcal{P}$ remains within $T\cup C\cup P$. Before proving that no uv-path $\mathcal{P}$ of length at most 6 visits I, thereby ruling out case A, we state a couple of useful observations.

▶ **Observation 4.** *There is at most one path of length 2 between $(\{d,e\},i,j,k)\in P$ and $(a,b,c)\in T$, namely $(\{d,e\},i,j,k)-(a,b,i,j,k)-(a,b,c)$, which in particular implies that $a\in\{d,e\}$.*

More basically, the only neighbors of $(a,b,c)\in T$ (at distance 1, so not in I) are of the form $(a,b,i,j,k)\in C$. We can generalize this observation to paths contained in $T\cup C$.

▶ **Observation 5.** *For every path within $G[T\cup C]$, all the vertices of the path have the same first two vector fields.*

Case A: $\mathcal{P}$ visiting I

As $\mathcal{P}$ cannot visit I twice, if it visits I then it has length exactly 6 and is one of the following kinds: (case 1) $T-I-T$, (case 2) $T-C-P-I-P-C-T$, or (case 3) $T-I-P-C-T$ (recall that the edges in $E(I,T)$ have weight 3). An important feature of such paths is that no index-switching edge can be used, thus the three last index fields (when they exist) have to remain the same.

Case 1. A path $(a,b,c)-(p_1,p_2,i,j,k)-(d,c,b)$ would in particular imply that $a[p_1]=b[p_1]=c[p_1]=d[p_1]=1$, contradicting the orthogonality of a,b,c,d.

Case 2. By Observation 4 applied to both ends of the path, $\mathcal{P}$ is of the form $(a,b,c)-(a,b,i,j,k)-(\{a,e\},i,j,k)-(p_1,p_2,i,j,k)-(\{d,f\},i,j,k)-(d,c,i,j,k)-(d,c,b)$ with some $e,f\in S$. The existence of the vertices $(a,b,i,j,k),(d,c,i,j,k)\in C$ implies that $a[i]=a[j]=a[k]=d[i]=d[j]=d[k]=1$, and that b and c have value 1 on at least two indices (with multiplicity) of multiset $\{i,j,k\}$. In particular, there is an $h\in\{i,j,k\}$ such that $a[h]=b[h]=c[h]=d[h]=1$, a contradiction to a,b,c,d being orthogonal.

Case 3. By Observation 4 applied to the second half of the path, $\mathcal{P}$ has then the form $(a,b,c)-(p_1,p_2,i,j,k)-(\{d,e\},i,j,k)-(d,c,i,j,k)-(d,c,b)$. The first three vertices yield a contradiction. Indeed, the existence of edge $(p_1,p_2,i,j,k)-(\{d,e\},i,j,k)$ implies that $d[p_z]=1$ for some $z\in\{1,2\}$, while the existence of $(a,b,c)-(p_1,p_2,i,j,k)$ implies that $a[p_z]=b[p_z]=c[p_z]=1$.

Case B: paths $\mathcal{P}$ within $T\cup C\cup P$

We now consider paths $\mathcal{P}$ in $G[T\cup C\cup P]$. Since $a\neq d$, $\mathcal{P}$ has to visit P, since otherwise the first vector field cannot change, by Observation 5. We then observe that no shortest uv-path visits T a third time (one more time than the two endpoints u and v). A TT-path visiting T a third time would contain a segment $C-T-C$ that can be shortcut into $C-C$. Indeed, $(a,b,i,j,k)-(a,b,c)-(a,b,i',j',k')$ has a chord $(a,b,i,j,k)-(a,b,i',j',k')$ which is an index-switching edge of C.

We further distinguish two cases: (case 1) $\mathcal{P}$ does not contain any index-switching edge, or (case 2) $\mathcal{P}$ contains at least one index-switching edge.

Case 1. In that case, $\mathcal{P}$ is of the form $T-C-P-C-T$ or $T-C-P-C-P-C-T$. Either way, we consider the unique neighbors of u and v in $\mathcal{P}$. These neighbors have to be $(a,b,i,j,k)\in C$ and $(d,c,i,j,k)\in C$ for some $i,j,k\in[\ell]$. Indeed no index-switching edge nor return to T is allowed here. Thus we conclude as in case A.2.

Case 2. We now assume that $\mathcal{P}$ contains at least one index-switching edge (of C). In that case, as $\mathcal{P}$ has length at most 6, it can visit P only once. Hence $\mathcal{P}$ is of the kind $T-C-C-P-C-C-T$, where one of the two edges $C-C$ is optional. We consider the last vertex $u'\in C$ before visiting P, and the first vertex $v'\in C$ after visiting P. There is, by design, no index-switching edge between u' and v' on path $\mathcal{P}$. Thus by Observation 5, there are $i,j,k\in[\ell]$ such that $u'=(a,b,i,j,k)$ and $v'=(d,c,i,j,k)$. We then conclude as in case A.2.

4 Removing the weights

So far we showed the announced lower bound for sparse undirected *weighted* Diameter. We show how to tune the previous construction to get the same lower bound for sparse undirected unweighted Diameter. The weighted graph G had only non-trivial edge weights in $E(T,I)$. We now describe how to replace these weighted edges, to get an unweighted graph $G'=\rho'(S)$.

4.1 Unweighted construction

We start with a short summary of the changes. We will replace T by three copies T, T', T'' with an induced perfect matching between T and T', and between T' and T''. We link T'' to I as we linked T to I, and T and C, and T' and C, as we linked T and C. We finally add a set I' of vertices with empty vector tuple (like I) that we link to T'' and I only.

Addition to the vertex set

We add three sets to $V(G)$ to get $V(G')$: two identical copies of T, denoted by T' and T'', and a set I' isomorphic to $[\ell]$. More precisely, for every $i \in [\ell]$, we add vertex (i) to I'. Thus I' has no vector field and a unique index field. We use a subscript to distinguish the *homologous* vertices in T, T', T''. Vertices $(a,b,c)_T \in T, (a,b,c)_{T'} \in T', (a,b,c)_{T''} \in T''$ are the three vertices of G' corresponding to the same vertex (a,b,c) of G.

Edition of the edge set

We first remove the edges of G with weight 3 (between T and I). For every $\{a,b,c\} \in \binom{S}{3}$, we add the edges $(a,b,c)_T - (a,b,c)_{T'}$ and $(a,b,c)_{T'} - (a,b,c)_{T''}$. We also add edges between T' and C, the same way we have defined edges between T and C. That is, $(a,b,c)_{T'} - (a,b,i,j,k)$ is an edge if and only if $(a,b,c)_T - (a,b,i,j,k)$ is an edge. Let us recall that the existence of this edge (and of its endpoint in C) implies that a,b,c have value 1 on indices $\{i,j,k\}$, three times, at least twice, and at least once, respectively, and that there is an $h \in \{i,j,k\}$ such that $a[h] = b[h] = c[h]$.

We add edges (of weight 1) between T'' and I, the same way we defined the weight-3 edges of G between T and I. Thus there is an edge $(a,b,c)_{T''} - (p_1,p_2,i,j,k)$ whenever $a[p_1] = b[p_1] = c[p_1] = a[p_2] = b[p_2] = c[p_2] = 1$. We further add an edge between $(i) \in I'$ and $(a,b,c)_{T''}$ whenever $a[i] = b[i] = 1$. Finally we add all the index-switching edges in I', and we make I and I' fully adjacent, that is, we turn $G'[I' \cup I]$ into a clique.

This finishes the edition to the unweighted construction. See Figure 3 for an illustration.

New vertex and edge count

We added to $V(G)$ $O(N^3)$ vertices in $T' \cup T''$, and ℓ, in I'. Thus G' has also $O(|V(G)|) = O(N^3 + N^2\ell^3 + \ell^5) = O(N^3)$ vertices. We added to $E(G)$ $O(N^3 + N^3\ell^3)$ edges incident to T', and $O(N^3\ell + \ell^6 + \ell^2)$ edges incident to I'. (The edges between T'' and I were already counted in G between T and I.) Thus G' has $O(|E(G)|) = O(N^3\ell^5 + N^2\ell^6 + \ell^{10}) = \tilde{O}(N^3)$ edges. Again G' can be computed in time $\tilde{O}(N^3)$.

4.2 The absence of orthogonal quadruple implies diameter at most 4

In case S has no orthogonal quadruple, we use similar arguments as in G to find paths of length at most 4 between every pair of vertices in G'. We first show that I' is at distance at most 3 of every vertex of G'.

▶ **Lemma 6.** *$N[I'] \supseteq I' \cup I \cup T''$, $N^2[I'] \supseteq I' \cup I \cup T'' \cup P \cup T'$, and $N^3[I'] = V(G')$.*

Proof. The inclusions are actually equalities. $N[I'] \supseteq I' \cup I \cup T''$ since I is fully adjacent to I' and every $(a,b,c)_{T''} \in T''$ is adjacent to some $(i) \in I'$, for otherwise a,b is an orthogonal pair. $N^2[I'] \supseteq N[I' \cup I \cup T''] \supseteq I' \cup I \cup T'' \cup P \cup T'$ since every $(\{a,b\},i,j,k) \in P$ is adjacent to $(i,i,i,j,k) \in I$ and every $(a,b,c)_{T'} \in T'$ is adjacent to $(a,b,c)_{T''} \in T''$. Finally, $N^3[I'] \supseteq N[I' \cup I \cup T'' \cup P \cup T'] = V(G')$ since every $(a,b,i,j,k) \in C$ is adjacent to $(\{a,a\},i,j,k) \in P$ and every $(a,b,c)_T \in T$ is adjacent to $(a,b,c)_{T'} \in T'$. ◀

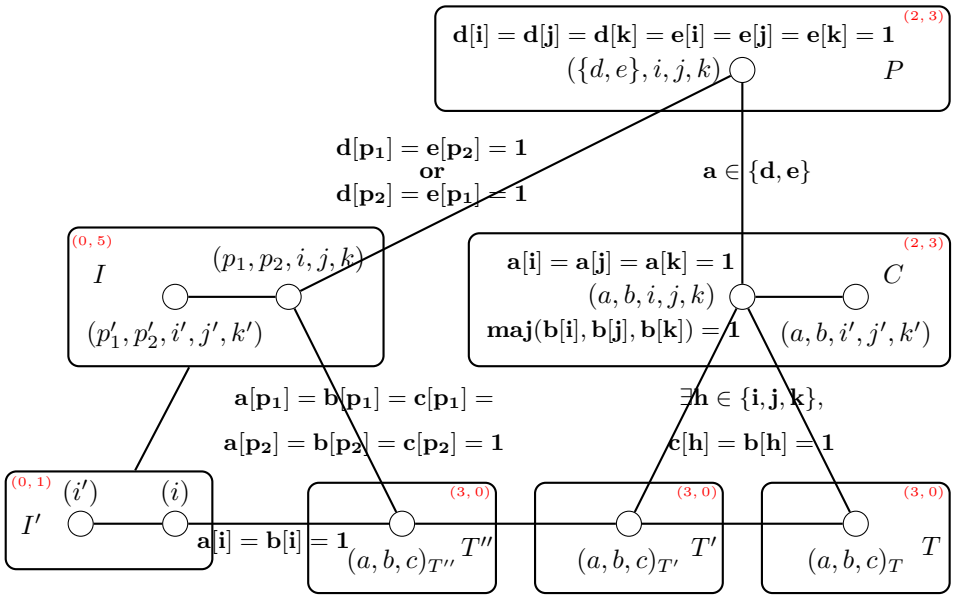


Figure 3 The unweighted construction G'. In bold, the conditions for the existence of a vertex or of an edge. The pairs in red recall, for vertices of the corresponding set, the length of their vector tuple followed by the length of their index tuple.

We also show the following inclusions.

▶ **Lemma 7.** *$N[I] \supset P \cup T''$, and $N^2[I] \supset P \cup C \cup T'' \cup T'$.*

Proof. $N[I] \supset P$, $N^2[I] \supset C$, and $N[T''] \supset T'$ have all been shown in Lemma 6. Therefore we shall just prove that $N[I] \supset T''$. Indeed every vertex $(a, b, c)_{T''} \in T''$ is adjacent to some $(i, i, i, i, i) \in I$, since otherwise a, b, c is an orthogonal triple. ◀

For the case disjunction, initially imagine the K_7 with loops on vertices T, T', T'', C, P, I, I', where edges correspond to the kinds of pairs that are left to check. The following paragraphs remove all its edges in the order: all edges incident to I and to I', all remaining edges incident to P and to T'' but TP and TT'', all remaining edges incident to C, all remaining edges incident to T' as well the loop on T, the edge TP, and finally the edge TT''.

Between $u \in I \cup I'$ and $v \in V(G')$

As $G'[I \cup I']$ is a clique and, by Lemma 6, $N^3[I'] = V(G')$, then $N^4[u] = V(G')$ holds for every vertex $u \in I \cup I'$.

Between $u \in P \cup T''$ and $v \in P \cup C \cup T'' \cup T'$

For every $u \in P \cup T''$, by Lemma 7 and the fact that $G'[I]$ is a clique, $N^2[u] \supset I$ and, again by Lemma 7, $N^4[u] \supset P \cup C \cup T'' \cup T'$. In particular there is a path of length at most 4 from u to any vertex $v \in P \cup C \cup T'' \cup T'$.

The following two cases work as in G, since $(a, b, c)_T$ and $(a, b, c)_{T'}$ are twins in $G'[T \cup T' \cup C \cup P]$.

Between $u \in C$ and $v \in T \cup T' \cup C$

This holds by replacing the occurrence of (c,d,e) by $(c,d,e)_T$ or $(c,d,e)_{T'}$, and every occurrence of T by $T \cup T'$, in the paragraph *Between $u \in C$ and $v \in T \cup C$* of the weighted construction.

Between $u \in T \cup T'$ and $v \in T \cup T'$

Again this holds by replacing occurrences of (a,b,c) (resp. (c,d,e)) by $(a,b,c)_T$ or $(a,b,c)_{T'}$ (resp. $(c,d,e)_T$ or $(c,d,e)_{T'}$).

Between $u \in T$ and $v \in P$

This works as in G by following three edges of weight 1 from T to I, instead of a single edge of weight 3. For every $u = (a,b,c)_T \in T$ and $v = (\{d,e\},i,j,k) \in P$, there is a path $u = (a,b,c)_T - (a,b,c)_{T'} - (a,b,c)_{T''} - (p_1,p_2,i,j,k) - (\{d,e\},i,j,k) = v$ in G', with $p_1 = \mathrm{ind}(a,b,c,d)$ and $p_2 = \mathrm{ind}(a,b,c,e)$.

Between $u \in T$ and $v \in T''$

This case is the real novelty compared to G, and the reason for introducing I'. For every $u = (a,b,c)_T \in T$ and $v = (d,e,f)_{T''} \in T''$, there is a path $u = (a,b,c)_T - (a,b,c)_{T'} - (a,b,c)_{T''} - (i) - (d,e,f)_{T''} = v$ in G', with $i = \mathrm{ind}(a,b,d,e)$. The last two edges exist since $a[i] = b[i] = d[i] = e[i] = 1$.

4.3 The presence of orthogonal quadruple implies diameter at least 7

Again we assume that there is an orthogonal quadruple $a,b,c,d \in S$ such that a,b,c,d are pairwise distinct. We claim that there is no path of length at most 6 in G' between $u = (a,b,c)_T$ and $v = (d,c,b)_T$. Since the distance between T and $I \cup I'$ is at least 3, any TT-path of length at most 6 visits $I \cup I'$ at most once. For the sake of contradiction, let $\mathcal{P}$ be such a path that we further assume shortest (hence in particular chordless) and, among shortest uv-paths, having the fewest edges in $E(T',C)$. We will show that $\mathcal{P}$ cannot visit I', nor use any edge of $E(T',C)$. Finally we observe that TT-paths of length at most 6 in G' respecting these two interdictions are in length-preserving one-to-one correspondence with TT-paths in G.

$\mathcal{P}$ cannot visit I'

The only possible kind of a TT-path of length at most 6 visiting I' is $T - T' - T'' - I' - T'' - T' - T$. This forces $\mathcal{P}$ to be of the form $(a,b,c)_T - (a,b,c)_{T'} - (a,b,c)_{T''} - (i) - (d,c,b)_{T''} - (d,c,b)_{T'} - (d,c,b)_T$ for some $i \in [\ell]$. However the third and fourth edges imply that there is an $i \in [\ell]$ such that $a[i] = b[i] = c[i] = d[i] = 1$, a contradiction to the orthogonality of a,b,c,d.

$\mathcal{P}$ cannot use any edge of $E(T',C)$

Assuming that $\mathcal{P}$ contains at least one edge in $E(T',C)$, we first show that it has to contain a subpath $C - T' - T'' - I \cup I'$ or $I \cup I' - T'' - T' - C$. Let $w = (a',b',c')_{T'} \in T' \cap V(\mathcal{P})$ be a vertex of $\mathcal{P}$ with one neighbor $x \in C \cap V(\mathcal{P})$ on $\mathcal{P}$. The other neighbor y of w on $\mathcal{P}$ is necessarily in T''. Indeed if $y \in T$, then $y = (a',b',c')_T$, and $xy \in E(G')$ is a chord. If

instead $y \in C$, then one can replace the subpath $x - (a', b', c')_{T'} - y$ by $x - (a', b', c')_T - y$, contradicting the minimality of the number of used edges in $E(T', C)$ (since this number decreases by 2).

Thus the only possibility is that $y \in T''$. Then the other neighbor of y on $\mathcal{P}$ (other than w) has to be in $I \cup I'$, since otherwise $\mathcal{P}$ is not a *simple* path. Hence $\mathcal{P}$ contains a subpath of the kind $C - T' - T'' - I \cup I'$ (or the reverse, $I \cup I' - T'' - T' - C$). Now we observe that C is at distance at least 1 from T, while $I \cup I'$ is at distance at least 3 from T. Therefore such a path $\mathcal{P}$ would have length at least 7.

Such a path $\mathcal{P}$ would also exist in G

We can now assume that $\mathcal{P}$ does not use any vertex of I' nor any edge of $E(T', C)$. Every such *simple* TT-path (visiting I at most once) also *exists* in the weighted graph G, with the same length. To see it, we notice that if $\mathcal{P}$ contains an edge $(a', b', c')_T - (a', b', c')_{T'}$, then it has to contain a subpath of the form $(a', b', c')_T - (a', b', c')_{T'} - (a', b', c')_{T''} - (p_1, p_2, i, j, k) \in I$, and is emulated in G by taking the weight-3 edge $(a', b', c') - (p_1, p_2, i, j, k)$. However we showed in the previous section that no uv-path of length at most 6 exists in G.

References

1 Donald Aingworth, Chandra Chekuri, Piotr Indyk, and Rajeev Motwani. Fast Estimation of Diameter and Shortest Paths (Without Matrix Multiplication). *SIAM J. Comput.*, 28(4):1167–1181, 1999. doi:10.1137/S0097539796303421.

2 Arturs Backurs, Liam Roditty, Gilad Segal, Virginia Vassilevska Williams, and Nicole Wein. Towards tight approximation bounds for graph diameter and eccentricities. In Ilias Diakonikolas, David Kempe, and Monika Henzinger, editors, *Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2018, Los Angeles, CA, USA, June 25-29, 2018*, pages 267–280. ACM, 2018. doi:10.1145/3188745.3188950.

3 Édouard Bonnet. Inapproximability of Diameter in super-linear time: Beyond the 5/3 ratio. *CoRR, To appear at STACS 2021*, abs/2008.11315, 2020. arXiv:2008.11315.

4 Massimo Cairo, Roberto Grossi, and Romeo Rizzi. New Bounds for Approximating Extremal Distances in Undirected Graphs. In Robert Krauthgamer, editor, *Proceedings of the Twenty-Seventh Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2016, Arlington, VA, USA, January 10-12, 2016*, pages 363–376. SIAM, 2016. doi:10.1137/1.9781611974331.ch27.

5 Marco L. Carmosino, Jiawei Gao, Russell Impagliazzo, Ivan Mihajlin, Ramamohan Paturi, and Stefan Schneider. Nondeterministic extensions of the strong exponential time hypothesis and consequences for non-reducibility. In Madhu Sudan, editor, *Proceedings of the 2016 ACM Conference on Innovations in Theoretical Computer Science, Cambridge, MA, USA, January 14-16, 2016*, pages 261–270. ACM, 2016. doi:10.1145/2840728.2840746.

6 Shiri Chechik, Daniel H. Larkin, Liam Roditty, Grant Schoenebeck, Robert Endre Tarjan, and Virginia Vassilevska Williams. Better Approximation Algorithms for the Graph Diameter. In Chandra Chekuri, editor, *Proceedings of the Twenty-Fifth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2014, Portland, Oregon, USA, January 5-7, 2014*, pages 1041–1052. SIAM, 2014. doi:10.1137/1.9781611973402.78.

7 Marek Cygan, Holger Dell, Daniel Lokshtanov, Dániel Marx, Jesper Nederlof, Yoshio Okamoto, Ramamohan Paturi, Saket Saurabh, and Magnus Wahlström. On Problems as Hard as CNF-SAT. *ACM Trans. Algorithms*, 12(3):41:1–41:24, 2016. doi:10.1145/2925416.

8 Marek Cygan, Fedor V. Fomin, Lukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michal Pilipczuk, and Saket Saurabh. *Parameterized Algorithms*. Springer, 2015. doi:10.1007/978-3-319-21275-3.

9 Mina Dalirrooyfard and Nicole Wein. Tight Conditional Lower Bounds for Approximating Diameter in Directed Graphs. *CoRR*, abs/2011.03892, 2020. arXiv:2011.03892.

10 Russell Impagliazzo and Ramamohan Paturi. On the complexity of k-sat. *J. Comput. Syst. Sci.*, 62(2):367–375, 2001. doi:10.1006/jcss.2000.1727.

11 Russell Impagliazzo, Ramamohan Paturi, and Francis Zane. Which Problems Have Strongly Exponential Complexity? *J. Comput. Syst. Sci.*, 63(4):512–530, 2001. doi:10.1006/jcss.2001.1774.

12 Ray Li. Improved SETH-hardness of unweighted Diameter. *CoRR*, abs/2008.05106, 2020. arXiv:2008.05106.

13 Ray Li. Settling SETH vs. Approximate Sparse Directed Unweighted Diameter (up to (NU)NSETH). *CoRR*, abs/2008.05106, 2020. arXiv:2008.05106.

14 Daniel Lokshtanov, Dániel Marx, and Saket Saurabh. Lower bounds based on the Exponential Time Hypothesis. *Bull. EATCS*, 105:41–72, 2011. URL: http://eatcs.org/beatcs/index.php/beatcs/article/view/92.

15 Liam Roditty and Virginia Vassilevska Williams. Fast approximation algorithms for the diameter and radius of sparse graphs. In Dan Boneh, Tim Roughgarden, and Joan Feigenbaum, editors, *Symposium on Theory of Computing Conference, STOC'13, Palo Alto, CA, USA, June 1-4, 2013*, pages 515–524. ACM, 2013. doi:10.1145/2488608.2488673.

16 Aviad Rubinstein and Virginia Vassilevska Williams. SETH vs Approximation. *SIGACT News*, 50(4):57–76, 2019. doi:10.1145/3374857.3374870.

17 Ryan Williams. A new algorithm for optimal 2-constraint satisfaction and its implications. *Theor. Comput. Sci.*, 348(2-3):357–365, 2005. doi:10.1016/j.tcs.2005.09.023.

18 Virginia Vassilevska Williams. On some fine-grained questions in algorithms and complexity. In *Proceedings of the ICM*, volume 3, pages 3431–3472. World Scientific, 2018.

Twin-width III: Max Independent Set, Min Dominating Set, and Coloring

Édouard Bonnet
Univ Lyon, CNRS, ENS de Lyon, Université Claude Bernard Lyon 1, LIP UMR5668, France

Colin Geniet
University of Warsaw, Poland

Eun Jung Kim
Université Paris-Dauphine, PSL University, CNRS UMR7243, LAMSADE, Paris, France

Stéphan Thomassé
Univ Lyon, CNRS, ENS de Lyon, Université Claude Bernard Lyon 1, LIP UMR5668, France

Rémi Watrigant
Univ Lyon, CNRS, ENS de Lyon, Université Claude Bernard Lyon 1, LIP UMR5668, France

Abstract

We recently introduced the notion of twin-width, a novel graph invariant, and showed that first-order model checking can be solved in time $f(d,k)n$ for n-vertex graphs given with a witness that the twin-width is at most d, called d-contraction sequence or d-sequence, and formulas of size k [Bonnet et al., FOCS '20]. The inevitable price to pay for such a general result is that f is a tower of exponentials of height roughly k. In this paper, we show that algorithms based on twin-width need not be impractical. We present $2^{O(k)}n$-time algorithms for k-Independent Set, r-Scattered Set, k-Clique, and k-Dominating Set when an $O(1)$-sequence of the graph is given in input. We further show how to solve the weighted version of k-Independent Set, Subgraph Isomorphism, and Induced Subgraph Isomorphism, in the slightly worse running time $2^{O(k \log k)}n$. Up to logarithmic factors in the exponent, all these running times are optimal, unless the Exponential Time Hypothesis fails. Like our FO model checking algorithm, these new algorithms are based on a dynamic programming scheme following the sequence of contractions forward.

We then show a second algorithmic use of the contraction sequence, by starting at its end and rewinding it. As an example of such a reverse scheme, we present a polynomial-time algorithm that properly colors the vertices of a graph with relatively few colors, thereby establishing that bounded twin-width classes are χ-bounded. This significantly extends the χ-boundedness of bounded rank-width classes, and does so with a very concise proof. It readily yields a constant approximation for Max Independent Set on K_t-free graphs of bounded twin-width, and a $2^{O(\mathrm{OPT})}$-approximation for Min Coloring on bounded twin-width graphs. We further observe that a constant approximation for Max Independent Set on bounded twin-width graphs (but arbitrarily large clique number) would actually imply a PTAS.

The third algorithmic use of twin-width builds on the second one. Playing the contraction sequence backward, we show that bounded twin-width graphs can be edge-partitioned into a linear number of bicliques, such that both sides of the bicliques are on consecutive vertices, in a fixed vertex ordering. This property is trivially shared with graphs of bounded average degree. Given that biclique edge-partition, we show how to solve the unweighted Single-Source Shortest Paths and hence All-Pairs Shortest Paths in time $O(n \log n)$ and time $O(n^2 \log n)$, respectively. In sharp contrast, even Diameter does not admit a truly subquadratic algorithm on bounded twin-width graphs, unless the Strong Exponential Time Hypothesis fails.

The fourth algorithmic use of twin-width builds on the so-called *versatile tree of contractions* [Bonnet et al., SODA '21], a branching and more robust witness of low twin-width. We present constant-approximation algorithms for Min Dominating Set and related problems, on bounded twin-width graphs, by showing that the integrality gap is constant. This is done by going down the versatile tree and stopping accordingly to a problem-dependent criterion. At the reached node, a greedy approach yields the desired approximation.

48th International Colloquium on Automata, Languages, and Programming (ICALP 2021).
Editors: Nikhil Bansal, Emanuela Merelli, and James Worrell; Article No. 35; pp. 35:1–35:20

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

2012 ACM Subject Classification Theory of computation → Graph algorithms analysis; Theory of computation → Fixed parameter tractability

Keywords and phrases Twin-width, Max Independent Set, Min Dominating Set, Coloring, Parameterized Algorithms, Approximation Algorithms, Exact Algorithms

Digital Object Identifier 10.4230/LIPIcs.ICALP.2021.35

Category Track A: Algorithms, Complexity and Games

Related Version *Full Version*: https://arxiv.org/abs/2007.14161

1 Introduction

As the title suggests, this is the third paper of a series [4, 3] devoted to a new graph invariant called *twin-width*. All the results presented in this paper are self-contained as the relevant background is given in Section 2. In the same section, the reader can find the definitions of *contraction sequences* and *twin-width*. For now, we are content with some intuition on these notions. This will be enough to sketch the ideas and techniques leading to our results, while sparing this introduction from too much formalism.

The twin-width of a graph is a non-negative integer measuring its distance to being a cograph. Among the several characterizations of cographs, a possible definition goes as follows. A graph is a *cograph* if one can find therein two twins,[1] identify them, and iterate this process until there is only one vertex left. This corresponds to what we define as a 0-sequence in Section 2, witnessing that cographs have twin-width 0. Conversely it is also true that graphs with twin-width 0 are cographs. We generalize this identification process by allowing a controlled error on the contracted pairs of vertices. An error graph or *red graph* keeps the faulty adjacencies appearing between a contracted pair and the vertices that are neighbor of only one vertex of the pair. A d-sequence is an indentification or contraction sequence such that the maximum degree of the error graph never exceeds d. The existence of such a sequence entails that the initial graph has twin-width at most d.

As it turns out, many graph classes have bounded twin-width: planar graphs and more generally proper minor-closed classes, bounded rank-width or clique-width graphs, proper hereditary subclasses of permutation graphs, unit interval graphs, and some particular class of cubic expanders, to name only a few.[2] Considering the wide variety of these classes, it might seem that our cograph generalization has gone too far to allow for a unified algorithmic treatment of bounded twin-width graphs. The first paper of the series [4] and the current one show that this is not the case. Graphs of bounded twin-width admit algorithms whose running times are provably unattainable in general graphs. We will now detail that point.

After defining any graph parameter κ, a natural question is whether some computationally hard problems can be solved more efficiently on graphs where κ is bounded. When this turns out to be the case for several problems, it may sometimes lead to a powerful meta-theorem. A standard way of capturing a large set of problems within the same framework is through the use of logic formulas over graphs, or more generally over relational structures. In the language of parameterized algorithms, one may ask for the existence of a Fixed-Parameter Tractable (FPT) algorithm parameterized by κ and the size of the graph formula φ to be tested: More precisely, an algorithm deciding in time $f(|\varphi|, \kappa(G))n^{O(1)}$, or better

[1] i.e., two vertices with the same neighborhood beside them
[2] A more exhaustive list is given in Theorem 7.

$f(|\varphi|, \kappa(G))n$, whether an n-vertex graph G satisfies φ, where f is some computable function. Certainly the most famous result of that kind is the celebrated Courcelle's theorem, where the parameter κ is tree-width, and the formula φ ranges over Monadic Second Order logic (MSO_2) formulas [5]. On a slightly less general logic (namely MSO_1, where quantification over edge sets is disallowed), the result holds for the smaller parameter clique-width [6]. It implies, for instance, that deciding whether a graph on n vertices contains a subset of k pairwise non-adjacent vertices (i.e., solving k-Independent Set) can be done in linear time on graphs of constant clique-width, while in general graphs it cannot be solved in polynomial time unless P=NP, nor in time $f(k)n^{O(1)}$ unless FPT=W[1]. Such a result is unlikely for twin-width as k-Independent Set remains NP-hard in planar graphs, which have constant twin-width. Nevertheless, when parameterized by the solution size k, an FPT algorithm is known in planar graphs, and more generally in any proper minor-closed graph class. Actually, on the latter class, every problem expressible by a first-order (FO) formula φ can be solved in FPT time parameterized by $|\varphi|$ [9]. In the first paper of our series [4], we extended this result and obtained the following meta-theorem for twin-width.

▶ **Theorem 1** ([4]). *Given an n-vertex graph G, a d-sequence of G, and a first-order formula φ, one can decide $G \models \varphi$ in time $f(|\varphi|, d)n$ for some computable function f.*

The main drawback of this kind of algorithm is the obtained running time: The function f is a tower of exponentials whose height depends on the size of the formula. This is an unavoidable price to pay to solve at once all graph problems expressible in first-order logic. Indeed, it is known that testing first-order formulas on trees requires a running time whose dependence in the size of the formula is a non-elementary function, unless P = NP [10]. Furthermore the running time of our FO model checking algorithm does not get better on "seemingly simpler" formulas, such as for instance, with few quantifier alternations.

Our results

We show that twin-width and its associated contraction sequence can also give rise to practical algorithms for some individual classic graph problems. In particular, we consider the following NP-complete problems, given a graph G and an integer k, decide if:

- k-Independent Set: there are k pairwise non-adjacent vertices.
- k-Clique: there are k pairwise adjacent vertices.
- (k, r)-Scattered Set: there are k vertices pairwise at distance at least r.
- k-Dominating Set: there is a set S of k vertices such that for every vertex v of G, either $v \in S$ or v has a neighbor in S.
- (k, r)-Dominating Set: there is a set S of k vertices such that every vertex of G is at distance at most r of some vertex in S.

These problems, parameterized by k, are W[1]-hard (the last two are even W[2]-complete), thus unlikely to admit an FPT algorithm, i.e., one with running time $f(k)n^{O(1)}$, on general graphs. We obtain single-exponential parameterized algorithms for all these problems when a contraction sequence witnessing "twin-width at most d" is given. When considering the unparameterized optimization variant, we denote these five problems by Max Independent Set (and MIS for short), Max Clique, Distance-$(r-1)$ MIS, Min Dominating Set, and Min r-Dominating Set, respectively.

▶ **Theorem 2.** *Given an n-vertex graph G and a d-sequence $G = G_n, \ldots, G_1 = K_1$, the above-mentioned five problems can be solved in time $2^{O_d(k)}n$.*

We then consider some W[1]-complete generalizations of k-Independent Set or of k-Clique. Namely:

- Weighted Max Independent Set: given a graph G with a weight function on vertices $w : V(G) \to \mathbb{R}$ and an integer k, decide whether there exists a set S of size exactly k of pairwise non-adjacent vertices such that $\sum_{v \in S} w(v)$ is maximum.
- Induced Subgraph Isomorphism: given a graph H on k vertices and a graph G, decide whether there exists a set $S \subseteq V(G)$ such that $G[S]$, the subgraph of G induced by S, is isomorphic to H.
- Subgraph Isomorphism: given a graph H on k vertices and a graph G, decide whether there exists a set $S \subseteq V(G)$ such that H is isomorphic to a subgraph of $G[S]$.

Unlike the other two problems, Subgraph Isomorphism is *not* a generalization of k-Independent Set. Though it does generalize k-Clique. Once the formal definition of a contraction sequence is given, it will be clear that a d-sequence for G readily yields a d-sequence for its complement, $\overline{G}$. Thus in the context of bounded twin-width graphs, an algorithm solving Subgraph Isomorphism can be used to solve k-Independent Set. For these three problems, we now get slightly superexponential parameterized algorithms.

▶ **Theorem 3.** *Given an n-vertex graph G and a d-sequence $G = G_n, \ldots, G_1 = K_1$, the above-mentioned three problems can be solved in time $2^{O_d(k \log k)} n$.*

The algorithms behind Theorems 2 and 3 follow the same general plan. Let us consider the n successive red graphs $R_n, \ldots, R_1$ (error graphs) obtained after each vertex contraction.[3] R_n is the edgeless n-vertex graph (since there are initially no errors) and R_1 is the 1-vertex graph. We maintain optimum partial solutions populating connected subgraphs of bounded size in each R_i. Initially in R_n, the connected subgraphs are only made of single vertices (there are no edges). So the optimum partial solutions are trivial to compute. The partial solutions for R_i are built from the partial solutions of R_{i+1} in the following way. Every partial solution *not* involving the newly contracted vertex is simply kept. Every partial solution involving the newly contracted vertex is computed by merging a bounded number of previous partial solutions on pairwise disconnected sets. The key is that, by design, there is no error between the latter partial solutions. Thus the presence or absence of edges can be decided regardless of the forgotten choices of precise vertices within the solution. Eventually a (partial) solution is computed in R_1, which constitutes an actual solution in the entire initial graph G. In a nutshell, the algorithms may be summarized as dynamic programming over connected sets of the red graphs.

For k-Independent Set there is not much more to it than the previous sketch. For (Induced) Subgraph Isomorphism the algorithms become more technical. Also conceptually, partial solutions are no longer necessarily feasible. For k-Dominating Set some new challenges appear. The partial solutions and their actual specification are not straightforward to define, as it is for k-Independent Set.

One may wonder if subexponential parameterized algorithms are possible for any of the eight problems considered so far. We will observe that even k-Independent Set cannot be solved in time $2^{o(k/\log k)} n^{O(1)}$ on graphs given with an $O(1)$-sequence, unless the Exponential Time Hypothesis fails. With a similar argument, the same lower bound applies to k-Dominating Set. Thus, up to logarithmic factors in the exponent, the running times of Theorems 2 and 3 are optimal. Actually we will see that even algorithms running in time $2^{o(n/\log n)}$ are unlikely.

[3] A reader who would want precise definitions at this point is welcome to read first the couple of paragraphs of Section 2.1.

All the previous algorithms exploit the contraction sequence forward. They follow the identification process from the initial graph G to the 1-vertex graph. What if we would start at the end, and maintain solutions as the vertices are iteratively split until the initial graph G is formed? We exemplify the idea of using the contraction sequence backward with an essentially greedy coloring procedure that is not optimal but still uses relatively few colors.

Let us be more specific. A proper k-coloring of a graph G is a mapping $c : V(G) \to \{1, \dots, k\}$ such that $c(u) \neq c(v)$ whenever $uv \in E(G)$. The chromatic number, denoted by $\chi(G)$, is the smallest integer k such that G admits a proper k-coloring. It can be seen that $\chi(G) \geqslant \omega(G)$, where $\omega(G)$ denotes the size of a largest clique in G, whereas many constructions of triangle-free (that is, with $\omega(G) \leqslant 2$) graphs G with arbitrarily large $\chi(G)$ are known. A class of graphs $\mathcal{C}$ is *χ-bounded* if there is a function f such that for any graph $G \in \mathcal{C}$, we have $\chi(G) \leqslant f(\omega(G))$. Our coloring algorithm $(d+2)$-colors any triangle-free graph of twin-width at most d, and more generally $(d+2)^{\omega(G)-1}$-colors any graph G given with a d-sequence. In particular, it shows the following.

▶ **Theorem 4.** *Every graph class with bounded twin-width is χ-bounded.*

Algorithmically this has some direct consequences for approximating the chromatic number, as well as, in the subcase of K_t-free graphs, the independence number.

The same idea of considering the contraction sequence backward is then used to show that every graph given with an $O(1)$-sequence admits an edge partition into $O(n)$ bicliques, each side of which is on consecutive vertices, for a fixed vertex ordering. We use this edge partition to tackle the unweighted version of some classic polynomial-time solvable problems:

- Single-Source Shortest Paths: given a graph G and a source s, find a shortest-path tree rooted at s, spanning the connected component of s.
- All-Pairs Shortest Paths: given a graph G, find the distances in G between every pair of vertices.
- Diameter: given a graph G, report the largest distance in G between two vertices.

We show how breadth-first search (BFS) can be mimicked, when replacing "traversing an edge" by "traversing a biclique all at once". A subtlety of the algorithm, beside the necessary data structures to get Single-Source Shortest Paths sublinear in the total number of edges, lies in the fact that bicliques, contrary to single edges, can be traversed twice (once in both directions) before being discarded.

▶ **Theorem 5.** *If the input graph comes with an $O(1)$-sequence, Single-Source Shortest Paths can be solved in $O(n \log n)$ time, thus All-Pairs Shortest Paths and Diameter can be solved in $O(n^2 \log n)$ time. In contrast, Diameter cannot be solved in $O(n^{2-\varepsilon})$ for any $\varepsilon > 0$, even in that scenario, unless the Strong Exponential Time Hypothesis fails.*

Our algorithm inherently relies on unweighted edges. Nonetheless vertex-weights can be supported with the same running time.

Min Dominating Set is known to be as approximable as the Set Cover problem. Thus, by classic papers by Johnson [14] and by Lovász [15], it admits a $\ln n$-approximation and the integrality gap (i.e., the ratio between the optimum of the original problem and the optimum of the LP relaxation) of its standard LP formulation is also $\ln n$. In sharp contrast, unless P=NP, Min Dominating Set cannot be approximated in polynomial-time within factor $(1 - o(1)) \ln n$ on n-vertex general graphs [7].

We show that, on bounded twin-width classes, the integrality gap of Min Dominating Set is constant. This uses the *versatile trees of contractions* developed in the second paper of the series [3]. These are more robust witnesses of low twin-width which, instead of providing

a single contraction in a given trigraph, give linearly many disjoint ones. Placing ourselves at a right node of the versatile tree, we show that a greedy strategy in the corresponding trigraph yields a constant approximation in the original graph.

▶ **Theorem 6.** *If the input graph comes with an $O(1)$-sequence, Min Dominating Set, Distance-2 MIS, and more generally Min r-Dominating Set, Distance-$2r$ MIS for every positive r, admit $O(1)$-approximation algorithms.*

These results are particular cases of the fact that when the twin-width of a matrix A is bounded, there is a linear gap between the packing number and the minimum hitting set of the hypergraph with incidence matrix A. Bounded twin-width matrices might more generally provide linear programs with bounded duality gap. It is noteworthy that Max Independent Set (which corresponds to Distance-1 MIS) is *not* covered by the previous theorem. We further give some evidence that MIS may have a very different approximability status than Min Dominating Set on bounded twin-width graphs.

2 Preliminaries

We denote by $[i, j]$ the set of integers $\{i, i+1, \ldots, j-1, j\}$, and by $[i]$ the set of integers $[1, i]$. If $\mathcal{X}$ is a set of sets, we denote by $\cup\mathcal{X}$ their union. The notation $O_d(\cdot)$ gives an asymptotic behavior when d is seen as a constant. The notation $O^*(\cdot)$ suppresses polynomial factors.

Unless stated otherwise, all graphs are assumed undirected and simple, that is, they do not have parallel edges or self-loops. We denote by $V(G)$ and $E(G)$ the set of vertices and edges respectively of a graph G. For $S \subseteq V(G)$, we denote the *open neighborhood* (or simply *neighborhood*) of S by $N_G(S)$, i.e., the set of neighbors of S deprived of S, and the *closed neighborhood* of S by $N_G[S]$, i.e., the set $N_G(S) \cup S$. We may omit the subscript if the graph is clear from the context. We denote by $G[S]$ the subgraph of G induced by S, and $G - S := G[V(G) \setminus S]$. A *connected subset* (or *connected set*) $S \subseteq V(G)$ is one such that $G[S]$ is connected. Two distinct vertices u, v such that $N(u) = N(v)$ are called *false twins*, and *true twins* if $N[u] = N[v]$. Two vertices are *twins* if they are false twins or true twins.

A graph is *H-free* if it does not contain H as an induced subgraph. However we make an exception for $H = K_{t,t}$. A $K_{t,t}$-free graph is a graph with no biclique $K_{t,t}$ *as a subgraph*. A class[4] $\mathcal{C}$ of graphs *has property Π* if every graph of $\mathcal{C}$ has property Π. A class is *hereditary* if it is closed under taking induced subgraphs.

2.1 Trigraphs, contraction sequences, and twin-width of a graph

A *trigraph* G has vertex set $V(G)$, (black) edge set $E(G)$, and red edge set $R(G)$ (the error edges), with $E(G)$ and $R(G)$ being disjoint. The *set of neighbors* $N_G(v)$ of a vertex v in a trigraph G consists of all the vertices adjacent to v by a black or red edge. A d-trigraph is a trigraph G such that the *red graph* $(V(G), R(G))$ has degree at most d. In that case, we also say that the trigraph has *red degree* at most d. A (vertex) *contraction* or *identification* in a trigraph G consists of merging two (non-necessarily adjacent) vertices u and v into a single vertex z, and updating the edges of G in the following way. Every vertex of the symmetric difference $N_G(u) \triangle N_G(v)$ is linked to z by a red edge. Every vertex x of the intersection $N_G(u) \cap N_G(v)$ is linked to z by a black edge if both $ux \in E(G)$ and $vx \in E(G)$, and by a red

[4] That is, a set of graphs closed under isomorphism.

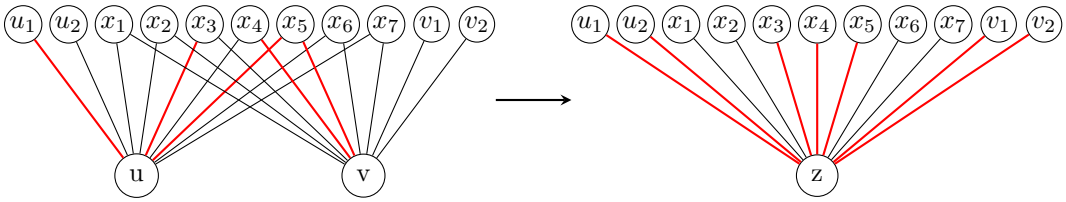


Figure 1 Contraction of vertices u and v, and how the edges of the trigraph are updated.

edge otherwise. The rest of the edges (not incident to u or v) remain unchanged. We insist that the vertices u and v (together with the edges incident to these vertices) are removed from the trigraph. See Figure 1 for an illustration.

A *d-sequence* (or *contraction sequence*) is a sequence of d-trigraphs $G_n, G_{n-1}, \dots, G_1$, where $G_n = G$, $G_1 = K_1$ is the graph on a single vertex, and G_{i-1} is obtained from G_i by performing a single contraction of two (non-necessarily adjacent) vertices. We observe that G_i has precisely i vertices, for every $i \in [n]$. The twin-width of G, denoted by $tww(G)$, is the minimum integer d such that G admits a d-sequence.

For $u \in V(G_i)$, we denote by $u(G)$ the subset of $V(G)$ that was contracted to the single vertex u in $G_n, G_{n-1}, \dots, G_i$. Twin-width and d-sequences can be equivalently seen as a partition refinement process on $V(G)$. We start with the finest partition $\mathcal{P}_n = \{\{v\} : v \in V(G)\}$, and end with the coarsest partition $\mathcal{P}_1 = \{V(G)\}$. There is a *partition sequence* $\mathcal{P}_n, \mathcal{P}_{n-1}, \dots, \mathcal{P}_2, \mathcal{P}_1$ mimicking the contraction sequence, where the contraction of $u, v \in V(G_i)$ corresponds to the merge of parts $u(G_i), v(G_i) \in \mathcal{P}_i$ to form the part $u(G_i) \cup v(G_i) = z(G_{i-1}) \in \mathcal{P}_{i-1}$, while all the other parts are unchanged from P_i to P_{i-1}. The red degree (bounded by d) of a part $P \in \mathcal{P}_i$ now corresponds to the number of other parts $P' \in \mathcal{P}_i$ which are not fully adjacent nor fully non-adjacent to P in G. We may denote by $G_{\mathcal{P}}$ the trigraph corresponding to partition $\mathcal{P}$ over $V(G)$. Thus $G_i = G_{\mathcal{P}_i}$.

2.2 Classes with bounded twin-width and how the sequences are given

The current paper is devoted to presenting efficient algorithms when the input has bounded twin-width, and the contraction sequence is given. It is therefore important to know how realistic this scenario is. Fortunately, in the first two papers of the series [4, 3] we showed that many central sparse and dense (di)graph classes have bounded twin-width.

▶ **Theorem 7** ([4, 3]). *The following classes have bounded twin-width, and $O(1)$-sequences for n-vertex members can be computed in $O(n^2)$ time.*

- *Bounded clique-width/rank-width, and more generally, boolean-width graphs,*
- *every hereditary proper subclass of permutation graphs,*
- *posets of bounded antichain size (seen as digraphs),*
- *unit interval graphs,*
- *K_t-minor free graphs,*
- *map graphs (given with an embedding),*
- *subgraphs of d-dimensional grids,*
- *K_t-free unit d-dimensional ball graphs,*
- *$\Omega(\log n)$-subdivisions of all the n-vertex graphs,*
- *cubic expanders defined by iterative random 2-lifts from K_4,*
- *strong products of two bounded twin-width classes one of which has also bounded degree,*
- *any subgraph closure of a $K_{t,t}$-free bounded twin-width class, and*
- *any first-order transduction of a bounded twin-width class.*

2.3 Selected results for the short version

Due to space constraints, we select a representative sample of the results announced in the introduction. This sample covers at least partially Theorems 2–4 and 6. We present the following four items on bounded twin-width classes, where the input graph comes with an $O(1)$-sequence; each item sharply contrasting with what is possible on general graphs.

- In Section 3 we give a linear FPT algorithm for k-Independent Set.
- In Section 4 we give a linear FPT algorithm for k-Dominating Set.
- In Section 5 we show a constant approximation for Min Dominating Set.
- In Section 6 we show that bounded twin-width graphs are χ-bounded.

That selection presents our new conceptual ideas in their simplest form, while echoing the title of the paper. For more details on these results or for the proofs *not* covered in the short version, we refer the reader to the long version in appendix.

3 Practical algorithm for k-Independent Set

The running time analysis of the forthcoming algorithm is based on a folklore bound on the number of connected subsets of size at most k in a bounded-degree graph.

▶ **Lemma 8.** *The number of connected vertex sets of size at most k, intersecting a set X, in a graph of maximum degree d is at most $(d^{2k-2}+1)|X|$. Furthermore they can be enumerated in time $O(d^{2k-2}|X|)$.*

We show how to solve k-Independent Set by dynamic programming on the connected subsets of size at most k in the red graphs of a d-sequence given with the input graph.

▶ **Theorem 9.** *Given an n-vertex graph G, a positive integer k, and a d-sequence $G = G_n, \dots, G_1 = K_1$, k-Independent Set can be solved in time $O(k^2 d^{2k} n) = 2^{O_d(k)} n$.*

Proof. Our algorithm maintains a set of *optimum partial solutions* in the current trigraph, starting from G, and progressively going along the d-sequence. Let us start with a definition of the partial solutions and of their optimality.

A *partial solution* in the trigraph G_i is a pair (T, S) where $T \subseteq V(G_i)$ is a vertex set inducing a connected subgraph in the red graph $(V(G_i), R(G_i))$, and $S \subseteq V(G)$ is an independent set of G such that $S \subseteq \bigcup_{u \in T} u(G)$ and for every $u \in T$, $S \cap u(G) \neq \emptyset$. A partial solution (T, S) is said *optimum* if there is no partial solution (T, S') such that $|S| < |S'|$. A set $T \subseteq V(G_i)$ is said *realizable* (in G_i) if there is an $S \subseteq V(G)$ such that (T, S) is a partial solution in G_i. Notice that *not* every connected subset in the red graph is realizable. For instance, it is easy to engineer a situation where there is no independent set intersecting the three vertices of a 3-vertex red path. Initially, in G, the only connected subgraphs of the red graph are singletons (since there is no red edge). So there are exactly n (optimum) partial solutions in $G = G_n$: Each vertex v of G induces a partial solution $(\{v\}, \{v\})$. We denote by $\mathcal{S}_n$ this set of n optimum partial solutions. It boils down to determining if there is a partial solution $(\_, S)$ in G_1 (or actually in any G_i) with $|S| \geqslant k$. For i going from $n-1$ down to 1, we will build a set of optimum partial solutions $\mathcal{S}_i$ in G_i from the set $\mathcal{S}_{i+1}$, keeping the invariant that for every realizable set $T \subseteq V(G_i)$, there is a unique optimum partial solution (T, S) stored in $\mathcal{S}_i$ (and no other partial solution in $\mathcal{S}_i$).

We shall then describe how we update the set of optimum partial solutions after a single contraction. Two partial solutions $(T, \_)$ and $(T', \_)$ in G_i are *disjoint* if $T \cap T' = \emptyset$, and *separate*, if they are disjoint and there is no red edge $uu' \in R(G_i)$ with $u \in T$ and

$u' \in T'$. Two separate partial solutions $(T, \_)$ and $(T', \_)$ are *compatible* if there is no edge $uu' \in E(G_i) \cup R(G_i)$ with $u \in T$ and $u' \in T'$. The *union* of two compatible partial solutions (T_1, S_1) and (T_2, S_2) as $(T_1, S_1) \cup (T_2, S_2) := (T_1 \cup T_2, S_1 \cup S_2)$. By definition, such a union is *not* a partial solution since T induces two connected components in its current red graph. Nevertheless we will build the new (connected) partial solutions of G_i by making unions of up to $d+2$ pairwise compatible partial solutions in G_{i+1}. These unions will be connected in G_i, hence will correspond to partial solutions as well.

Let us be more specific. Say $u, v \in V(G_{i+1})$ are contracted into $z \in V(G_i)$ to form G_i. We say that a partial solution $(T, \_)$ in G_i *intersects* a set $X \subseteq V(G_i)$ if $T \cap X \neq \emptyset$. We initialize $\mathcal{S}_i$ with all the partial solutions of $\mathcal{S}_{i+1}$ not intersecting $\{u, v\}$. We now add one partial solution in $\mathcal{S}_i$ per realizable set $T \ni z$ in G_i, of size at most k. For every $T \subseteq V(G_i)$ such that $z \in T$ and T induces a connected subgraph on at most k vertices in the red graph $(V(G_i), R(G_i))$, we observe three possibilities for a potential partial solution (T, S). Either S intersects $u(G)$ and $v(G)$, or it intersects only $u(G)$, or it intersects only $v(G)$. (It is not possible that $S \cap (u(G) \cup v(G)) = \emptyset$ since T contains z.) Therefore we take the best (meaning with the largest S, breaking ties arbitrarily) of the potential partial solutions $\bigcup \mathrm{dec}(T \setminus \{z\} \cup \{u, v\}), \bigcup \mathrm{dec}(T \setminus \{z\} \cup \{u\}), \bigcup \mathrm{dec}(T \setminus \{z\} \cup \{v\})$, where $\mathrm{dec}(X)$ is the set with one partial solution per connected component of X in its red graph (here $(V(G_{i+1}), R(G_{i+1}))$. See Figure 2 for an illustration of this decomposition.

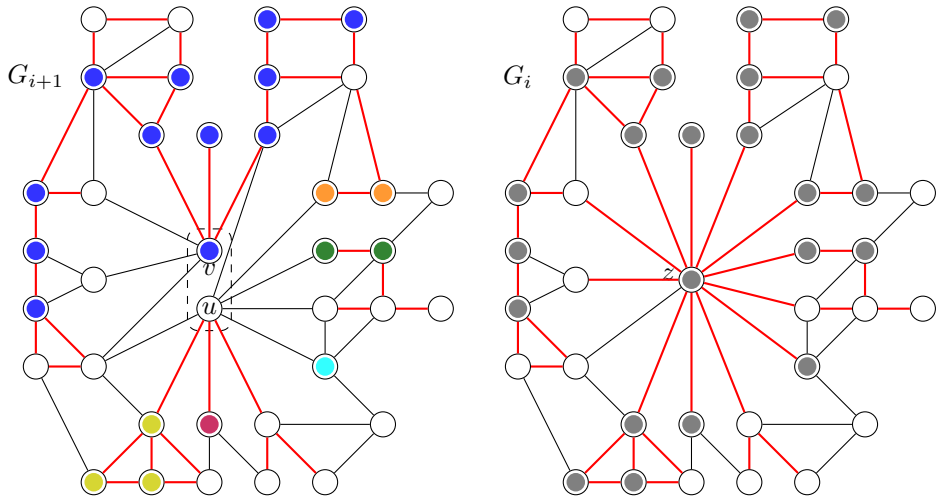


Figure 2 Right: In gray, a connected vertex set T in the red graph of G_i in the vicinity of the just contracted vertex $z \in T$. Left: The decomposition $\mathrm{dec}(T \setminus \{z\} \cup \{v\})$ in the previous trigraph G_{i+1}, where each color represents a connected component. If every color class is a realizable set in G_{i+1}, then T is realizable in G_i, with (optimum) partial solution $\bigcup \mathrm{dec}(T \setminus \{z\} \cup \{v\})$. Note that, due to black edges between u and some vertices of T, the partial solutions in $\mathrm{dec}(T \setminus \{z\} \cup \{u, v\})$ and in $\mathrm{dec}(T \setminus \{z\} \cup \{u\})$ cannot be pairwise compatible.

In the very possible event that at least one such connected component of X is not realizable, $\mathrm{dec}(X) = \text{None}$. The union $\bigcup \mathrm{dec}(X)$ of all the partial solutions of $\mathrm{dec}(X)$ is None if $\mathrm{dec}(X) = \text{None}$ or if there is at least one black edge between two connected components. Otherwise $\bigcup \mathrm{dec}(X)$ is a pair (T, S) as defined in the previous paragraph, since the partial solutions of $\mathrm{dec}(X)$ are pairwise compatible. Since T is chosen connected in $(V(G_i), R(G_i))$, (T, S) is indeed a partial solution in G_i. If $\bigcup \mathrm{dec}(T \setminus \{z\} \cup \{u, v\}), \bigcup \mathrm{dec}(T \setminus \{z\} \cup \{u\}), \bigcup \mathrm{dec}(T \setminus \{z\} \cup \{v\})$ all three evaluate to None, then $\mathrm{best}\{\bigcup \mathrm{dec}(T \setminus \{z\} \cup \{u, v\}), \bigcup \mathrm{dec}(T \setminus \{z\} \cup \{u\}), \bigcup \mathrm{dec}(T \setminus$

$\{z\} \cup \{v\})\}$ also returns None. This would mean that T is not realizable. If instead T is realizable, we get a partial solution (T, S) that we put in $\mathcal{S}_i$. If $|S| \geqslant k$, we already have a large enough independent set; the algorithm outputs it and terminates.

If we finally build $\mathcal{S}_1$, and no independent set of size at least k was found, we output S, the unique set such that $(\_, S) \in \mathcal{S}_1$. $\mathcal{S}_1$ is indeed a singleton since there is only one realizable set in G_1. That finishes the description of the algorithm.

Details on the correctness and running time can be found in the long version. The correctness uses the classic inductive arguments for an algorithm based on dynamic programming. The claimed running time for `k-IndSet` essentially follows from Lemma 8.

Optimizations. We suggest some improvements or variations of `k-IndSet` to generally improve over the worst-case running time of the inner for loop. A lot of sets T will trivially be *not* realizable because they induce a black edge. When enumerating the walks starting at z of length at most $2k-3$, one can abort every branch $zv_1 \dots v_h$ inducing at least one black edge. It can even be done in a way that the enumeration takes time $O(t)$ where t is the number of sets $T \ni z$ of size at most k, such that T is connected in the red graph, and an independent set in the black graph.

Even if a set T satisfies those properties, we have no guarantee that T is realizable. In very dense instances, it is imaginable that the realizable sets are very rare. In that case, we will lose a lot of time generating sets T to observe immediately after that there is no associated partial solution (T, S). An alternative to `k-IndSet` is to build the new partial solutions of $\mathcal{S}_i$ directly as unions of pairwise compatible partial solutions of $\mathcal{S}_{i+1}$, without anticipating the nature of the possibly realizable set $T \subseteq V(G)$.

Let us be more precise. Let R_z be the set of red neighbors of z in G_i. For every set of at most $\max(2, d+1)$ partial solutions $(T_1, S_1), \dots, (T_h, S_h) \in \mathcal{S}_{i+1}$ intersecting R_z, at least one of which intersects $\{u, v\}$, if the partial solutions are pairwise compatible, we update the realizable set $\bigcup_{i\in[h]} T_i$ with the partial solution $\bigcup_{i\in[h]}(T_i, S_i)$ if $\bigcup_{i\in[h]} S_i$ is larger than the current best solution. Following the first improvement, we can only generate the sets that are pairwise compatible. As we know, there are at most three ways to reach a given set $T \subseteq V(G_i)$ as a union of pairwise compatible partial solutions in $\mathcal{S}_{i+1}$. The running time of this variation of `k-IndSet` is $O^*(\Sigma_{i\in[n]}|\mathcal{S}_i^{\text{new}}|)$, where $S_i^{\text{new}} := \mathcal{S}_i \setminus \mathcal{S}_{i-1}$ (and $S_n^{\text{new}} := \mathcal{S}_n$) represents the new partial solutions computed at step i. In practice, this can be significantly better than $O(k^2 d^{2k} n)$. Such a dynamic programming, only generating "positive" subinstances, dubbed *positive-instance driven* by Tamaki, led to a breakthrough and current state-of-the-art practical algorithm for computing optimally the treewidth of a graph [17]. ◀

Without too many changes, `k-IndSet` may support weights, that is, find an independent set of size exactly $\min(k, \alpha(G))$ with largest total weight. Instead of keeping one solution S per realizable set T, we keep up to k solutions, one per pair (T, j) with $j \in [|T|, k]$. A partial solution (T, j, S) is defined as before except S is required to have size exactly j. To compute the new partial solutions, we add a third nested for loop after line 6: We iterate over all the ways of distributing $j \leqslant k$ units between the red connected components induced by $T' \in \{T \setminus \{z\} \cup \{u, v\}, T \setminus \{z\} \cup \{u\}, T \setminus \{z\} \cup \{v\}\}$ so that each connected component gets a positive integer (at least equal to its size). We then add to $\mathcal{S}_i$ one partial solution (T, j, S) (if at least one exists) maximizing the weight of S for fixed T and j. We also skip lines 8 and 9 of `k-IndSet`.

This comes with a slight increase in the running time. Namely, there is an extra $2^{O(k \log k)}$ factor accounting for the ordered partition of integer $j \leqslant k$ into positive integers. Thus the overall running time with weights is $2^{O(k \log k)} d^{2k} n$.

As twin-width and d-sequences are preserved when complementing the graph, we also solve k-Clique in the same running time. One may wonder if the dependency in k of our $2^{O_d(k)}n$-time algorithm can be improved. It turns out that this running time is essentially optimal. Due to the Sparsification Lemma [13] and folklore reductions, MIS restricted to subcubic n-vertex graphs cannot be solved in $2^{o(n)}$, under the Exponential Time Hypothesis[5] (ETH) [12]. Thus, by the classic self-reduction consisting of performing an even subdivision of each edge [16], MIS cannot be solved in time $2^{o(n/\log n)}$ on $2\lceil \log n \rceil$-subdivisions of n-vertex subcubic graphs, unless the ETH fails. In [3], we show how to find $O(1)$-sequences in polynomial time for $2\lceil \log n \rceil$-subdivisions of n-vertex graphs. Therefore this lower bound holds even if we are given the d-sequence. In particular, no algorithm solves k-Independent Set in time $2^{o_d(k/\log k)}n^{O(1)}$, unless the ETH fails.

4 A practical algorithm for k-Dominating Set

We solve k-Dominating Set with a more involved instantiation of the scheme of the previous section. We face some new conceptual difficulties compared to the algorithm for k-Independent Set. For one thing, the partial solutions that we maintain are not feasible solutions in the whole graph. Also we now consider balls of radius $f(d)k$ in the red graphs, and not merely of radius k. In general, the arguments are more subtle to handle partially and fully dominated vertex sets, as well as the solution trace. This entails a worse dependency in d, but the same essentially optimal $2^{O(k)}n$ when d is treated as a constant.

▶ **Theorem 10.** *Given an n-vertex graph G, a positive integer k, and a d-sequence $G = G_n, \dots, G_1 = K_1$, k-Dominating Set can be solved in time $O(2^{2(d^2+1)(2+\log d)k}n) = 2^{O_d(k)}n$.*

Proof. As was the case with k-Independent Set, the algorithm sequentially considers each trigraph in the d-sequence $G_n, \dots, G_1$ starting from G_n, and inductively updates a set of optimal partial solutions of the trigraph G_i to yield the next set for G_{i-1}. We recall that $E(G_i)$ and $R(G_i)$ respectively refer to the black and red edge set of the trigraph G_i. The ball of radius at most r in the red graph $(V(G_i), R(G_i))$ centered at a vertex $x \in V(G_i)$ is denoted as $B_i^r(x)$.

Profile of a partial solution. A *profile (of a partial solution)* of G_i is a triple (T, D, M) of vertex sets of $V(G_i)$ such that (i) T forms a connected set in the red graph $(V(G_i), R(G_i))$, (ii) $D, M \subseteq T$, and (iii) $\bigcup_{x\in D} B_i^2(x) \subseteq T$. The first entry T of a profile $P = (T, D, M)$ is called the *ground set* of P, and the size of P is defined as the size of its ground set. A profile (T, D, M) is said to be a *k-profile* if $|D| \leqslant k$. When the profile under consideration is clear from the context, we denote $T \setminus D$ and $T \setminus M$ by $\bar{D}$ and $\bar{M}$ respectively.

We say that *a profile (T, D, M) is realizable with $S \subseteq V(G)$* if the following conditions hold.

1. $S \subseteq \bigcup_{x\in T} x(G)$,
2. for every $x \in V(G_i)$, $x \in D$ if and only if $x(G) \cap S \neq \emptyset$, and
3. for every $x \in V(G_i)$, $x \in M$ if and only if $x(G)$ is (fully) dominated by S.

A profile is said to be *realizable* if there exists S with which it is realizable.

Suppose that $x, y \in V(G_{i+1})$ are contracted to yield G_i with z being the new vertex. For a vertex set $T \subseteq V(G_i)$ connected in the red graph $V(G_i, R_i)$ and containing z, let $T_1, \dots, T_\ell$ be the red connected components of $T' = (T \setminus z) \cup \{x, y\}$ in G_{i+1}, i.e. the partition of T'

[5] The assumption that there is a constant $\delta > 0$, such that 3-SAT cannot be solved in time $2^{\delta n}$.

into maximal vertex sets each of which is connected in $V(G_{i+1}, R_{i+1})$. The number of these red subgraphs does not exceed $d+2$ because each T_i either contains x or y, or one of the newly created red neighbors of z. Notice also that ℓ can be equal to 1, which means that x and y belong to the same connected component of $(V(G_{i+1}), R(G_{i+1}))$.

For a k-profile (T, D, M) of G_i such that $z \in T$, we say that a set $\mathcal{P} = \{(T_1, D_1, M_1), \ldots, (T_\ell, D_\ell, M_\ell)\}$ of k-profiles of G_{i+1} is *consistent with* (T, D, M) if the following holds. Let $T' := (T \setminus z) \cup \{x, y\}$, $D' := \bigcup_{j=1}^{\ell} D_j$ and $M' := \bigcup_{j=1}^{\ell} M_j$.

1. The ground sets of the profiles in $\mathcal{P}$ are precisely the red components of T' in G_{i+1}.
2. $D \setminus z = D' \setminus \{x, y\}$.
3. $z \in D$ if and only if $x \in D'$ or $y \in D'$.
4. For every $u \in T \setminus z$, $u \in M$ if and only if $u \in M'$ or there exists $v \in D'$ such that uv is a black edge in G_{i+1}.
5. $z \in M$ if and only if for each $u \in \{x, y\}$, it holds that: $u \in M'$ or there exists $v \in D'$ such that uv is a black edge in G_{i+1}.

Algorithm, and how to compute τ_i from τ_{i+1}. At each iteration along the d-sequence, we maintain one mapping τ_i from k-profiles $P = (T, D, M)$ of G_i with $|T| < (d^2+1)k$ to a subset of $\bigcup_{t \in T} t(G)$. The assignment $\tau_i(P) = nil$ is interpreted as that P is not realizable whereas $\tau_i(P) \neq nil$ is intended to be a minimum-size vertex set of $V(G)$ realizing P. Again let G_i be obtained by contracting the vertices $x, y \in V(G_{i+1})$ and z be the new vertex. Our goal is to compute τ_i from τ_{i+1}, assuming τ_{i+1} has been computed correctly. Note that a k-profile $P = (T, D, M)$ of G_i such that $z \notin T$ is also a profile of G_i, and trivially one is realizable with S if and only if the other is realizable with S. Therefore, τ_i simply inherits the assignment of τ_{i+1} in this case as depicted in lines 6-7.

If $P = (T, D, M)$ has z in its ground set, the algorithm `k-DomSet` inspects all sets $\mathcal{P}$ of k-profiles of G_{i+1} consistent with (T, D, M) and among the unions $\bigcup_{P \in \mathcal{P}} \tau_{i+1}(P)$ over all such $\mathcal{P}$, outputs the best one as $\tau_i(T, D, M)$, that is, the one of minimum cardinality is chosen. If $\bigcup_{P \in \mathcal{P}} \tau_{i+1}(P) = nil$ for each consistent $\mathcal{P}$, the algorithm concludes that (T, D, M) is not realizable and assigns nil. The case when $\mathcal{P}$ contains a k-profile P with ground set of size at least $(d^2+1)k$, a special step is taken as τ_{i+1} is not defined on such P. In this situation, a vertex $v \in T' \setminus \bigcup_{t \in D'} B^2_{i+1}(t)$ is chosen, and the query at $(T' \setminus v, D' \setminus v, M' \setminus v)$ is made instead. Lines 15-18 handle this case. The uniqueness of k-profile in $\mathcal{P}$ in line 16 and the existence of such v in line 17 will be discussed in the correctness proof.

Correctness. To show the correctness of Algorithm 1, it suffices to prove the following.

> ($\star$) For every $i \in [n]$ and every k-profile P of G_i, we have $\tau_i(P) \neq nil$ if and only if P is realizable with a set of size at most k. Furthermore, if $\tau_i(P) \neq nil$, then $\tau_i(P)$ is a set of minimum size with which P is realizable.

We prove ($\star$) by induction. In the base case when $i = n$, the claim trivially holds. Assume $i < n$ and let x, y be the vertices of G_{i+1} which were contracted to yield G_i, where z is the newly obtained vertex of G_i. By induction hypothesis, for any k-profile (T, D, M) of G_i with $z \notin T$ the claim holds as it is a k-profile of G_{i+1} as well.

Therefore, we assume that $z \in T$ and let $T' = (T \setminus z) \cup \{x, y\}$.

▷ Claim 11. Assume that ($\star$) holds for all $i' > i$ and let $P = (T, D, M)$ be a k-profile of G_i. If P is realizable with a set of size at most k, then $\tau_i(P) \neq nil$.

Proof. Suppose that $P = (T, D, M)$ is realizable with $S \subseteq V(G)$ of size at most k. Let $T_1, \ldots, T_\ell$ be the red connected components of T' in G_i, and let $S_j = S \cap \bigcup_{t \in T_j} t(G)$ for every $j \in [\ell]$. The pairs T_j and S_j for $j = 1, \ldots, \ell$ define a set of ℓ k-profiles (T_j, D_j, M_j) of G_{i+1} in a canonical way: D_j is precisely the set of vertices $t \in T_j$ such that $t(G) \cap S_j$ and M_j is the set of vertices $t \in T_j$ such that $t(G)$ is (fully) dominated by S_j. By construction, each k-profile (T_j, D_j, M_j) is realizable with S_j.

We argue that the set $\mathcal{P} = \{(T_j, D_j, M_j) : j \in [\ell]\}$ is consistent with $P = (T, D, M)$. The first and the second conditions for consistency are clearly satisfied. To verify the third condition, consider a vertex $u \in T$ distinct from z and without loss of generality we assume $u \in T_{j^*}$. If $u \in M$ and $u \notin M_{j^*}$, this means that S_{j^*} does not dominate $u(G)$ because S_{j^*} realizes $(T_{j^*}, D_{j^*}, M_{j^*})$. From $u \in M$ and the fact that S realizes (T, D, M), we know that S dominates $u(G)$ and thus there is at least one vertex $S \setminus S_{j^*}$ which is adjacent (in G) with some vertex of $u(G)$. Consider an arbitrary vertex $v \in T$ to which some of $S \setminus S_{j^*}$ contracts to, and observe that $v \notin T_{j^*}$. This means that uv is a black edge. The converse direction of the third condition is clearly met. The fourth condition of consistency can be verified similarly as the third condition. If $\mathcal{P}$ does not contain any k-profile whose ground set has size at least $(d^2 + 1)k$, now the claim is immediate because each (T_j, D_j, M_j) is realizable with S_j: by induction hypothesis, we have $\tau_{i+1}(T_j, D_j, M_j) \neq nil$, and thus $\tau_i(T, D, M)$ is set to $\neq nil$ at line 14.

Suppose that $\mathcal{P}$ contains a k-profile whose ground set has size at least $(d^2 + 1)k$. One can easily see that in this case, $\ell = 1$ or equivalently T' is a red connected component in $(V(G_{i+1}), R(G_{i+1}))$ consisting of exactly $(d^2+1)k$ vertices. Since the union of at most k balls of radius at most 2 which is connected in $(V(G_{i+1}), R(G_{i+1}))$ have less than $(d^2+1)k$ vertices, there exists $v \in T' \setminus \bigcup_{t \in D'} B^2_{i+1}(t)$. Moreover, by the choice of v, $(T' \setminus v, D' \setminus v, M' \setminus v)$ is now a k-profile of G_{i+1}. To conclude that $\tau_i(T, D, M) \neq nil$, it suffices to prove that $\tau_{i+1}(T' \setminus v, D' \setminus v, M' \setminus v) \neq nil$. We do this by showing that (T, D, M), (T', D', M') and $(T' \setminus v, D' \setminus v, M' \setminus v)$ are equivalent in regards to realizability.

The equivalence of the first two is obvious. For the equivalence of the last two, note that if S realizes (T', D', M'), S does not intersect $v(G)$, and thus S trivially realizes $(T' \setminus v, D' \setminus v, M' \setminus v)$. Conversely, suppose that $(T' \setminus v, D' \setminus v, M' \setminus v)$ is realizable with S'. The crucial observation is that v has no red neighbor in D' since otherwise, v belongs to the union $\bigcup_{t \in D'} B^2_{i+1}(t)$, contradicting the choice of v. Therefore, we know that $v \in M'$ if and only if there exists $u \in D' \setminus v$ such that uv is a black edge. In the case when $v \in M'$, there exists a black neighbor $u \in D' \setminus v$ of v, and any S' realizing $(T' \setminus v, D' \setminus v, M' \setminus v)$ intersects $u(G)$. If follows that S' fully dominates $v(G)$ and S' realizes (T', D', M'). Else if $v \notin M'$, this means that not only the red neighbors of v are disjoint from D' but also no black neighbor of v is contained in D'. As a consequence $v(G)$ is not dominated by S', thus S' realizes (T', D', M'). This proves the equivalence of (T', D', M') and $(T' \setminus v, D' \setminus v, M' \setminus v)$, and completes the proof of the claim. ◁

To establish the other direction, suppose that $\tau_i(T, D, M) \neq nil$ and let $\mathcal{P}^*$ be the set consistent with P such that $\tau_i(T, D, M) = \bigcup_{P \in \mathcal{P}^*} \tau_{i+1}(P)$ or $\tau_i(T, D, M) = \tau_{i+1}(T' \setminus v, D' \setminus v, M' \setminus v)$ for some v. Such $\mathcal{P}^*$ clearly exists since otherwise only nil can be output. In the former case, it is tedious to verify that if each (T_i, D_i, M_i) of $\mathcal{P}^*$ is realizable with S_i, then $\bigcup_{i \in [\ell]} S_i$ realizes (T, D, M). In the latter case, we simply recall that (T, D, M) and $(T' \setminus v, D' \setminus v, M' \setminus v)$ are equivalent in regards to realizability. This completes the proof of the first statement of $(\star)$. The second statement immediately follows.

Algorithm 1 `k-DomSet`.

Input : A graph G, a positive integer k, and a d-sequence $G = G_n, \dots, G_1 = K_1$.
Output : A dominating set of G of size at most k, or report *nil* (No-instance).

1 **for** $v \in V(G_n)$ **do**
2 $\tau_n(\{v\},\{v\},\{v\}) = \{v\}$, $\tau_n(\{v\},\emptyset,\emptyset) = \emptyset$, $\tau_n(P) = nil$ for all other k-profiles P
3 **for** $i = n-1 \to 1$ **do**
4 $x, y \leftarrow$ contracted pair in $G_{i+1} \to G_i$
5 $z \leftarrow$ contraction of x and y in G_i
6 **for** *every k-profile (T, D, M) of G_i of size less than $(d^2+1)k$ s.t. $z \notin T$* **do**
7 $\tau_i(T, D, M) \leftarrow \tau_{i+1}(T, D, M)$
8 **for** *every k-profile (T, D, M) of G_i of size less than $(d^2+1)k$ s.t. $z \in T$* **do**
9 $\tau_i(T, D, M) \leftarrow nil$
10 $T' \leftarrow (T \setminus z) \cup \{x, y\}$
11 **for** *every set $\mathcal{P}$ of k-profiles of G_{i+1} consistent with (T, D, M)* **do**
12 **if** *each k-profile of $\mathcal{P}$ has size less than $(d^2+1)k$* **then**
13 **if** *$\tau_{i+1}(P) \neq nil$ for all $P \in \mathcal{P}$* **then**
14 $\tau_i(T, D, M) \leftarrow \text{best}\{\tau_i(T, D, M), \bigcup_{P\in\mathcal{P}} \tau_{i+1}(P)\}$
15 **else**
16 Let (T', D', M') be the unique k-profile contained in $\mathcal{P}$.
17 Choose $v \in T' \setminus \bigcup_{t\in D'} B^2_{i+1}(t)$
18 $\tau_i(T, D, M) \leftarrow \text{best}\{\tau_i(T, D, M), \tau_{i+1}(T' \setminus v, D' \setminus v, M' \setminus v)\}$
19 **if** *$\tau_i(T, D, M) \neq nil$ and has size larger than k* **then**
20 $\tau_i(T, D, M) \leftarrow nil$
21 **return** $\tau_1(V(G_1), V(G_1), V(G_1))$

Running time. In an actual implementation of Algorithm 1, we maintain a single mapping τ. As we proceed from G_{i+1} to G_i, we modify the domain of τ consisting of k-profiles so that new k-profiles involving z are added and after calculating the assignments for the new k-profiles, all the domains and corresponding assignments involving x or y shall be discarded. Therefore, it suffices to check the running time for updating τ, which is performed in the inner loop of lines 6-20. By Lemma 8, there are $O(d^{2(d^2+1)k-2} \cdot 2^{2(d^2+1)k})$ new profiles of G_i to compute. For each k-profile (T, D, M) with $z \in T$, the ground sets $T_1, \dots, T_\ell$ of a potentially consistent set $\mathcal{P}$ is already determined. Hence, we exhaust all possibilities of appending each T_i by M_i and D_i to form a k-profile and the inner loop of 8-20 will consider at most $2^{(d^2+1)k} \cdot 2^{(d^2+1)k}$ sets $\mathcal{P}$. The consistency of $\mathcal{P}$ with (T, D, M) can be routinely verified. This establishes the claimed running time. ◀

5 Approximation algorithms

5.1 Constant approximation for Min Dominating Set

In this section, we prove that Min Dominating Set has bounded integrality gap in classes of bounded twin-width. A constant factor approximation algorithm readily follows. We will use the following technical lemma from the second paper of the series.

▶ **Theorem 12** (Section 3, Lemma 20 in [3]). *For every integer t, there are integers s and t' such that every graph G with a t-sequence admits a rooted tree $\mathcal{T}$ with the following properties.*

- *Every node of $\mathcal{T}$ is labeled by a t'-trigraph.*
- *The root of $\mathcal{T}$ is labeled by G.*
- *All the leaves of $\mathcal{T}$ are labeled by the 1-vertex graph K_1.*
- *If a node x of $\mathcal{T}$ is labeled by H, and a child node of x is labeled by H', there is a t'-contraction in H that yields H'. In particular $|V(H)| = |V(H')| + 1$.*
- *Every internal node of $\mathcal{T}$ labeled by H has at least $|V(H)|/s$ children coming from t'-contractions on pairwise disjoint pairs of vertices of H.*

Such a tree is called an *s-versatile tree of t'-contractions*. Informally Theorem 12 says that, by degrading the twin-width bound, one can move away from the "linear nature" of the contraction sequence to a profusely branching contraction witness.

Theorem 12 is effective: The s-versatile tree of t'-contractions can be computed in polynomial time, if a t-sequence for G is provided.

▶ **Theorem 13.** *In classes of bounded twin-width, Min Dominating Set has bounded integrality gap.*

Proof. Let G be a graph of twin-width at most t. By Theorem 12, there exist t', s functions of t only such that G admits an s-versatile tree of t'-contraction. Let $w^* : V(G) \to [0,1]$ be the weight function of a minimum fractional dominating set, with total weight γ^*. Thus w^* is an optimum solution of the linear program

$$\text{minimize} \sum_{x \in V(G)} w(x)$$
$$\text{with } \forall x \in V(G), \sum_{y \in N[x]} w(y) \geqslant 1, \text{ and } 0 \leqslant w(x) \leqslant 1,$$

and $\gamma^* = \sum_{x \in V(G)} w^*(x)$. The weight function w^* is extended to subsets of vertices by sum. We assume that G has at least one vertex, so $\gamma^* \geqslant 1$.

We now greedily perform contractions in G following the versatile tree of contractions with a restriction: contractions involving a part of total weight at least $\frac{1}{2(t'+1)}$ are forbidden. Let us explain what this means in more detail. We start at the root, labeled G, of the versatile tree. We move to a(ny) child node along an edge corresponding to a non-forbidden t'-contraction. A *non-forbidden* contraction is one of u, v with $w^*(u(G)) < \frac{1}{2(t'+1)}$ and $w^*(v(G)) < \frac{1}{2(t'+1)}$. We iterate that until we get stuck (every child of the current node entails a forbidden contraction).

We adopt the partition viewpoint of the t'-sequence. Let $\mathcal{P}$ be the partition of $V(G)$ obtained when this process finishes, and let $G_{\mathcal{P}}$ be the corresponding trigraph (that is, the label of the node where we stop). We observe that we cannot end at a leaf of the versatile tree. Indeed that would mean that the last contraction merged a bipartition $\{X, Y\}$ of $V(G)$ into $\{V(G)\}$. As $\gamma^* \geqslant 1$, this would imply that $w^*(X) \geqslant 1/2$ or $w^*(Y) \geqslant 1/2$, contradicting $\max(w^*(X), w^*(Y)) < \frac{1}{2(t'+1)}$.

▷ Claim 14. The partition $\mathcal{P}$ has at most $2s(t'+1)\gamma^*$ classes.

Proof. As we explained, we cannot end up with a partition $\mathcal{P}$ at a leaf of the versatile tree. Thus at least $|\mathcal{P}|/s$ disjoint pairs of vertices are t'-contractions in $G_{\mathcal{P}}$. Therefore all these contractions must be forbidden by our restriction imposed on the weights. It follows that at least $|\mathcal{P}|/s$ parts of $\mathcal{P}$ have weight at least $\frac{1}{2(t'+1)}$. Since the sum of all weights in $\mathcal{P}$ is γ^*, it follows that $|\mathcal{P}| \leqslant 2s(t'+1)\gamma^*$. ◁

▷ Claim 15. Let $P \in \mathcal{P}$ be any part. Either $w^*(P) < \frac{1}{t'+1}$ or P is a singleton.

Proof. Let $P \in \mathcal{P}$, and assume that P is not a singleton. Then P has been obtained by contracting two parts P_1, P_2 during the contraction sequence leading to $\mathcal{P}$. The restriction on the contraction sequence ensures that $w^*(P_1) < \frac{1}{2(t'+1)}$ and $w^*(P_2) < \frac{1}{2(t'+1)}$. Therefore $w^*(P) = w^*(P_1) + w^*(P_2) < \frac{1}{t'+1}$. ◁

Let $D \subseteq V(G)$ be obtained by picking arbitrarily one vertex x_P in each part $P \in \mathcal{P}$. By Claim 14, $|D| \leqslant 2s(t'+1)\gamma^*$, which is linear in γ^* when t is fixed. Let us prove that D is a dominating set. We let $P \in \mathcal{P}$, and prove that all vertices of P are dominated by D.

Suppose first that there exists $P' \in \mathcal{P}$ such that P, P' is a black edge in $G_{\mathcal{P}}$. Then $x_{P'} \in P'$ is adjacent to all vertices of P, which are thus dominated by D.

Hence we may instead assume that P does not have any black neighbor in $G_{\mathcal{P}}$. Consider any vertex $y \in P$, and let $P_1, \dots, P_k$ the parts of $\mathcal{P} \setminus \{P\}$ such that there exists an edge between y and some vertex of P_i. Then $P_1, \dots, P_k$ are neighbors of P in $G_{\mathcal{P}}$, and must be red neighbors since P has no black neighbor. Since $G_{\mathcal{P}}$ is a t'-trigraph, it follows that $k \leqslant t'$.

We now claim that one of the parts $P, P_1, \dots, P_k$ must be a singleton. Indeed, since w^* is a fractional dominating set, and since $P \cup \bigcup_{i=1}^{k} P_i$ contains y and its neighborhood, it must be that $w^*(P) + \sum_{i=1}^{k} w^*(P_i) \geqslant 1$. Because $k \leqslant t'$, it follows that one part among $P, P_1, \dots, P_k$ has weight at least $\frac{1}{t'+1}$. By Claim 15, that same part P_h must be a singleton. Let z be the single vertex in P_h. Necessarily $z \in D$. If this singleton part is P, then $z = y$. Otherwise z is a neighbor of y by definition of $P_1, \dots, P_k$. In either case y is dominated in D by z. ◀

5.2 A constant approximation for MIS would imply a PTAS

A pessimistic stance on the result of this section is that, perhaps surprisingly, the constant approximation of Min Dominating Set is unlikely to be generalizable to the closely related MIS. We indeed observe that the self-improving reduction of Feige et al. [8] preserves the twin-width. As a consequence a constant approximation for MIS would provide a polynomial-time approximation scheme (PTAS).

▶ **Theorem 16.** *If Max Independent Set on graphs of twin-width at most d has a constant-approximation algorithm, then it admits a PTAS.*

For G_1 and G_2 two non-empty graphs, and $u \in V(G_1)$, we denote by $G_1(u \leftarrow G_2)$ the substitution in G_1 of u by G_2. That is, u is replaced by G_2, and every vertex of $V(G_1) \setminus \{u\}$ initially adjacent to u is made adjacent to the whole $V(G_2)$.

▶ **Lemma 17.** $tww(G_1(u \leftarrow G_2)) = \max(tww(G_1), tww(G_2))$.

For G a graph, let G^t be the graph on the vertex set $V(G)^t$, such that for $\bar{x} = (x_1, \dots, x_t)$, $\bar{y} = (y_1, \dots, y_t)$ distinct vertices, $\bar{x}\bar{y} \in E(G^t)$ if and only if $x_i y_i \in E(G)$ where i is the smallest index such that $x_i \neq y_i$. This definition can be restated inductively: G^0 is the 1-vertex graph, and G^t is obtained from G by substituting each vertex by a copy of G^{t-1}. With the notations of the initial definition, for $x \in V(G)$, the set of vertices of G^t of the form $(x, x_2, \dots, x_t)$ is a copy isomorphic to G^{t-1}.

The following holds as a direct consequence of Lemma 17.

▶ **Lemma 18.** *For any graph G and integer $t > 0$, $tww(G^t) = tww(G)$.*

We now show that the independence number of G^t is tightly related to the one of G.

▶ **Lemma 19.** *For any graph G, both following conditions hold.*

1. *Given any independent set of size k in G, one can compute an independent of size k^t in G^t, in time $O(k^t)$.*
2. *Given any independent set of size k' in G^t, one can compute an independent of size at least $\sqrt[t]{k'}$ in G, in time $O(k')$.*

As an immediate corollary, $\alpha(G^t) = \alpha(G)^t$ where, we recall, $\alpha(H)$ denotes the size of a maximum independent set in H.

Proof of Theorem 16. Assume there is a polynomial-time β-approximation for MIS on graphs of twin-width at most d. Let G be a graph with twin-width at most d. By Lemma 18 the algorithm can be ran on G^t to obtain an independent set of size at least $\frac{\alpha(G^t)}{\beta} = \frac{\alpha(G)^t}{\beta}$. By Lemma 19, this independent set in G^t can be turned into an independent set in G of size at least $\alpha(G)/\sqrt[t]{\beta}$. This gives a polynomial-time $\sqrt[t]{\beta}$-approximation for arbitrary t. Thus the approximation ratio can be made arbitrarily close to 1. ◀

6 Bounded twin-width classes are χ-bounded

So far, our algorithms use the contraction sequence (or tree) "forward". This is the original scheme of Guillemot and Marx [11], and of our model checking algorithm [4]. We now see how it can be useful to consider the contraction process "backward". We start with the case of triangle-free graphs, which will be the base case for the proof of the χ-boundedness.

▶ **Theorem 20.** *Every triangle-free graph with twin-width at most d is $(d+2)$-colorable.*

Proof. Let G be an n-vertex triangle-free graph of twin-width at most d, and let $G = G_n, \dots, G_1 = K_1$ be a d-sequence of G. We show how to color G with $d+2$ colors starting from G_1, and iteratively coloring G_{i+1} based on the coloring of G_i. We give the unique vertex of $G_1 = K_1$ color 1. This defines coloring C_1. For every i from 1 to $n-1$, let z be the vertex of G_i split into $u, v \in V(G_{i+1})$. In coloring C_{i+1}, every vertex of $V(G_{i+1}) \setminus \{u, v\}$ keeps the color it received by C_i. Vertex u receives color $C_i(z)$. Finally, v receives color $C_i(z)$ if uv is a non-edge in G_{i+1}, and the smallest positive integer *not* appearing in its neighborhood (black *and* red neighbors) in G_{i+1}, otherwise. We will now show that C_n is a proper coloring of G using at most $d+2$ distinct colors.

We show by induction on i that C_i is a proper $(d+2)$-coloring of the *graph* $G'_i := (V(G_i), E(G_i) \cup R(G_i))$. Coloring C_1 is indeed proper in G'_1 and uses $1 \leqslant d+2$ color. We assume that C_i is a proper $(d+2)$-coloring of G'_i, and distinguish two cases. If there is a black edge $yz \in E(G_i)$ (recall that z is the vertex split into u, v), then uv has to be a non-edge in G_{i+1}. Otherwise there is at least one edge between $u(G)$ and $v(G)$, and this edge forms a triangle with any vertex in $y(G)$. Thus in that case, $C_{i+1}(u) = C_{i+1}(v) = C_i(z)$. So the number of distinct colors given by C_{i+1} is still at most $d+2$ (see Figure 3).

And C_{i+1} is a proper coloring of G'_{i+1} since $N_{G'_{i+1}}(\{u, v\}) = N_{G'_i}(z)$. If instead z has only red neighbors in G_i, then z has at most d neighbors in G'_i. Furthermore let us assume that $uv \in E(G'_{i+1})$, otherwise we conclude as previously. In that case, v is properly colored by C_{i+1} in G'_{i+1} by construction, and vertex u as well, since $N_{G'_{i+1}}(u) \setminus \{v\} \subseteq N_{G'_i}(z)$. Finally $C_{i+1}(v)$ is the smallest positive integer not appearing in a set of at most $d+1$ positive integers. Thus $C_{i+1}(v) \leqslant d+2$, and C_{i+1} is overall a proper $(d+2)$-coloring of G'_{i+1} (see Figure 4).

In particular, C_n is a proper $(d+2)$-coloring of $G'_n = G_n = G$. ◀

As a side note, it is, to our knowledge, possible that every triangle-free K_t-minor free graph has twin-width $O(t)$. If this turns out to be true, it offers a seemingly different approach to getting improved bounds in the triangle-free case of the Hadwiger's conjecture:

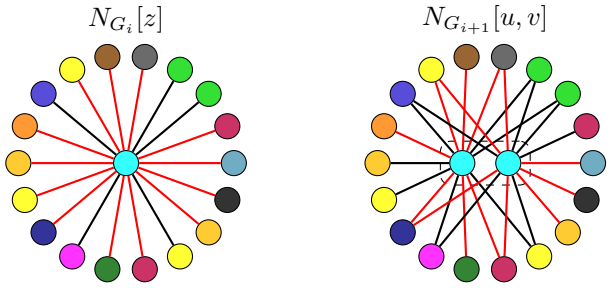


Figure 3 Split, when z is incident to a black edge in G_i. As G is triangle-free, there cannot be an edge (red or black) between u and v. Thus both u and v can take the color of z, which does not appear in their neighborhood.

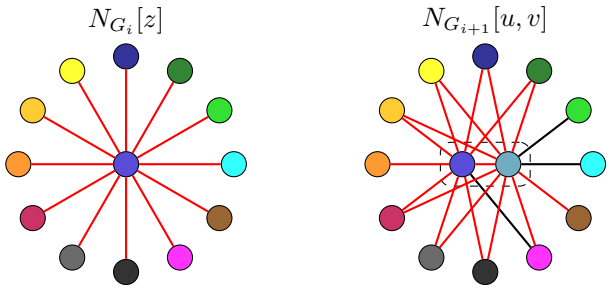


Figure 4 Split, when z is only incident to red edges. Even if the red neighbors of z have d distinct colors, vertex v can find a color in $[d+2]$ which avoids these d colors plus the color of z and u.

Instead of trying to color these graphs, one could try to design contraction sequences for them. We now show how to color any K_t-free graph G given with a d-sequence, with at most $(d+2)^{t-2}$ colors.

▶ **Theorem 21.** *For every integer $t \geqslant 3$, every K_t-free graph with twin-width at most d is $(d+2)^{t-2}$-colorable.*

Proof. Let $G_n, \dots, G_1$ be a d-sequence of a K_t-free graph G with $t \geqslant 3$. In Theorem 20, whenever a vertex $x \in V(G_{i+1})$ was incident to a black edge for the first time (going from G_1 to G_n), the color of all the vertices in $x(G)$ was eventually set to the same value, namely $C_{i+1}(x)$. Now such a set $x(G)$ is not necessarily an independent set, but rather induces a K_{t-1}-free graph. Indeed, a K_{t-1} in $G[x(G)]$ would form a K_t in G with any vertex of $y(G)$, where $xy \in E(G_{i+1})$. By induction on t, we may color $G[x(G)]$ with tuples of at most $t-3$ integers of $[d+2]$, and prepends $C_{i+1}(x)$ to these tuples. The base case $t=3$ is Theorem 20. We make the general idea a bit more precise.

For every $i \in [n]$, we define G_i^* as the *graph* obtained from G_i by blowing every vertex $x \in V(G_i)$ into $G[x(G)]$ whenever x is incident to a black edge, and then turning every red edge into a black edge. We define the successive colorings $C'_1, \dots, C'_n$ of $G_1^*, \dots, G_n^*$, respectively, following the algorithm of Theorem 20. While there are no black edge in the current trigraph G_i, we set $C'_i := C_i$, where C_i is the coloring in the triangle-free case. Say, at least one black edge appears for the first time in G_{i+1} (this is well-defined since G_n has only black edges). Again we adopt the convention that $z \in V(G_i)$ was split into $u, v \in V(G_{i+1})$. Let S be the set of (at most $d+2$) vertices with an incident black edge in G_{i+1}. (One may notice that $S \subseteq \{u,v\} \cup N_{G_i}(z)$ and $S \cap \{u,v\} \neq \emptyset$.) Every vertex $w \in V(G_{i+1}) \setminus S$ receives

color $C_{i+1}(w)$. As we observed, for every $x \in S$, $G[x(G)]$ is K_{t-1}-free. By induction there is a coloring C^x of $G[x(G)]$ with tuples of at most $t-3$ integers from $[d+2]$. We *permanently* color every vertex $y \in x(G)$ by $(C_{i+1}(x), C^x(y))$. This defines the coloring C'_{i+1} of G^*_{i+1}.

We continue to follow Theorem 20, with the ensuing precisions. We go through all the splits, including the ones between two permanently colored vertices, since they may make some other vertices incident to a black edge for the first time. If the split vertex $z \in V(G_j)$ is *not* such that $z(G)$ was already permanently colored, the colors of the new vertices $u, v \in V(G_{j+1})$ are chosen according to the rules of Theorem 20 where we consider the trigraphs G_j and G_{j+1} (and not the graphs G^*_j and G^*_{j+1}), and the coloring C_j of $V(G_j)$ is defined as: $C_j(y)$ is the first coordinate of $C'_j(y)$ (or $C'_j(y)$ itself if it is not a tuple) if $y \in V(G^*_j)$, and the first coordinate of the color of any vertex in $y(G)$, otherwise. (One may observe that C_j is *not* necessarily a proper coloring of $(V(G_j), E(G_j) \cup R(G_j))$, but all the conflict edges lie within a permanently colored subgraph.) Every time a vertex x becomes incident to a black edge, we permanently color $x(G)$. This defines the sequence of colorings $C'_1, \dots, C'_n$.

We show by induction on i that C'_i properly colors G^*_i. Coloring C'_1 is indeed a proper coloring of $G^*_1 = K_1$. We assume that C'_i is a proper coloring of G^*_i, and let xy be any edge in $E(G^*_{i+1})$. By the outermost induction on t, if xy lies within a K_{t-1}-free graph permanently colored, then $C'_{i+1}(x) \neq C'_{i+1}(y)$. If instead x and y belong to two distinct vertices of G_{i+1}, by the proof of Theorem 20 and the fact that C'_i is a proper coloring of G^*_i, the first coordinate of $C'_{i+1}(x)$ and of $C'_{i+1}(y)$ differ. In particular C'_n is a proper coloring of $G^*_n = G_n = G$. We pad every tuple $C'_n(x)$ of length $t' < t$ with $t - t'$ entries 1. From the previous proof, it can be observed that this new coloring of G is still proper, and uses at most $(d+2)^{t-2}$ colors. ◀

Theorem 21 directly implies that, provided $O(1)$-sequences are given, Min Coloring can be $2^{O(\mathrm{OPT})}$-approximated on bounded twin-width graphs, and Max Independent Set can be $O(1)$-approximated on K_t-free graphs of bounded twin-width. It would be interesting to determine if bounded twin-width classes are polynomially χ-bounded, that is, satisfies for some constant c, $\chi(G) = O(\omega(G)^c)$ for every graph G in the class. Bounded clique-width or rank-width classes were shown polynomially χ-bounded only recently [2]. We show however that bounded twin-width classes satisfy the related *strong Erdős-Hajnal property*. We recall that a class $\mathcal{C}$ of graphs satisfies the *strong Erdős-Hajnal property* if there exists an $\varepsilon > 0$ such that every $G \in \mathcal{C}$ contains two disjoint subsets of vertices X, Y, both of size at least $\varepsilon|V(G)|$, with either all edges or no edges between X and Y. The strong Erdős-Hajnal property of a hereditary class implies the existence of a clique or a stable set of polynomial size, that is, the Erdős-Hajnal property [1].

▶ **Theorem 22.** *The class of graphs with twin-width at most d satisfies the strong Erdős-Hajnal property with $\varepsilon = 1/(d+4)$.*

Proof. Let G be an n-vertex graph with twin-width at most d. Consider in a fixed d-sequence $G_n, \dots, G_1$ the maximum index i such that there is a vertex $z \in V(G_i)$ satisfying $|z(G)| \geqslant n/(d+4)$. Since $X := z(G)$ is the union of $u(G)$ and $v(G)$ for some $u, v \in V(G_{i+1})$, its size is at most $2n/(d+4)$. Vertex z has at most d red neighbors in G_i. These neighbors constitute a set $S \subseteq V(G)$ of at most $d \cdot n/(d+4)$ vertices. Thus $|V(G) \setminus (z(G) \cup S)| \geqslant n - 2n/(d+4) - dn/(d+4) = 2n/(d+4)$. By construction, every vertex in $V(G) \setminus (z(G) \cup S)$ is fully adjacent to X or fully non-adjacent to X. Let $Y \subseteq V(G) \setminus (z(G) \cup S)$ be the subset of all vertices in the majority regarding these two outcomes. Set Y has size at least $n/(d+4)$ vertices and X, Y is therefore an appropriate pair. ◀

References

1 Noga Alon, János Pach, Rom Pinchasi, Radoš Radoičić, and Micha Sharir. Crossing patterns of semi-algebraic sets. *Journal of Combinatorial Theory, Series A*, 111(2):310–326, 2005. doi:10.1016/j.jcta.2004.12.008.

2 Marthe Bonamy and Michal Pilipczuk. Graphs of bounded cliquewidth are polynomially χ-bounded. *CoRR*, abs/1910.00697, 2019. arXiv:1910.00697.

3 Édouard Bonnet, Colin Geniet, Eun Jung Kim, Stéphan Thomassé, and Rémi Watrigant. Twin-width II: small classes. In *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1977–1996, 2021. doi:10.1137/1.9781611976465.118.

4 Édouard Bonnet, Eun Jung Kim, Stéphan Thomassé, and Rémi Watrigant. Twin-width I: tractable FO model checking. In *61st IEEE Annual Symposium on Foundations of Computer Science, FOCS 2020, Durham, NC, USA, November 16-19, 2020*, pages 601–612. IEEE, 2020. doi:10.1109/FOCS46700.2020.00062.

5 Bruno Courcelle. The monadic second-order logic of graphs. I. Recognizable sets of finite graphs. *Information and Computation*, 85(1):12–75, 1990. doi:10.1016/0890-5401(90)90043-H.

6 Bruno Courcelle, Johann A. Makowsky, and Udi Rotics. Linear time solvable optimization problems on graphs of bounded clique-width. *Theory Comput. Syst.*, 33(2):125–150, 2000. doi:10.1007/s002249910009.

7 Irit Dinur and David Steurer. Analytical approach to parallel repetition. In David B. Shmoys, editor, *Symposium on Theory of Computing, STOC 2014, New York, NY, USA, May 31 - June 03, 2014*, pages 624–633. ACM, 2014. doi:10.1145/2591796.2591884.

8 Uriel Feige, Shafi Goldwasser, László Lovász, Shmuel Safra, and Mario Szegedy. Approximating Clique is almost NP-complete (preliminary version). In *32nd Annual Symposium on Foundations of Computer Science, San Juan, Puerto Rico, 1-4 October 1991*, pages 2–12. IEEE Computer Society, 1991. doi:10.1109/SFCS.1991.185341.

9 Jörg Flum and Martin Grohe. Fixed-parameter tractability, definability, and model-checking. *SIAM J. Comput.*, 31(1):113–145, 2001. doi:10.1137/S0097539799360768.

10 Markus Frick and Martin Grohe. The complexity of first-order and monadic second-order logic revisited. *Ann. Pure Appl. Log.*, 130(1-3):3–31, 2004. doi:10.1016/j.apal.2004.01.007.

11 Sylvain Guillemot and Dániel Marx. Finding small patterns in permutations in linear time. In *Proceedings of the Twenty-Fifth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2014, Portland, Oregon, USA, January 5-7, 2014*, pages 82–101, 2014. doi:10.1137/1.9781611973402.7.

12 Russell Impagliazzo and Ramamohan Paturi. On the Complexity of k-SAT. *J. Comput. Syst. Sci.*, 62(2):367–375, 2001. doi:10.1006/jcss.2000.1727.

13 Russell Impagliazzo, Ramamohan Paturi, and Francis Zane. Which problems have strongly exponential complexity? *J. Comput. Syst. Sci.*, 63(4):512–530, 2001. doi:10.1006/jcss.2001.1774.

14 David S. Johnson. Approximation algorithms for combinatorial problems. *J. Comput. Syst. Sci.*, 9(3):256–278, 1974. doi:10.1016/S0022-0000(74)80044-9.

15 László Lovász. On the ratio of optimal integral and fractional covers. *Discret. Math.*, 13(4):383–390, 1975. doi:10.1016/0012-365X(75)90058-8.

16 Svatopluk Poljak. A note on stable sets and colorings of graphs. *Commentationes Mathematicae Universitatis Carolinae*, 15(2):307–309, 1974.

17 Hisao Tamaki. Positive-instance driven dynamic programming for treewidth. *J. Comb. Optim.*, 37(4):1283–1311, 2019. doi:10.1007/s10878-018-0353-z.

Almost-Optimal Deterministic Treasure Hunt in Arbitrary Graphs

Sébastien Bouchard
Univ. Bordeaux, Bordeaux INP, CNRS, LaBRI, UMR5800, F-33400 Talence, France

Yoann Dieudonné
MIS Lab., Université de Picardie Jules Verne, Amiens, France

Arnaud Labourel
Aix Marseille Univ, Université de Toulon, CNRS, LIS, Marseille, France

Andrzej Pelc
Département d'informatique, Université du Québec en Outaouais, Gatineau, Canada

Abstract

A mobile agent navigating along edges of a simple connected graph, either finite or countably infinite, has to find an inert target (treasure) hidden in one of the nodes. This task is known as treasure hunt. The agent has no *a priori* knowledge of the graph, of the location of the treasure or of the initial distance to it. The cost of a treasure hunt algorithm is the worst-case number of edge traversals performed by the agent until finding the treasure. Awerbuch, Betke, Rivest and Singh [3] considered graph exploration and treasure hunt for finite graphs in a restricted model where the agent has a fuel tank that can be replenished only at the starting node s. The size of the tank is $B = 2(1+\alpha)r$, for some positive real constant α, where r, called the radius of the graph, is the maximum distance from s to any other node. The tank of size B allows the agent to make at most $\lfloor B \rfloor$ edge traversals between two consecutive visits at node s.

Let $e(d)$ be the number of edges whose at least one extremity is at distance less than d from s. Awerbuch, Betke, Rivest and Singh [3] conjectured that it is impossible to find a treasure hidden in a node at distance at most d at cost nearly linear in $e(d)$. We first design a deterministic treasure hunt algorithm working in the model without any restrictions on the moves of the agent at cost $\mathcal{O}(e(d) \log d)$, and then show how to modify this algorithm to work in the model from [3] with the same complexity. Thus we refute the above twenty-year-old conjecture. We observe that no treasure hunt algorithm can beat cost $\Theta(e(d))$ for all graphs and thus our algorithms are also almost optimal.

2012 ACM Subject Classification Theory of computation → Design and analysis of algorithms

Keywords and phrases treasure hunt, graph, mobile agent

Digital Object Identifier 10.4230/LIPIcs.ICALP.2021.36

Category Track A: Algorithms, Complexity and Games

Related Version *Full Version*: https://arxiv.org/abs/2010.14916

Funding *Andrzej Pelc*: Partially supported by NSERC discovery grant 2018-03899 and by the Research Chair in Distributed Computing at the Université du Québec en Outaouais.

1 Introduction

The background. A mobile agent has to find an inert target (treasure) in some environment that can be a network modeled by a graph or a terrain in the plane. This task, known as *treasure hunt*, has important applications when the environment is dangerous for humans. When a miner is lost in a contaminated mine, it may have to be found by a robot, and the length of the robot's trajectory should be as short as possible, in order to minimize rescuing

time. In this example, a graph models the corridors of the mine with nodes representing crossings. Another application of treasure hunt in graphs is searching for a data item in a communication network modeled by a graph.

The models and the problem. We consider a simple connected undirected locally finite graph $\mathcal{G} = (V_{\mathcal{G}}, E_{\mathcal{G}})$, i.e., a graph with nodes of finite degrees. Such a graph can be either finite or countably infinite. A mobile agent (robot) starts at a node s of $\mathcal{G}$, called the *source node*, and moves along its edges. The maximum distance of any node from s is denoted by r and called the *radius* of the graph (the radius of countably infinite graphs is infinite). We make the same assumption as in [3] that the agent has unbounded memory and can recognize already visited nodes and traversed edges. This is formalized as follows. Nodes of $\mathcal{G}$ have distinct labels that are positive integers. Each edge has ports at both of its extremities. Ports corresponding to edges incident to a node of degree δ are numbered $0, 1, \ldots, \delta - 1$ in an arbitrary way. At the beginning, the agent situated at node s sees its degree. The agent executes a deterministic algorithm: at each step, it selects a port number on the basis of currently available information, and traverses the corresponding edge. When the agent enters the adjacent node, it learns its label, its degree, and the incoming port number. Each node of $V_{\mathcal{G}}$ will be identified with its label, and each edge of $E_{\mathcal{G}}$ will be identified as the quadruple (v, w, p, q), where $v < w$ are labels of the edge extremities, p is its port number at node v and q is its port number at node w.

The above simple model will be called *unrestricted*. However, some authors imposed additional restrictions, in the case when the graph is finite. The authors of [3] used a restriction of moves of the agent that we will call the *fuel-restricted model*. They assumed that the agent has a fuel tank that can be replenished only at the starting node s of the agent. The size of the tank is $B = 2(1+\alpha)r$, for some positive real constant α, where r is the radius of the graph. The tank of size B allows the agent to make at most $\lfloor B \rfloor$ edge traversals between two consecutive visits at node s. The restriction used in [11] was of a different kind. We will call it the *rope-restricted model*. It was assumed in [11] that the agent is *tethered*, i.e., attached to s by a rope that it unwinds by a length 1 with every forward edge traversal and rewinds by a length of 1 with every backward edge traversal. The rope is infinitely extendible but has to satisfy the following constraint: the segment of the rope unwinded by the agent must never be longer than $L = (1 + \alpha)r$, for some positive real constant α. Hence the agent is forced to match every forward edge traversal of an edge with a backward edge traversal, rewinding the rope, in a first-in last-out stack order.

The task of treasure hunt, in any of the above three models, is defined as follows. An adversary hides the treasure in some node of the underlying graph $\mathcal{G}$. The agent has no *a priori* knowledge of the graph, of the location of the treasure or of the initial distance to it, and has to find the treasure. The *cost* of a treasure hunt algorithm is the worst-case number of edge traversals performed by the agent until finding the treasure.

In order to state our problem we need the notion of a *ball*. Given a non-negative integer k, a graph G, and a node u, the ball $B_k(G, u)$ is defined as the subgraph $K = (V_K, E_K)$ of G, where V_K is the set of all nodes at distance at most k from u in G, and E_K is the set of all edges of G whose at least one extremity is at distance smaller than k from u in G. (Thus $B_k(G, u)$ is the subgraph of G induced by nodes at distance at most k without edges joining nodes at distance exactly k from u in G). The number of edges in ball $B_k(\mathcal{G}, s)$, where s is the source node, will be denoted by $e(k, \mathcal{G})$. Whenever the graph $\mathcal{G}$ is clear from the context, we will write $e(k)$ instead of $e(k, \mathcal{G})$.

The main problem considered in this paper is inspired by the following conjecture of Awerbuch, Betke, Rivest and Singh [3], formulated for their fuel-restricted model:

Is it possible (we conjecture not) to find a treasure in time nearly linear in the number of those vertices and edges whose distance to the source is less than or equal to that of the treasure? [1]

Our results. Our main result refutes the above twenty-year-old conjecture. Let d be any integer such that $1 < d \leq r$, where r is the radius of the underlying graph $\mathcal{G}$. We first design a deterministic treasure hunt algorithm working in the unrestricted model and always finding a treasure located at distance at most d from the source node, at cost $\mathcal{O}(e(d) \log d)$. We then show how to modify this algorithm to work in the fuel-restricted and rope-restricted models with the same complexity. Since $d \leq e(d)$, the cost of our algorithms differs from $e(d)$ only by a logarithmic factor, and hence it is nearly linear in $e(d)$, contrary to the conjecture. Due to the ignorance of the agent concerning the graph in which it operates, it can be easily shown that no treasure hunt algorithm can beat cost $\Theta(e(d))$ for all graphs and thus our algorithms are also almost optimal. The main difficulty is to design the algorithm for the unrestricted model. This algorithm is then suitably modified for each of the two restricted models.

Solving the problem of treasure hunt at a cost quasi-linear in $e(d)$ required to respect two fundamental principles, whose joint implementation seemed precarious in the light of the existing literature.

The first one is a *prudence principle*. It consists in never getting "for too long" beyond the unknown distance d in order to guarantee a cost that depends on $e(d)$. This can be ideally achieved by emulating BFS. However, since in such an emulation the agent must physically move from one node to the next, it may be forced to traverse $\Omega(e(d)^2)$ edges before finding the treasure, in some graphs. In particular, this could be the case when $\mathcal{G}$ is an infinite line.

The second principle is what we could call an *efficiency principle*. It consists in getting a cost that is asymptotically close to the number of edges of the subgraph that has been explored till finding the treasure, if the treasure is far away. This can be ideally achieved using the treasure hunt algorithm of [11], the cost of which is linear in the number of edges of the explored subgraph. However, using this algorithm, the agent may go for too long beyond the unknown distance d and consequently the cost of treasure hunt could not be upper bounded by any function of $e(d)$. The key challenge overcome by our work was combining these two principles within the same algorithm. It is precisely the combination of prudence with efficiency that finally made possible the design of an almost-optimal treasure hunt algorithm.

Due to lack of space, the proofs of several results are omitted.

Related work. The task of treasure hunt, i.e., finding an inert target hidden in some environment, has been studied for over fifty years [5, 6, 7]. The environment where the target is hidden may be a graph or a plane, and the search may be deterministic or randomized. The book [1] surveys both treasure hunt and the related rendezvous problem, where the target and the searching agent are both mobile and they cooperate to meet. This book is concerned mostly with randomized search strategies. In [23, 26] the authors studied relations between treasure hunt and rendezvous in graphs. The authors of [4] studied the task of treasure hunt on the line and in the grid, and initiated the study of the task of searching for an unknown line in the plane. This research was continued, e.g., in [16, 21].

Several papers considered treasure hunt in the plane, see surveys [14, 15]. In [20], the author designs an optimal algorithm to sweep the plane in order to locate an unknown fixed target, where locating means getting the agent originating at point O to a point P such that the target is in the segment OP. In [13], the authors generalized the search problem in the

[1] Here time is what we call cost, i.e., the worst-case number of edge traversals until finding the treasure.

plane to the case of several searchers. Efficient treasure hunt in the plane, under complete ignorance of the searching agent, was studied in [24]. Treasure hunt on the line (called the cow-path problem [17]) has been also generalized to the environment consisting of multiple rays originating at a single point [2, 10, 22, 25].

In [12], the authors considered treasure hunt in several classes of graphs including trees. Treasure hunt in trees was studied in [8, 9, 18]. In [8, 9], the authors considered complete b-ary trees, and in [18], treasure hunt was studied in symmetric trees, with possibly multiple treasures.

In [19, 23], treasure hunt in graphs was considered under the advice paradigm, where a given number of bits of advice can be given to the agent, and the issue is to minimize this number of bits. The impact of different types of knowledge on the efficiency of the treasure hunt problem restricted to symmetric trees was studied in [18].

The two papers closest to the present work are [3, 11]. Both of them are mainly interested in exploration of finite unknown graphs but they both get interesting corollaries for the treasure hunt problem. [3] adopts the fuel-restricted model and [11] adopts the rope-restricted model. In [3], the authors get a treasure hunt algorithm working at cost $O(E + V^{1+o(1)})$, where E (resp. V) is the number of edges (resp. nodes) in a ball $B_\Delta(\mathcal{G}, s)$, with $\Delta \leq d + o(d)$, if the treasure is at distance at most d from the starting node of the agent. Since $e(\Delta)$ may be a lot larger than $e(d)$, this does not permit to bound the cost of the algorithm by any function of $e(d)$. This impossibility may be the reason for their conjecture that we refute in this paper. In [11], the authors design, for any constant $0 < \alpha < 1$, a treasure hunt algorithm whose cost is linear in $e((1 + \alpha)d)$. Again, since $e((1 + \alpha)d)$ may be much larger than $e(d)$, this does not permit to bound the cost of the algorithm by any function of $e(d)$.

2 Preliminaries

In this section we introduce some conventions, definitions and procedures that will be used to describe and analyze our algorithm.

Consider any graph $H = (V_H, E_H) \subseteq \mathcal{G}$. If H is finite, its size i.e., its number of edges is denoted by $|H|$. A graph is said to be *empty* if it contains no node. In the rest of this section, we assume that H is not empty.

Let u and v be two (not necessarily distinct) nodes of H. We say that a sequence of i integers $(x_1, x_2, \ldots, x_i)$ is a *path* (of length i) in H from node u to v iff (1) $i = 0$ and $u = v$, or (2) there exists an edge e in H between node u and a node w of H such that the port number of edge e at node u is x_1 and $(x_2, \ldots, x_i)$ is a path from node w to v in H. The lexicocraphically smallest shortest path from node u to v in H, if any, is denoted by $P_H(u, v)$, and the length of this path is denoted by $|P_H(u, v)|$. The distance between u and v in H is denoted by $d_H(u, v)$ and is equal to $|P_H(u, v)|$ if $P_H(u, v)$ exists, ∞ otherwise. If H is finite and connected, the eccentricity $\epsilon_H(u)$ of node u is defined as $\max_{w \in V_H} d_H(u, w)$. The degree of u in H will be denoted by $deg_H(u)$, or simply by $deg(u)$ if $H = \mathcal{G}$. We say that node u is *incomplete* (resp. *complete*) in H if $deg_H(u) < deg(u)$ (resp. $deg_H(u) = deg(u)$). We also say that a port p is free at node u in H, if $p \leq deg(u) - 1$ and there is no edge $(u, *, p, *)$ or $(*, u, *, p)$ in E_H.

We will often need to handle subgraphs of $\mathcal{G}$ through union and intersection operations. More precisely, given two subgraphs $H' = (V_{H'}, E_{H'})$ and $H'' = (V_{H''}, E_{H''})$ of $\mathcal{G}$, the union of (resp. the intersection of) H' and H'' is denoted by $H' \sqcup H''$ (resp. $H' \sqcap H''$) and is equal to $(V_{H'} \cup V_{H''}, E_{H'} \cup E_{H''})$ (resp. $(V_{H'} \cap V_{H''}, E_{H'} \cap E_{H''})$).

We define the boundary of a ball $B_f(\mathcal{G}, s)$, where s is the source node, as the set of nodes of $B_f(\mathcal{G}, s)$ that are incomplete in $B_f(\mathcal{G}, s)$.

To design our algorithm, we will also make use of three basic routines presented below. The first routine is `MoveTo`(H, v). Assuming that the agent currently occupies a node w of H and $P_H(w, v)$ exists, this routine moves the agent from node w to node v by following path $P_H(w, v)$. The second routine is `IncompleteNodes`(v, H, l) where l is a positive integer. This routine returns the set of all nodes w of H such that $d_H(v, w) \leq l$ and w is incomplete in H. The third routine is `Nodes`$(\mathcal{S})$, where $\mathcal{S}$ is a finite set of finite subgraphs of $\mathcal{G}$. This routine returns the union of all nodes in all subgraphs from $\mathcal{S}$.

Given an execution $\mathcal{E}$ of a series of instructions, the cost of $\mathcal{E}$ is the number of edge traversals performed by the agent during $\mathcal{E}$.

We will use the following convention. The agent will sometimes need to use Depth First Search traversal of graphs (not performed physically, but performed as a computation in the memory of the agent). Such a traversal depends on the order in which edges incident to a given node are traversed for the first time. We fix this order as the increasing order of port numbers at the given node. In this way the traversal is unambiguous, and we call it DFS.

3 Intuition

The purpose of this section is to sketch an intuitive overview of our algorithm that allows to find the treasure at an almost-optimal cost in the unrestricted model. To this end and to simplify the discussion, we will assume that the underlying graph $\mathcal{G}$ is countably infinite with nodes of finite degrees. We will rely on the notion of *largest explored ball*. By "largest explored ball", at a given phase of treasure hunt, we mean the ball $B_f(\mathcal{G}, s)$ where f is the largest integer such that each edge of $B_f(\mathcal{G}, s)$ has been traversed at least once. This largest integer f is the radius of the largest explored ball.

At a high level, our algorithm works in phases $i = 1, 2, 3, \ldots$ and immediately stops as soon as the treasure is found. At the beginning of phase i, the agent is located at node s and the radius of the largest explored ball is equal to f_i. The goal for the agent is to terminate the phase at node s while satisfying at least one of the following three conditions unless, of course, the treasure has been found before.

- *Condition 1.* The agent has entirely explored ball $B_{f_i+1}(\mathcal{G}, s)$, $e(f_i + 1) \geq 2e(f_i)$ and the cost of the phase is $\mathcal{O}(e(f_i + 1))$.
- *Condition 2.* The agent has entirely explored ball $B_{2f_i}(\mathcal{G}, s)$, $f_i \geq 1$ and the cost of the phase is $\mathcal{O}(e(f_i))$.
- *Condition 3.* The agent has entirely explored ball $B_{f_i+k}(\mathcal{G}, s)$ for some positive integer k, $e(f_i + k + 1) \geq 2e(f_i)$, $f_i \geq 2$, and the cost of the phase is $\mathcal{O}(e(f_i) \log f_i)$.

Actually, the conditions we really seek to meet in our algorithm are a little more intricate than those presented above, because we needed stronger technical requirements to refute the conjecture of Awerbuch, Betke, Rivest and Singh [3]. However, this would add an unnecessary level of complexity to understand the intuition, hence we omit these technical details here.

Before seeing how we implement our strategy, let us briefly examine why it permits us to get a cost quasi-linear in $e(d)$. Since $f_1 = 0$ and the radius of the largest explored ball increases by at least one during each phase in which the treasure is not found, the agent necessarily finds the treasure by the end of some phase $\lambda \leq d$, and $f_i < f_\lambda < d$ for every $1 \leq i < \lambda$. During each phase satisfying Condition 1, the size of the largest explored ball at least doubles, which means that the total cost of these phases is upper bounded by twice the

worst-case cost of the last phase satisfying Condition 1 i.e., $\mathcal{O}(e(f_\lambda + 1))$. Concerning the phases fulfilling Condition 2, their number is at most $\mathcal{O}(\log(f_\lambda + 1))$ and the cost of each of them cannot be more than $\mathcal{O}(e(f_\lambda))$, which implies that their total cost is $\mathcal{O}(e(f_\lambda)\log(f_\lambda+1))$. It remains to consider the case of the phases satisfying Condition 3. Given such a phase i, we have the guarantee that the size of the largest explored ball at least doubles between the beginning of phase i and the end of phase $i+1$, provided phase $i+1$ exists and is not prematurely interrupted by the discovery of the treasure. Indeed, at the end of phase i, the agent has at least entirely explored ball $B_{f_i+k}(\mathcal{G}, s)$ for some positive integer k and $e(f_i + k + 1) \geq 2e(f_i)$, while at the end of the (not prematurely interrupted) phase $i+1$ the agent has at least entirely explored ball $B_{f_{i+1}+1}(\mathcal{G}, s)$ with $f_{i+1} \geq f_i + k$. Using this, it can be shown that the total cost of the phases satisfying Condition 3 is at most four times the worst-case cost of the last phase satisfying this condition i.e., $\mathcal{O}(e(f_\lambda)\log(f_\lambda + 1))$. Given that the last phase λ can be viewed as a truncated phase that should have normally satisfied one of the three conditions, our sketch of analysis leads to the conclusion that the cost incurred by the agent till the discovery of the treasure is in $\mathcal{O}(e(f_\lambda + 1)\log(f_\lambda + 1))$, which is $\mathcal{O}(e(d)\log d)$ and is in line with our expectations.

Having justified the pertinence of such a strategy, we can turn our attention to its implementation. To do so, we need to introduce a technical building block, which we call `GlobalExpansion`(l, m) and to which we will go back at the end of this section to give additional details. Always executed from the source node s, it is a function that returns a boolean and whose two input parameters are positive integers except m that may be sometimes equal to the special symbol $\perp$. Assuming that $B_f(\mathcal{G}, s)$ is the largest explored ball, the execution of `GlobalExpansion`$(l, \perp)$ permits the agent to traverse all the edges of $B_{f+l}(\mathcal{G}, s)$ that are outside of $B_f(\mathcal{G}, s)$ before coming back to node s. Under the same assumption, the execution of `GlobalExpansion`(l, m), when m is a positive integer, consists for the agent in acting as if m was $\perp$ but with the following extra requirement: as soon as more than m distinct edges outside of $B_f(\mathcal{G}, s)$ have been traversed during the execution of the function, the agent backtracks to node s and aborts this execution. If m is $\perp$ or at least large enough to avoid an aborted execution, the agent ends up exploring $B_{f+l}(\mathcal{G}, s)$ and the function returns true. Otherwise, the function returns false. It should be stressed that all of this is made while guaranteeing two properties. The first one is that the agent is always in $B_{f+2l-1}(\mathcal{G}, s)$ during the execution of `GlobalExpansion`(l, m). The second is that the cost of the execution of `GlobalExpansion`(l, m) is $\mathcal{O}(e(f + 2l - 1))$ (resp. $\mathcal{O}(\min\{e(f) + m, e(f + 2l - 1)\})$) when $m = \perp$ (resp. $m \neq \perp$). Both these properties will turn out to be crucial to ensure a proper design of the phases. Finally, even if by chance the agent could explore a larger ball, we will assume for the ease of our intuitive explanations that $B_{f+l}(\mathcal{G}, s)$ (resp. $B_f(\mathcal{G}, s)$) is the largest ball explored by the agent at the end of `GlobalExpansion`(l, m) in the case where the returned value is true (resp. false).

Let us consider a phase i of our algorithm and, in order not to burden the text with a lot of "unless the treasure is found", let us assume that the treasure will not be found by the end of it. Phase i is made of at most three successive attempts, each of them aiming at fulfilling at least one of the three conditions described earlier, with the help of our building block. In the first attempt, the agent executes `GlobalExpansion`$(1, \perp)$ from node s, the cost of which is $\mathcal{O}(e(f_i + 1))$. At the end of this execution, the agent is at node s and $B_{f_i+1}(\mathcal{G}, s)$ has been entirely explored by the agent. If $e(f_i + 1) \geq 2e(f_i)$ or $f_i \leq 1$, the first attempt is a success as Condition 1 or Condition 2 is verified, and the agent directly switches to phase $i + 1$. Otherwise, the attempt is a failure, but we can nonetheless observe that the cost incurred because of the attempt is just $\mathcal{O}(e(f_i))$ because $e(f_i + 1) < 2e(f_i)$.

If the first attempt has failed, the agent starts the second attempt of phase i that consists of an execution of function `GlobalExpansion`$(f_i - 1, e(f_i))$. The hope here is to expand by a distance of $f_i - 1$ the radius of the largest explored ball, which is $B_{f_i+1}(\mathcal{G}, s)$. According to the properties of `GlobalExpansion` and the fact that $e(f_i + 1) < 2e(f_i)$, the cost of this execution, and thus of the second attempt, is $\mathcal{O}(e(f_i))$. If `GlobalExpansion`$(f_i - 1, e(f_i))$ returns true, then at the end of the second attempt, the radius of the largest explored ball is $2f_i$. Hence, the cost of the first two attempts being equal to $\mathcal{O}(e(f_i))$ and f_i being at least 2, Condition 2 is satisfied and the agent starts phase $i + 1$ without making the third attempt.

On the other hand, if `GlobalExpansion`$(f_i - 1, e(f_i))$ returns false, it is a different story. Indeed, the largest explored ball is still only $B_{f_i+1}(\mathcal{G}, s)$ and we cannot ensure the fulfillment of Condition 1 or Condition 2. This is exactly where Condition 3 comes into the picture. In order to remedy the failures of the two previous attempts, the agent will start a third and last attempt which consists of a dichotomic process that is described in Algorithm 1. At the end of this process, Condition 3 is guaranteed to be satisfied.

Algorithm 1 Third attempt.

```
1 floor := f_i + 1; ceil := 3f_i − 2; l := ⌊(ceil − floor)/2⌋;
2 while l ≥ 1 and |B_floor(G, s)| < 2e(f_i) do
3     success := GlobalExpansion(l, e(f_i));
4     if success = true then
5         floor := floor + l; l := ⌊(ceil − floor)/2⌋;
6     else
7         ceil := floor + 2l − 1; l := ⌊l/2⌋;
```

In order to better understand why we can get such a guarantee, let us take a look at the properties that are satisfied during the third attempt and at its end.

Since the execution of `GlobalExpansion`$(f_i - 1, e(f_i))$ returned false, the agent has explored at least $e(f_i)$ distinct edges outside of ball $B_{f_i+1}(\mathcal{G}, s)$ during the second attempt. Moreover, during this execution, the agent was always in $B_{3f_i-2}(\mathcal{G}, s)$ according to the properties of `GlobalExpansion`. As a result, in view of line 1 of Algorithm 1, we necessarily have the following feature before the execution of the while loop of Algorithm 1: $B_{floor}(\mathcal{G}, s)$ is the largest explored ball and $e(ceil) \geq 2e(f_i)$. Actually, by carefully examining the pseudocode of the while loop and using again the properties of `GlobalExpansion`, it can be inductively proven that this feature is a loop invariant. Alone, this loop invariant is not enough to bring the sought guarantee, but as highlighted below, it is of precious help to do the job.

The number of iterations of the while loop can be shown to be $\mathcal{O}(\log f_i)$. Furthermore, at the beginning of each iteration, $B_{floor}(\mathcal{G}, s)$ has size smaller than $2e(f_i)$ in view of the condition of the while loop, and is the largest explored ball in view of the loop invariant. Hence, according to the cost property of `GlobalExpansion`, each execution of `GlobalExpansion`$(l, e(f_i))$ costs at most $\mathcal{O}(e(f_i))$ like the previous two attempts, which gives a total cost of $\mathcal{O}(e(f_i) \log f_i)$ of the whole phase. This corresponds exactly to the target value of Condition 3. Along with this, at the end of the while loop, the size of $B_{floor}(\mathcal{G}, s)$ is at least $2e(f_i)$, or $l < 1$. In the first case, we immediately have $e(floor + 1) \geq 2e(f_i)$, while in the second case it can be shown that $ceil \leq floor + 1$. This, combined with the fact that $e(ceil)$ is always at least $2e(f_i)$ (by the loop invariant) and the fact that $floor$ is always at

least $f_i + 1$, allows us to show the last missing piece of the puzzle, which is precisely this: when Algorithm 1 terminates, ball $B_{f_i+k}(\mathcal{G}, s)$ is entirely explored and $e(f_i + k + 1) \geq 2e(f_i)$ for some integer $k \geq 1$.

To conclude with the intuitive explanations, let us give, as promised, some more insight concerning the building block `GlobalExpansion`(l, m). At first glance, one might think that `GlobalExpansion` could be directly derived from the exploration algorithm `CFX`(v, r, α) of [11], which permits to explore a ball $B_r(\mathcal{G}, v)$ at a cost of $\mathcal{O}\left(\frac{|B_{(1+\alpha)r}(\mathcal{G},v)|}{\alpha}\right)$ for any given real $\alpha > 0$ (this corresponds to a cost of $\mathcal{O}\left(\frac{e((1+\alpha)r)}{\alpha}\right)$ when $v = s$) provided $\alpha r \geq 1$. Indeed, the task of `GlobalExpansion`(l, m) that consists in expanding the radius f of the largest explored ball by a distance l in the case where m is appropriately set, can be done with `CFX`$(s, f + l, \alpha)$. However, in this case we want the cost of this expansion to be $\mathcal{O}(e(f + 2l - 1))$, which is an important property of our strategy. This cannot be guaranteed using `CFX`$(s, f + l, \alpha)$ because, in order to get a cost depending on $e(f + 2l - 1)$, we would have to set α to a value lower than $\frac{l-1}{f+l}$, which cannot lead to a cost that is linear in $e(f + 2l - 1)$, as $\frac{l-1}{f+l}$ can be arbitrarily small. True, during the design we could have been "less demanding" about some of the properties of `GlobalExpansion`(l, m), but not significantly enough to permit the use of `CFX`$(s, f + l, \alpha)$ without spoiling the validity or the cost complexity of our strategy. Another solution that may come to mind would be to apply `CFX`(v, l, α) from each node v located on the boundary of the largest explored ball $B_f(\mathcal{G}, s)$. Visiting each node of the boundary can be done in $\mathcal{O}(e(f))$. Hence, this solution looks attractive because by setting α to $\frac{1}{2}$ or less (which overcomes the above problem of the arbitrarily small value) and provided the zones explored by the different executions of `CFX` do not overlap, we would get a cost that is linear in $e(f + 2l - 1)$. The bad news is that there may be overlaps. Of course, some overlaps can be easily avoided, especially those appearing within $B_f(\mathcal{G}, v)$, but some others cannot without running the risk of missing some nodes of $B_{f+l}(\mathcal{G}, s)$ that are outside of $B_f(\mathcal{G}, s)$. These "necessary overlaps" may be pernicious and may occur in a way that prevents us from guaranteeing a cost of $\mathcal{O}(e(f + 2l - 1))$.

So, what did we do? Although it was not possible to use `CFX` as a black box, we managed to tailor `GlobalExpansion` by adapting to our needs an elegant algorithmic technique used in `CFX`. Through a set of judiciously pruned trees spanning some already explored area, it allowed us to satisfy the desired cost property of `GlobalExpansion` by controlling and amortizing efficiently the number of times the same edges are traversed. The technique in question is detailed in the next section that presents the pseudocode of our treasure hunt algorithm.

4 Algorithm

Solving the treasure hunt problem in the unrestricted model can be done by executing Algorithm `TreasureHunt`(x) described below in Algorithm 2 and by interrupting it as soon as the treasure is found.

Algorithm 2 `TreasureHunt`(x).

```
1 v := the current node;
2 M := ({v}, ∅); /* M is a global variable                    */
3 repeat
4     Search(x);
```

The input parameter x is a positive real constant. It is a technical ingredient that will have an impact on the maximal distance at which the agent can be from node s. In our present context, parameter x does not really matter and it can be fixed as *any* positive real constant. In fact, it will show its full significance in Section 6 that is dedicated to the same problem in restricted models: there, we will reuse `TreasureHunt`(x) in a context where x will have to be carefully chosen. The variable $\mathcal{M}$ in line 2 of Algorithm 2 is a global variable that will always correspond to some explored subgraph of $\mathcal{G}$. For this reason, it will recurrently appear in most of the pseudocodes of the functions described thereafter.

As the reader can see, the execution of Algorithm `TreasureHunt`(x) essentially consists of a series of executions of procedure `Search`(x), whose pseudocode is described in Algorithm 3: these executions correspond to what we called "phases" in our intuitive explanations of Section 3. Procedure `Search`(x) should be seen as the organizer of our solution. At the beginning of each call to `Search`(x), $\mathcal{M}$ is some explored ball $B_f(\mathcal{G}, s)$ and the goal of the call is to make this ball grow while satisfying some conditions. These conditions, whose simplified version we gave at the beginning of Section 3, are formally described in Lemma 4.

Algorithm 3 `Search`(x).

```
1  v := the current node; m := |M|;
2  floor := ε_M(v); ceil := ⌊(1 + x) · floor⌋;
3  success := GlobalExpansion(1, ⊥);
4  floor := floor + 1; i := 0; l := ⌊(ceil − floor)/2⌋;
5  while l ≥ 1 and |M| < 2m and (i ≠ 1 or success = false) do
6      success := GlobalExpansion(l, m);
7      if success = true then
8          floor := floor + l; l := ⌊(ceil − floor)/2⌋;
9      else
10         ceil := floor + 2l − 1; l := ⌊l/2⌋;
11     M := B_floor(M, v);
12     i := i + 1;
```

Although there are some technical differences, we can discern, throughout the lines of Algorithm 3, the three attempts outlined in Section 3 that rely on function `GlobalExpansion`. Roughly speaking, line 3 of Algorithm 3 relates to the first attempt, the first iteration of the while loop of Algorithm 3 relates to the second attempt, and the other iterations relate to the third attempt.

The pseudocode of function `GlobalExpansion`(l, m) is given by Algorithm 4. It has primarily the same specifications as those given in Section 3 except that we did not implement the case where $m = \perp$ and $l \geq 2$ as it was not necessary for our purpose. Hence, the function precisely handles the case where $l = 1$ and $m = \perp$, and the case where $l \geq 1$ and $m \neq \perp$. The general scheme of the function is as follows. At the beginning, the agent knows a ball $B_f(\mathcal{G}, s)$ that is stored in variable $\mathcal{M}$ and the objective is to expand the radius of this ball by a distance l, without exploring more than m edges outside of $B_f(\mathcal{G}, s)$, if $m \neq \perp$. To do this, the agent visits the nodes $L[1], L[2], \ldots$ (stored in the array L) of the boundary of $B_f(\mathcal{G}, s)$ and executes from these nodes function `CDFS` (described in Algorithm 5 and whose name stands for Constrained DFS) or function `LocalExpansion` (described in Algorithm 6) depending on the initial values of l and m. Each of these executions, which starts and ends at the same node, locally contributes to the global expansion of the ball. In the case where

$m \neq \perp$, variable b of Algorithm 4 is updated with the return value of the two aforementioned functions, and corresponds at each stage to the remaining number of new edges the agent is authorized to traverse outside of $B_f(\mathcal{G}, s)$. If b becomes negative before the end of the while loop of Algorithm 4, the objective of expansion is simply not reached. Note that, in order to avoid that the moves from one node of the boundary of $B_f(\mathcal{G}, s)$ to the next get too costly, they are made according to a precise order that results from the definition of L given in line 2 of Algorithm 4.

Algorithm 4 `GlobalExpansion`(l, m).

```
1  v := the current node;
2  L := the array containing all the nodes of the boundary of M sorted in the order of
     the first visit through the DFS traversal of M from node v;
3  T := the tree produced by the DFS traversal of M from node v;
4  i := 1; b := m; T := ∅; /* T is a global variable                              */
5  while i ≤ |L| and (b ≥ 0 or b = ⊥) do
6      MoveTo(T, L[i]);
7      if l = 1 then
8          if b = ⊥ then
               /* We run CDFS(1, deg(L[i])) without using its return value.   */
9              (∗, ∗) := CDFS(1, deg(L[i]));
10         else
               /* We run CDFS(1, b) without using the second term of its
                  return value.                                                */
11             (b, ∗) := CDFS(1, b);
12     else
13         b := LocalExpansion(l, b);
14     i := i + 1;
15 MoveTo(T, v);
16 return the logical value of "b ≥ 0 or b = ⊥";
```

As one can see in lines 9 and 11 of Algorithm 4, the implementation of the case $l = 1$ in Algorithm 4 directly relies on function `CDFS`. We will see below that this function is also involved in the trickier case where $l \geq 2$ and $m \neq \perp$ through the calls to function `LocalExpansion`. Function `CDFS`(l, b) permits the agent to perform a depth-first search in the zone that does not belong to $\mathcal{M}$ when it starts executing it. During the execution of this function $\mathcal{M}$ grows, augmented with the edges that are traversed by the agent. The two input parameters $l \geq 1$ and $b \geq 0$ are integers that bring constraints to the execution of the depth-first search. The first indicates the limit depth of the search, while the second indicates an upper bound on the number of distinct edges the agent can traverse during the search: when this bound is violated, the agent stops the search and goes back to the node it occupied at the beginning of the search. The return value of `CDFS`(l, b) is a couple (n, T). The first term n is an integer such that $b - n$ is the number of distinct edges that have been traversed during the execution of `CDFS`(l, b). If the bound b has been respected then $n \geq 0$, otherwise $n = -1$. Concerning the second term T of the return value, it simply corresponds to the resulting DFS tree of the execution of `CDFS`(l, b). If $n \geq 0$ and v is the occupied node at the start of `CDFS`(l, b), then for every node u such that $d_T(u, v) < l$, u is complete in $\mathcal{M}$ at the end of `CDFS`(l, b). Note that in the particular case where $l = 1$ and $m = \perp$ in Algorithm 4,

the second argument of each call to `CDFS` is always set to the degree of the node from which the function is executed (cf. line 9 of Algorithm 4) in order to ensure that this node becomes complete in $\mathcal{M}$ at the end of the call.

Algorithm 5 `CDFS`(l, b).

```
1  v := the current node; T := ({v}, ∅); bound := b;
2  if l > 0 then
3      Mark node v;
4      while node v is incomplete in M and bound ≥ 0 do
5          pt_1 := the smallest free port at node v in M;
6          Take port pt_1;
7          w := the current node;
8          pt_2 := the port by which the agent has just entered node w;
9          if v < w then
10             K := ({v, w}, {(v, w, pt_1, pt_2)});
11         else
12             K := ({v, w}, {(w, v, pt_2, pt_1)});
13         M := M ⊔ K; bound := bound − 1;
14         if w is not marked then
15             (bound, T') := CDFS(l − 1, bound);
16             T := T ⊔ T' ⊔ K;
17         Take port pt_2;
18     Unmark node v;
19 return (bound, T);
```

The case where $l \geq 2$ and $m \neq \perp$ in Algorithm 4 relies on function `LocalExpansion`. It is exactly here that we make use of the algorithmic technique of [11] mentioned at the end of Section 3, which is based on a set of adequately pruned trees. In our solution, this set corresponds to the variable $\mathcal{T}$. It is a global variable like $\mathcal{M}$ and it is initialized to $\emptyset$ at the beginning of each call to `GlobalExpansion` (cf. line 4 of Algorithm 4). Let us consider the ith call LE_i to `LocalExpansion`(l, b) made from node $L[i]$ during an execution of `GlobalExpansion`(l, m). At the end of LE_i, the return value of `LocalExpansion`(l, b) is an integer $n \geq -1$ such that $b - n$ is the number of distinct edges that have been traversed during LE_i and that were not in $\mathcal{M}$ at the start of LE_i. Besides, in the case where $n \geq 0$, at the end of LE_i we can guarantee that for each incomplete node u of $\mathcal{M}$, $d_{\mathcal{M}}(L[i], u) > l$ or u is one of the last $|L| - i$ nodes of L (i.e., a node of L from which the agent has not yet executed `LocalExpansion`(l, b)).

To see the algorithmic technique in question at work, let us focus on an iteration I of the first while loop of Algorithm 6 occuring in LE_i. This iteration starts at node $L[i]$ and we can show that at the beginning of I, we necessarily have the following properties.

- $\mathcal{T}$ is a set of node disjoint trees that are all subgraphs of $\mathcal{M}$.
- For each tree Tr of $\mathcal{T}$, $|Tr| \geq \lfloor \frac{l}{8} \rfloor$ if Tr contains a node different from $L[i]$.
- Every incomplete node of $\mathcal{M}$ belongs to a tree of $\mathcal{T}$ or is one of the last $|L| - i$ nodes of L.

Algorithm 6 `LocalExpansion`(l, b).

1 $bound := b$; $v :=$ the current node;
2 **if** *v is incomplete in $\mathcal{M}$* **and** *no tree of $\mathcal{T}$ contains node v* **then**
3 $\mathcal{T} := \mathcal{T} \cup \{(\{v\}, \emptyset)\}$;
4 **while** $\texttt{IncompleteNodes}(v, \mathcal{M}, l) \cap \texttt{Nodes}(\mathcal{T}) \neq \emptyset$ **and** $bound \geq 0$ **do**
5 $u :=$ the node with the smallest label in $\texttt{IncompleteNodes}(v, \mathcal{M}, l) \cap \texttt{Nodes}(\mathcal{T})$;
6 $\texttt{MoveTo}(\mathcal{M}, u)$;
7 $\texttt{Prune}(l)$;
8 $bound := \texttt{Explore}(l, bound)$;
9 Remove from $\mathcal{T}$ every tree for which all the nodes are complete in $\mathcal{M}$;
10 **while** *there are two trees T and T' in $\mathcal{T}$ having a common node* **do**
11 $T'' :=$ the spanning tree produced by the BFS traversal of $T \sqcup T'$ from the node having the smallest label in $T \sqcup T'$;
12 $\mathcal{T} := (\mathcal{T} \setminus \{T, T'\}) \cup \{T''\}$;
13 Execute in the reverse order all the edge traversals that have been made since the beginning of the current iteration of the while loop;
14 **return** *bound*;

Let us examine what happens during iteration I. At the beginning of I, the agent follows a path of length at most l from node $L[i]$ to a node u that is incomplete in $\mathcal{M}$ (cf. line 5 of Algorithm 6). By the first and third properties and the condition at line 4 of Algorithm 6, node u belongs to a unique tree $T_u \subseteq \mathcal{G}$ of $\mathcal{T}$. Once the agent occupies node u, the tree T_u is pruned via the procedure $\texttt{Prune}(l)$ at line 7 of Algorithm 6. The pseudocode of procedure `Prune` is detailed in Algorithm 7.

Algorithm 7 `Prune`(l).

1 $v :=$ the current node;
2 $T_v :=$ the tree of $\mathcal{T}$ containing node v;
3 $\mathcal{T} := \mathcal{T} \setminus \{T_v\}$;
4 Root T_v at node v;
5 **foreach** *node u of T_v such that $d_{T_v}(u, v) = \max\{1, \lfloor \frac{l}{4} \rfloor\}$* **do**
6 $T_u :=$ the subtree of T_v rooted at u;
7 **if** $\epsilon_{T_u}(u) \geq \lfloor \frac{l}{4} \rfloor - 1$ **then**
8 $\mathcal{T} := \mathcal{T} \cup \{T_u\}$;
9 Remove from T_v all nodes that belong to T_u and all edges that are incident to a node of T_u;
10 $\mathcal{T} := \mathcal{T} \cup \{T_v\}$;

In the context of iteration I, the pruning operation will transform T_u into a tree T'_u such that $\epsilon_{T'_u}(u) \leq \lfloor \frac{l}{2} \rfloor - 1$, while preserving the three properties listed above: this offers two important advantages to which we will return at the end of this section. Once the pruning is done, the agent applies function $\texttt{Explore}(l, bound)$, whose pseudocode is given in Algorithm 8.

Algorithm 8 `Explore`(l, b).

```
1 bound := b; i := 1; v := the current node;
2 T := the tree of 𝒯 containing node v;
3 V := array containing all the nodes of T sorted in the order of the first visit through
    the DFS traversal of T from node v;
4 while i ≤ |V| and bound ≥ 0 do
5     MoveTo(T, V[i]);
6     if node V[i] is incomplete in ℳ then
7         (bound, T') := CDFS(⌊l/2⌋, bound);
8         𝒯 := 𝒯 ∪ {T'};
9 return bound;
```

In the pseudocodes of `LocalExpansion` and of `Explore`, variable *bound* corresponds at any stage to the number of remaining edges the agent is authorized to traverse outside of $B_f(\mathcal{G}, s)$. In the context of iteration I, function `Explore`$(l, bound)$ permits the agent to explore tree T'_u and to execute function `CDFS`$(\lfloor \frac{l}{2} \rfloor, bound)$ from the nodes of T'_u that are incomplete in $\mathcal{M}$, as long as variable *bound* remains non-negative. These executions of `CDFS` occuring during the exploration of T'_u create in turn trees that are added to $\mathcal{T}$ (cf. line 8 of Algorithm 8) and that contain the new incomplete nodes of $\mathcal{M}$. If the return value of function `Explore`$(l, bound)$ is non-negative, we can show that all the nodes of T'_u have become complete in $\mathcal{M}$. Under the same condition, we will also guarantee that each tree Tr, which has been added to $\mathcal{T}$ during the execution of function `Explore`, contains an incomplete node only if $|Tr| \geq \lfloor \frac{l}{8} \rfloor$. Both these guarantees combined with lines 9 to 12 of Algorithm 6 will allow us to show that our three properties will be satisfied for the next iteration I', if any, even if it occurs in another call to `LocalExpansion` (in the same execution of `GlobalExpansion`(l, m)). In particular, this is made possible by the fact that $\mathcal{T}$ is never reset between the calls to `LocalExpansion` during the execution of the while loop of Algorithm 4.

To fully appreciate the process accomplished during I, we need to come back to the two aforementioned advantages that are brought by the pruning operation. The first advantage concerns the height of T'_u. The fact that $\epsilon_{T'_u}(u) \leq \lfloor \frac{l}{2} \rfloor - 1$ is a key element to control the maximal distance between the agent and node s. Without this, the agent could go too far from node s and we would not be able to guarantee that the agent explores only edges of $B_{f+2l-1}(\mathcal{G}, s)$ during the execution of `GlobalExpansion`(l, m) (which is a crucial property as pointed out in Section 3). The second advantage concerns the size of T'_u. The pruning operation preserves the second property, and thus (1) T'_u corresponds to a tree containing only node $L[i]$ or (2) $|T'_u| \geq \lfloor \frac{l}{8} \rfloor$. This implies that the cost resulting from the moves of line 6 of Algorithm 6 and line 5 of Algorithm 8 is linear in the size of T'_u. Besides, if *bound* is still non-negative at the end of `Explore`$(l, bound)$, all the nodes of T'_u have become complete (it is in particular the case for node u) and the tree is removed from $\mathcal{T}$ through line 9 of Algorithm 6. After this removal, no edge of T'_u will be an edge of another tree of $\mathcal{T}$ till the end of the execution of `GlobalExpansion`(l, m). As a result, if the return value of `Explore`$(l, bound)$ is non-negative in I, we can associate the moves of line 6 of Algorithm 6 and line 5 of Algorithm 8 to at least one node that becomes complete during I and to at least $\lfloor \frac{l}{8} \rfloor$ edges that will no longer be edges of any tree of $\mathcal{T}$ till the end of the execution of `GlobalExpansion`(l, m). In our analysis, this association will enable us to amortize efficiently the number of times the agent retraverses the edges that have been already explored during any previous iteration of the considered while loop. This will be a decisive argument to show the cost of $\mathcal{O}(e(f) + m)$ for the execution of `GlobalExpansion`(l, m) in the case where $l \geq 2$ and $m \neq \perp$.

5 Correctness and complexity analysis

In this section, we give a sketch of the proof of correctness and of complexity of Algorithm `TreasureHunt`(x) in the unrestricted model. `TreasureHunt`(x) is an exploration algorithm that can be executed also if there is no treasure in $\mathcal{G}$. We first establish several exploration properties of our algorithm or of its components assuming that there is no treasure in $\mathcal{G}$. In fact, this assumption concerns all the lemmas (and only them) of this section. After the series of lemmas, we show the main result of this section, namely Theorem 6, which specifies that our algorithm allows to find the treasure at a cost quasi-linear in $e(d)$.

Throughout the proof of correctness, we will often have to consider the value of the global variable $\mathcal{M}$ before or after some executions. To this end, we introduce the following convention: given an execution $\mathcal{E}$ of Algorithm `TreasureHunt`(x) or some part of it, we denote by $M_1(\mathcal{E})$ the value of $\mathcal{M}$ at the beginning of $\mathcal{E}$ and by $M_2(\mathcal{E})$ the value of $\mathcal{M}$ at the end of $\mathcal{E}$.

We start by giving two lemmas concerning the function `CDFS`(l, b). They list some properties that are useful to prove Lemma 3. They are direct consequences of Algorithm 5 and can be easily proved by induction on l.

▶ **Lemma 1.** *Consider an execution $\mathcal{E}$ of function* `CDFS`(l, b) *from a node u of $\mathcal{G}$ where $l \geq 1$ and $b \geq 0$ are integers. Assume that $M_1(\mathcal{E}) \subseteq \mathcal{G}$. Execution $\mathcal{E}$ terminates at node u, and the agent always knows a path of length at most l from node u to its current node during $\mathcal{E}$.*

▶ **Lemma 2.** *Consider an execution $\mathcal{E}$ of function* `CDFS`(l, b) *from a node u of $\mathcal{G}$ where $l \geq 1$ and $b \geq 0$ are integers. Assume that $M_1(\mathcal{E}) \subseteq \mathcal{G}$. Function* `CDFS`$(l, b)$ *returns a couple (i, Tr) such that the following properties are satisfied.*

- *Let G be the subgraph of $\mathcal{G}$ that has been explored during $\mathcal{E}$. $G \subseteq B_l(\mathcal{G}, u)$, $|M_1(\mathcal{E}) \sqcap G| = 0$, $M_1(\mathcal{E}) \sqcup G = M_2(\mathcal{E})$, Tr is a spanning tree of G and $i = b - |G| \geq -1$.*
- *The cost of $\mathcal{E}$ is $2|G|$ and $\epsilon_{Tr}(u) \leq l$.*
- *If $i \geq 0$ then for every node v of Tr such that $d_{Tr}(u, v) < l$, v is complete in $M_2(\mathcal{E})$. If $i = -1$, then there exists a node v of Tr such that $d_{Tr}(u, v) \leq l - 1$ and v is incomplete in $M_2(\mathcal{E})$.*

The following lemma establishes the properties of function `GlobalExpansion`(l, m). It is prerequisite to prove Lemma 4 that concerns procedure `Search`(x).

▶ **Lemma 3.** *Consider an execution $\mathcal{E}$ of function* `GlobalExpansion`(l, m) *from the source node s, where l is a positive integer and m is either a positive integer or $\perp$. Assume that $M_1(\mathcal{E}) = B_f(\mathcal{G}, s)$ for some integer $f \geq 0$.*

- *if $m \neq \perp$, or $m = \perp$ and $l = 1$, then $\mathcal{E}$ terminates at node s and during $\mathcal{E}$ the agent always knows a path in $\mathcal{G}$ of length at most $f + 2l - 1$ from node s to its current node.*
- *If $m = \perp$ and $l = 1$ then the cost of $\mathcal{E}$ is $\mathcal{O}(e(f + 1))$ and $B_{f+1}(\mathcal{G}, s) = M_2(\mathcal{E})$.*
- *If $m \neq \perp$ and function* `GlobalExpansion`(l, m) *returns true (resp. false) then $B_{f+l}(\mathcal{G}, s) \subseteq M_2(\mathcal{E})$ (resp. $B_f(\mathcal{G}, s) \subseteq M_2(\mathcal{E})$ and $e(f + 2l - 1) > e(f) + m$) and the cost of $\mathcal{E}$ is $\mathcal{O}(e(f) + m)$.*

Below is the lemma establishing the properties of procedure `Search`(x).

▶ **Lemma 4.** *Consider an execution $\mathcal{E}$ of procedure* `Search`(x) *from the source node s, for any real constant $x > 0$. Assume that $M_1(\mathcal{E}) = B_f(\mathcal{G}, s)$ for some integer $f \geq 0$.*

- *The execution terminates at node s and during the execution the agent always knows a path in $\mathcal{G}$ of length at most $\max\{f + 1, \lfloor (1 + x)f \rfloor\}$ from node s to its current node.*
- *There exists an integer $f' > f$ such that $M_2(\mathcal{E}) = B_{f'}(\mathcal{G}, s)$ and at least one of the following properties holds:*

1. *The cost of* $\mathcal{E}$ *is* $\mathcal{O}(e(f+1))$ *and* $xf < 3$.
2. *The cost of* $\mathcal{E}$ *is* $\mathcal{O}(e(f))$ *and* $f' > (1+\frac{x}{3})f$.
3. *The cost of* $\mathcal{E}$ *is* $\mathcal{O}(e(f)\log(f+2))$ *and* $e(f'+1) \geq 2e(f)$.
4. *The cost of* $\mathcal{E}$ *is* $\mathcal{O}(e(f+1))$ *and* $e(f+1) \geq 2e(f)$.

If we put aside the initial assignments of lines 1 and 2 in Algorithm 2, the execution of procedure `TreasureHunt`(x) from the source node s in $\mathcal{G}$ can be viewed as a sequence of consecutive executions of procedure `Search`(x): the i*th* execution of `Search`(x) in this sequence will be denoted by S_i.

The following lemma is a small technical observation concerning the execution of `TreasureHunt`(x) from the source node s. Since, at the beginning of this execution, variable $\mathcal{M}$ is equal to $B_0(\mathcal{G}, s)$, the lemma can be easily proved by induction on i using Lemma 4.

▶ **Lemma 5.** *Consider an execution of procedure* `TreasureHunt`(x) *from the source node* s*, for any real constant* $x > 0$*. For every integer* $i \geq 1$*,* S_i *starts and ends at node* s*, and there are two integers* $f_{i+1} > f_i \geq i-1$ *such that* $M_1(S_i) = B_{f_i}(\mathcal{G}, s)$ *and* $M_2(S_i) = B_{f_{i+1}}(\mathcal{G}, s)$.

Using Lemmas 4 and 5, we can prove the main result of this section that is stated in the following theorem.

▶ **Theorem 6.** *Consider a graph* $\mathcal{G}$ *of unknown radius* r *in which a treasure is located at an unknown distance at most* $1 < d \leq r$ *from the starting node* s *of an agent. For any real constant* $x > 0$*, procedure* `TreasureHunt`(x) *allows the agent to find the treasure at cost* $\mathcal{O}(e(d)\log d)$.

6 Treasure hunt with restrictions

Theorem 6 holds for the task of treasure hunt without any restrictions on the moves of the agent, for all locally finite graphs, both finite and infinite. In this section we show how to modify our treasure hunt algorithm to make it work under the fuel-restricted and the rope-restricted models for finite graphs.

Strictly speaking, the fuel-restricted model was defined in [3] assuming that both the constant $\alpha > 0$ and the radius r were known to the agent. On the other hand, the rope-restricted model was defined in [11] for any known constant $\alpha > 0$ and for unknown radius r. We will show that, for each of these restrictive models and for any known constant $\alpha > 0$, we can design a treasure hunt algorithm with the promised efficiency even when r is unknown. To this end, we need to modify the restriction of the fuel-restricted model from [3], avoiding to reveal r to the agent by showing it the size of the tank. We fix a positive constant α, known to the agent, and we proceed as follows. For the restricted tank case from [3], we assume that at any visit of s the agent can put as much fuel in the tank as it wants, but we show that if the (unknown) radius of the graph is r then the tank is never filled to more than $B = 2(1+\alpha)r$. The formalization of the rope-restricted model corresponds to its definition in [11]. Recall that the agent is attached at s by an infinitely extendible rope that it unwinds by a length 1 with every forward edge traversal and rewinds by a length of 1 with every backward edge traversal. Whenever the agent completely backtracks to s, the unwinded segment of the rope is of length 0. We show that if the (unknown) radius of the graph is r then the initial segment of the rope unwinded by the agent executing our algorithm will never be longer than $L = (1+\alpha)r$.

▶ **Theorem 7.** *Consider a graph $\mathcal{G}$ of unknown radius r in which a treasure is located at an unknown distance at most $1 < d \leq r$ from the starting node s of the agent. For any positive constant α, procedure* `TreasureHunt`$(\frac{\alpha}{2})$ *can be transformed into a procedure allowing the agent to find the treasure at cost $\mathcal{O}(e(d)\log d)$ in the rope-restricted model (resp. fuel-restricted model) without ever using a segment of the rope longer than $(1+\alpha)r$ (resp. without filling the tank to more than $2(1+\alpha)r$ at any visit of s).*

Proof. The execution of procedure `TreasureHunt`$(\frac{\alpha}{2})$ from node s corresponds to a sequence $S = (S_1, S_2, \ldots, S_{|S|})$ of executions of `Search`$(\frac{\alpha}{2})$, in which the $|S|$th execution of `Search`$(\frac{\alpha}{2})$ is interrupted prematurely because of the discovery of the treasure.

We denote by G_0 the graph consisting only of node s, and for every $1 \leq i \leq |S|$, we denote by G_i the subgraph of $\mathcal{G}$ that has been explored from the beginning of S_1 to the end of S_i. For every $1 \leq i \leq |S|$, the cost of S_i will be denoted by c_i.

According to Lemma 5, for every $1 \leq i \leq |S|$, S_i starts and ends at node s (except $S_{|S|}$ that ends at the node containing the treasure), there is an integer $f_i \geq 0$ such that $M_1(S_i) = B_{f_i}(\mathcal{G}, s)$ and if $i < |S|$, $M_2(S_i) = M_1(S_{i+1})$. Moreover, the value of $\mathcal{M}$ is always a subgraph of $\mathcal{G}$ whose nodes and edges have been all explored by the agent, and thus, for every $1 \leq i \leq |S|$, $B_{f_i}(\mathcal{G}, s)$ is a subgraph of G_{i-1}, f_i is unique and $f_i < d$ (or otherwise the treasure would have been found before the start of S_i which leads to a contradiction with the existence of this execution). Hence, from the fact that $d \leq r$, we get the following claim.

▷ Claim 8. For every $1 \leq i \leq |S|$, $\max\{f_i + 1, \lfloor(1+\alpha)f_i\rfloor\} \leq (1+\alpha)r$

First, we describe a new algorithm $\mathcal{A}$ that permits to find the treasure, in the model without constraints, with asymptotically the same cost as that of `TreasureHunt`$(\frac{\alpha}{2})$. This new algorithm consists in executing `TreasureHunt`$(\frac{\alpha}{2})$ with some changes in order to guarantee an extra property that will be important for our purpose. More precisely, an execution of $\mathcal{A}$ from node s is a sequence of executions $(S'_1, S'_2, \ldots, S'_{|S|})$ in which each S'_i has cost $\mathcal{O}(c_i)$ and corresponds to an emulation of execution S_i. In particular, for every $1 \leq i \leq |S|$, S'_i starts and ends at node s (except $S'_{|S|}$ that ends at the node containing the treasure), $M_1(S'_i) = M_1(S_i) = B_{f_i}(\mathcal{G}, s)$, and at the end of S'_i, G_i has been entirely explored. Obviously, all of this would not be interesting without the additional crucial property brought by S'_i that will be called the *frequent return property* and that is the following. Let Sk be the stack initially empty in which we push (resp. pop) the last traversed edge if it corresponds to a forward (resp. backward) edge traversal. During S'_i, the size of Sk is 0 at least once during any block of $2\max\{f_i + 1, \lfloor(1+\alpha)f_i\rfloor\}$ consecutive edge traversals, and is never greater than $\max\{f_i + 1, \lfloor(1+\alpha)f_i\rfloor\}$. Moreover, at the beginning of S'_i, the size of Sk is 0, and if $i < |S|$, it is also 0 at the end of S'_i.

Note that Algorithm $\mathcal{A}$ is a solution with the desired cost in the rope-restricted model, that will never use a segment of the rope longer than $(1+\alpha)r$, as for all $1 \leq i \leq |S|$, we have $\max\{f_i+1, \lfloor(1+\alpha)f_i\rfloor\} \leq (1+\alpha)r$ according to Claim 8. By requiring the agent, each time the size of Sk is 0 in S'_i, to refuel its tank up to the limit of $2 \cdot \max\{f_i + 1, \lfloor(1+\alpha)f_i\rfloor\}$ (when the size of Sk is 0, the agent is at node s), we also get our objective with algorithm $\mathcal{A}$ in the fuel-restricted model, as the agent never runs out of fuel and $2\cdot\max\{f_i+1, \lfloor(1+\alpha)f_i\rfloor\} \leq 2(1+\alpha)r$.

Let us describe how we can construct our emulations while ensuring the features mentioned above. Consider the emulation S'_i of S_i. Assume that at the beginning of S'_i, G_{i-1} has been entirely explored, the size of Sk is 0 and $M_1(S'_i) = M_1(S_i) = B_{f_i}(\mathcal{G}, s)$. These assumptions are trivially satisfied if $i = 1$. We will show below that, at the end of S'_i, G_i is entirely explored and if $i < |S|$ the size of Sk is 0. We will also show that if $i < |S|$ then $M_1(S'_{i+1}) = B_{f_{i+1}}(\mathcal{G}, s)$. We consider two cases.

The first case is when $\alpha f_i \geq 2$. We assume for simplicity that the number of edge traversals in S_i is a positive multiple of $\lfloor \frac{\alpha f_i}{2} \rfloor$. As we will explain in detail, in this case the agent executes S_i but interrupts it after each block of $\lfloor \frac{\alpha f_i}{2} \rfloor$ edge traversals, except the last one, to make a "return trip" to node s before resuming S_i from where it was interrupted. The goal of these return trips is to satisfy the frequent return property. Once the agent has executed all instructions of S_i, it is either at the node containing the treasure or at node s. In the first case, we know that $i = |S|$ and S'_i is simply over. In the second case $i < |S|$, but we do not have the guarantee that the size of Sk is 0. Hence, if the agent occupies node s once it has executed all instructions of S_i, it then finishes S'_i with what we call a *close period* in which it executes in the reverse order some of the last edge traversals so that the size of Sk becomes 0 at the end of S'_i.

Denote by v_k the node in which the kth interruption occurs, and by P_k the path of length at most $\lfloor (1 + \frac{\alpha}{2}) f_i \rfloor$ from node s to v_k that is known by the agent when the interruption occurs. Note that P_k necessarily exists in view of Lemma 4, of the initial assumptions concerning S'_i and of the fact that no edge traversal of S_i has been skipped before the kth interruption. Also note that if there are several paths that can play the role of P_k, we simply choose the lexicographically smallest shortest path among them.

Each interruption is composed of two parts. In the first interruption, the first part consists in backtracking to node s by executing in the reverse order the last $\lfloor \frac{\alpha f_i}{2} \rfloor$ edge traversals. The second part consists in going back to node v_1 using path P_1 to resume S_i. For the kth interruption with $k > 1$, the first part consists in backtracking to node s by executing in the reverse order the last $|P_{k-1}| + \lfloor \frac{\alpha f_i}{2} \rfloor$ edge traversals, and the second part consists in going back to node v_k using path P_k to resume S_i. Finally, the close period simply consists in backtracking to node s by executing in the reverse order the last $\lfloor \frac{\alpha f_i}{2} \rfloor$ edge traversals if S_i is made of only one block of $\lfloor \frac{\alpha f_i}{2} \rfloor$ edge traversals. Otherwise, it consists in backtracking to node s by executing in the reverse order the last $|P_{k^*-1}| + \lfloor \frac{\alpha f_i}{2} \rfloor$ edge traversals where $k^* = \frac{c_i}{\lfloor \frac{\alpha f_i}{2} \rfloor}$ is the number of blocks of $\lfloor \frac{\alpha f_i}{2} \rfloor$ edge traversals in S_i.

It follows by induction on the number of interruptions that the size of Sk is 0 at the end of the first part of each interruption. Using this, the fact that Sk is empty at the beginning of S'_i and the fact that for every $1 < k \leq \frac{c_i}{\lfloor \frac{\alpha f_i}{2} \rfloor}$, $|P_{k-1}| + \lfloor \frac{\alpha f_i}{2} \rfloor \leq \lfloor (1 + \alpha) f_i \rfloor$, it follows that the frequent return property is satisfied during S'_i.

Moreover, it follows from the above explanation that at the end of S'_i, G_i is entirely explored and the agent is at the node containing the treasure, if $i = |S|$. If $i < |S|$, it also follows that the size of Sk is 0 at the beginning of the next emulation S'_{i+1}, and $M_1(S'_{i+1}) = M_1(S_{i+1}) = B_{f_{i+1}}(\mathcal{G}, s)$ because $M_2(S'_i) = M_2(S_i) = M_1(S_{i+1})$. Finally, concerning the cost of S'_i observe that the number of interruptions is $\frac{c_i}{\lfloor \frac{\alpha f_i}{2} \rfloor} - 1$ and during each interruption as well as during the close period the agent makes at most $2\lfloor (1+\alpha) f_i \rfloor$ edge traversals. The cost of S'_i is then upper bounded by $c_i + \frac{c_i}{\lfloor \frac{\alpha f_i}{2} \rfloor} 2 \lfloor (1+\alpha) f_i \rfloor \leq (1 + \frac{2(1+\alpha) f_i}{\lfloor \frac{\alpha f_i}{2} \rfloor}) c_i$. If $2 \leq \alpha f_i < 4$, then $\frac{2}{\alpha} \leq f_i < \frac{4}{\alpha}$, which implies that the cost is at most $(1 + \frac{8(1+\alpha)}{\alpha}) c_i$. Otherwise, $\alpha f_i \geq 4$ and the cost is then upper bounded by $(1 + \frac{2(1+\alpha) f_i}{\frac{\alpha f_i}{2} - 1}) c_i \leq (1 + \frac{2(1+\alpha)}{\frac{\alpha}{2} - \frac{1}{f_i}}) c_i$ which is also at most $(1 + \frac{8(1+\alpha)}{\alpha}) c_i$, as $\frac{1}{f_i} \leq \frac{\alpha}{4}$. Hence, the cost of S'_i is $\mathcal{O}(c_i)$ as α is a constant, which concludes the first case.

The second case is when $\alpha f_i < 2$. Here, we could not apply the same strategy as that of the first case because we have $\lfloor \frac{\alpha f_i}{2} \rfloor = 0$. Consequently, we adopt a slightly different strategy in which the agent executes S_i but interrupts it before each of its edge traversals. As explained in detail below, the kth interruption either consists of a return trip to node s before resuming S_i and making the kth edge traversal of S_i, or it consists in going to the node the agent should occupy at the end of the kth edge traversal of S_i but without

taking the corresponding edge: the agent then resumes S_i as if it had just performed the kth edge traversal of S_i (essentially it just makes some computations before interrupting again S_i for the next edge traversal, if any). We will show that the latter situation will occur only when the "skipped edge" has already been traversed before by the agent. Once S_i has been entirely processed, S'_i is simply over if the agent is located at the node containing the treasure. Otherwise, the agent is at node s and $i < |S|$. In this case, it executes (similarly as in the previous case) a close period in order to guarantee that the size of Sk is 0 at the end of S'_i.

Let us first focus on the interruptions. We denote by $(u_1, u_2, u_3, \ldots, u_{c_i+1})$ the sequence (with repetitions), in the chronological order, of the nodes that are visited during S_i, and by $(e_1, e_2, e_3, \ldots, e_{c_i})$ the sequence (with repetitions), in the chronological order, of the edges that are traversed during S_i. Consider the kth interruption occuring at node u_k just before the kth edge traversal of S_i and assume that at the beginning of this interruption, the property $H(k)$, consisting of the following three conditions, is satisfied:

- The agent has made $D_k \leq f_i + 1$ edge traversals since the last time when Sk was empty (this could be the current time).
- The sequence of edges $(e_1, e_2, \ldots, e_{k-1})$ has been previously explored by the agent.
- The size of Sk has been 0 at least once during any previous block of $2(f_i + 1)$ consecutive edge traversals and has never been greater than $f_i + 1$.

Note that at the beginning of the first interruption, property $H(1)$ immediately holds. We will show below that property $H(k + 1)$ is satisfied at the beginning of the $(k + 1)$th interruption, if any.

In the kth interruption, the agent first checks whether it knows a path of length at most f_i from node s to node u_k. If this is the case, the agent executes in the reverse order the last D_k edge traversals, at the end of which it is at node s and Sk is empty. Then, the agent comes back to u_k using the known path of length at most f_i from node s to node u_k (as when $\alpha f_i \geq 2$, if there are several such paths, the agent chooses the lexicographically smallest shortest among them). Once this is done, the interruption is over: the agent resumes S_i and makes the kth edge traversal to reach node u_{k+1}. We can easily show that at the end of this edge traversal, and thus at the beginning of the next interruption if any, property $H(k + 1)$ is satisfied.

So, assume that at the beginning of the kth interruption, the agent does not know a path of length at most f_i from node s to node u_k. In view of the fact that $D_k \leq f_i + 1$, the shortest path from node s to node u_k that is known by the agent has actually length exactly $f_i + 1$. Before explaining what the agent does, let us give some properties that necessarily hold in this situation. We have the following claim, whose proof is omitted.

▷ Claim 9. e_k belongs to G_{i-1} or to the sequence $(e_1, e_2, \ldots, e_{k-1})$.

From the above claim, it follows that at the beginning of the kth interruption the agent has already traversed edge e_k before, and already knows which edge of G_{i-1} or of $(e_1, e_2, \ldots, e_{k-1})$ corresponds to it. Thus, at the beginning of the kth interruption, the agent can already determine a path of length at most $f_i + 1$ from node s to node u_{k+1} because in view of Lemma 4 it must know such a path when reaching node u_{k+1} and because the traversal of e_k does not bring extra topological information on $\mathcal{G}$.

Now we are are able to formulate what the agent does, when it has noticed that it does not know a path of length at most f_i from node s to node u_k. It executes in the reverse order the last D_k edge traversals, at the end of which it is at node s and Sk is empty. Then, instead of coming back to u_k, it goes directly to node u_{k+1} using the known path (highlighted

in the previous paragraph) of length at most $f_i + 1$ from node s to node u_{k+1}. Once this is done, the interruption is over: the agent resumes S_i and acts as if it had just traversed edge e_k (as previously mentioned, it just performs some computations before interrupting again S_i for the next edge traversal, if any). It follows that at the end of the interruption, and thus at the beginning of the following one if any, $H(k+1)$ is satisfied. We have shown by induction on k that, at the beginning of the kth interruption, for any $k \geq 1$, the property $H(k)$ is satisfied. This closes the description of the interruptions.

It remains to deal with the close period. At the beginning of it, property $H(c_i + 1)$ is satisfied, which implies that the agent has performed $D_{c_i+1} \leq f_i + 1$ edge traversals since the last time when Sk was empty. Hence, during the close period, the agent simply executes in the reverse order the last D_{c_i+1} edge traversals, at the end of which Sk is empty. In view of this, of the fact that Sk is empty at the beginning of S'_i, and of property $H(c_i + 1)$, the frequent return property is satisfied during S'_i.

It follows from the above explanation that at the end of S'_i, G_i is entirely explored and the agent is at the node containing the treasure if $i = |S|$. If $i < |S|$, it also follows that the size of Sk is 0 at the beginning of the next emulation S'_{i+1}, and $M_1(S'_{i+1}) = M_1(S_{i+1}) = B_{f_{i+1}}(\mathcal{G}, s)$ because $M_2(S'_i) = M_2(S_i) = M_1(S_{i+1})$. Finally, concerning the cost of S'_i, observe that the number of interruptions is c_i and during the close period as well as during each interruption the agent makes at most $2(f_i + 1)$ edge traversals. The cost of S'_i is then upper bounded by $2(f_i + 1)c_i + 2f_i + 2$ which is at most $2(\frac{2}{\alpha} + 1)c_i + \frac{4}{\alpha} + 2$, as $f_i < \frac{2}{\alpha}$ in the currently analysed case. Hence, the cost of S'_i is $\mathcal{O}(c_i)$. This concludes the second case and thus concludes the proof of the theorem. ◀

References

1. Steve Alpern and Shmuel Gal. *The Theory of Search Games and Rendezvous*. Kluwer Academic Publisher, 2003. doi:10.1007/b100809.
2. Spyros Angelopoulos, Diogo Arsénio, and Christoph Dürr. Infinite linear programming and online searching with turn cost. *Theor. Comput. Sci.*, 670:11–22, 2017. doi:10.1016/j.tcs.2017.01.013.
3. Baruch Awerbuch, Margrit Betke, Ronald L. Rivest, and Mona Singh. Piecemeal graph exploration by a mobile robot. *Inf. Comput.*, 152(2):155–172, 1999. doi:10.1006/inco.1999.2795.
4. Ricardo A. Baeza-Yates, Joseph C. Culberson, and Gregory J. E. Rawlins. Searching in the plane. *Inf. Comput.*, 106(2):234–252, 1993. doi:10.1006/inco.1993.1054.
5. Anatole Beck. On the linear search problem. *Israel Journal of Mathematics*, 2:221–228, 1964. doi:10.1007/BF02759737.
6. Anatole Beck and Donald J. Newman. Yet more on the linear search problem. *Israel Journal of Mathematics*, 8:419–429, 1970. doi:10.1007/BF02798690.
7. Richard E. Bellman. An optimal search. *SIAM Review*, 5(3):274, 1963. doi:10.1137/1005070.
8. Pallab Dasgupta, P. P. Chakrabarti, and S. C. De Sarkar. Agent searching in a tree and the optimality of iterative deepening. *Artif. Intell.*, 71(1):195–208, 1994. doi:10.1016/0004-3702(94)90066-3.
9. Pallab Dasgupta, P. P. Chakrabarti, and S. C. De Sarkar. A correction to "agent searching in a tree and the optimality of iterative deepening". *Artif. Intell.*, 77(1):173–176, 1995. doi:10.1016/0004-3702(95)00089-W.
10. Erik D. Demaine, Sándor P. Fekete, and Shmuel Gal. Online searching with turn cost. *Theor. Comput. Sci.*, 361(2-3):342–355, 2006. doi:10.1016/j.tcs.2006.05.018.
11. Christian A. Duncan, Stephen G. Kobourov, and V. S. Anil Kumar. Optimal constrained graph exploration. *ACM Trans. Algorithms*, 2(3):380–402, 2006. doi:10.1145/1159892.1159897.

12 Rudolf Fleischer, Thomas Kamphans, Rolf Klein, Elmar Langetepe, and Gerhard Trippen. Competitive online approximation of the optimal search ratio. *SIAM J. Comput.*, 38(3):881–898, 2008. doi:10.1137/060662204.

13 G. Matthew Fricke, Joshua P. Hecker, Antonio D. Griego, Linh T. Tran, and Melanie E. Moses. A distributed deterministic spiral search algorithm for swarms. In *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS 2016, Daejeon, South Korea, October 9-14, 2016*, pages 4430–4436. IEEE, 2016. doi:10.1109/IROS.2016.7759652.

14 Shmuel Gal. Search games: A review. In *Search Theory: A Game Theoretic Perspective*, pages 3–15. Springer, 2013. doi:10.1007/978-1-4614-6825-7_1.

15 Subir Kumar Ghosh and Rolf Klein. Online algorithms for searching and exploration in the plane. *Comput. Sci. Rev.*, 4(4):189–201, 2010. doi:10.1016/j.cosrev.2010.05.001.

16 Artur Jez and Jakub Lopuszanski. On the two-dimensional cow search problem. *Inf. Process. Lett.*, 109(11):543–547, 2009. doi:10.1016/j.ipl.2009.01.020.

17 Ming-Yang Kao, John H. Reif, and Stephen R. Tate. Searching in an unknown environment: An optimal randomized algorithm for the cow-path problem. *Inf. Comput.*, 131(1):63–79, 1996. doi:10.1006/inco.1996.0092.

18 David G. Kirkpatrick and Sandra Zilles. Competitive search in symmetric trees. In Frank Dehne, John Iacono, and Jörg-Rüdiger Sack, editors, *Algorithms and Data Structures - 12th International Symposium, WADS 2011, New York, NY, USA, August 15-17, 2011. Proceedings*, volume 6844 of *Lecture Notes in Computer Science*, pages 560–570. Springer, 2011. doi:10.1007/978-3-642-22300-6_47.

19 Dennis Komm, Rastislav Královic, Richard Královic, and Jasmin Smula. Treasure hunt with advice. In Christian Scheideler, editor, *Structural Information and Communication Complexity - 22nd International Colloquium, SIROCCO 2015, Montserrat, Spain, July 14-16, 2015, Post-Proceedings*, volume 9439 of *Lecture Notes in Computer Science*, pages 328–341. Springer, 2015. doi:10.1007/978-3-319-25258-2_23.

20 Elmar Langetepe. On the optimality of spiral search. In Moses Charikar, editor, *Proceedings of the Twenty-First Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2010, Austin, Texas, USA, January 17-19, 2010*, pages 1–12. SIAM, 2010. doi:10.1137/1.9781611973075.1.

21 Elmar Langetepe. Searching for an axis-parallel shoreline. *Theor. Comput. Sci.*, 447:85–99, 2012. doi:10.1016/j.tcs.2011.12.069.

22 Alejandro López-Ortiz and Sven Schuierer. The ultimate strategy to search on m rays? *Theor. Comput. Sci.*, 261(2):267–295, 2001. doi:10.1016/S0304-3975(00)00144-4.

23 Avery Miller and Andrzej Pelc. Tradeoffs between cost and information for rendezvous and treasure hunt. *J. Parallel Distributed Comput.*, 83:159–167, 2015. doi:10.1016/j.jpdc.2015.06.004.

24 Andrzej Pelc. Reaching a target in the plane with no information. *Inf. Process. Lett.*, 140:13–17, 2018. doi:10.1016/j.ipl.2018.04.006.

25 Sven Schuierer. Lower bounds in on-line geometric searching. *Comput. Geom.*, 18(1):37–53, 2001. doi:10.1016/S0925-7721(00)00030-4.

26 Amnon Ta-Shma and Uri Zwick. Deterministic rendezvous, treasure hunts, and strongly universal exploration sequences. *ACM Transactions on Algorithms*, 10(3):12, 2014. doi:10.1145/2601068.

Conditional Dichotomy of Boolean Ordered Promise CSPs

Joshua Brakensiek
Computer Science Department, Stanford University, CA, USA

Venkatesan Guruswami
Computer Science Department, Carnegie Mellon University, Pittsburgh, PA, USA

Sai Sandeep
Computer Science Department, Carnegie Mellon University, Pittsburgh, PA, USA

Abstract

Promise Constraint Satisfaction Problems (PCSPs) are a generalization of Constraint Satisfaction Problems (CSPs) where each predicate has a strong and a weak form and given a CSP instance, the objective is to distinguish if the strong form can be satisfied vs. even the weak form cannot be satisfied. Since their formal introduction by Austrin, Guruswami, and Håstad [1], there has been a flurry of works on PCSPs, including recent breakthroughs in approximate graph coloring [4, 24, 35]. The key tool in studying PCSPs is the algebraic framework developed in the context of CSPs where the closure properties of the satisfying solutions known as *polymorphisms* are analyzed.

The polymorphisms of PCSPs are significantly richer than CSPs – even in the Boolean case, we still do not know if there exists a dichotomy result for PCSPs analogous to Schaefer's dichotomy result [32] for CSPs. In this paper, we study a special case of Boolean PCSPs, namely Boolean *Ordered* PCSPs where the Boolean PCSPs have the predicate $x \leq y$. In the algebraic framework, this is the special case of Boolean PCSPs when the polymorphisms are *monotone functions*. We prove that Boolean Ordered PCSPs exhibit a computational dichotomy assuming the Rich 2-to-1 Conjecture [9] which is a perfect completeness surrogate of the Unique Games Conjecture.

In particular, assuming the Rich 2-to-1 Conjecture, we prove that a Boolean Ordered PCSP can be solved in polynomial time if for every $\epsilon > 0$, it has polymorphisms where each coordinate has *Shapley value* at most ϵ, else it is NP-hard. The algorithmic part of our dichotomy result is based on a structural lemma showing that Boolean monotone functions with each coordinate having low Shapley value have arbitrarily large threshold functions as minors. The hardness part proceeds by showing that the Shapley value is consistent under a uniformly random 2-to-1 minor. As a structural result of independent interest, we construct an example to show that the Shapley value can be inconsistent under an adversarial 2-to-1 minor.

2012 ACM Subject Classification Theory of computation → Constraint and logic programming; Theory of computation → Approximation algorithms analysis; Theory of computation → Problems, reductions and completeness; Applied computing → Economics

Keywords and phrases promise constraint satisfaction, Boolean ordered PCSP, Shapley value, rich 2-to-1 conjecture, random minor

Digital Object Identifier 10.4230/LIPIcs.ICALP.2021.37

Category Track A: Algorithms, Complexity and Games

Related Version *Full Version*: `https://arxiv.org/abs/2102.11854` [7]

Funding *Joshua Brakensiek*: Research supported in part by an NSF Graduate Research Fellowship.
Venkatesan Guruswami: Research supported in part by NSF grant CCF-1908125 and a Simons Investigator award.
Sai Sandeep: Research supported in part by NSF grants CCF-1563742 and CCF-1908125.

Acknowledgements We thank Libor Barto, whose talk [2] and insightful discussions inspired our work.

1 Introduction

Constraint satisfaction problems (CSP) have played a very influential role in the theory of computation, providing an excellent testbed for the development of both algorithmic and hardness techniques, which then extend to more general settings. A CSP over domain D is specified by a finite collection $\mathbb{A}$ of predicates over D, and is denoted as CSP($\mathbb{A}$). Given an input containing n variables with constraints on the variables using these predicates, the objective is to identify if we can assign values from D to the variables that satisfies all the constraints. Examples of CSPs include classical problems such as 3-SAT and 3-Coloring of graphs.

When the domain is Boolean, Schaefer [32] proved that every CSP is either in P or is NP-Complete. Feder and Vardi [15] conjectured that the same should hold over arbitrary domains as well. They also showed that the then known algorithmic results all follow by the algebraic closure properties of the CSPs. This notion was formalized by Jeavons, Cohen, and Gyssens [17, 18] and other works [11] that crystallized the *(universal) algebraic approach* to CSPs. In the algebraic approach, the higher-order closure properties obeyed by the predicates, namely their *polymorphisms*, are studied. A polymorphism is a function that when applied coordinate-wise to arbitrary satisfying assignments to the predicate, is guaranteed to produce an output that satisfies the predicate. For example, consider an arbitrary instance I of the 2-SAT problem over n variables, and suppose that $\mathbf{x}, \mathbf{y}, \mathbf{z} \in \{0,1\}^n$ are three assignments that satisfy all the constraints in I. Now, if we compute $\mathbf{u} \in \{0,1\}^n$ that is obtained by setting $u_i = \mathsf{MAJ}(x_i, y_i, z_i)$ for all $i \in [n]$, the assignment $\mathbf{u}$ also satisfies all the constraints of I. Thus, the majority function on 3 bits is a polymorphism of the 2-SAT CSP. On the other hand, for the 3-SAT problem, it is not hard to prove that the only polymorphisms are the dictator functions. The algebraic approach has been immensely successful and culminated in the recent resolution of Feder-Vardi conjecture by Bulatov [10] and Zhuk [36]. Further, these proofs yield a precise understanding of the mathematical structure underlying efficient algorithms: if the CSP has a "non-trivial" polymorphisms, the CSP is polytime solvable, and otherwise, it is NP-complete.

In this paper, we study Promise Constraint Satisfaction Problems (PCSPs) that vastly generalize the CSPs. In the PCSPs, each predicate has a weak and a strong form–given an instance of PCSP containing n variables with the constraints, the goal is to distinguish between the case that the stronger form can be satisfied vs. even the weaker one cannot be satisfied. A classical example of PCSP is the approximate graph coloring, where given a graph G, the goal is to distinguish between the cases that G can be colored with c colors vs. it cannot be colored with s colors for some $c \leq s$. Another example is the (1-in-3 SAT, NAE-3-SAT), wherein given a 1-in-3-SAT instance that is promised to be satisfiable, the objective is to assign $0, 1$ values to the variables such that each constraint is satisfied as a NAE-3-SAT instance, i.e., both 0 and 1 occur in every constraint. While the individual CSPs, namely 1-in-3-SAT and NAE-3-SAT are both NP-hard, the above PCSP is in P. The study of PCSPs was formally initiated by Austrin, Guruswami, and Håstad [1]. and since then, there has been a lot of recent interest in PCSPs, including the development of a systematic theory in [4, 6] and leading to breakthroughs in approximate graph coloring [4, 24, 35].

The central question in the study of PCSPs is whether there exists a complexity dichotomy for PCSPs i.e. if every PCSP is either in P or is NP-complete. As is the case with CSPs, the key tool towards establishing such a dichotomy result is the algebraic approach. The Galois correspondence from the CSP world extends to PCSPs, i.e., the polymorphisms fully capture the computational complexity of the underlying PCSP [6, 30]. This has been extended to show

that just the identities satisfied by the polymorphisms suffice to capture the computational complexity of the underlying PCSP [4]. However, the polymorphisms of PCSPs are much richer, and characterizing which polymorphisms lead to algorithms and which ones lead to hardness has been a challenging problem. Conceptually, the principal difficulty is that the polymorphisms for CSPs are closed under composition (hence referred to as *clones*), whereas for PCSPs, this is no longer the case.

As a result, even in the Boolean case, we do not have a dichotomy theorem for PCSPs. Towards establishing a potential Boolean PCSP dichotomy, progress has been made by Ficak, Kozik, Olsák and Stankiewicz [16], who obtained a dichotomy result when each predicate is symmetric. In this paper, we study Boolean PCSPs that contain the *simplest non-symmetric predicate*, $x \to y$. We call such Boolean PCSPs *Ordered* as we can also view the implication constraint as an ordering requirement $x \leq y$.

Ordered Boolean PCSPs have come under recent study. The work of Petr [29] (inspired by work of Barto [2, 3]) considered a special class of Ordered Boolean PCSPs which have an additional predicate $x \neq y$ (this corresponds to allowing negations in the constraints) as well as the requirement that the majority on three bits is *not* a polymorphism. In this setting Petr was able to show that such Ordered Boolean PCSPs are NP-hard. However, the approach considered does not seem immediately extendable to analyzing general Ordered Boolean PCSPs [3].

The main motivation for studying these PCSPs comes from the fact that adding the additional $x \leq y$ predicate is equivalent to restricting the polymorphisms of the PCSPs to be *monotone functions*. Monotonicity is an influential theme in the study of Boolean functions and complexity theory, and understanding the structure of polymorphisms in the monotone case is an important (and certainly necessary) subcase towards a general characterization of polymorphisms vs. tractability for arbitrary Boolean PCSPs. For the special case of Boolean Ordered PCSPs which include negation constraints, it was conjectured in [3] that polynomial time tractability is characterized by the existence of majority polymorphisms of arbitrarily large arity.

Our main result is that Boolean Ordered PCSPs exhibit a dichotomy, under the recently introduced *Rich 2-to-1 Conjecture* of Braverman, Khot, and Minzer [9].

▶ **Theorem 1.** *Assuming the Rich 2-to-1 Conjecture, every Ordered Boolean PCSP is either in P or is NP-Complete. Furthermore, an Ordered PCSP Γ is in P if and only if for every $\epsilon > 0$, there are polymorphisms of Γ with every coordinate having Shapley value at most ϵ. Equivalently, Γ is in P if and only if it has threshold polymorphisms of arbitrarily large arity.*

As a concrete example, recall the earlier mentioned example of (1-in-3-SAT, NAE-3-SAT). As it has threshold polymorphisms of arbitrarily large arity, it remains polynomial time solvable even after adding the predicate $x \to y$. However, if we also add another two-variable predicate $x \neq y$, the PCSP no longer has threshold polymorphisms, and by our above result, it becomes NP-Complete.

We obtain the conditional dichotomy result by analyzing the polymorphisms of the Ordered PCSPs. The key idea in the algebraic approach to PCSPs is that the PCSP is tractable if the polymorphisms are close to symmetric, and the PCSP is hard if all the polymorphisms have a small number of "important" coordinates. More concretely, on the algorithmic front, it has been proved that symmetric polymorphisms of arbitrarily large arities lead to polynomial time algorithms for PCSPs [8]. On the hardness side, if all the polymorphisms depend on a bounded number of coordinates, then the underlying PCSP is NP-hard [1]. This has been extended to various other notions, including combinatorial ones

such as C-fixing [5], and topological ones such as having a bounded number of coordinates with non-zero winding number [24]. In this paper, we study the monotone polymorphisms using analytical techniques.

In particular, we use Shapley value to analyze the monotone polymorphisms. For a monotone function $f : \{0,1\}^n \to \{0,1\}$, Shapley value of a coordinate i is the probability that on a random path from $\{0,0,\ldots,0\}$ to $\{1,1,\ldots,1\}$, the function value turns from 0 to 1 when we switch the ith coordinate to 1. Initially studied to understand the power of an individual in voting systems [33], Shapley value has now found applications in various settings, especially in game theory [26, 27]. In our setting, there are two advantages of using Shapley value to study the polymorphisms. First, it is a relative measure of the importance of a coordinate, as opposed to other notions of Influence which are absolute. This helps in bounding the number of coordinates with Shapley value above a certain threshold. Second, it is a versatile measure with combinatorial and analytical interpretations [12] which helps in proving that Shapley value stays consistent under function minors[1], a key property necessary in both the algorithm and the hardness.

Algorithm Overview. We obtain our algorithmic result by using the Basic Linear Programming with Affine relaxation (BLP+Affine relaxation), combined with a structural result regarding the monotone functions with bounded Shapley value. As mentioned earlier, PCSPs with symmetric polymorphisms of arbitrarily large arities can be solved in polynomial time using the BLP+Affine relaxation algorithm [8]. Our main structural result is that Boolean functions with bounded Shapley value have arbitrarily large threshold functions as minors. Since the set of polymorphisms of a PCSP are closed under taking minors, this proves that the underlying PCSP Γ has arbitrarily large threshold functions as polymorphisms, which then implies that Γ is in P. The key tool underlying our structural result is a result of Kalai [19] that states that under certain conditions, monotone Boolean functions with arbitrarily small Shapley value have a sharp threshold.

Hardness Overview. We obtain our hardness result assuming the Rich 2-to-1 Conjecture. Braverman, Khot, and Minzer [9] introduced the conjecture as a perfect completeness surrogate of the well known Unique Games Conjecture [21]. They also proved that the conjecture is equivalent to Unique Games Conjecture when we relax the perfect completeness requirement. The reduction from the Rich 2-to-1 Conjecture to PCSPs follows using the standard Label Cover-Long Code paradigm. The key ingredient in this reduction is a decoding of the Long Codes to a bounded number of coordinates that is consistent under function minors. We decode each Long Code function to the coordinates with $\Omega(1)$ Shapley value – as the sum of Shapley values of all the coordinates of any monotone function is equal to 1, there is a bounded number of such coordinates. We argue about the consistency of this decoding using a structural result that states that under a uniformly random minor, Shapley value is roughly preserved.

On the necessity of "richness" in 2-to-1 Conjecture. A natural question is whether our hardness result can be obtained using a weaker assumption such as the 2-to-1 conjecture (whose imperfect completeness version was recently established [13, 14, 22, 23]). We shed some light on this question by showing that there are monotone Boolean functions $f : \{0,1\}^{2n} \to \{0,1\}$ and $g : \{0,1\}^n \to \{0,1\}$ such that g is a minor of f with respect to the

[1] A minor(formally defined in Section 2) of a function $f : \{0,1\}^m \to \{0,1\}$ is a function $g : \{0,1\}^n \to \{0,1\}$ of smaller arity $n \leq m$ obtained from f by identifying sets of variables together.

2-to-1 function π, both the functions f and g have exactly one coordinate i_1, i_2 respectively, with $\Omega(1)$ Shapley value, and yet $\pi(i_1) \neq i_2$. Such an adversarial example is interesting from two angles: first, it shows that even using the 2-to-1 conjecture, the Shapley value based decoding is not consistent. Second, it gives an example of agents pairing up maliciously to completely alter the Shapley value. The underlying phenomenon is that the rich 2-to-1 games have "subcode-covering" property, which is absent in the standard 2-to-1 games, helping in preserving the consistency of any biased influence measure such as the Shapley value.

Organization. In Section 2, we formally define PCSPs, polymorphisms, and Shapley value. We present the algorithmic and hardness parts of our dichotomy result in Section 3 and Section 4 respectively. We present the adversarial example of a 2-to-1 minor that alters the Shapley value in Section 5.

2 Preliminaries

Notations. We use $[n]$ to denote the set $\{1, 2, \ldots, n\}$. For a k-ary relation $A \subseteq [q]^k$, we abuse the notation and use A both as a subset of $[q]^k$, and also as a predicate $A : [q]^k \to \{0, 1\}$. For a vector $\mathbf{x} = (x_1, x_2, \ldots, x_n) \in \{0, 1\}^n$, we use $\mathsf{hw}(\mathbf{x})$ to denote $\sum_{i=1}^{n} x_i$. For two vectors $\mathbf{x}, \mathbf{y} \in \{0, 1\}^n$, we say that $\mathbf{x} \leq \mathbf{y}$ if $x_i \leq y_i$ for all $i \in [n]$. A Boolean function $f : \{0, 1\}^n \to \{0, 1\}$ is called monotone if $f(\mathbf{x}) \leq f(\mathbf{y})$ for all $\mathbf{x} \leq \mathbf{y}$.

PCSPs and Polymorphisms. We first define Constraint Satisfaction Problems (CSP).

▶ **Definition 2** (CSP). *Given a k-ary relation $A : D^k \to \{0, 1\}$ over a domain D, the Constraint Satisfaction Problem (CSP) associated with the predicate A takes a set of variables $V = \{v_1, v_2, \ldots, v_n\}$ as input which are to be assigned values from D. There are m constraints $(e_1, e_2, \ldots, e_m)$ each consisting of $e_i = ((e_i)_1, (e_i)_2, \ldots, (e_i)_k) \subseteq V^k$ that indicate that the corresponding assignment should belong to A. The objective is to identify if there is an assignment $V \to D$ that satisfies all the constraints.*

In general, we can have multiple relations $A_1, A_2, \ldots, A_l$, and different constraints can use different relations. We denote such a CSP by $CSP(A_1, A_2, \ldots, A_l)$.

We formally define Promise Constraint Satisfaction Problems (PCSP).

▶ **Definition 3** (PCSP). *In a Promise Constraint Satisfaction Problem $PCSP(\Gamma)$ over a pair of domains D_1, D_2, we have a set of pairs of relations $\Gamma = \{(A_1, B_1), (A_2, B_2), \ldots, (A_l, B_l)\}$ such that for every $i \in [l]$, A_i is a subset of $D_1^{k_i}$ and B_i is a subset of $D_2^{k_i}$. Furthermore, there is a homomorphism $h : D_1 \to D_2$ such that for all $i \in [l]$ and $x \in D_1^{k_i}$, $x \in A_i$ implies $h(x) \in B_i$. Given a $CSP(A_1, A_2, \ldots, A_l)$ instance, the objective is to distinguish between the two cases:*

1. *There is an assignment to the variables from D_1 that satisfies every constraint when viewed as $CSP(A_1, A_2, \ldots, A_l)$.*
2. *There is no assignment to the variables from D_2 that satisfies every constraint when viewed as $CSP(B_1, B_2, \ldots, B_l)$.*

We now define Boolean Ordered PCSPs.

▶ **Definition 4** (Boolean Ordered PCSP). *A PCSP $PCSP(\Gamma)$ over a pair of domains D_1, D_2 with the set of pairs of relations $\Gamma = \{(A_1, B_1), (A_2, B_2), \ldots, (A_l, B_l)\}$ is said to be Boolean Ordered if the following hold.*

1. *The domains are both Boolean i.e., $D_1 = D_2 = \{0, 1\}$.*
2. *There exists $i \in [l]$ such that $A_i = B_i = \{(0, 0), (0, 1), (1, 1)\}$.*

Associated with every PCSP, there are polymorphisms that capture the closure properties of the satisfying solutions to the PCSP. More formally, we can define polymorphisms of a PCSP as follows.

▶ **Definition 5** (Polymorphisms). *For $PCSP(\Gamma)$ with $\Gamma = \{((A_1, B_1), (A_2, B_2), \ldots, (A_l, B_l))\}$ where for every $i \in [l]$, $A_i : [q_1]^{k_i} \to \{0,1\}, B_i : [q_2]^{k_i} \to \{0,1\}$, a polymorphism of arity n is a function $f : [q_1]^n \to [q_2]$ that satisfies the below property for all $i \in [l]$. For all $(\boldsymbol{v}_1, \boldsymbol{v}_2, \ldots, \boldsymbol{v}_{k_i})$ such that for all $j \in [n], ((\boldsymbol{v}_1)_j, (\boldsymbol{v}_2)_j, \ldots, (\boldsymbol{v}_{k_i})_j) \in A_i$, we have*

$$(f(\boldsymbol{v}_1), f(\boldsymbol{v}_2), \ldots, f(\boldsymbol{v}_{k_i})) \in B_i$$

We use $\mathrm{Pol}(\Gamma)$ *to denote the family of all the polymorphisms of $PCSP(\Gamma)$.*

A crucial property satisfied by $\mathrm{Pol}(\Gamma)$ is that the family of functions is closed under taking minors. We first define the minor of a function formally.

▶ **Definition 6** (Minor of a function). *For a Boolean function $f : [q]^n \to [q']$, the function $g : [q]^m \to [q']$ is said to be a minor of f with respect to the function $\pi : [n] \to [m]$ if*

$$g(x_1, x_2, \ldots, x_m) = f(x_{\pi(1)}, x_{\pi(2)}, \ldots, x_{\pi(n)}) \, \forall x_1, x_2, \ldots, x_m \in [q]$$

We say that a function g is a minor of f if there exists some π such that g is a minor of f with respect to π.

We are often interested in 2-to-1 minors. A function g is said to be a 2-to-1 minor of f if there exists a 2-to-1 function π such that g is a minor of f with respect to π, where 2-to-1 function is defined below.

▶ **Definition 7** (2-to-1 function). *A function $\pi : [2n] \to [n]$ is said to be a 2-to-1 function if*

$$|\pi^{-1}(i)| = 2 \ \forall i \in [n]$$

We use $\mathcal{F}_{2\to1}(n)$ to denote the set of all the 2-to-1 functions from $[2n]$ to $[n]$.

By the definition of the polymorphisms, we can infer that if $f \in \mathrm{Pol}(\Gamma)$ for a PCSP Γ, then for all functions g such that g is a minor of f, we have $g \in \mathrm{Pol}(\Gamma)$. Such a family of functions that is closed under taking minors is called as a *minion*. We often refer to the family of polymorphisms of a PCSP as the polymorphism minion.

We refer the reader to [4] for an extensive introduction to PCSPs and polymorphisms.

Shapley value. Let $f : \{0,1\}^n \to \{0,1\}$ be a monotone Boolean function. We can view the monotone Boolean function f as a voting scheme between two parties, and n agents: the winner of the voting scheme when the ith agent votes for $\mathbf{x}_i \in \{0,1\}$ is $f(\mathbf{x})$. The relative power of an agent in a voting scheme is typically measured using the Shapley-Shubix Index, also known as Shapley Value.

Informally speaking, the Shapley Value of a coordinate i is the probability that the ith agent is the altering vote when we start with all zeroes and flip the votes in a uniformly random order. More formally,

▶ **Definition 8** (Shapley value). *Let $f : \{0,1\}^n \to \{0,1\}$ be a monotone Boolean function. Let $\sigma \in S_n$ be a uniformly random permutation of $[n]$. For an integer $j \in [n]$, let P_j denote the set of first j elements of σ i.e., $P_j := \{\sigma(1), \sigma(2), \ldots, \sigma(j)\}$. The Shapley value $\Phi_f(i)$ of the coordinate $i \in [n]$ is defined as*

$$\Phi_f(i) := Pr_\sigma \{\exists j \in [n] : \sigma(j) = i, f(P_{j-1}) = 0, f(P_j) = 1\}$$

We also give an alternate definition of Shapley value using the notion of boundary of a coordinate. For a monotone Boolean function $f : \{0,1\}^n \to \{0,1\}$ and coordinate $i \in [n]$, let $\mathcal{B}_f(i)$ denote the boundary of the coordinate i i.e.

$$\mathcal{B}_f(i) := \{S \subseteq [n] \setminus \{i\} : f(\{i\} \cup S) = 1, f(S) = 0\}$$

By the monotonicity of f, we can infer that $\mathcal{B}_f(i)$ satisfies the following sandwich property that will be useful later.

▶ **Proposition 9.** *Let $f : \{0,1\}^n \to \{0,1\}$ be a monotone Boolean function and let $i \in [n]$. Then, for every pair of sets $S_1, S_2 \in \mathcal{B}_f(i)$ with $S_1 \subseteq S_2$, we have $S \in \mathcal{B}_f(i)$ for all S such that $S_1 \subseteq S \subseteq S_2$.*

Proof. By the monotonicity of f, we have $f(S \cup \{i\}) \geq f(S_1 \cup \{i\}) = 1$, and thus, $f(S \cup \{i\}) = 1$. Similarly, we have $f(S) \leq f(S_2) = 0$, and thus, $f(S) = 0$. ◀

For an index $j \in \{0, 1, \ldots, n-1\}$, let $\mu_f(j)^{(i)}$ denote the fraction of subsets of $[n]$ of size j that are in $\mathcal{B}_f(i)$ i.e.

$$\mu_f(j)^{(i)} := \left|\mathcal{B}_f(i) \cap \tbinom{[n]}{j}\right| \Big/ \tbinom{n}{j}.$$

We can rewrite the definition of Shapley value of the ith coordinate as the following [34]:

$$\Phi_f(i) = \frac{\sum_{j=0}^{n-1} \mu_f(j)^{(i)}}{n}. \tag{1}$$

3 Algorithm when Shapley values are small

In this section, we show that monotone Boolean functions where each coordinate has bounded Shapley value has arbitrarily large threshold functions as minors, thereby proving the algorithmic part of our dichotomy result.

Let L be a positive integer and $0 \leq \tau \leq L$ be a non-negative integer. We let $\mathsf{THR}_{L,\tau} : \{0,1\}^L \to \{0,1\}$ be the threshold function on L variables with threshold τ. More formally,

$$\mathsf{THR}_{L,\tau}(\mathbf{x}) := \begin{cases} 1 \text{ if } \mathsf{hw}(\mathbf{x}) \geq \tau \\ 0 \text{ otherwise.} \end{cases}$$

For a monotone Boolean function $f : \{0,1\}^n \to \{0,1\}$ and real number $p \in [0,1]$, let $P_p(f)$ denote the expected value of $f(x)$ where each element $x_i, i \in [n]$ is independently set to be 1 with probability p and 0 with probability $1-p$. For every monotone function f, the function $P_p(f)$ is a strictly monotone continuous function in p on the interval $[0,1]$. The value $p_c = p_c(f)$ at which $P_{p_c}(f) = \frac{1}{2}$ is called the *critical probability* of f.

Using the Russo-Margulis Lemma [25, 31] and Poincaré Inequality, we can show the following lemma that we need later.

▶ **Lemma 10** (Exercise 8.29(e) in [28]). *Let f be a non-constant monotone Boolean function with critical probability $p_c \leq \frac{1}{2}$. Let $p_1 := \frac{1}{(2\nu)^2} p_c$ for $\nu > 0$. If $p_1 \leq \frac{1}{2}$, then $P_{p_1}(f) \geq 1 - \nu$.*

We now define the threshold interval of f.

▶ **Definition 11.** *For a monotone function f and $0 < \epsilon < \frac{1}{2}$, we define $T_\epsilon(f) := p_2 - p_1$, where p_2 and p_1 are such that $P_{p_1}(f) = \epsilon, P_{p_2}(f) = 1 - \epsilon$.*

Kalai [19] proved the following result regarding monotone Boolean functions.

▶ **Theorem 12.** *For every $a, \epsilon, \gamma > 0$, there exists $\delta := \delta(a, \epsilon, \gamma) > 0$ such that for every monotone Boolean function $f : \{0,1\}^n \to \{0,1\}$ with $\Phi_f(i) \leq \delta$ for all $i \in [n]$ and $a \leq p_c(f) \leq 1 - a$, then $T_\epsilon(f) \leq \gamma$.*

We will use this result to show that for every monotone function where each coordinate has bounded Shapley value has arbitrarily large threshold functions as minor.

▶ **Lemma 13.** *For every $L \geq 2$, there exists a $\delta := \delta(L) > 0$ such that the following holds. For any monotone Boolean function $f : \{0,1\}^n \to \{0,1\}$ with*

$$\Phi_f(i) \leq \delta \ \ \forall i \in [n]$$

there exists a positive integer $L' \in \{L, L+1\}$ and a non-negative integer τ such that $\mathsf{THR}_{L',\tau}$ is a minor of f.

Proof. We obtain $\delta := \delta(L) > 0$ from Theorem 12 by setting $\epsilon = \frac{1}{2^{L+1}}, \gamma = a = \frac{1}{L^3}$. Our goal is to show that for this parameter δ, for every monotone Boolean function f with each coordinate having Shapley value at most δ, there exists $L' \in \{L, L+1\}$ and τ such that $\mathsf{THR}_{L',\tau}$ is a minor of f.

We assume that f is a non-constant function, else we have a trivial minor by setting $\tau = 0$ or $\tau = L'$. Let p_c be the critical probability of f.

Case 1: $p_c < a = \frac{1}{L^3}$. Let $p_1 = L^2 p_c < \frac{1}{L}$. Using Lemma 10, we can conclude that $P_{p_1}(f) \geq 1 - \frac{1}{2L}$. As $P_p(f)$ is monotone, we get that $P_{\frac{1}{L}}(f) > 1 - \frac{1}{2L}$. We let $g : \{0,1\}^L \to \{0,1\}$ be a uniformly random minor of f i.e. we choose the function $\pi : [n] \to [L]$ by choosing each value $\pi(i)$ uniformly and independently at random from $[L]$, and we let g to be the minor of f with respect to π.

Note that for every $i \in [L]$, the distribution of $g(\{i\})$ over the random minor g is the same as sampling a random input to f where we set each bit to 1 with probability $\frac{1}{L}$. As $P_{\frac{1}{L}}(f) \geq 1 - \frac{1}{2L}$, we get that for each $i \in [L]$, $g(\{i\}) = 1$ with probability at least $1 - \frac{1}{2L}$. By union bound, with probability at least $\frac{1}{2}$, $g(\{i\}) = 1$ for all $i \in [L]$. As $f(0, 0, \ldots, 0) = 0$, $g(\phi) = 0$ as well. Thus, with probability at least $\frac{1}{2}$, $g = \mathsf{THR}_{L,1}$. Hence, $\mathsf{THR}_{L,1}$ is a minor of f.

Case 2: $p_c > 1 - a = 1 - \frac{1}{L^3}$. Let $f^\dagger$ be the Boolean dual of f defined as $f^\dagger(x) = 1 - f(\overline{x})$. Note that $P_p(f^\dagger) = 1 - P_{1-p}(f)$ for all $p \in [0,1]$. Thus, $p_c(f^\dagger) = 1 - p_c < a$. Using the previous case, we can infer that $\mathsf{THR}_{L,1}$ is a minor of $f^\dagger$ with respect to a funtion $\pi : [n] \to [L]$. The same function π proves that $\mathsf{THR}^\dagger_{L,1} = \mathsf{THR}_{L,L}$ is a minor of f.

Case 3: $a \leq p_c \leq 1 - a$. Using Theorem 12, we obtain p_1 such that $P_{p_1}(f) \leq \epsilon$, and $P_{p_1+\gamma} \geq 1 - \epsilon$, where $\epsilon = \frac{1}{2^{L+1}}, \gamma = \frac{1}{L^3}$. As $\gamma < \frac{1}{L(L+1)}$, there exists $L' \in \{L, L+1\}$ and $\tau \in [L']$ such that $p_1 + \gamma < \frac{\tau}{L'}$ and $p_1 > \frac{\tau-1}{L'}$. Thus, we get that $P_{\frac{\tau}{L'}}(f) > 1 - \epsilon$ and $P_{\frac{\tau-1}{L'}} < \epsilon$. Let $g : \{0,1\}^{L'} \to \{0,1\}$ be a uniformly random minor of f i.e. we choose $\pi : [n] \to [L']$ by setting each value uniformly and independently at random from $[L']$ and set g to be the minor of f with respect to π. For a vector $\mathbf{x} \in \{0,1\}^{L'}$ with $\mathsf{hw}(\mathbf{x}) = \tau$, with probability greater than $1 - \frac{1}{2^{L+1}}$, $g(\mathbf{x}) = 1$. Similarly, for $\mathbf{x} \in \{0,1\}^{L'}$ with $\mathsf{hw}(\mathbf{x}) = \tau - 1$, with probability greater than $1 - \frac{1}{2^{L+1}}$, $g(\mathbf{x}) = 0$. Thus, with non-zero probability, $g(\mathbf{x}) = 1$ for all $x \in \{0,1\}^{L'}$ with $\mathsf{hw}(\mathbf{x}) = \tau$ and $g(\mathbf{x}) = 0$ for all $\mathbf{x} \in \{0,1\}^{L'}$ with $\mathsf{hw}(\mathbf{x}) = \tau - 1$. In other words, with non-zero probability, g is equal to $\mathsf{THR}_{L',\tau}$. Thus, $\mathsf{THR}_{L',\tau}$ is a minor of f. ◀

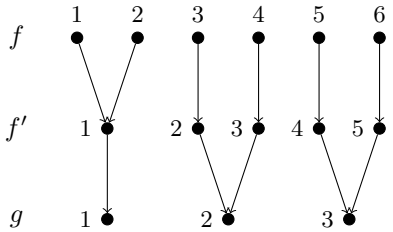


Figure 1 An illustration of the two step minor approach: Here $f : \{0,1\}^6 \to \{0,1\}$ is a Boolean function, $f' : \{0,1\}^5 \to \{0,1\}$ is a minor of f with respect to the function $\pi_1 : [6] \to [5]$ with $\pi_1(i) = \max(i-1, 1)$, and g is a minor of f' with respect to the function $\pi_2 : [5] \to [3]$ with $\pi_2(i) = \lceil \frac{i+1}{2} \rceil$.

Using the existence of arbitrarily large arity threshold minors, the algorithmic part of our Dichotomy result follows immediately.

▶ **Theorem 14.** *Let Γ be a Promise CSP template. Suppose that for every $\epsilon > 0$, there exists a function $f \in \mathrm{Pol}(\Gamma), f : \{0,1\}^n \to \{0,1\}$ such that $\Phi_i(f) \leq \epsilon$ for all $i \in [n]$. Then,* $\mathrm{PCSP}(\Gamma) \in \mathsf{P}$.

Proof. Using Lemma 13, we can conclude that there are infinitely many positive integers L such that there exists $\tau \in \{0, 1, \ldots, L\}$ with $\mathsf{THR}_{L,\tau} \in \mathrm{Pol}(\Gamma)$. As the threshold functions are symmetric[2], $\mathrm{Pol}(\Gamma)$ has symmetric polymorphisms of infinitely many arities. Thus, using the BLP+Affine algorithm of [8], $\mathrm{PCSP}(\Gamma)$ can be solved in polynomial time. ◀

We remark that the above result is inspired by a special case shown by Barto [2] that a Boolean Ordered PCSP is polytime tractable if it has cyclic polymorphisms of arbitrarily large arities.

4 Hardness Assuming Rich 2-to-1 Conjecture

In this section, we prove the hardness part of our dichotomy result. First, we prove that Shapley value is preserved under uniformly random 2-to-1 minors, and then we use this to show the hardness assuming the Rich 2-to-1 Conjecture.

4.1 Shapley value under random 2-to-1 minor

Let $f : \{0,1\}^{2n} \to \{0,1\}$ be a monotone Boolean function with $\Phi_f(1) \geq \lambda$ for some absolute constant $\lambda > 0$. Let $g : \{0,1\}^n \to \{0,1\}$ be a minor of f with respect to the uniformly random 2-to-1 function $\pi : [2n] \to [n]$. Our goal in this subsection is to show that $\mathbb{E}_\pi[\Phi_g(\pi(1))] \geq \gamma$ for some function $\gamma := \gamma(\lambda) > 0$. We prove this in two steps. (See Figure 1)

1. First, we consider the minor of f, $f' : \{0,1\}^{2n-1} \to \{0,1\}$ obtained with respect to $\pi_1 : [2n] \to [2n-1]$ where $\pi_1(1) = \pi_1(2) = 1, \pi_1(i) = i - 1 \, \forall i \in \{3, 4, \ldots, 2n\}$. We show that $\Phi_{f'}(1) \geq \frac{\lambda}{2}$.
2. Next, we consider a minor g of f' obtained with respect to the function $\pi_2 : [2n-1] \to [n]$ which has $\pi_2(1) = 1$ while the rest $2n-2$ values are chosen using a uniformly random partition of $[2n-2]$ into $n-1$ pairs. We show that $\mathbb{E}_{\pi_2}[\Phi_g(1)] \geq \gamma$ for some function $\gamma := \gamma(\lambda) > 0$.

[2] A function $f : \{0,1\}^n \to \{0,1\}$ is said to be symmetric if it is unchanged by any permutation of the input variables.

Note that the process of first taking the f' minor and then obtaining g by partitioning $[2n-2]$ into $n-1$ uniformly random pairs is equivalent to taking a uniformly random 2-to-1 minor of f. Thus, the two steps together prove the required Shapley value property of the uniformly random 2-to-1 minor.

The first step is captured by the following pair of lemmas, which we prove in the full version.

▶ **Lemma 15.** *Let $f : \{0,1\}^{2n} \to \{0,1\}$ and $f' : \{0,1\}^{2n-1} \to \{0,1\}$ be monotone Boolean functions such that f' is a minor of f with respect to the function $\pi_1 : [2n] \to [2n-1]$ defined as $\pi_1(i) = \max(i-1, 1)$. If $\Phi_f(1) \geq \lambda$, then $\Phi_{f'}(1) \geq \frac{\lambda}{2}$.*

▶ **Lemma 16.** *Let $f' : \{0,1\}^{2n-1} \to \{0,1\}$ be a monotone Boolean function such that $\Phi_{f'}(1) = \lambda$ with $\lambda \geq \frac{1}{n}$. For an integer $j \in \{0, 1, \ldots, 2n-2\}$, let $\mu'(j) = \mu_{f'}(j)^{(1)}$. Then, there exists an absolute constant $\gamma := \gamma(\lambda) > 0$ such that*

$$\frac{\sum_{j=0}^{n-1} \mu'(2j)}{n} \geq \gamma$$

We now prove the second step in the proof.

▶ **Lemma 17.** *Suppose that $f' : \{0,1\}^{2n-1} \to \{0,1\}$ is a monotone Boolean function such that $\Phi_{f'}(1) \geq \lambda$ with $\lambda \geq \frac{1}{n}$. Let g be a random minor of f' with respect to $\pi_2 : [2n-1] \to [n]$ which is obtained by setting $\pi_2(1) = 1$, and for every $i > 1$, we randomly choose $j_1, j_2 \in [2n-1] \setminus \{1\}$ (without replacements) and set $\pi_2(j_1) = \pi_2(j_2) = i$. In other words, we choose a uniformly random partition of $[2n-1] \setminus \{1\}$ into $n-1$ pairs $P_2, P_3, \ldots, P_n$ and set $\pi_2(j) = i \, \forall j \in P_i$. Then, there exists $\gamma := \gamma(\lambda) > 0$ such that*

$$\mathbb{E}_{\pi_2}[\Phi_g(1)] \geq \gamma \ .$$

Proof. For ease of notation, we let $\mu'(j) = \mu_{f'}(j)^{(1)}$ and $\mu_g(j) = \mu_g(j)^{(1)}$. For a set $S \subseteq [n] \setminus \{1\}$ and a function $\pi_2 : [2n-1] \to [n]$ with $\pi_2(1) = 1$, and $|\pi_2^{-1}(i)| = 2$ for all $i \in \{2, 3, \ldots, n\}$, let $\pi_2^{-1}(S)$ be the $2|S|$ sized subset of $\{2, 3, \ldots, 2n-1\}$ defined as follows:

$$\pi_2^{-1}(S) := \{\pi_2^{-1}(i) : i \in S\}$$

For every set $S \subseteq \{2, 3, \ldots, n\}$, when $\pi_2 : [2n-1] \to [n]$ is a uniformly random 2-to-1 minor with $\pi_2(1) = 1$, and the rest $2n-2$ elements are partitioned into $n-1$ pairs uniformly at random, the set $\pi_2^{-1}(S)$ is distributed uniformly in $\binom{[2n-1]\setminus\{1\}}{2|S|}$. Also note that $S \in \mathcal{B}_g(1)$ if and only if $\pi^{-1}(S) \in \mathcal{B}_{f'}(1)$. Thus, for every set $S \subseteq \{2, 3, \ldots, n\}$, the probability that $S \in \mathcal{B}_g(1)$ (over the choice of π_2) is equal to $\mu'(2|S|)$. Summing over all such sets of size j, we get that for every $j \in \{0, 1, \ldots, n-1\}$, the expected value of $\mu_g(j)$ is equal to $\mu'(2j)$.

$$\mathbb{E}_{\pi_2}[\mu_g(j)] = \mu'(2j) \, \forall j \in \{0, 1, \ldots, n-1\}$$

By using Lemma 16, we can infer that there exists $\gamma = \gamma(\lambda) > 0$ such that $\sum_{j=0}^{n-1} \mathbb{E}_{\pi_2}[\mu_g(j)] = \sum_{j=0}^{n-1} \mu'(2j) \geq \gamma n$. Using Equation (1), we get

$$\mathbb{E}_{\pi_2}[\Phi_g(1)] = \mathbb{E}_{\pi_2}\left[\frac{\sum_{j=0}^{n-1} \mu_g(j)}{n}\right] = \frac{\sum_{j=0}^{n-1} \mathbb{E}_{\pi_2}[\mu_g(j)]}{n} \geq \gamma. \qquad \blacktriangleleft$$

Lemma 15 and Lemma 17 together prove that Shapley value behaves well under uniformly random 2-to-1 minors for monotone Boolean functions.

▶ **Lemma 18.** *Suppose that* $f : \{0,1\}^{2n} \to \{0,1\}$ *is a monotone Boolean function such that* $\Phi_f(1) \geq \lambda$ *for some absolute constant* $\lambda > 0$ *with* $\lambda \geq \frac{1}{n}$*. Then, there exists* $\gamma := \gamma(\lambda) > 0$ *such that*

$$\mathbb{E}_\pi[\Phi_g(\pi(1))] \geq \gamma$$

where g *is a minor of* f *with respect to the uniformly random* 2*-to-*1 *function* π*.*

Proof. Combining Lemma 15 and Lemma 17, we can conclude that for every $i \in [2n], i > 1$, when $\pi : [2n] \to [n]$ is a uniformly random 2-to-1 minor conditioned on the fact that $\pi(1) = \pi(i)$, we have $\mathbb{E}_\pi[\Phi_g(\pi(1))] \geq \gamma$. Taking average over all the $i \in [2n], i > 1$, we get a proof that the same inequality holds when π is a uniformly random 2-to-1 minor. ◀

4.2 Reduction

We first formally define the Label Cover problem and state the Rich 2-to-1 Conjecture.

▶ **Definition 19** (Label Cover)**.** *In the Label Cover problem* $\mathcal{G} = (G, \Sigma_L, \Sigma_R, \Pi)$*, the input is a bipartite graph* $G = (L \cup R, E)$ *with projection constraint* $\Pi_e : \Sigma_L \to \Sigma_R$ *on every edge* $e \in E$*. A labeling* σ *which assigns values from* Σ_L *to* L *and from* Σ_R *to* R *satisfies the constraint* Π_e *on the edge* $e = (u, v)$ *if* $\Pi_e(\sigma(u)) = \sigma(v)$*. The objective is to identify if there is a labeling that satisfies all the constraints.*

For every constant $\epsilon > 0$, it is NP-hard to distinguish between the case that a given Label Cover instance has a labeling that satisfies all the constraints vs. no labeling can satisfy more than ϵ fraction of the constraints. This hardness result for Label Cover has been instrumental in showing numerous strong, and sometimes optimal, inapproximability results for various computational problems. However, standard Label Cover seems insufficient as a starting point towards proving hardness results for approximate graph coloring and other 2-variable PCSPs. To circumvent this, the hardness of Label Cover on structured instances such as Unique Games, smooth Label Cover, etc. has been studied.

In his celebrated work proposing the Unique Games Conjecture [20], Khot also proposed the "2-to-1 conjecture" that the strong hardness of Label Cover holds when all the constraints of the Label Cover are 2-to-1 functions. The imperfect completeness version of this conjecture was recently established in a striking sequence of works [13, 14, 22, 23]. Braverman, Khot, and Minzer [9] put forth a stronger conjecture that states that the hardness of Label Cover holds when the distribution of 2-to-1 functions on edges incident on every vertex $u \in L$ is uniform over $\mathcal{F}_{2\to1}$.

▶ **Definition 20** (Rich 2-to-1 Label Cover instances)**.** *We call a Label Cover instance* $\mathcal{G} = (G, \Sigma_L, \Sigma_R, \Pi)$ *with* $G = (L \cup R, E)$ *a rich* 2*-to-*1 *instance if the following hold.*

1. *There exists an integer* Σ *such that* $\Sigma_L = [2\Sigma]$*,* $\Sigma_R = [\Sigma]$*, and every projection constraint* Π_e*,* $e \in E$ *is a* 2*-to-*1 *function.*
2. *For every vertex* $u \in L$*, the distribution of* 2*-to-*1 *functions* $\mathcal{P}_u$ *obtained by first sampling a uniformly random neighbor* v *of* u*, and then picking* $\Pi_e, e = (u, v)$*, is uniform over* $\mathcal{F}_{2\to1}(\Sigma)$*.*

▶ **Conjecture 21** (Rich 2-to-1 Conjecture with Perfect Completeness, [9])**.** *For every* $\epsilon > 0$*, there exists an integer* $\Sigma = \Sigma(\epsilon)$ *such that given a rich* 2*-to-*1 *Label Cover instance* $\mathcal{G}$*, it is NP-Hard to distinguish between the following.*

1. *There is a labeling that satisfies all the constraints of* $\mathcal{G}$*.*
2. *No labeling can satisfy more than* ϵ *fraction of the constraints of* $\mathcal{G}$*.*

We are now ready to state the hardness part of our dichotomy. It is proved using the Label Cover-Long Code framework. This reduction is standard in the PCSP literature, see e.g., [4], and we include the proof in the full version of the paper.

▶ **Theorem 22.** *Assume the Rich 2-to-1 Conjecture. Let PCSP*(Γ) *be a Boolean Ordered PCSP such that there exists an absolute constant* $\lambda > 0$ *with* $\max_{i\in[n]} \Phi_f(i) \geq \lambda$ *for all functions* $f : \{0,1\}^n \to \{0,1\}$, $f \in \mathrm{Pol}(\Gamma)$. *Then PCSP*$(\Gamma)$ *is NP-Hard.*

5 Adversarial 2-to-1 minor

We construct an example of a 2-to-1 minor where the Shapley value alters completely after taking the minor.

▶ **Theorem 23.** *Let* $n \geq 2$ *be a positive integer. There exist two monotone Boolean functions* $f : \{0,1\}^{2n} \to \{0,1\}$ *and* $g : \{0,1\}^n \to \{0,1\}$ *such that* g *is a 2-to-1 minor of* f *with respect to the 2-to-1 function* $\pi : [2n] \to [n]$ *defined as* $\pi(i) = \lceil \frac{i}{2} \rceil$. *Furthermore,*

1. $\Phi_g(1) = \Omega(1)$, *and* $\Phi_g(j) = o(1)$ *for all* $j > 1$.
2. $\Phi_f(3) = \Omega(1)$, *and* $\Phi_f(i) = o(1)$ *for all* $i \in [2n], i \neq 3$.

Proof. Similar to the proof of Theorem 22, we construct the minor function pair in two steps.

1. First, we construct Boolean monotone functions $f : \{0,1\}^{2n-1} \to \{0,1\}$ and $g : \{0,1\}^n \to \{0,1\}$ such that g is a minor of f with respect to the function $\pi : [2n-1] \to [n]$ defined as $\pi(1) = 1, \pi(i) = \lceil \frac{i+1}{2} \rceil$ for all $i > 1$. Furthermore, $\Phi_g(1) = \Omega(1)$, and $\Phi_g(j) = o(1)$ for all $j > 1$. We also have $\Phi_f(2) = \Omega(1)$, and $\Phi_f(i) = o(1)$ for all $i \in [2n-1], i \neq 2$.
2. We define the function $f' : \{0,1\}^{2n} \to \{0,1\}$ as

   $$f'(y_1, y_2, \ldots, y_{2n}) = f(y_1, y_3, \ldots, y_{2n})$$

   Note that g is a minor of f' with respect to the 2-to-1 function $\pi : [2n] \to [n]$ defined as $\pi(i) = \lceil \frac{i}{2} \rceil$. Furthermore, by definition, we have $\Phi_{f'}(3) = \Omega(1)$, and $\Phi_{f'}(i) = o(1)$ for all $i \in [2n], i \neq 3$.

Henceforth, our goal is to construct a pair of functions as in the first step above.

We define a partial Boolean function to be a function from $\{0,1\}^n \to \{0,1,?\}$. A partial Boolean function on n variables is monotone if for all $\mathbf{p} \in \{0,1\}^n$ and $\mathbf{q} \in \{0,1\}^n$ such that $\mathbf{p} \leq \mathbf{q}$, if $f(\mathbf{p}) = 1$, then $f(\mathbf{q}) = 1$, and if $f(\mathbf{q}) = 0$, then $f(\mathbf{p}) = 0$.

Now, consider $g : \{0,1\}^n \to \{0,1\}$ to be

$$g(\mathbf{x}) = \begin{cases} 1 \text{ if } \sum_{j=2}^{n} x_j \geq \frac{51n}{100} \\ 0 \text{ if } \sum_{j=2}^{n} x_j \leq \frac{49n}{100} \\ x_1 \text{ if } \frac{49n}{100} < \sum_{j=2}^{n} x_j < \frac{51n}{100} \end{cases}$$

By definition, g is a monotone function, and using Equation (1), we can infer that $\Phi_g(1) = \frac{1}{50}$, and $\Phi_g(j) < \frac{1}{n}$ for all $j > 1$.

We now construct f in three steps. Start with $f = '?'$.

1. (Preserving the minor) First, set the value of entries of f that are of the form $(x_1, x_2, x_2, \cdots, x_n, x_n)$ as

   $$f(x_1, x_2, x_2, \ldots, x_n, x_n) = g(x_1, x_2, \ldots, x_n) \ \forall \mathbf{x} \in \{0,1\}^n$$

   We then extend it both upwards and downwards i.e. if $f(\mathbf{p})$ is set to 1 and $\mathbf{p} \leq \mathbf{q}$, then set $f(\mathbf{q}) = 1$ as well, and similarly, if $f(\mathbf{q})$ is set to 0, and $\mathbf{p} \leq \mathbf{q}$, then we set $f(\mathbf{p}) = 0$. This ensures that g is a minor of f and that the partial function f is monotone.

2. (Destroying the influence of 1) Next, we ensure that the Shapley value of the coordinate 1 is low by the following operation: consider all $\mathbf{y}$ such that $f(\mathbf{y})$ has not been set in the first step, $y_1 = 0$ and $f(1, y_2, \cdots, y_{2n-1})$ is already set to 1 in the first step. Then set $f(\mathbf{y})$ to be 1. Similarly, if $\mathbf{y}$ satisfies $y_1 = 1$ and $f(0, y_2, \cdots, y_{2n-1})$ is already set to 0 in the first step, set $f(\mathbf{y})$ to be 0 if it has not been set in the first step.
   We claim that the updated partial function f is still a monotone partial function. Consider $\mathbf{p}, \mathbf{q} \in \{0,1\}^{2n-1}$ such that $\mathbf{p} \leq \mathbf{q}$. Suppose that $f(\mathbf{p})$ is set to be 1. If it is set in the first step, as we extended the partial function upwards in the first step, $f(\mathbf{q}) = 1$ as well. If $f(\mathbf{p})$ is set to be 1 in the second step, it implies that $f(\mathbf{p}')$ has been set to 1 in the first step, where $\mathbf{p}'$ is obtained from $\mathbf{p}$ by setting p_1 to be 1. Let $\mathbf{q}' \in \{0,1\}^{2n-1}$ be obtained from $\mathbf{q}$ by setting $q_1 = 1$. As $\mathbf{p}' \leq \mathbf{q}'$, $f(\mathbf{q}')$ has been set to 1 in the first step as well. Thus, $f(\mathbf{q})$ is set to be 1 in the second step. The same argument can be used to show that if $f(\mathbf{q}) = 0$, then $f(\mathbf{p}) = 0$ as well.
3. (Adding influence to 2) For all $\mathbf{y}$ for which $f(\mathbf{y}) =' ?'$ set $f(\mathbf{y}) = y_2$. The fact that the final function f is monotone follows from observing that any completion of a partial monotone function using a monotone function results in a monotone function.

Finally, our goal is to argue about the Shapley value of the coordinates of the function f. First, we show that the Shapley value of the coordinate 1 in f is $o(1)$. Suppose there exists $\mathbf{p} = (0, y_2, y_3, \cdots, y_{2n-1})$ and $\mathbf{q} = (1, y_2, y_3, \cdots, y_{2n-1})$ such that $f(\mathbf{p}) = 0$ and $f(\mathbf{q}) = 1$. We claim that both the values $f(\mathbf{p})$ and $f(\mathbf{q})$ are set in the first step of the above procedure. Suppose for contradiction that this is not the case. If neither of them is set in the first step, then they will not be set in the second step either, and in the third step, both of them will be assigned the same value, a contradiction. If exactly one of them is set in the first step, then in the second step, the other value would be set to be equal to it, a contradiction as well. Thus, both the values $f(\mathbf{p})$ and $f(\mathbf{q})$ are set in the first step.

Let $B = \mathcal{B}_g(1) \subseteq \{0,1\}^{n-1}$ be the boundary of the coordinate 1 in g. As $f(\mathbf{q})$ is set to be 1 in the first step, there exists $\mathbf{x} \in \{0,1\}^n$ such that $g(\mathbf{x}) = 1$ and $(x_1, x_2, x_2, \cdots, x_n, x_n) \leq \mathbf{q}$. As $\mathbf{x}$ is not less than or equal to $\mathbf{p}$, we can conclude that $x_1 = 1$ and $g(0, x_2, x_3, \cdots, x_n) = 0$. In other words, $(x_2, x_3, \cdots, x_n) \in B$. Similarly, there exists $\mathbf{x}'$ such that $g(\mathbf{x}') = 0$ and $(x'_1, x'_2, x'_2, \cdots, x'_n, x'_n) \geq \mathbf{p}$. By the same argument as above, we can conclude that $(x'_2, x'_3, \cdots, x'_n) \in B$. Combining the both, we can conclude that there exist $\mathbf{x}, \mathbf{x}' \in B$ such that $(x_2, x_2, x_3, x_3, \ldots, x_n, x_n) \leq (y_2, y_3, \cdots, y_{2n-2}) \leq (x'_2, x'_2, x'_3, x'_3, \ldots, x'_n, x'_n)$. Note that if the above inequality is true for a $(y_2, y_3, \cdots, y_{2n-2})$, we directly get that $(y_2, y_3, \cdots, y_{2n-2})$ is in the boundary of the coordinate 1 in f.

Observe that the boundary of coordinate 1 in g is the set of vectors $(x_2, x_3, \cdots, x_n)$ such that $\frac{49}{100}n \leq \sum_{j=2}^{n} x_j \leq \frac{51}{100}n$. By the previous argument, we can deduce that the boundary $B' = \mathcal{B}_f(1)$ of the coordinate 1 in f is the set of vectors $\mathbf{y} = (y_2, y_3, \cdots, y_{2n-1})$ that satisfy the following property: The number of $i \in [n-1]$ such that both $y_{2i} = y_{2i+1} = 1$ is at least $\frac{49}{100}n$. Similarly, the number of $i \in [n-1]$ such that $y_{2i} = y_{2i+1} = 0$ is at least $\frac{49}{100}n$. Observe that this implies that we require that $\frac{49}{50}n \leq \sum_{j=2}^{2n-1} y_j \leq \frac{51}{50}n$. However, for every integer l such that $\frac{49}{50}n \leq l \leq \frac{51}{50}n$, when we sample a uniformly random vector $\mathbf{y} = (y_2, y_3, \ldots, y_{2n-1})$ with $\sum_{j=2}^{2n-1} y_j = l$, the probability that the number of $i \in [n-1]$ such that both $y_{2i} = y_{2i+1} = 1$ is at least $\frac{49}{100}n$ is $o(\frac{1}{n})$. Thus, using Equation (1), we can infer that the Shapley value of the coordinate 1 in f is $o(1)$.

We now show that the coordinate 2 has $\Omega(1)$ Shapley value in f. Consider $\mathbf{y} = (y_1, y_2, \ldots, y_{2n-1})$ such that $\frac{49n}{50} < \mathsf{hw}(\mathbf{y}) \leq \frac{51n}{50}$. If the number of i such that both $y_{2i} = y_{2i+1} = 1$ is less than $\frac{49}{100}n$, we have $(y_1, y_3, \ldots, y_{2n-1}) \in \mathcal{B}_f(2)$. However, for every integer l such that $\frac{49}{50}n \leq l \leq \frac{51}{50}n$, when we sample a uniformly random $\mathbf{y}$ with

$\mathsf{hw}(\mathbf{y}) = l$, with probability $1 - o(1)$, the number of i such that both $y_{2i} = y_{2i+1} = 1$ is less than $\frac{49}{100}n$. Thus, using Equation (1), we can infer that the Shapley value of the coordinate 2 is $\Omega(1)$ in the function f. Finally, by symmetry, we can observe that $\Phi_f(i) = \Phi_f(3)$ for all $i \geq 3$, and thus, $\Phi_f(i) = o(1)$ for all $i \geq 3$. ◀

References

1 Per Austrin, Venkatesan Guruswami, and Johan Håstad. $(2+\epsilon)$-SAT is NP-hard. *SIAM J. Comput.*, 46(5):1554–1573, 2017.

2 Libor Barto. Cyclic operations in promise constraint satisfaction problems. *Dagstuhl workshop The Constraint Satisfaction Problem: Complexity and Approximability, Schloss Dagstuhl, Germany*, 2018. Available at `https://www2.karlin.mff.cuni.cz/~barto/Articles/Barto_Dagstuhl18.pdf`.

3 Libor Barto. Personal communication, 2018-20.

4 Libor Barto, Jakub Bulín, Andrei Krokhin, and Jakub Opršal. Algebraic approach to promise constraint satisfaction. *arXiv preprint*, 2018. `arXiv:1811.00970`.

5 Joshua Brakensiek and Venkatesan Guruswami. New hardness results for graph and hypergraph colorings. In *31st Conference on Computational Complexity, CCC 2016*, pages 14:1–14:27, 2016.

6 Joshua Brakensiek and Venkatesan Guruswami. Promise constraint satisfaction: Structure theory and a symmetric Boolean dichotomy. In *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2018*, pages 1782–1801, 2018.

7 Joshua Brakensiek, Venkatesan Guruswami, and Sai Sandeep. Conditional dichotomy of boolean ordered promise csps. *arXiv preprint*, 2021. `arXiv:2102.11854`.

8 Joshua Brakensiek, Venkatesan Guruswami, Marcin Wrochna, and Stanislav Zivný. The power of the combined basic linear programming and affine relaxation for promise constraint satisfaction problems. *SIAM J. Comput.*, 49(6):1232–1248, 2020.

9 Mark Braverman, Subhash Khot, and Dor Minzer. On rich 2-to-1 games. In *12th Innovations in Theoretical Computer Science Conference, ITCS 2021*, volume 185 of *LIPIcs*, pages 27:1–27:20, 2021.

10 Andrei A. Bulatov. A dichotomy theorem for nonuniform CSPs. In *58th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2017*, pages 319–330. IEEE Computer Society, 2017.

11 Andrei A. Bulatov, Peter Jeavons, and Andrei A. Krokhin. Classifying the complexity of constraints using finite algebras. *SIAM J. Comput.*, 34(3):720–742, 2005.

12 Anindya De, Ilias Diakonikolas, and Rocco A. Servedio. The inverse Shapley value problem. *Games Econ. Behav.*, 105:122–147, 2017.

13 Irit Dinur, Subhash Khot, Guy Kindler, Dor Minzer, and Muli Safra. On non-optimally expanding sets in grassmann graphs. In *Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2018,*, pages 940–951. ACM, 2018.

14 Irit Dinur, Subhash Khot, Guy Kindler, Dor Minzer, and Muli Safra. Towards a proof of the 2-to-1 games conjecture? In *Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2018,*, pages 376–389. ACM, 2018.

15 Tomás Feder and Moshe Y. Vardi. The computational structure of monotone monadic SNP and constraint satisfaction: A study through datalog and group theory. *SIAM J. Comput.*, 28(1):57–104, 1998.

16 Miron Ficak, Marcin Kozik, Miroslav Olsák, and Szymon Stankiewicz. Dichotomy for symmetric boolean PCSPs. In *46th International Colloquium on Automata, Languages, and Programming, ICALP 2019*, volume 132 of *LIPIcs*, pages 57:1–57:12, 2019.

17 Peter Jeavons. On the algebraic structure of combinatorial problems. *Theor. Comput. Sci.*, 200(1-2):185–204, 1998.

18 Peter Jeavons, David A. Cohen, and Marc Gyssens. Closure properties of constraints. *J. ACM*, 44(4):527–548, 1997.

19 Gil Kalai. Social indeterminacy. *Econometrica*, 72(5):1565–1581, 2004.

20 Subhash Khot. Hardness results for coloring 3-colorable 3-uniform hypergraphs. In *The 43rd Annual IEEE Symposium on Foundations of Computer Science, FOCS 2002*, pages 23–32. IEEE, 2002.

21 Subhash Khot. On the power of unique 2-prover 1-round games. In *Proceedings on 34th Annual ACM Symposium on Theory of Computing, STOC 2002*, pages 767–775. ACM, 2002.

22 Subhash Khot, Dor Minzer, and Muli Safra. On independent sets, 2-to-2 games, and grassmann graphs. In *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2017,*, pages 576–589. ACM, 2017.

23 Subhash Khot, Dor Minzer, and Muli Safra. Pseudorandom sets in Grassmann graph have near-perfect expansion. In *59th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2018*, pages 592–601. IEEE Computer Society, 2018.

24 Andrei A. Krokhin and Jakub Opršal. The complexity of 3-colouring H-colourable graphs. In *60th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2019*, pages 1227–1239. IEEE Computer Society, 2019.

25 G. A. Margulis. Probabilistic characteristics of graphs with large connectivity (in Russian). *Probl. Pered. Inform.*, 10:101–108, 1974.

26 Tomasz P. Michalak, Aadithya V. Karthik, Piotr L. Szczepanski, Balaraman Ravindran, and Nicholas R. Jennings. Efficient computation of the Shapley value for game-theoretic network centrality. *J. Artif. Intell. Res.*, 46:607–650, 2013.

27 Ramasuri Narayanam and Yadati Narahari. A Shapley value-based approach to discover influential nodes in social networks. *IEEE Trans Autom. Sci. Eng.*, 8(1):130–147, 2011.

28 Ryan O'Donnell. *Analysis of Boolean Functions*. Cambridge University Press, 2014.

29 Jan Petr. Monotone functions avoiding majorities, 2020. Undergraduate Thesis. Univerzita Karlova, Matematicko-fyzikální fakulta.

30 Nicholas Pippenger. Galois theory for minors of finite functions. *Discrete Mathematics*, 254(1-3):405–419, 2002.

31 Lucio Russo. An approximate zero-one law. *Z. Wahrscheinlichkeitstheorie und Verwandte Gebiete*, 61(1):129–139, 1982.

32 Thomas J. Schaefer. The complexity of satisfiability problems. In *Proceedings of the 10th Annual ACM Symposium on Theory of Computing, STOC 1978*, pages 216–226. ACM, 1978.

33 L. S. Shapley and Martin Shubik. A method for evaluating the distribution of power in a committee system. *American Political Science Review*, 48(3):787–792, 1954.

34 Robert Weber. Probabilistic values for games. Cowles Foundation Discussion Papers 471R, Cowles Foundation for Research in Economics, Yale University, 1977. URL: `https://EconPapers.repec.org/RePEc:cwl:cwldpp:471r`.

35 Marcin Wrochna and Stanislav Zivný. Improved hardness for H-colourings of G-colourable graphs. In *Proceedings of the 2020 ACM-SIAM Symposium on Discrete Algorithms, SODA 2020*, pages 1426–1435. SIAM, 2020.

36 Dmitriy Zhuk. A proof of the CSP dichotomy conjecture. *J. ACM*, 67(5):30:1–30:78, 2020.

Parameterized Applications of Symbolic Differentiation of (Totally) Multilinear Polynomials

Cornelius Brand
Charles University, Prague, Czech Republic

Kevin Pratt
Carnegie Mellon University, Pittsburgh, PA, USA

Abstract

We study the following problem and its applications: given a homogeneous degree-d polynomial g as an arithmetic circuit C, and a $d \times d$ matrix X whose entries are homogeneous linear polynomials, compute $g(\partial/\partial x_1, \ldots, \partial/\partial x_n) \det X$. We show that this quantity can be computed using $2^{\omega d}|C|\operatorname{poly}(n,d)$ arithmetic operations, where ω is the exponent of matrix multiplication. In the case that C is skew, we improve this to $4^d|C|\operatorname{poly}(n,d)$ operations, and if furthermore X is a Hankel matrix, to $\varphi^{2d}|C|\operatorname{poly}(n,d)$ operations, where $\varphi = \frac{1+\sqrt{5}}{2}$ is the golden ratio.

Using these observations we give faster parameterized algorithms for the matroid k-parity and k-matroid intersection problems for linear matroids, and faster deterministic algorithms for several problems, including the first deterministic polynomial time algorithm for testing if a linear space of matrices of logarithmic dimension contains an invertible matrix. We also match the runtime of the fastest deterministic algorithm for detecting subgraphs of bounded pathwidth with a new and simple approach. Our approach generalizes several previous methods in parameterized algorithms and can be seen as a relaxation of Waring rank based methods [Pratt, FOCS19].

2012 ACM Subject Classification Theory of computation → Design and analysis of algorithms

Keywords and phrases Parameterized Algorithms, Algebraic Algorithms, Longest Cycle, Matroid Parity

Digital Object Identifier 10.4230/LIPIcs.ICALP.2021.38

Category Track A: Algorithms, Complexity and Games

Funding *Cornelius Brand*: The research was supported by OP RDE project No. CZ.02.2.69/0.0/0.0/18_053/0016976 International mobility of research, technical and administrative staff at Charles University.

Acknowledgements We would like to thank Ryan O'Donnell and several anonymous reviewers for their many helpful comments on earlier drafts of this paper. In particular we thank an anonymous reviewer for suggesting the name "totally multilinear."

1 Introduction

Let $\mathcal{S}_d^n := \mathbb{Q}[x_1, \ldots, x_n]_d$ denote the vector space of homogeneous polynomials of degree d in n variables with rational coefficients. We define the *apolar inner product* $\langle \cdot, \cdot \rangle : \mathcal{S}_d^n \times \mathcal{S}_d^n \to \mathbb{Q}$ via

$$\langle f, g \rangle := f\left(\frac{\partial}{\partial x_1}, \ldots, \frac{\partial}{\partial x_n}\right) g. \tag{1}$$

Explicitly, if $f = \sum_{i_1,\ldots,i_n} a_{i_1,\ldots,i_n} x_1^{i_1} \cdots x_n^{i_n}$ and $g = \sum_{i_1,\ldots,i_n} b_{i_1,\ldots,i_n} x_1^{i_1} \cdots x_n^{i_n}$, then

$$\langle f, g \rangle = \sum_{i_1,\ldots,i_n} i_1! \cdots i_n! a_{i_1,\ldots,i_n} b_{i_1,\ldots,i_n}.$$

This inner product originated in 19th century invariant theory [39] and has become a source of interest in computer science due to algorithmic applications. In a typical application, one first identifies some easy-to-evaluate generating polynomial g whose coefficients encode solutions to a combinatorial problem. This information can then be recovered by computing $\langle f, g\rangle$ for a suitable choice of f. While this quantity can be $\#P$ hard to compute exactly, in special cases it can be efficiently approximated. This approach has led to new algorithms for problems as disparate as approximating permanents and mixed discriminants [25], sampling from determinantal point processes [3], Nash social welfare maximization [5], and approximately counting subgraphs [37].

As a motivating example, given a directed graph G with n vertices, let A_G be the $n \times n$ matrix with entry (i, j) equal to the variable x_i if there is an edge from vertex v_i to vertex v_j, and zero otherwise. By the trace method,

$$\operatorname{tr}(A_G^d) = \sum_{\substack{(v_{i_1}, v_{i_2}, \ldots, v_{i_d}) \in G, \\ v_{i_d} = v_{i_1}}} x_{i_1} \cdots x_{i_d} \in \mathcal{S}_d^n.$$

Now let $A \in \mathbb{Q}^{d \times n}$ be a matrix any d columns of which are linearly independent. Let $X = A \cdot \operatorname{diag}(x_1, \ldots, x_n) \cdot A^{\mathrm{T}}$. By the Cauchy-Binet formula,

$$\det X = \sum_{S \in \binom{[n]}{d}} \det(A_S)^2 \prod_{i \in S} x_i.$$

(Here A_S refers to the $d \times d$ submatrix of A with columns indexed by the set S.) Since any d columns in A are linearly independent, $\det(A_S)^2 > 0$ for all $S \in \binom{[n]}{d}$. Then note that $\langle \det(A_S)^2 \prod_{i \in S} x_i, \operatorname{tr}(A_G^d)\rangle$ is positive if there is a simple cycle on the vertices $\{v_i : i \in S\}$, and zero otherwise. It follows by linearity that $\langle \det X, \operatorname{tr}(A_G^d)\rangle > 0$ if and only if G contains a simple cycle of length d.

Motivated by this and other applications, we consider in our Theorems 7, 13, and 25 the algorithmic task of computing the inner product (1) when f is the determinant of a symbolic matrix (a matrix whose entries are homogeneous linear polynomials) and g is given as an arithmetic circuit. As one consequence, starting from the observation of the above example we give a deterministic $\varphi^{2d} \operatorname{poly}(n) < 2.62^d \operatorname{poly}(n)$-time algorithm for detecting simple cycles of length d in an n vertex graph. Here $\varphi := \frac{1+\sqrt{5}}{2}$ is the golden ratio. Our algorithm generalizes to detecting subgraphs of bounded pathwidth, unexpectedly matching the runtime of the fastest known algorithm for this problem of [20].

Our main conceptual contribution is the observation that the following algebraic question is central to a handful of methods in parameterized algorithms. It is motivated by the observation that in order to obtain a (possibly randomized) algorithm for detecting cycles, it would suffice to compute $\langle f, \operatorname{tr}(A_G^d)\rangle$ for any $f \in \mathcal{S}_d^n$ that is supported exactly on the set of all degree-d square-free monomials; $\det X$ is just one such polynomial. We call such polynomials *totally multilinear*, and denote the set of all such polynomials in $\mathcal{S}_d^n$ by $\mathcal{T}_{n,d}$.

▶ **Question 1.** *Let $\mathcal{T}_{n,d}$ be the set of all $f \in \mathcal{S}_d^n$ such that $f = \sum_{S \in \binom{[n]}{d}} c_S \prod_{i \in S} x_i$, where $c_S \neq 0$ for all S. What is $B(d, n) := \min(\dim \operatorname{Diff}(f) : f \in \mathcal{T}_{n,d})$? Here $\operatorname{Diff}(f)$ denotes the vector space spanned by the partial derivatives of all orders of f, including f itself.* [1]

Our algorithms, color coding [1], the group algebra approach [31], and the exterior algebra methods of [12, 11] are all closely related to the existence of polynomials in $\mathcal{T}_{n,d}$ (or related sets) with "unusually small" spaces of partial derivatives (see Section 5). In [37] it was shown

[1] For example, $\operatorname{Diff}(x_1x_2)$ is the vector space spanned by x_1x_2, x_1, x_2, and 1.

that a related quantity, namely the minimum *Waring rank* of any $f \in \mathcal{T}_{n,d}$, gives upper bounds on the complexity of certain parameterized problems. In general, the Waring rank of $f \in \mathcal{S}_d^n$ is lower bounded by $\frac{1}{d} \dim \mathrm{Diff}(f)$, and this bound is almost never optimal [32, Section 3.2]. In this paper, we exploit that fact that, provided f can be "efficiently differentiated," this lower bound can be used to *upper bound* the complexity of these parameterized problems! For instance, our $\varphi^{2d}\,\mathrm{poly}(n)$-time cycle detection algorithm relies on the fact that there is a spanning set of size $\varphi^{2d} < 2.62^d$ for the space of partial derivatives of the polynomial $\det X$ above, when A is a Vandermonde matrix, that we can differentiate $\det X$ "efficiently" with respect to. In contrast, the best best-known upper bound on the Waring rank of this polynomial is 6.75^d [37, Theorem 41].

We show in Proposition 10 that $B(n,d) \leq O(2.6^d)$. Additionally, it is not difficult to show that $B(n,d) \geq 2^d$. A proof of this fact is as follows. First, observe that $\dim \mathrm{Diff}(f)$ does not increase under setting variables to zero. Hence for any $f \in \mathcal{T}_{n,d}$, $\dim \mathrm{Diff}(f) \geq \dim \mathrm{Diff}(c \cdot x_1 x_2 \cdots x_d)$ for some nonzero constant c. As $\mathrm{Diff}(c \cdot x_1 x_2 \cdots x_d)$ has as a basis the collection of products of subsets of the variables $x_1, \ldots, x_d$, the claim follows.

1.1 Previous approaches to computing the apolar inner product

One special case of (1) that has been the source of several recent breakthroughs is when f and g are *real stable* polynomials with nonnegative coefficients; see e.g. [27, 4]. In this case $\langle f, g \rangle$ can be approximated (up to a factor of $e^{d+\varepsilon}$) in polynomial time [2, Theorem 1.2]. For the cases we consider, however, f and g will not both be real stable.

Another approach is based on Waring rank upper bounds [9, 26, 23, 37]. The Waring rank of $f \in \mathcal{S}_d^n$, denoted $\mathbf{R}(f)$, is defined as the minimum r such that $f = \sum_{i=1}^r c_i \ell_i^d$ for linear forms $\ell_1, \ldots, \ell_r \in \mathcal{S}_1^n$ and scalars $c_1, \ldots, c_r$. For example, the identity

$$x_1 x_2 x_3 = \frac{1}{24}\left[(x_1 + x_2 + x_3)^3 - (x_1 + x_2 - x_3)^3 - (x_1 - x_2 + x_3)^3 - (-x_1 + x_2 + x_3)^3\right]$$

shows that $\mathbf{R}(x_1 x_2 x_3) \leq 4$. Waring rank has been studied since the 1850's [29, Introduction] and has gained recent attention for its applications to algebraic complexity, see e.g. [13, 14]. Its relevance to the inner product (1) is due to the following fact, which can be verified by a straightforward calculation: if $f = \sum_{i=1}^r c_i (a_{i,1} x_1 + \cdots + a_{i,n} x_i)^d$, then for all $g \in \mathcal{S}_d^n$,

$$\langle f, g \rangle = d! \sum_{i=1}^r c_i g(a_{i,1}, \ldots, a_{i,n}).$$

Hence upper bounds on $\mathbf{R}(f)$ yield black-box algorithms for computing $\langle f, g \rangle$. Furthermore, it was shown in [37, Theorem 6] that with only evaluation access to g, $\mathbf{R}(f)$ queries are *required* to compute this inner product. Unfortunately, $\mathbf{R}(f)$ is usually prohibitively large; for instance, the Waring rank of almost all $f \in \mathcal{S}_d^n$ is at least $\lceil \binom{n+d-1}{d} / n \rceil$ [32, Section 3.2].

It is also worth pointing out the very recent works of Arvind et al. [7, 6], in particular, [7, Remark 4.3.], which set up a very similar framework based on non-commutative polynomials and algebraic branching programs, on which they prove bounds that also follow from bounds on spaces of partial derivatives.

1.2 Our approach

Given g as an arithmetic circuit C, we compute $\langle f, g \rangle$ symbolically. Our algorithms inductively compute at each gate in C the result of differentiating f by the polynomial computed by C at that gate[2]. At the output gate of C we will therefore have computed $\langle g, f \rangle = \langle f, g \rangle$.

[2] By "differentiating f by g," we mean applying the differential operator $g(\partial/\partial x_1, \ldots, \partial/\partial x_n)$ to f.

At intermediate gates we compute and store elements of $\mathrm{Diff}(f)$, the vector space of partial derivatives of f, which we represent with respect to some spanning set for this space. This kind of symbolic manipulation of partial derivatives is reminiscent of the Baur-Strassen Theorem and its applications (see for example [17]).

We will be particularly interested in the case when f is the determinant of a symbolic matrix X. The advantage of this case is that for a symbolic $d \times d$ matrix X, the vector space spanned by the partial derivatives of $\det X$ of all orders has dimension at most 4^d, and in some algorithmically relevant cases this bound can be significantly improved. So while one might naïvely represent an element in this space as a linear combination of $\binom{n+d}{d}$ monomials, doing so generally includes a significant amount of unnecessary information. Instead, we represent elements in this space as linear combinations of minors (determinants of submatrices) of X, which are specified by pairs of increasing sequences.

We will start by giving in our Theorem 7 a simple algorithm for the special but important case when g is computed by a *skew* circuit, meaning one of the two operands to each multiplication gate is a variable or a scalar:

▶ **Theorem 7.** *Let C be a skew arithmetic circuit computing $g \in \mathcal{S}_d^n$, and let $X = (\ell_{i,j})_{i,j\in[d]}$ be a symbolic matrix with entries in $\mathcal{S}_1^n$. Then we can compute $\langle \det X, g\rangle$ with $4^d|C|\operatorname{poly}(n,d)$ arithmetic operations.*

Our algorithm for Theorem 7 only uses linear algebra and basic properties of differentials.

Of particular interest will be the case of Theorem 7 when X is a Hankel matrix, meaning that $X_{i,j} = X_{i+k,j-k}$ for all $k = 0, \dots, j-i$. For example, the generic 3×3 Hankel matrix is

$$\begin{bmatrix} x_1 & x_2 & x_3 \\ x_2 & x_3 & x_4 \\ x_3 & x_4 & x_5 \end{bmatrix}.$$

This has applications to problems such as detecting cycles in graphs and more generally detecting square-free monomials in arithmetic circuits (Corollary 19). We show the following improvement in this case:

▶ **Theorem 13.** *Let C be a skew arithmetic circuit computing $g \in \mathcal{S}_d^n$, and let $X = (\ell_{i,j})_{i,j\in[d]}$ be a symbolic Hankel matrix with entries in $\mathcal{S}_1^n$. Then we can compute $\langle \det X, g\rangle$ with $\varphi^{2d}|C|\operatorname{poly}(n,d)$ arithmetic operations. Here $\varphi := \frac{1+\sqrt{5}}{2}$ is the golden ratio.*

The improvement in Theorem 13 over Theorem 7 is facilitated by the fact that the space of partial derivatives of the determinant has dimension about 4^d, whereas the dimension of the space of partial derivatives of the determinant of a Hankel matrix is upper bounded by φ^{2d} (Proposition 10).

Let us point out that Hankel matrices (in their guise as squares of the Vandermonde) made appearances already in the exterior-algebraic framework [12, 11], so that their usefulness in our applications might perhaps not come as a complete surprise. Before this work, however, any connections between the exterior and the partial-differential approach remained unclear, and their exact nature remains to be determined.

Still, we do gain one such connection here: For general (not necessarily skew) circuits, we exploit a connection between the *apolar algebra* of the determinant and the exterior algebra (Lemma 24) to show the following:

▶ **Theorem 25.** *Let C be an arithmetic circuit computing $g \in \mathcal{S}_d^n$, and let $X = (\ell_{i,j})_{i,j\in[d]}$ be a symbolic matrix with entries in $\mathcal{S}_1^n$. Then we can compute $\langle \det X, g\rangle$ with $2^{\omega d}|C|\operatorname{poly}(n,d)$ arithmetic operations, where $\omega < 2.373$ is the exponent of matrix multiplication.*

1.3 Applications

Theorem 7 yields faster algorithms for the k-matroid intersection and matroid k-parity problems:

▶ **Problem 1** (Matroid k-Parity). *Suppose we are given a matrix $B \in \mathbb{Q}^{km \times kn}$ representing a matroid M with groundset $[kn]$, and a partition π of $[kn]$ into parts of size k. Decide if the union of any m parts in π are independent in M.*

▶ **Problem 2** (k-Matroid Intersection). *Suppose we are given matrices $B_1, \ldots, B_k \in \mathbb{Q}^{m \times n}$ representing matroids $M_1, \ldots, M_k$ with the common groundset $[n]$. Decide if $M_1, \ldots, M_k$ share a common base.*

We show in Theorems 17 and 18 that these can be solved in time $4^{km}\,\mathrm{poly}(N)$, where N denotes the size of the input. When $k = 2$ these are the classic matroid parity and intersection problems and can be solved in polynomial time, but for $k > 2$ they are NP-hard. The first algorithms for general k faster than naïve enumeration were given by Barvinok in [8], and had runtimes $(km)^{2k+1}4^{km}\,\mathrm{poly}(N)$ and $(km)^{2k}4^{k^2m}\,\mathrm{poly}(N)$, respectively. A parameterized algorithm for Problem 1 was also given by Marx in [36] where it was used to give fixed-parameter tractable algorithms for several other problems, including Problem 2. The fastest algorithms prior to our work were due to Fomin et al. [20] and had runtime $2^{km\omega}\,\mathrm{poly}(N)$, where $\omega < 2.373$ is the exponent of matrix multiplication [33].

By combining Theorem 7 with a known construction of the determinant as a skew circuit [35], we obtain a faster deterministic algorithm for the following problem:

▶ **Problem 3** (SING). *Given matrices $A_1, \ldots, A_n \in \mathbb{Q}^{d \times d}$, decide if their span contains an invertible matrix. Equivalently, decide if $\det \sum_{i=1}^{n} x_i A_i \not\equiv 0$.*

We show that SING can be solved in $4^d\,\mathrm{poly}(N)$ time in our Corollary 16. In particular, this establishes that SING $\in \mathcal{P}$ for subspaces of matrices of logarithmic dimension. The fastest previous deterministic algorithm, due to an observation of Gurvits in [24], had runtime $2^d d!\,\mathrm{poly}(N)$ and made use of an upper bound of $2^d d!$ on $\mathbf{R}(\det_d)$. This problem was originally studied by Edmonds for its application to matching problems [18]. While it is known to admit a simple randomized polynomial time algorithm as was first observed by Lovász [34], a *deterministic* polynomial time algorithm would imply circuit lower bounds that seem far beyond current reach [30]. As a result, variants of SING have attracted attention, leading to a recent breakthrough in the non-commutative setting [22].

Using Theorem 13, we give in Corollary 19 a deterministic $\varphi^{2d}\,\mathrm{poly}(|C|)$-time algorithm for detecting square-free monomials of degree-d in a polynomial with non-negative coefficients computed by a skew arithmetic circuit. Combining this with observations in [11], we obtain the following applications:

▶ **Corollary 20.** *The following problems admit deterministic algorithms running in time $\varphi^{2d}\,\mathrm{poly}(n)$:*

1. *Deciding whether a given directed n-vertex graph has a directed spanning tree with at least d non-leaf vertices,*
2. *Deciding whether a given edge-colored, directed n-vertex graph has a directed spanning tree containing at least d colors,*
3. *Deciding whether a given planar, edge-colored, directed n-vertex graph has a perfect matching containing at least d colors.*

The previous fastest algorithms for these problems had runtimes $3.19^d\,\mathrm{poly}(n)$, $4^d\,\mathrm{poly}(n)$, and $4^d\,\mathrm{poly}(n)$, respectively [11]. This built upon work of Gutin et al. [28] Problem (1) is the best studied among these, with [28, Table 1] listing eleven articles on this problem in the last fourteen years. It is worth noting that our improvements do not rely on any problem-specific adaptations.

Theorem 13 also yields a $\varphi^{2d}\,\mathrm{poly}(n)$-time deterministic algorithm for detecting simple cycles of length d in an n vertex directed graph (and paths, and more generally subgraphs of bounded pathwidth). While it is known that simple cycles of length d can be detected in randomized time $2^d\,\mathrm{poly}(n)$ [41] ($1.66^d\,\mathrm{poly}(n)$ for undirected graphs [10]), it is a major open problem to achieve the same runtime deterministically. Finding a better upper bound on $B(n,d)$ witnessed by a polynomial with nonnegative coefficients seems to us a promising approach for obtaining faster deterministic algorithms for this problem.

Our cycle detection algorithm brushes up against the fastest known deterministic algorithm for this problem which has runtime $2.55^d\,\mathrm{poly}(n)$ [40], and unexpectedly matches the runtime of a previous algorithm [20] while using a different (shorter) approach. Our approach differs from those of previous algorithms which have been based on paradigms such as *color coding*, *divide and color*, and *representative families* [16, Chapter 5] [43]. Whereas these methods make use of explicit constructions of pseudorandom objects such as perfect hash families, universal sets, and representative sets, our algorithm makes use of algebraic-combinatorial identities. This approach was foreshadowed in [12, Theorem 2]. It is important to note that our algorithm only works for unweighted graphs (or weighted graphs with integer weights bounded by $\mathrm{poly}(n)$), while several previous algorithms work for weighted graphs. The algorithm of [20] also extends more generally to detect subgraphs of bounded treewidth.

1.4 Algebraic considerations; the potential for improvement

In Section 4 we note that our algorithms for computing special cases of (1) yield algorithms for performing arithmetic in a certain algebra $\mathcal{A}_f$ associated to f, namely the *apolar algebra* of f. We show in our Lemma 24 that the apolar algebra of the determinant is isomorphic to the diagonal subalgebra of the tensor square of the exterior algebra. This algebra was previously identified in [12] for its applications to detecting subgraphs of bounded pathwidth. By combining this observation with known algorithms for arithmetic in the exterior algebra, we derive our general algorithm for computing $\langle \det X, g\rangle$.

To obtain faster deterministic algorithms for several problems such as detecting simple cycles, we ask Question 1. Our Corollary 11 shows that $B(d,n)$ is at most $(\sqrt{27}/2)^d d$. This upper bound is witnessed by the polynomial

$$\sum_{1\le i_1<\cdots<i_d\le n}\prod_{j<k}(i_j-i_k)^2\prod_{j=1}^{d}x_{i_j}.$$

▶ Remark 4. The discrepancy between $(\sqrt{27}/2)^d d \le O(2.6^d)$ and the base of the exponent $\varphi^{2d} > 2.6^d$ in our runtimes is due to the fact that we do not know how to differentiate in nearly-linear time with respect to a basis for the above polynomial; see Further Questions (6). Instead, we compute them with respect to a larger spanning set.

1.5 Paper outline

In the next section we prove Theorems 7 and 13. We will motivate them with the running application of detecting simple cycles, giving in Corollary 14 our $\varphi^{2d}\,\mathrm{poly}(n)$-time algorithm. The rest of our applications can be found in Section 3, and follow quickly from Theorems 7

and 13, using little more than the Cauchy-Binet formula. In Section 4 we define the apolar algebra of a polynomial. We relate the apolar algebra of the determinant to the tensor square of the exterior algebra in Lemma 24, and use this to prove Theorem 25. We then explain the connection to other methods in parameterized algorithms in Section 5.

2 Computing the apolar inner product for skew circuits

We start by giving an algorithm for computing (1) in the case that g is the determinant of a symbolic matrix and f is computed by a skew arithmetic circuit C. This is a warmup for the special case when g is the determinant of a symbolic Hankel matrix.

We fix the following notation for the rest of the paper. We denote by $|C|$ the total number of gates in the circuit C. Let $I(d,k) \subseteq [d]^k$ be the set of strictly increasing sequences of length k with elements in $[d]$; when $k=0$ we include the empty sequence in this set. Given a $d \times d$ matrix X and tuples $\alpha, \beta \in I(d,k)$, we denote by $X[\alpha|\beta]$ the minor (determinant of a submatrix) of X with rows indexed by α and columns indexed by β. We declare the "empty minor" $X[\,|\,]$ to equal one. We use the convention of writing $\alpha_1, \ldots, \widehat{\alpha_i}, \ldots, \alpha_k$ to denote the sequence $\alpha_1, \ldots, \alpha_{i-1}, \alpha_{i+1}, \ldots, \alpha_k$ obtained from α by omitting α_i. We call a monomial $x_1^{a_1} \cdots x_n^{a_n}$ *square-free* if $a_i \in \{0,1\}$ for all i.

For $f \in \mathcal{S}_d^n$, $\mathrm{Diff}(f)$ denotes the vector space spanned by the partial derivatives of f of all orders (this includes f itself). For example, $\mathrm{Diff}(x_1x_2)$ is the vector space spanned by x_1x_2, x_1, x_2, and 1. The next observation is a simple bound on this quantity for determinants of symbolic matrices, and has been essentially observed several times previously (e.g. [38, Lemma 1.3]).

▶ **Proposition 5.** *Let $X = (\ell_{i,j})_{i,j\in[d]}$ be a symbolic matrix with entries in $\mathcal{S}_1^n$. Then $\mathrm{Diff}(\det X)$ is contained in the space of minors of X. Hence*

$$\dim \mathrm{Diff}(\det X) \leq \sum_{i=0}^{d} \binom{d}{i}^2 = \binom{2d}{d} < 4^d.$$

Proof. Let $\mathfrak{S}_d$ denote the symmetric group on d elements. By the Leibniz formula for the determinant and the product rule, for any $l \in [n]$,

$$\begin{aligned}\frac{\partial \det X}{\partial x_l} &= \sum_{\sigma\in\mathfrak{S}_d} \mathrm{sgn}(\sigma) \sum_{i=1}^{d} \frac{\partial \ell_{i,\sigma(i)}}{\partial x_l} \prod_{j\neq i} \ell_{j,\sigma(j)} = \sum_{1\leq i,j\leq d} \frac{\partial \ell_{i,j}}{\partial x_l} \sum_{\sigma\in\mathfrak{S}_d,\sigma(i)=j} \mathrm{sgn}(\sigma) \prod_{m\neq i} \ell_{m,\sigma(m)} \\ &= \sum_{1\leq i,j\leq d} (-1)^{i+j} \frac{\partial \ell_{i,j}}{\partial x_l} X[1,\ldots,\widehat{i},\ldots,d|1,\ldots,\widehat{j},\ldots,d].\end{aligned}$$

Note that $\frac{\partial \ell_{i,j}}{\partial x_l}$ is just a scalar. To see the last equality, consider the matrix $X^{(ij)}$ obtained by setting the (i,j)th entry of X to 1, and all other entries in the ith row of X to 0. Then $\det X^{(ij)} = \sum_{\sigma\in\mathfrak{S}_d,\sigma(i)=j} \mathrm{sgn}(\sigma) \prod_{m\neq i} \ell_{m,\sigma(m)}$, but at the same time by Laplace expansion along the ith row of $X^{(ij)}$, $\det X^{(ij)} = (-1)^{i+j} X[1,\ldots,\widehat{i},\ldots,d|1,\ldots,\widehat{j},\ldots,d]$.

This shows that the space of order-1 partial derivatives of $\det X$ is contained in the span of the degree-$(d-1)$ minors of X. That $\mathrm{Diff}(\det X)$ is contained in the space of minors of X follows by repeated application of this fact. Furthermore, since square $k \times k$ submatrices of X can be identified by pairs of elements in $I(d,k)$ (their row and column indices), the vector space spanned by all minors of X has dimension at most $\sum_{k=0}^{d} |I(d,k)|^2 = \sum_{k=0}^{d} \binom{d}{k}^2 = \binom{2d}{d}$. ◀

▶ **Lemma 6.** *Given as input a symbolic matrix $X = (\ell_{i,j})_{i,j\in[d]}$ with entries in $\mathcal{S}_1^n$, a linear combination P of minors of X, and $l \in [n]$, we can compute a representation for $\frac{\partial P}{\partial x_l}$ as a linear combination of minors of X with $4^d \operatorname{poly}(n,d)$ arithmetic operations.*

Proof. Let $P = \sum_{k=0}^{d} \sum_{\alpha,\beta\in I(d,k)} c_{\alpha,\beta} X[\alpha|\beta]$ and let $a_{i,j}^{(l)}$ be the coefficient of x_l in $\ell_{i,j}$ (so the input consists of l and the vectors $(c_{\alpha,\beta}) \in \mathbb{Q}^{\binom{2d}{d}}$, $(a_{i,j}^{(l)}) \in \mathbb{Q}^{d^2 n}$). Then by the same considerations as in the proof of Proposition 5,

$$\frac{\partial P}{\partial x_l} = \sum_{k=1}^{d} \sum_{\alpha,\beta\in I(d,k)} \sum_{1\le i,j\le k} c_{\alpha,\beta}(-1)^{i+j} a_{i,j}^{(l)} X[\alpha_1,\dots,\widehat{\alpha}_i,\dots,\alpha_k|\beta_1,\dots,\widehat{\beta}_j,\dots,\beta_k].$$

Note that for $\alpha,\beta \in I(d,k)$, the coefficient of $X[\alpha|\beta]$ in the above equals

$$\sum_{1\le i,j\le k} \sum_{\substack{\alpha',\beta'\in I(d,k+1)\\ \alpha=\alpha'_1,\dots,\widehat{\alpha'_i},\dots,\alpha'_{k+1}\\ \beta=\beta'_1,\dots,\widehat{\beta'_j},\dots,\beta'_{k+1}}} (-1)^{i+j} a_{i,j}^{(l)} c_{\alpha',\beta'}.$$

The numbers of pairs of sequences α',β' considered by the inner sum is naïvely bounded by d^4 (there are d positions in α where we could try to insert a number in $[d]$ into to get an increasing sequence, and similarly for β), and hence the coefficient of each minor can be computed with $O(d^6)$ arithmetic operations. Since there are $\binom{2d}{d}$ minors, all coefficients can be computed with the stated number of operations. ◀

▶ **Theorem 7.** *Let C be a skew arithmetic circuit computing $g \in \mathcal{S}_d^n$, and let $X = (\ell_{i,j})_{i,j\in[d]}$ be a symbolic matrix with entries in $\mathcal{S}_1^n$. Then we can compute $\langle \det X, g\rangle$ with $4^d|C|\operatorname{poly}(n,d)$ arithmetic operations.*

Proof. Say that gate v in C computes the polynomial C_v. We will compute the inner product (1) inductively: at gate v we will compute and store C_v^∂, a representation for $C_v(\frac{\partial}{\partial x_1},\dots,\frac{\partial}{\partial x_n})\det X$ as a linear combination of minors of X. C_v^∂ will be stored as a vector of length $\binom{2d}{d}$ indexed by pairs of row and column sets. At the end of the algorithm we will have computed $f(\frac{\partial}{\partial x_1},\dots,\frac{\partial}{\partial x_n})\det X$ (which by symmetry of the apolar inner product equals $\langle \det X, f\rangle$) at the output gate.

We start by computing and storing $\frac{\partial}{\partial x_l}\det X$ at input gate x_l, which by Lemma 6 can be done in $4^d \operatorname{poly}(n,d)$ time. Now suppose that gate v takes input from gates v' and v'', and that we have already computed $C_{v'}^\partial$ and $C_{v''}^\partial$. To compute C_v^∂, there are two cases to consider:

1. $C_v = x_i \cdot C_{v'}$. Then $C_v^\partial = \frac{\partial}{\partial x_i} C_{v'}(\frac{\partial}{\partial x_1},\dots,\frac{\partial}{\partial x_n})\det X = \frac{\partial}{\partial x_i} C_{v'}^\partial$. Using Lemma 6 this can be computed with $4^d \operatorname{poly}(n,d)$ operations.
2. $C_v = C_{v'} + C_{v''}$. Since differentiation is linear, $C_v^\partial = C_{v'}^\partial + C_{v''}^\partial$. Since $C_{v'}^\partial$ and $C_{v''}^\partial$ are vectors of length $\binom{2d}{d}$, it takes $\binom{2d}{d}$ operations to add them.

Hence at each gate we use at most $4^d \operatorname{poly}(n,d)$ arithmetic operations, for a total of $4^d \operatorname{poly}(n,d)|C|$. ◀

We now show how Theorem 7 can be applied to obtain a deterministic algorithm for detecting simple cycles in graphs. This motivates the following improvement.

▶ **Proposition 8.** *Let G be a graph on n vertices. We can decide in $4^d \operatorname{poly}(n)$ time if G contains a simple cycle of length d.*

Proof. Let $V \in \mathbb{Q}^{d\times n}$ be the Vandermonde matrix with $V_{i,j} = j^i$. Let $X = V \cdot \operatorname{diag}(x_1, \ldots, x_n) \cdot V^{\mathrm{T}}$. By the Cauchy-Binet formula,

$$\det X = \sum_{\alpha \in I(n,d)} V[1, \ldots, d|\alpha]^2 \prod_{i\in\alpha} x_i.$$

Since any d columns in V are linearly independent, $V[1, \ldots, d|\alpha]^2 > 0$ for all $\alpha \in I(n, d)$. Furthermore, observe that $\operatorname{tr}(A_G^d)$ has nonnegative coefficients and contains a square-free monomial if and only if G contains a simple cycle of length d. It follows that $\langle \det X, \operatorname{tr}(A_G^d)\rangle \neq 0$ if and only if G contains such a cycle. In addition, $\operatorname{tr}(A_G^d)$ can be naïvely computed by a skew circuit of size $O(dn^3)$. The theorem follows by applying Theorem 7, noting that we only perform arithmetic with $\operatorname{poly}(n)$-bit integers. ◀

Note that the (i, j)th entry in the matrix X in the proof of Proposition 8 equals $\sum_{k=1}^n k^{i+j} x_k$, and therefore X is Hankel. We now show how this additional structure can be exploited.

Fix linear forms $\ell_1, \ldots, \ell_{2d-1} \in \mathcal{S}_1^n$, and let C_d be the symbolic matrix

$$\begin{bmatrix} \ell_1 & \ell_2 & \ell_3 & \cdots & \cdots & \cdots & \ell_{2d-2} & \ell_{2d-1} \\ \ell_2 & \ell_3 & \iddots & \iddots & \iddots & \iddots & \iddots & 0 \\ \ell_3 & \iddots & \iddots & \iddots & \iddots & \iddots & \iddots & 0 \\ \vdots & \iddots & \iddots & \iddots & \iddots & \iddots & \iddots & \vdots \\ \ell_{2d-2} & \iddots & \iddots & \iddots & \iddots & \iddots & \iddots & \vdots \\ \ell_{2d-1} & 0 & 0 & \cdots & \cdots & \cdots & \cdots & 0 \end{bmatrix}. \tag{2}$$

The minors of the form $C_d[1, 2, \ldots, k|b_1, \ldots, b_k]$, where $k \leq d$ and $b_k \leq 2d - k$, are called *maximal*. For brevity we will let $[\alpha|\beta] \coloneqq C_d[\alpha|\beta]$, and if $[\alpha|\beta]$ is maximal (so $\alpha = 1, \ldots, k$) we further simplify this to $[\beta]$. Let H_d be the submatrix of C_d with row and column subscripts $1, \ldots, d$. It is readily seen that H_d is a Hankel matrix.

We will need the following fact of Conca [15, Lemma 2.1(a)]. For a subset I, we let $e(I)$ be its indicator vector.

▶ **Lemma 9.** *Let $\alpha = \alpha_1, \ldots, \alpha_t$ and $\beta = \beta_1, \ldots, \beta_t$ be sequences of positive integers. Then for all $k = 1, \ldots, t$,*

$$\sum_{I\subseteq[t],|I|=k} [\alpha + e(I)|\beta] = \sum_{J\subseteq[t],|J|=k} [\alpha|\beta + e(J)].$$

Proof. We denote by α_I the subsequence of α indexed by the set I, and by $\alpha^{\hat{I}}$ the subsequence indexed by the complement of I in $[t]$. We let $\alpha + 1 = \alpha_1 + 1, \ldots, \alpha_t + 1$.

First, expanding $[\alpha + e(I)|\beta]$ with respect to the rows indexed by $\alpha_I + 1$:

$$\sum_I [\alpha + e(I)|\beta] = \sum_I \sum_J (-1)^{|I|}(-1)^{|J|}[\alpha_I + 1|\beta_J][\alpha^{\hat{I}}|\beta^{\hat{J}}].$$

Since C_d is Hankel, $[\alpha_I + 1|\beta_J] = [\alpha_I|\beta_J + 1]$. So

$$\sum_I [\alpha + e(I)|\beta] = \sum_J \sum_I (-1)^{|I|}(-1)^{|J|}[\alpha_I|\beta_J + 1][\alpha^{\hat{I}}|\beta^{\hat{J}}] = \sum_J [\alpha|\beta + e(J)]$$

where in the final equality we recognize that the middle equation equals $[\alpha|\beta + e(J)]$ expanded with respect to the columns indexed by $\beta_J + 1$. ◀

▶ **Corollary 10.** $\mathrm{Diff}(\det H_d)$ *is contained in the space of maximal minors of* C_d*. Furthermore, the number of maximal minors of* C_d *is at most* φ^{2d}*.*

Proof. By Proposition 5, the space of partial derivatives of $\det H_d$ is spanned by the minors of C_d. We now show that the maximal minors of C_d span the space of minors of H_d. We will follow the proof of Corollary 2.2 in [15].

Let $[\alpha_1, \ldots, \alpha_t | \beta_1, \ldots, \beta_t]$ be a minor of C_d, where $t \leq d$ and $\alpha_1 = 1$. Note that any minor of H_d can be expressed in this form by shifting the corresponding submatrix in C_d up and to the right. We also assume α and β are strictly increasing sequences (if this is not the case then $[\alpha|\beta]$ vanishes). We now give an inductive procedure that expresses $[\alpha|\beta]$ as a linear combination of maximal minors.

If $\alpha_t = t$, then this minor is maximal and we are done. Otherwise, let h be the smallest index where $\alpha_h > h$. We now apply Lemma 9 to the minor $[\alpha_1, \ldots, \alpha_{h-1}, \alpha_h - 1, \ldots, \alpha_t - 1 | \beta]$ with $k = t - h + 1$. Doing so we obtain an expression for $[\alpha|\beta]$ as a linear combination of minors of C_d, each of which in turn have a larger "h." We conclude by induction.

Finally, note that the number of maximal minors of degree k is $|I(2d-k, k)| = \binom{2d-k}{k}$. Hence the total number of maximal minors equals $\sum_{k=0}^{d} \binom{2d-k}{k} < \varphi^{2d}$. In the last step we used the facts that the dth Fibonacci number satisfies $F_d = \sum_{k=0}^{\lfloor \frac{d-1}{2} \rfloor} \binom{d-k-1}{k}$, and that $F_d \leq \varphi^{d-1}$. ◀

The next observation was noted in [37, Theorem 43].

▶ **Corollary 11.**

$$\dim \mathrm{Diff}(\det H_d) \leq \sum_{i=0}^{d} \min \left\{ \binom{d+i}{d-i}, \binom{2d-i}{i} \right\} \leq (\sqrt{27}/2)^d \cdot d.$$

Proof. By Corollary 10, for $i \leq d/2$ the degree-i piece of $\mathrm{Diff}(H_d)$ has dimension at most $\binom{2d-i}{i}$. We conclude by the fact that $\mathrm{Diff}(H_d)_i \cong \mathrm{Diff}(H_d)_{d-i}$, i.e., the sequence of dimensions of space of partial derivatives is symmetric about $d/2$ [29, Definition 1.9]. The inequality on the right follows from Stirling's formula. ◀

▶ **Lemma 12.** *Given as input a linear combination* P *of maximal minors of* C_d *and* $l \in [n]$*, we can compute a representation for* $\frac{\partial P}{\partial x_l}$ *as a linear combination of maximal minors of* C_d *with* $\varphi^{2d} \operatorname{poly}(n, d)$ *arithmetic operations.*

Proof. For brevity we will write $[\alpha]$ for the minor $C_d[1, \ldots, |\alpha| | \alpha]$. Let $P = \sum_{k=0}^{d} \sum_{\beta \in I(2d-k,k)} c_\beta [\beta]$, and say that the coefficient of x_l in $(C_d)_{i,j}$ is $a_{i,j}^{(l)}$. As in Lemma 6,

$$\frac{\partial P}{\partial x_l} = \sum_{k=1}^{d} \sum_{\beta \in I(2d-k,k)} c_\beta \sum_{1 \leq i,j \leq k} (-1)^{i+\beta_j} a_{i,\beta_j}^{(l)} [1, \ldots, \widehat{i}, \ldots, k | \beta_1, \ldots, \widehat{\beta}_j, \ldots, \beta_k].$$

Note that the only minors with nonzero coefficient in this expression are of the form $[1, \ldots, \widehat{i}, \ldots, k|\gamma]$ for $k \in [d], i \in [k]$ and $\gamma \in I(2d-k, k-1)$. Call the coefficient of this minor in the above $b(i, \gamma)$. Then

$$b(i, \gamma) = \sum_{1 \leq j \leq k} \sum_{\substack{\beta \in I(2d-k,k) \\ \gamma = (\beta_1, \ldots, \widehat{\beta}_j, \ldots, \beta_k)}} c_\beta (-1)^{i+\beta_j} a_{i,\beta_j}^{(l)}.$$

The number of sequences β considered by the inner sum is at most $O(d^2)$, and hence $b(i,\gamma)$ can be computed with $O(d^3)$ additions and multiplications. We can thus compute

$$\frac{\partial P}{\partial x_l} = \sum_{k=1}^{d}\sum_{i=1}^{k}\sum_{\gamma\in I(2d-k,k-1)} b(i,\gamma)[1,\ldots,\widehat{i},\ldots,k|\gamma] \tag{3}$$

with $d^4\sum_{k=1}^{d}|I(2d-k,k-1)| \le \varphi^{2d}\operatorname{poly}(n,d)$ arithmetic operations. Note that this expresses $\frac{\partial P}{\partial x_l}$ as a linear combination of minors that are not necessarily maximal. We now fix this.

We first claim that for all $i\in[k]$ and $\beta\in I(2d-k,k-1)$,

$$[1,\ldots,\widehat{i},\ldots,k|\beta] = \sum_{J\subseteq[k-1],|J|=k-i}[e(J)+(1,\ldots,k-1)|\beta]$$

where $e(J)$ is the indicator vector of the set J. This holds since when $J=\{i,\ldots,k-1\}$, $e(J)+(1,\ldots,k-1)=(1,\ldots,\widehat{i},\ldots,k)$, and for all other J, $e(J)+(1,\ldots,k-1)$ will have a repeated value and hence $[e(J)+(1,\ldots,k-1)|\beta]=0$.

Given this claim, it follows from Lemma 9 that

$$[1,\ldots,\widehat{i},\ldots,k|\beta] = \sum_{J\subseteq[k-1],|J|=k-i}[\beta+e(J)],$$

and so letting Q_k be the degree-k part of Equation 3,

$$Q_k = \sum_{i=1}^{k+1}\sum_{\beta\in I(2d-k-1,k)} b(i,\beta)\sum_{J\subseteq[k],|J|=k+1-i}[\beta+e(J)].$$

We now show how to efficiently compute the coefficients of the maximal minors in this expression from the already computed $b(i,\gamma)$'s.

Let $0\le k\le d-1$ be fixed. For $\beta\in I(2d-k-1,k)$ and integers i,j where $0\le i\le j\le k$, let $D(\beta,i,j,k)\subseteq\{0,1\}^k$ be the set of binary vectors of length k containing exactly i ones, whose last $k-j$ entries are zero, and whose summation with β is strictly increasing everywhere except possibly at positions j and $j+1$ (that is, we may have $w_j+\beta_j=w_{j+1}+\beta_{j+1}$). Define

$$A^k(i,j) \coloneqq \sum_{\beta\in I(2d-k-1,k)} b(k+1-i,\beta)\sum_{w\in D(\beta,i,j,k)}[\beta+w].$$

Note that $\sum_{i=0}^{k}A^k(i,k)=Q_k$, so it suffices to show how to compute $A^k(i,j)$ for all i,j. We do this with a dynamic program. When we store $A^k(i,j)$ we will store all coefficients of maximal minors arising in the above definition, even though such a minor might contain a repeated column and hence equal zero. The minors arising in this definition are specified by sequences of length k with maximum value $2d-k$ that are strictly increasing everywhere but possibly at one position. Hence the number of such sequences is at most $k\binom{2d-k}{k}$.

For the base cases, we have

$$A^k(0,j) = \sum_{\beta\in I(2d-k-1,k)} b(k+1,\beta)[\beta],$$

$$A^k(i,i) = \sum_{\beta\in I(2d-k-1,k)} b(k+1-i,\beta)[\beta+e(\{1,\ldots,i\})].$$

Now suppose we have computed $A^k(i, j-1)$ and $A^k(i-1, j-1)$. Then

$$\begin{aligned} A^k(i,j) &= \sum_{\beta \in I(2d-k-1,k)} b(k+1-i,\beta) \left(\sum_{\substack{w \in B(\beta,i,j,k),\\ w_j=0}} [\beta + w] + \sum_{\substack{w \in D(\beta,i,j,k),\\ w_j=1}} [\beta + w] \right) \\ &= \sum_{\beta \in I(2d-k-1,k)} b(k+1-i,\beta) \sum_{\substack{w \in D(\beta,i,j-1,k),\\ \beta+w \text{ is strictly increasing}}} [\beta + w] \\ &\quad + \sum_{\beta \in I(2d-k-1,k)} b(k+1-i,\beta) \sum_{w \in D(\beta,i-1,j-1,k)} [\beta + w + e(\{j\})]. \end{aligned}$$

The first part of the sum can be computed from $A^k(i, j-1)$ by setting the coefficient of any maximal minor with a repeated column equal zero, and the second sum can be computed from $A^k(i-1, j-1)$ by setting the coefficient of $[\beta]$ to that of $[\beta - e(\{j\})]$. Hence $A^k(i, j)$ can be computed with $O(k\binom{2d-k}{k})$ arithmetic operations. It follows that we can represent $\frac{\partial P}{\partial x_l} = \sum_{i=0}^{d-1} Q_i$ in the space of maximal minors using $\varphi^{2d} \operatorname{poly}(n, d)$ arithmetic operations. ◀

With this we have the following analog of Theorem 7. We omit the proof as it is almost exactly the same, we just work in the space of maximal minors rather than minors, using Lemma 12 to differentiate instead of Lemma 6.

▶ **Theorem 13.** *Let C be a skew arithmetic circuit computing $g \in \mathcal{S}_d^n$, and let $X = (\ell_{i,j})_{i,j\in[d]}$ be a symbolic Hankel matrix with entries in $\mathcal{S}_1^n$. Then we can compute $\langle \det X, g \rangle$ with $\varphi^{2d}|C| \operatorname{poly}(n, d)$ arithmetic operations. Here $\varphi := \frac{1+\sqrt{5}}{2}$ is the golden ratio.*

▶ **Corollary 14.** *Let G be a graph on n vertices. We can decide in $\varphi^{2d} \operatorname{poly}(n)$ time if G contains a simple cycle of length d.*

Proof. Let $V \in \mathbb{Q}^{d\times n}$ be the Vandermonde matrix with $V_{i,j} = j^i$, and $X = V \cdot \operatorname{diag}(x_1, \ldots, x_n) \cdot V^{\mathrm{T}}$. By the same argument of Proposition 8, $\langle \det X, \operatorname{tr}(A_G^d) \rangle \neq 0$ if and only if G contains a simple cycle of length d. Note that the (i, j)th entry in X equals $\sum_{k=1}^n k^{i+j} x_k$, and therefore X is Hankel. We conclude by applying Theorem 13 to compute $\langle \det X, \operatorname{tr}(A_G)^d \rangle$, as $\operatorname{tr}(A_G^d)$ can be computed by a skew circuit of size $\operatorname{poly}(n)$. ◀

▶ Remark 15. This algorithm extends to detecting subgraphs of bounded pathwidth on d vertices by using the construction of the subgraph generating polynomial given in [12, Appendix B].

3 Applications

In this section we give our applications of Theorems 1 and 2.

▶ **Corollary 16.** *Given matrices $A_1, \ldots, A_n \in \mathbb{Q}^{d\times d}$, we can decide if their span contains an invertible matrix in time $4^d \operatorname{poly}(N)$, where N denotes the size of the input.*

Proof. Let $X = \sum_{i=1}^n x_i A_i$. First note that $\operatorname{span}(A_1, \ldots, A_n)$ contains an invertible matrix if and only if $\det X \not\equiv 0$. Writing $\det X = \sum_{\alpha\in[d]^n} c_\alpha x^\alpha$ for some coefficients c_α (at least one of which will be nonzero iff the answer is "yes"), observe that $\langle \det X, \det X \rangle = \sum_\alpha c_\alpha^2 \alpha!$. It follows that $\operatorname{span}(A_1, \ldots, A_n)$ contains an invertible matrix if and only if this quantity is nonzero.

It is shown in [35] that $\det_d$ can be expressed as a skew circuit of size $O(d^4)$, and the construction of this circuit is linear in the output size. Hence we can construct a circuit for $\det X$ by replacing the input variable x_{ij} in this circuit with the (i,j)th entry of X. The theorem follows by applying Theorem 7 to the matrix X and this circuit, noting that all numbers have bit-length $\mathrm{poly}(N)$ throughout the algorithm. ◀

▶ **Corollary 17.** *Suppose we are given a matrix $A \in \mathbb{Q}^{km \times kn}$, where $n \geq m$, representing a matroid M with groundset $[kn]$, and a partition π of $[kn]$ into parts of size k. Then we can decide if the union of any m parts in π are independent in M in time $4^{km}\,\mathrm{poly}(N)$, where N is the size of the input.* [3]

Proof. Let $g \coloneqq (\sum_{S \in \pi} \prod_{i \in S} x_i)^m$. It is easily seen that the square-free monomials appearing in g correspond to unions of m elements in π, and that g can be computed by a skew circuit of size $\mathrm{poly}(n)$. Next, let $X = A \cdot \mathrm{diag}(x_1, \ldots, x_n) \cdot A^{\mathrm{T}}$. By the Cauchy-Binet formula,

$$\det X = \sum_{S \in \mathrm{Bases}(M)} \det(B_S)^2 \prod_{i \in S} x_i,$$

Note that the same monomial appears in the expansion of g and $\det X$ exactly when there is such an independent set in M, and then since g and $\det X$ have non-negative coefficients, $\langle \det X, g \rangle \neq 0$ if and only if an independent set in M is the union of m blocks in π. We conclude by applying Theorem 7. ◀

Using the same trick as in [36] we can use Corollary 17 to solve the k-matroid intersection problem.

▶ **Corollary 18** (k-Matroid Intersection). *Suppose we are given matrices $B_1, \ldots, B_k \in \mathbb{Q}^{m \times n}$ representing matroids $M_1, \ldots, M_k$ with the common groundset $[n]$. We can decide if $M_1, \ldots, M_k$ share a common base in time $4^{km}\,\mathrm{poly}(N)$, where N is the size of the input.*

Proof. Let $M = \bigoplus_{i=1}^{k} B_k$ be the direct sum of the input matrices. We first partition $[kn]$ into n parts of size k as follows: for $i \in [n]$, let $S_i \coloneqq \{i, i+n, i+2n, \ldots, i+kn\}$. If a union of m of the blocks $S_1, \ldots, S_n$ are independent in the matroid represented by M, then $M_1, \ldots, M_k$ share a common base. Conversely, if these matroids share a common base, some union of the S_i's are independent in the matroid represented by M. We conclude by applying Corollary 17 to the matrix $M \in \mathbb{Q}^{km \times kn}$ and the partition $S_1, \ldots, S_n$. ◀

Finally, we have our applications of Theorem 13. These follow immediately by a reduction given in [11, Theorem 1] to the following "square-free monomial detection" algorithm.

▶ **Corollary 19.** *Let $g \in \mathbb{Q}[x_1, \ldots, x_n]_d$ be a homogeneous degree-d polynomial with nonnegative coefficients, computed by a skew arithmetic circuit C. Given as input C, we can decide in deterministic $\varphi^{2d}|C|\,\mathrm{poly}(n)$ time whether g contains a degree-d square-free monomial.*

Proof. Let $V \in \mathbb{Q}^{d \times n}$ be the Vandermonde matrix with $V_{i,j} = j^i$, and $X = V \cdot \mathrm{diag}(x_1, \ldots, x_n) \cdot V^{\mathrm{T}}$. By the Cauchy-Binet formula,

$$\det X = \sum_{S \subseteq \binom{[n]}{d}} \det(V_S)^2 \prod_{i \in S} x_i.$$

[3] Similar to the Theorem 1.1 of [36], this algorithm can be modified to work for finite fields of sufficiently large size with the addition of randomness.

Since any d columns in V are linearly independent, $\det(V_S)^2 > 0$ for all S. It follows that since g has nonnegative coefficients, $\langle \det X, g\rangle \neq 0$ if and only if g contains a square-free monomial. Note that the (i,j)th entry in X equals $\sum_{k=1}^{n} k^{i+j} x_k$, and therefore X is Hankel. The theorem follows by invoking Theorem 13. ◀

By [11, Theorem 1], we then have:

▶ **Corollary 20.** *The following problems admit deterministic algorithms running in time* $\varphi^{2d}\operatorname{poly}(n)$*:*

1. *Deciding whether a given directed n-vertex graph has a directed spanning tree with at least d non-leaf vertices,*
2. *Deciding whether a given edge-colored, directed n-vertex graph has a directed spanning tree containing at least d colors,*
3. *Deciding whether a given planar, edge-colored, directed n-vertex graph has a perfect matching containing at least d colors.*

4 A general algorithm: the apolar algebra of the determinant

4.1 Algebraic preliminaries

Let $\mathcal{R}^n := \mathbb{Q}[\partial_1, \ldots, \partial_n]$ be the ring of partial differential operators. Elements of this ring are just multivariate polynomials in the variables $\partial_1, \ldots, \partial_n$. For an n-tuple $\alpha \in \mathbb{N}^n$, we let ∂^α be the monomial $\partial_1^{\alpha_1} \cdots \partial_n^{\alpha_n}$, and let $|\alpha| = \sum_{i=1}^{n} \alpha_i$. For $h \in \mathcal{R}$ and $f \in \mathcal{S}$, we denote by $h \circ f$ the result of applying the differential operator h to f. For example,

$$(3 \cdot \partial_1\partial_2 + \partial_1^2) \circ x_1^2 x_2 = 3 \cdot \partial_1\partial_2 \circ x_1^2 x_2 + \partial_1^2 \circ x_1^2 x_2 = 6x_1 + 2x_2.$$

When h and f are homogeneous of the same degree, $h \circ f$ is a scalar. In this case $f(\partial_1, \ldots, \partial_n) \circ g = \langle f, g\rangle$, so computing $h \circ f$ is equivalent to computing the apolar inner product.

▶ **Definition 21.** *For $f \in \mathcal{S}_d^n$, we define* $\operatorname{Ann}(f)$ *as the ideal of elements in $\mathcal{R}^n$ annihilating f under differentiation. We define the* apolar algebra $\mathcal{A}_f$ *as the quotient* $\mathcal{R}^n / \operatorname{Ann}(f)$.

In other words, $\mathcal{A}_f$ is the ring of representatives of equivalence classes of differential operators subject to the equivalence relation $\sim$, where $h \sim h'$ if and only if $h \circ f = h' \circ f$. It follows that there is a vector space isomorphism $\mathcal{J}$ between $\mathcal{A}_f$ and $\operatorname{Diff}(f)$, sending $h \in \mathcal{A}_f$ to $h \circ f$. In particular, $(\mathcal{A}_f)_i \cong \operatorname{Diff}(f)_{d-i}$, where we denote by $(\mathcal{A}_f)_i$ the vector space of degree-i elements in $\mathcal{A}_f$.

▶ Remark 22. Multiplication in $\mathcal{A}_f$ corresponds to differentiating by f: for $h_1, h_2 \in \mathcal{A}_f$, $\mathcal{J}(h_1 \cdot h_2) = h_1 \circ (h_2 \circ f)$. It follows that Lemmas 6 and 12 are algorithms for multiplication by ∂_l in $\mathcal{A}_{\det X}$, with respect to the spanning sets of $\mathcal{A}_{\det X}$ given by the inverse images of the minors (or maximal minors) of X.

▶ **Definition 23.** $\Lambda(\mathbb{Q}^n) \otimes \Lambda(\mathbb{Q}^n)$ *is the algebra with the basis of formal variables* $\{(I|J) : I, J \subseteq [n]\}$*, and where multiplication is given by extending bilinearly the rule*

$$(I|J) \cdot (I'|J') = \begin{cases} 0 \text{ if } I \cap I' \neq \emptyset \text{ or } J \cap J' \neq \emptyset, \\ \operatorname{sgn}(I, I') \operatorname{sgn}(J, J')(I \cup I' | J \cup J') \text{ else} \end{cases}$$

where $\operatorname{sgn}(I, I') = (-1)^{|\{i \in I, i' \in I' : i > i'\}|}$.

▶ **Lemma 24.** *$\mathcal{A}_{\det_n}$ is isomorphic to the subalgebra of $\Lambda(\mathbb{Q}^n) \otimes \Lambda(\mathbb{Q}^n)$ generated by $\{v \otimes v : v \in \Lambda(\mathbb{Q}^n)\}$.*

Proof. We first claim that the set of monomials of the form $(I|J) \coloneqq \partial_{I_1,J_1} \cdots \partial_{I_k,J_k}$, where $I, J \in I(n,k)$ and $0 \leq k \leq n$, are a basis for $\mathcal{A}_{\det_n}$. This follows from the fact that there are $\binom{2n}{n}$ such monomials, $\dim \mathrm{Diff}(\det_n) = \binom{2n}{n}$, and the polynomials of the form $(I|J) \circ \det_n$ are linearly independent. The latter claim can be seen by noting that if $(I|J) \neq (I'|J')$, $(I|J) \circ \det_n$ and $(I'|J') \circ \det_n$ have disjoint sets of monomials appearing in their expansion.

Next we claim that the product of two basis elements $(I|J)$ and $(I'|J')$ is given by the rule

$$(I|J) \cdot (I'|J') = \begin{cases} 0 \text{ if } I \cap I' \neq \emptyset \text{ or } J \cap J' \neq \emptyset, \\ \operatorname{sgn}(I, I') \operatorname{sgn}(J, J')(I \cup I'|J \cup J') \text{ else} \end{cases}$$

where $\operatorname{sgn}(I, I')$ denotes the sign of the permutation that brings the sequence $I_1, \ldots, I_{k'}$ into increasing order, and $I \cup I'$ denotes the resulting sorted sequence. Indeed, if $I \cap I' \neq \emptyset$, then $(I|J)(I'|J')$ is divisible by the product of two variables that have the same first (row) index. But then $(I|J)(I'|J') \circ \det_n = 0$, since all monomials in the determinant have different row indices. The second case follows from the fact that for $I, J \in I(n,k)$ and $\tau \in \mathfrak{S}_k$, $(I|J) \circ \det_n = \operatorname{sgn} \tau^{-1} \cdot (\tau(I)|J) \circ \det_n$, which follows from the Leibniz formula for the determinant.

It follows from these observations that $\mathcal{A}_{\det_n}$ is the claimed subalgebra of $\Lambda(\mathbb{Q}^n) \otimes \Lambda(\mathbb{Q}^n)$. ◀

▶ **Theorem 25.** *Let C be an arithmetic circuit computing $g \in \mathcal{S}_d^n$, and let $X = (\ell_{i,j})_{i,j \in [d]}$ be a symbolic matrix with entries in $\mathcal{S}_1^n$. Then we can compute $\langle \det X, g \rangle$ with $2^{\omega d}|C| \operatorname{poly}(n,d)$ arithmetic operations, where $\omega < 2.373$ is the exponent of matrix multiplication.*

Proof. Assume that the entries of X are linearly independent. If this is not the case, add a new variable $x_{i,j}$ to the (i,j)th entry of X. Note that since these variables do not appear in g, this does not change the value of $\langle \det X, g \rangle$.

Let $A : \mathcal{S}_1^n \to \mathbb{Q}[y_{1,1}, \ldots, y_{d,d}]_1$ be the linear transformation sending the linear form $X_{i,j}$ to the variable $y_{i,j}$. By [19, Corollary 3.1], $\langle \det X, g \rangle = \langle \det Y, g((A^{-1})^T(x_1, \ldots, x_n)) \rangle$, where $Y_{i,j} = y_{i,j}$. So we will first modify C by applying the linear transformation $(A^{-1})^T$ to the input gates, obtaining a new circuit C'. We then will evaluate C' over $\mathcal{A}_{\det Y}$, using the monomial basis of Lemma 24. Additions in $\mathcal{A}_{\det Y}$ can be done in time linear in the number of basis elements, which is bounded above by 4^d. By identifying $\mathcal{A}_{\det_d}$ with the subalgebra of diagonal elements in $\Lambda(\mathbb{Q}^d) \otimes \Lambda(\mathbb{Q}^d)$, and using the $2^{\omega d} \operatorname{poly}(d)$-time algorithm of [42, Theorem 14] for multiplying elements in $\Lambda(\mathbb{Q}^d) \otimes \Lambda(\mathbb{Q}^d)$, we can multiply elements in $\mathcal{A}_{\det_d}$ with $2^{\omega d} \operatorname{poly}(d)$ operations. Note that the highest degree element in this basis is $q \coloneqq \partial_{1,1}\partial_{2,2} \cdots \partial_{d,d}$. The output gate of of C' therefore equals $(A^{-1})^T \cdot g \bmod \operatorname{Ann}(\det Y) = \frac{\langle \det X, g \rangle q}{\langle \det Y, q \rangle} = \langle \det X, g \rangle q$. ◀

5 Totally Multilinear Polynomials and Previous Methods

We now explain how previous algorithms relate to answers to Question 1. Recall for comparison that Proposition 10 implies that $B(n,d) \leq O(2.6^d)$, and is witnessed by the polynomial

$$f = \sum_{1 \leq i_1 < \cdots < i_d \leq n} \prod_{j<k} (i_j - i_k)^2 \prod_{j=1}^{d} x_{i_j}.$$

Furthermore, we showed in Lemma 12 that given an element of $\mathrm{Diff}(f)$, the linear map $\partial_i : \mathrm{Diff}(f) \to \mathrm{Diff}(f)$ could be computed in time $\varphi^{2d}\,\mathrm{poly}(n)$ with respect to the spanning set of maximal minors.

5.1 Color coding

Let $\mathcal{F}$ be an (n, d) *perfect hash family*, that is, a family of functions from $[n]$ to $[d]$ such that for any subset of $[n]$ of size d, some function in $\mathcal{F}$ is injective on $[d]$. It can be shown by a straightforward random argument that there exists such an $\mathcal{F}$ with size at most $e^d\,\mathrm{poly}(n)$ [1].

Now for $\pi \in \mathcal{F}$ and $i \in [d]$, define the linear forms $\ell_{\pi,i} = \sum_{j \in \pi^{-1}(i)} x_j$. Let $f = \sum_{\pi \in \mathcal{F}} \prod_{i=1}^{d} \ell_{\pi,i}$. Then from the definition of a perfect hash family it follows that $f \in \mathcal{T}_{n,d}$. As the space of partial derivatives of a product of d linear forms has dimension at most 2^d, $\dim \mathrm{Diff}(f) \leq |\mathcal{F}| 2^d \leq (2e)^d\,\mathrm{poly}(n)$.

Explicitly, $\mathrm{Diff}(f)$ is spanned by $\prod_{i \in S} \ell_{\pi,i}$ for all $S \subseteq [d]$ and $\pi \in \mathcal{F}$. In addition, the operator ∂_j can efficiently be computed with respect to this spanning set; this follows from the fact that $\partial_j \prod_{i \in S} \ell_{\pi,i} = \prod_{i \in S : j \notin \pi^{-1}(i)} \ell_{\pi,i}$ (i.e., the matrix representing ∂_i is sparse). Hence color coding can be interpreted in our framework as using the polynomial f.

5.2 Waring rank

In [37] an improvement to the color-coding construction was given. Let $\mathcal{F}$ be an $(n, d, 1.55d)$-*splitter*, that is, a family of functions from $[n]$ to $[1.55d]$ such that for any subset S of $[1.55d]$ of size d, there exists some $\pi \in \mathcal{F}$ that is injective on S.

Let $e_{n,d}$ denote the elementary symmetric polynomial of degree d in n variables. For $\pi \in \mathcal{F}$ and $i \in [1.55d]$, define the linear forms $\ell_{\pi,i} = \sum_{j \in \pi^{-1}(i)} x_j$. Let

$$f = \sum_{\pi \in \mathcal{F}} e_{1.55d,d}(\ell_{\pi,1}, \ldots, \ell_{\pi,1.55d}).$$

Since $\mathcal{F}$ is a splitter it follows that $f \in \mathcal{T}_{n,d}$. By using bounds on $|\mathcal{F}|$ and the Waring rank of $e_{n,d}$, it was shown in [37][Theorem 7] that the Waring rank of f is at most $4.075^d\,\mathrm{poly}(n)$. Since in general $\dim \mathrm{Diff}(f) \leq \mathbf{R}(f)/(d+1)$, we conclude that $B(n,d) \leq \dim \mathrm{Diff}(f) \leq 4.075^d\,\mathrm{poly}(n)$.

5.3 Abelian 2 Groups

Let k be a field of characteristic 2 of size at least n, and let A be a $d \times n$ matrix, any d columns of which are linearly independent. It was shown in Section 3.3 of [37] that the polynomial $f = \sum_{S \in \binom{[n]}{d}} \det(A_S)^2 \prod_{i \in S} x_i$ has Waring rank at most $2^d - 1$, and hence $\dim \mathrm{Diff}(f) \leq (d+1)(2^d - 1)$.

6 Further questions

1. We showed that $2^d \leq B(n,d) < (\sqrt{27}/2)^d d$. Can these bounds be improved?
2. Let X be a generic Hankel matrix. Can the bound $\dim \mathrm{Diff}(\det X) \leq (\sqrt{27}/2)^d d$ be improved? We suspect that the base of the exponent is optimal.
3. Let X be a generic Hankel matrix. Can the linear map $A_i : \mathrm{Diff}(\det X) \to \mathrm{Diff}(\det X)$ given by differentiation with respect to the any variable x_i be computed in linear time with respect to a spanning set of size $(\sqrt{27}/2)^d d$ (rather than φ^{2d})?
4. Do the methods of [21, 43, 40] have an interpretation in our framework?

References

1 Noga Alon, Raphael Yuster, and Uri Zwick. Color-coding. *Journal of the ACM (JACM)*, 42(4):844–856, 1995.

2 Nima Anari and Shayan Oveis Gharan. A generalization of permanent inequalities and applications in counting and optimization. In *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing*, pages 384–396. ACM, 2017.

3 Nima Anari, Shayan Oveis Gharan, and Alireza Rezaei. Monte Carlo Markov chain algorithms for sampling strongly Rayleigh distributions and determinantal point processes. In *Conference on Learning Theory*, pages 103–115, 2016.

4 Nima Anari, Shayan Oveis Gharan, and Cynthia Vinzant. Log-concave polynomials, entropy, and a deterministic approximation algorithm for counting bases of matroids. In *2018 IEEE 59th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 35–46. IEEE, 2018.

5 Nima Anari, Shayan Oveis Gharan, Amin Saberi, and Mohit Singh. Nash social welfare, matrix permanent, and stable polynomials. In *8th Innovations in Theoretical Computer Science Conference (ITCS 2017)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2017.

6 Vikraman Arvind, Abhranil Chatterjee, Rajit Datta, and Partha Mukhopadhyay. Fast exact algorithms using Hadamard product of polynomials. In Arkadev Chattopadhyay and Paul Gastin, editors, *39th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS 2019, December 11-13, 2019, Bombay, India*, volume 150 of *LIPIcs*, pages 9:1–9:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019. doi:10.4230/LIPIcs.FSTTCS.2019.9.

7 Vikraman Arvind, Abhranil Chatterjee, Rajit Datta, and Partha Mukhopadhyay. On explicit branching programs for the rectangular determinant and permanent polynomials. *Chic. J. Theor. Comput. Sci.*, 2020, 2020. URL: http://cjtcs.cs.uchicago.edu/articles/2020/2/contents.html.

8 Alexander I Barvinok. New algorithms for linear-matroid intersection and matroid k-parity problems. *Mathematical Programming*, 69(1-3):449–470, 1995.

9 Alexander I Barvinok. Two algorithmic results for the traveling salesman problem. *Mathematics of Operations Research*, 21(1):65–84, 1996.

10 Andreas Björklund. Determinant sums for undirected hamiltonicity. In *51th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2010, October 23-26, 2010, Las Vegas, Nevada, USA*, pages 173–182, 2010. doi:10.1109/FOCS.2010.24.

11 Cornelius Brand. Patching colors with tensors. In *27th Annual European Symposium on Algorithms, ESA 2019, September 09-11, 2019, Munich, Germany*, 2019.

12 Cornelius Brand, Holger Dell, and Thore Husfeldt. Extensor-coding. In *Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2018, Los Angeles, CA, USA, June 25-29, 2018*, pages 151–164, 2018. doi:10.1145/3188745.3188902.

13 Peter Bürgisser, Christian Ikenmeyer, and Greta Panova. No occurrence obstructions in geometric complexity theory. *Journal of the American Mathematical Society*, 32(1):163–193, 2019.

14 Luca Chiantini, Jonathan D Hauenstein, Christian Ikenmeyer, Joseph M Landsberg, and Giorgio Ottaviani. Polynomials and the exponent of matrix multiplication. *Bulletin of the London Mathematical Society*, 50(3):369–389, 2018.

15 Aldo Conca. Straightening law and powers of determinantal ideals of Hankel matrices. *Advances in Mathematics*, 138(2):263–292, 1998.

16 Marek Cygan, Fedor V. Fomin, Lukasz Kowalik, Daniel Lokshtanov, Daniel Marx, Marcin Pilipczuk, Michal Pilipczuk, and Saket Saurabh. *Parameterized Algorithms*. Springer, 2015. doi:10.1007/978-3-319-21275-3.

17 Marek Cygan, Harold N Gabow, and Piotr Sankowski. Algorithmic applications of baur-strassen's theorem: Shortest cycles, diameter, and matchings. *Journal of the ACM (JACM)*, 62(4):1–30, 2015.

18 Jack Edmonds. Systems of distinct representatives and linear algebra. *J. Res. Nat. Bur. Standards Sect. B*, 71(4):241–245, 1967.

19 Richard Ehrenborg and Gian-Carlo Rota. Apolarity and Canonical Forms for Homogeneous Polynomials. *European Journal of Combinatorics*, 14(3):157–181, 1993. doi:10.1006/eujc.1993.1022.

20 Fedor V. Fomin, Daniel Lokshtanov, Fahad Panolan, and Saket Saurabh. Efficient computation of representative families with applications in parameterized and exact algorithms. *J. ACM*, 63(4):29:1–29:60, 2016. doi:10.1145/2886094.

21 Fedor V Fomin, Daniel Lokshtanov, and Saket Saurabh. Efficient computation of representative sets with applications in parameterized and exact algorithms. In *Proceedings of the twenty-fifth annual ACM-SIAM symposium on Discrete algorithms*, pages 142–151. SIAM, 2014.

22 Ankit Garg, Leonid Gurvits, Rafael Oliveira, and Avi Wigderson. Operator scaling: theory and applications. *Foundations of Computational Mathematics*, pages 1–68, 2019.

23 David G Glynn. Permanent formulae from the Veronesean. *Designs, codes and cryptography*, 68(1-3):39–47, 2013.

24 Leonid Gurvits. Classical deterministic complexity of Edmonds' problem and quantum entanglement. In *Proceedings of the Thirty-fifth Annual ACM Symposium on Theory of Computing*, STOC '03, pages 10–19, New York, NY, USA, 2003. ACM. doi:10.1145/780542.780545.

25 Leonid Gurvits. On the complexity of mixed discriminants and related problems. In *International Symposium on Mathematical Foundations of Computer Science*, pages 447–458. Springer, 2005.

26 Leonid Gurvits. Hyperbolic polynomials approach to Van der Waerden/Schrijver-Valiant like conjectures: sharper bounds, simpler proofs and algorithmic applications. In *Proceedings of the thirty-eighth annual ACM symposium on Theory of computing*, pages 417–426. ACM, 2006.

27 Leonid Gurvits. Van der waerden/schrijver-valiant like conjectures and stable (aka hyperbolic) homogeneous polynomials: one theorem for all. *The electronic journal of combinatorics*, 15(1):66, 2008.

28 Gregory Z. Gutin, Felix Reidl, Magnus Wahlström, and Meirav Zehavi. Designing deterministic polynomial-space algorithms by color-coding multivariate polynomials. *J. Comput. Syst. Sci.*, 95:69–85, 2018. doi:10.1016/j.jcss.2018.01.004.

29 Anthony Iarrobino and Vassil Kanev. *Power sums, Gorenstein algebras, and determinantal loci*. Springer Science & Business Media, 1999.

30 Valentine Kabanets and Russell Impagliazzo. Derandomizing polynomial identity tests means proving circuit lower bounds. *Computational Complexity*, 13(1-2):1–46, 2004.

31 Ioannis Koutis and Ryan Williams. Limits and applications of group algebras for parameterized problems. In *Automata, Languages and Programming, 36th International Colloquium, ICALP 2009, Rhodes, Greece, July 5-12, 2009, Proceedings, Part I*, pages 653–664, 2009. doi:10.1007/978-3-642-02927-1_54.

32 Joseph M Landsberg. Tensors: geometry and applications. *Representation theory*, 381:402, 2012.

33 François Le Gall. Powers of tensors and fast matrix multiplication. In *Proceedings of the 39th international symposium on symbolic and algebraic computation*, pages 296–303, 2014.

34 László Lovász. On determinants, matchings, and random algorithms. *Fundamentals of Computation Theory*, pages 565–574, 1979.

35 Meena Mahajan and V Vinay. A combinatorial algorithm for the determinant. In *In Proceedings of the 8th Annual ACM-SIAM Symposium on Discrete Algorithms*. Citeseer, 1997.

36 Dániel Marx. A parameterized view on matroid optimization problems. *Theor. Comput. Sci.*, 410(44):4471–4479, 2009. doi:10.1016/j.tcs.2009.07.027.

37 Kevin Pratt. Waring rank, parameterized and exact algorithms. In *60th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2019, Baltimore, MD, USA, November 9-12, 2019*, 2019.

38 Masoumeh Sepideh Shafiei. Apolarity for determinants and permanents of generic matrices. *Journal of Commutative Algebra*, 7(1):89–123, 2015.

39 J.J. Sylvester. On the principles of the calculus of forms. *Cambridge and Dublin Mathematical Journal*, 7:52–97, 1852.

40 Dekel Tsur. Faster deterministic parameterized algorithm for k-Path. *Theoretical Computer Science*, 790:96–104, 2019. doi:10.1016/j.tcs.2019.04.024.

41 Ryan Williams. Finding paths of length k in $O^*(2^k)$ time. *Inf. Process. Lett.*, 109(6):315–318, 2009. doi:10.1016/j.ipl.2008.11.004.

42 Michał Włodarczyk. Clifford algebras meet tree decompositions. *Algorithmica*, 81(2):497–518, 2019.

43 Meirav Zehavi. Mixing color coding-related techniques. In *Algorithms-ESA 2015*, pages 1037–1049. Springer, 2015.

A Linear-Time $n^{0.4}$-Approximation for Longest Common Subsequence

Karl Bringmann
Saarland University, Saarland Informatics Campus, Saarbrücken, Germany
Max-Planck-Institute for Informatics, Saarland Informatics Campus, Saarbrücken, Germany

Debarati Das
Basic Algorithm Research Copenhagen (BARC), University of Copenhagen, Denmark

Abstract

We consider the classic problem of computing the Longest Common Subsequence (LCS) of two strings of length n. While a simple quadratic algorithm has been known for the problem for more than 40 years, no faster algorithm has been found despite an extensive effort. The lack of progress on the problem has recently been explained by Abboud, Backurs, and Vassilevska Williams [FOCS'15] and Bringmann and Künnemann [FOCS'15] who proved that there is no subquadratic algorithm unless the Strong Exponential Time Hypothesis fails. This major roadblock for getting faster exact algorithms has led the community to look for subquadratic *approximation* algorithms for the problem.

Yet, unlike the edit distance problem for which a constant-factor approximation in almost-linear time is known, very little progress has been made on LCS, making it a notoriously difficult problem also in the realm of approximation. For the general setting (where we make no assumption on the length of the optimum solution or the alphabet size), only a naive $O(n^{\varepsilon/2})$-approximation algorithm with running time $\widetilde{O}(n^{2-\varepsilon})$ has been known, for any constant $0 < \varepsilon \leq 1$. Recently, a breakthrough result by Hajiaghayi, Seddighin, Seddighin, and Sun [SODA'19] provided a linear-time algorithm that yields a $O(n^{0.497956})$-approximation in expectation; improving upon the naive $O(\sqrt{n})$-approximation for the first time.

In this paper, we provide an algorithm that in time $O(n^{2-\varepsilon})$ computes an $\widetilde{O}(n^{2\varepsilon/5})$-approximation with high probability, for any $0 < \varepsilon \leq 1$. Our result (1) gives an $\widetilde{O}(n^{0.4})$-approximation in linear time, improving upon the bound of Hajiaghayi, Seddighin, Seddighin, and Sun, (2) provides an algorithm whose approximation scales with any subquadratic running time $O(n^{2-\varepsilon})$, improving upon the naive bound of $O(n^{\varepsilon/2})$ for any ε, and (3) instead of only in expectation, succeeds with high probability.

2012 ACM Subject Classification Theory of computation → Approximation algorithms analysis

Keywords and phrases approximation algorithm, longest common subsequence, string algorithm

Digital Object Identifier 10.4230/LIPIcs.ICALP.2021.39

Category Track A: Algorithms, Complexity and Games

Related Version *Full Version*: `https://arxiv.org/abs/2106.08195`

Funding *Karl Bringmann*: This work is part of the project TIPEA that has received funding from the European Research Council (ERC) under the European Unions Horizon 2020 research and innovation programme (grant agreement No. 850979).
Debarati Das: Work supported by Basic Algorithms Research Copenhagen, grant 16582 from the VILLUM Foundation.

Acknowledgements We thank an anonymous reviewer for suggesting how to turn the near-linear-time algorithm that we obtained in a previous version of this paper into a linear-time algorithm.

1 Introduction

The longest common subsequence (LCS) of two strings x and y is the longest string that appears as a subsequence of both strings. The length of the LCS of x and y, which we denote by $L(x, y)$, is one of the most fundamental measures of similarity between two strings and has drawn significant interest in last five decades, see, e.g. [35, 6, 26, 27, 30, 32, 10, 31, 11, 36, 22, 16, 28, 2, 19, 4, 20, 3, 33, 34, 24]. On strings of length n, the LCS problem can be solved exactly in quadratic time $O(n^2)$ using a classical dynamic programming approach [35]. Despite an extensive line of research the quadratic running time has been improved only by logarithmic factors [30]. This lack of progress is explained by a recent result showing that any truly subquadratic algorithm for LCS would falsify the Strong Exponential Time Hypothesis (SETH); this has been proven independently by Abboud et al. [2] and by Bringmann and Künnemann [19]. Further work in this direction shows that even a high polylogarithmic speedup for LCS would have surprising consequences [4, 3]. For the closely related edit distance the situation is similar, as the classic quadratic running time can be improved by logarithmic factors, but any truly subquadratic algorithm would falsify SETH [12].

These strong hardness results naturally bring up the question whether LCS or edit distance can be efficiently *approximated* (namely, whether an algorithm with truly subquadratic time $O(n^{2-\varepsilon})$ for any constant $\varepsilon > 0$, can produce a *good* approximation in the worst-case). In the last two decades, significant progress has been made towards designing efficient approximation algorithms for edit distance [14, 13, 15, 9, 7, 21, 23, 29, 17]; the latest achievement is a constant-factor approximation in almost-linear[1] time [8].

For LCS the picture is much more frustrating. The LCS problem has a simple $\widetilde{O}(n^{\varepsilon/2})$-approximation algorithm with running time $O(n^{2-\varepsilon})$ for any constant $0 < \varepsilon < 1$, and it has a trivial $|\Sigma|$-approximation algorithm with running time $O(n)$ for strings over alphabet Σ. Yet, improving upon these naive bounds has evaded the community until very recently, making LCS a notoriously hard problem to approximate. In 2019, Rubinstein et al. [33] presented a subquadratic-time $O(\lambda^3)$-approximation, where λ is the ratio of the string length to the length of the optimal LCS. For binary alphabet, Rubinstein and Song [34] recently improved the 2-approximation. In the general case (where λ and the alphabet size are arbitrary), the naive $O(\sqrt{n})$-approximation in near-linear[2] time was recently beaten by Hajiaghayi et al. [24], who designed a linear-time algorithm that computes an $O(n^{0.497956})$-approximation *in expectation*.[3] Nonetheless, the gap between the upper bound provided by Hajiaghayi et al. [24] and the recent results on hardness of approximation [1, 5] remains huge.

1.1 Our Contribution

We present a randomized $\widetilde{O}(n^{0.4})$-approximation for LCS running in linear time $O(n)$, where the approximation guarantee holds *with high probability*[4]. More generally, we obtain a tradeoff between approximation guarantee and running time: For any $0 < \varepsilon \leq 1$ we achieve approximation ratio $\widetilde{O}(n^{2\epsilon/5})$ in time $O(n^{2-\varepsilon})$. Formally we prove the following:

---

[1] By *almost-linear* we mean time $O(n^{1+\varepsilon})$ for a constant $\varepsilon > 0$ that can be chosen arbitrarily small.

[2] By *near-linear* we mean time $\widetilde{O}(n)$, where $\widetilde{O}$ hides polylogarithmic factors in n.

[3] While the SODA proceedings version of [24] claimed a high probability bound, the newer corrected Arxiv version [25] only claims that the algorithm outputs an $O(n^{0.497956})$-approximation in expectation. Personal communications with the authors confirm that the result indeed holds only in expectation, see also Remark 14.

[4] We say that an event happens *with high probability* (w.h.p.) if it has probability at least $1 - n^{-c}$, where the constant $c > 0$ can be chosen in advance.

▶ **Theorem 1.** *There is a randomized algorithm that, given strings x, y of length $n \geq 1$ and a time budget $T \in [n, n^2]$, with high probability computes a multiplicative $\widetilde{O}(n^{0.8}/T^{0.4})$-approximation of the length of the LCS of x and y in time $O(T)$.*

The improvement over the state of the art can be summarized as follows:

1. An improved approximation ratio for the linear time regime: from $O(n^{0.497956})$ [24] to $\widetilde{O}(n^{0.4})$;
2. The first algorithm which improves upon the naive bound *with high probability*[4];
3. A generalization to running time $O(n^{2-\varepsilon})$, breaking the naive approximation ratio $\widetilde{O}(n^{\varepsilon/2})$ in general.

2 Technical Overview

We combine classic exact algorithms for LCS with different subsampling strategies to develop several algorithms that work in different regimes of the problem. A combination of these algorithms then yields the full approximation algorithm.

Our Algorithm 1 covers the regime of short LCS, i.e., when the LCS has length at most n^γ for an appropriate constant $\gamma < 1$ depending on the running time budget. In this regime, we decrease the length of the string x by subsampling. This naturally allows to run classic exact algorithms for LCS on the subsampled string x (which now has significantly smaller size) and the original string y, while not deteriorating the LCS between the two strings too much.

For the remaining parts of the algorithm, the strings x and y are split into substrings $x_1, \ldots, x_{n/m}$ and $y_1, \ldots, y_{n/m}$ of length $m = n/\sqrt{T}$ where T denotes the total running time budget. For any block (i, j) we write L_{ij} for the length of the LCS of x_i and y_j. We call a set $\mathcal{S} = \{(i_1, j_1), \ldots, (i_k, j_k)\}$ with $i_1 < \ldots < i_k$ and $j_1 < \ldots < j_k$ a *block sequence*. Since we can assume the LCS of x and y to be long, it follows that there exists a good "block-aligned LCS", more precisely there exists a block sequence with large LCS sum $\sum_{(i,j)\in\mathcal{S}} L_{ij}$.

Now, a natural approach is to compute estimates $0 \leq \widetilde{L}_{ij} \leq L_{ij}$ for all blocks (i, j) and to determine the maximum sum $\widetilde{L} = \sum_{(i,j)\in\mathcal{S}} \widetilde{L}_{ij}$ over all block sequences $\mathcal{S}$. Once we have estimates $\widetilde{L}_{ij}$, the maximum $\widetilde{L}$ can be computed by dynamic programming in time $O((n/m)^2)$, which is $O(T)$ for our choice of m. In the following we describe three different strategies to compute estimates $\widetilde{L}_{ij}$. The major difficulty is that on average per block (i, j) we can only afford time $\widetilde{O}(1)$ to compute an estimate $\widetilde{L}_{ij}$.

The first strategy focuses on *matching pairs*. A matching pair of strings s, t is a pair of indices (a, b) such that $s[a] = t[b]$. We write M_{ij} for the number of matching pairs of the strings x_i and y_j. Our Algorithm 2 works well if some block sequence $\mathcal{S}$ has a large total number of matching pairs $\mu = \sum_{(i,j)\in\mathcal{S}} M_{ij}$. Here the key observation (Lemma 7) is that for each block (i, j) there exists a symbol that occurs at least $\frac{M_{ij}}{2m}$ times in both x_i and y_j. If M_{ij} is large, matching this symbol provides a good approximation for L_{ij}. Unfortunately, since we can afford only $\widetilde{O}(1)$ running time per block, finding a frequent symbol is difficult. We develop as a new tool an algorithm that w.h.p. finds a frequent symbol in each block with an above-average number of matching pairs, see Lemma 8.

For our remaining two strategies we can assume the optimal LCS L to be large and μ to be small (i.e., every block sequence has a small total number of matching pairs). In our Algorithm 3, we analyze the case where $\lambda = \sum_{i,j} L_{ij}$ is large. Here we pick some diagonal and run our basic approximation algorithm on each block along the diagonal. Since there are $O(n/m)$ diagonals, an above-average diagonal has a total LCS of $\Omega(\lambda/(n/m))$. If λ is

large then this provides a good estimation of the LCS. The main difficulty is how to find an above-average diagonal. A random diagonal has a good LCS sum in expectation, but not necessarily with good probability. Our solution is a non-uniform sampling, where we first test random blocks until we find a block with large LCS, and then choose the diagonal containing this seed block. This sampling yields an above-average diagonal with good probability.

Recall that there always exists a block sequence $\mathcal{G}$ with large LCS sum (see Lemma 11). The idea of our Algorithm 4 is to focus on a uniformly random subset of all blocks, where each block is picked with probability p. Then on each picked block we can spend more time (specifically time $\widetilde{O}(1/p)$) to compute an estimate $\widetilde{L}_{ij}$. Moreover, we still find a p-fraction of $\mathcal{G}$. We analyze this algorithm in terms of μ and λ (the choice of p depends on these two parameters) and show that it works well in the complementary regimes of Algorithms 1-3.

Comparison with the Previous Approach of Hajiaghayi et al. [24]. The general approach of splitting x and y into blocks and performing dynamic programming over estimates $\widetilde{L}_{ij}$ was introduced by Hajiaghayi et al. [24]. Moreover, our Algorithm 1 has essentially the same guarantees as [24, Algorithm 1], but ours is a simple combination of generic parts that we reuse in our later algorithms, thus simplifying the overall algorithm.

Our Algorithm 2 follows the same idea as [24, Algorithm 3], in that we want to find a frequent symbol in x_i and y_j and match only this symbol to obtain an estimate $\widetilde{L}_{ij}$. Hajiaghayi et al. find a frequent symbol by picking a *random* symbol σ in each block x_i, y_j; in expectation σ appears at least $\frac{M_{ij}}{2m}$ times in x_i and y_j. In order to obtain with high probability guarantees, we need to develop a new tool for finding frequent symbols not only in expectation but even with high probability, see Lemma 8 and Remark 14.

The remainder of the approach differs significantly; our Algorithms 3 and 4 are very different compared to [24, Algorithms 2 and 4]. In the following we discuss their ideas. In [24, Algorithm 2], they argue about the alphabet size, splitting the alphabet into frequent and infrequent letters. For infrequent letters the total number of matching pairs is small, so augmenting a classic exact algorithm by subsampling works well. Therefore, they can assume that every letter is frequent and thus the alphabet size is small. We avoid this line of reasoning. Finally, [24, Algorithm 4] is their most involved algorithm. Assuming that their other algorithms have failed to produce a sufficiently good approximation, they show that each part x_i and y_j can be turned into a *semi-permutation* by a little subsampling. Then by leveraging Dilworth's theorem and Tuŕan's theorem they show that most blocks have an LCS length of at least $n^{1/6}$; this can be seen as a *triangle inequality* for LCS and is their most novel contribution. This results in a highly non-trivial algorithm making clever use of combinatorial machinery.

We show that these ideas can be completely avoided, by instead relying on classic algorithms based on matching pairs augmented by subsampling. Specifically, we replace their combinatorial machinery by our Algorithms 3 and 4 described above (recall that Algorithm 3 considers a non-uniformly sampled random diagonal while Algorithm 4 subsamples the set of blocks to be able to spend more time per block). We stress that our solution completely avoids the concept of semi-permutation or any heavy combinatorial machinery as used in [24, Algorithm 4], while providing a significantly improved approximation guarantee.

Organization of the Paper. Section 3 introduces notation and a classical algorithm by Hunt and Szymanski. In Section 4 we present our new tools, in particular for finding frequent symbols. Section 5 contains our main algorithm, split into four parts that are presented in Sections 5.1, 5.3, 5.4, and 5.5, and combined in Section 5.6.

3 Preliminaries

For $n \in \mathbb{N}$ we write $[n] = \{1, 2, \ldots, n\}$. By the notation $\widetilde{O}$ and $\widetilde{\Omega}$ we hide factors of the form polylog(n). We use "with high probability" (w.h.p.) to denote probabilities of the form $1 - n^{-c}$, where the constant $c > 0$ can be chosen in advance.

String Notation. A string x over alphabet Σ is a finite sequence of letters in Σ. We denote its length by $|x|$ and its i-th letter by $x[i]$. We also denote by $x[i..j]$ the substring consisting of letters $x[i] \ldots x[j]$. For any indices $i_1 < i_2 < \ldots < i_k$ the string $z = x[i_1] \ldots x[i_k]$ forms a *subsequence* of x. For strings x, y we denote by $L(x, y)$ the length of the longest common subsequence of x and y. In this paper we study the problem of approximating $L(x, y)$ for given strings x, y of length n. We focus on the length $L(x, y)$, however, our algorithms can be easily adapted to also reconstruct a subsequence attaining the output length. If x, y are clear from the context, we may replace $L(x, y)$ by L. Throughout the paper we assume that the alphabet is $\Sigma \subseteq [O(n)]$ (this is without loss of generality after a $\widetilde{O}(n)$-time preprocessing).

Matching Pairs. For a symbol $\sigma \in \Sigma$, we denote the number of times that σ appears in x by $\#_\sigma(x)$, and call this the *frequency* of σ in x. For strings x and y, a *matching pair* is a pair (i, j) with $x[i] = y[j]$. We denote the number of matching pairs by $M(x, y)$. If x, y are clear from the context, we may replace $M(x, y)$ by M. Observe that $M = \sum_{\sigma \in \Sigma} \#_\sigma(x) \cdot \#_\sigma(y)$. Using this equation we can compute M in time $O(n)$.

Hunt and Szymanski [27] solved the LCS problem in time $\widetilde{O}(n + M)$. More precisely, their algorithm can be viewed as having a preprocessing phase that only reads y and runs in time $\widetilde{O}(|y|)$, and a query phase that reads x and y and takes time $\widetilde{O}(|x| + M)$.

▶ **Theorem 2** (Hunt and Szymanski [27]). *We can preprocess a string y in time $\widetilde{O}(|y|)$. Given a string x and a preprocessed string y, we can compute their LCS in time $\widetilde{O}(|x| + M)$.*

4 New Basic Tools

4.1 Basic Approximation Algorithm

Throughout this section we abbreviate $L = L(x, y)$ and $M = M(x, y)$. We start with the basic approximation algorithm that is central to our approach; most of our later algorithms use this as a subroutine. This algorithm subsamples the string x and then runs Hunt and Szymanski's algorithm (Theorem 2).

▶ **Lemma 3** (Basic Approximation Algorithm). *Let $x, y \in \Sigma^n$. We can preprocess y in time $\widetilde{O}(n)$. Given x, the preprocessed string y, and $\beta \geq 1$, in expected time $\widetilde{O}((n + M)/\beta + 1)$ we can compute a value $\widetilde{L} \leq L$ that w.h.p. satisfies $\widetilde{L} > \frac{L}{\beta} - 1$.*

Proof. In the preprocessing phase, we run the preprocessing of Theorem 2 on y.

Fix a constant $c \geq 1$. If $\beta \geq 1/(8c \log n)$, then in the query phase we simply run Theorem 2, solving LCS exactly in time $\widetilde{O}(|x| + M) = \widetilde{O}((n + M)/\beta + 1)$.

Otherwise, denote by x' a random subsequence of x, where each letter $x[i]$ is removed independently with probability $1 - p$ (i.e., kept with probability p) for $p := 8c \log(n)/\beta$. Note that $p \leq 1$ by our assumption on β. We can sample x' in expected time $O(|x'| + 1)$, since the difference from one unremoved letter to the next is geometrically distributed, and geometric random variates can be sampled in expected time $O(1)$, see, e.g., [18]. Note that this subsampling yields $\mathbb{E}[|x'|] = p|x| = \widetilde{O}(|x|/\beta)$ and $\mathbb{E}[M(x', y)] = p\,M = \widetilde{O}(M/\beta)$.

In the query phase, we sample x' and then run the query phase of Theorem 2 on x' and y. This runs in time $\widetilde{O}(|x'| + M(x', y) + 1)$, which is $\widetilde{O}((|x| + M)/\beta + 1)$ in expectation.

Finally, consider a fixed LCS of x and y, namely $z = x[i_1] \dots x[i_L] = y[j_1] \dots y[j_L]$ for some $i_1 < \dots < i_L$ and $j_1 < \dots < j_L$. Each letter $x[i_k]$ survives the subsampling to x' with probability p. Therefore, we can bound $L(x', y)$ from below by a binomial random variable $\mathrm{Bin}(L, p)$ (the correct terminology is that $L(x', y)$ statistically dominates $\mathrm{Bin}(L, p)$). Since $Z = \mathrm{Bin}(L, p)$ is a sum of independent $\{0, 1\}$-variables, multiplicative Chernoff applies and yields $\Pr[Z < \mathbb{E}[Z]/2] \leq \exp(-\mathbb{E}[Z]/8)$. If $L \geq \beta$ then $\mathbb{E}[Z] = Lp \geq 2L/\beta$ and $\mathbb{E}[Z] \geq 8c \log n$, and thus $\Pr[L(x', y) \geq L/\beta] \geq 1 - n^{-c}$. Otherwise, if $L < \beta$, then we can only bound $L(x', y) \geq 0$. In both cases, we have $L(x', y) > L/\beta - 1$ with high probability. ◀

The above lemma behaves poorly if $L \leq \beta$, due to the "-1" in the approximation guarantee. We next show that this can be avoided, at the cost of increasing the running time by an additive $\widetilde{O}(n)$.

▶ **Lemma 4** (Generalised Basic Approximation Algorithm). *Given $x, y \in \Sigma^n$ and $\beta \geq 1$, in expected time $\widetilde{O}(n + M/\beta)$ we can compute a value $\widetilde{L} \leq L$ that w.h.p. satisfies $\widetilde{L} \geq L/\beta$.*

Proof. We run the basic approximation algorithm from Lemma 3, which computes a value $\widetilde{L} \leq L$. Additionally, we compute the number of matching pairs $M = M(x, y)$ in time $\widetilde{O}(n)$. If $M > 0$, then there exists a matching pair, which yields a common subsequence of length 1. Therefore, if $M > 0$ we set $\widetilde{L} := \max\{\widetilde{L}, 1\}$.

In the proof of Lemma 3 we showed that if $L \geq \beta$ then w.h.p. we have $\widetilde{L} \geq L/\beta$. We now argue differently in the case $L < \beta$. If $L = 0$, then $\widetilde{L} \geq 0 = L/\beta$ and we are done. If $0 < L < \beta$, then there must exist at least one matching pair, so $M > 0$, so the second part of our algorithm yields $\widetilde{L} \geq 1 > L/\beta$. Hence, in all cases w.h.p. we have $\widetilde{L} \geq L/\beta$. ◀

We now turn towards the problem of deciding for given x, y and ℓ whether $L(x, y) \geq \ell$. To this end, we repeatedly call the basic approximation algorithm with geometrically decreasing approximation ratio β. Note that with decreasing approximation ratio we get a better approximation guarantee at the cost of higher running time. The idea is that if the LCS $L = L(x, y)$ is much shorter than the threshold ℓ, then already approximation ratio $\beta \approx \ell/L$ allows us to detect that $L < \ell$. This yields a running time bound depending on the gap L/ℓ.

▶ **Lemma 5** (Basic Decision Algorithm). *Let $x, y \in \Sigma^n$. We can preprocess y in time $\widetilde{O}(n)$. Given x, the preprocessed y, and a number $1 \leq \ell \leq n$, in expected time $\widetilde{O}((n + M)L/\ell + n/\ell)$ we can w.h.p. correctly decide whether $L \geq \ell$. Our algorithm has no false positives (and w.h.p. no false negatives).*

Proof. In the preprocessing phase, we run the preprocessing of Lemma 3. In the query phase, we repeatedly call the query phase of Lemma 3, with geometrically decreasing values of β:

1. Preprocessing: Run the preprocessing of Lemma 3.
2. For $\beta = n, n/2, n/4, \dots, 1$:
   3. Run the query phase of Lemma 3 with parameter β to obtain an estimate $\widetilde{L}$.
   4. If $\widetilde{L} \geq \ell$: return "$L \geq \ell$"
   5. If $\widetilde{L} \leq \ell/\beta - 1$: return "$L < \ell$"

Let us first argue correctness. Since Lemma 3 computes a common subsequence of x, y, we have $\widetilde{L} \leq L$. Thus, if $\widetilde{L} \geq \ell$, we correctly infer $L \geq \ell$. Moreover, w.h.p. $\widetilde{L}$ satisfies $\widetilde{L} > L/\beta - 1$. Therefore, if $\widetilde{L} \leq \ell/\beta - 1$, we can infer $L < \ell$, and this decision is correct with high probability. Finally, in the last iteration (where $\beta = 1$), we have $\ell/\beta - 1 = \ell - 1$, and thus one of $\widetilde{L} \geq \ell$ or $\widetilde{L} \leq \ell/\beta - 1$ must hold, so the algorithm indeed returns a decision.

The expected time of the query phase of Lemma 3 is $\widetilde{O}((n+M)/\beta+1)$. Since β decreases geometrically, the total expected time of our algorithm is dominated by the last call.

If $L \geq \ell$, the last call is at the latest for $\beta = 1$. This yields running time $\widetilde{O}(n+M) \leq \widetilde{O}((n+M)L/\ell)$.

If $L < \ell$, note that for any $\beta \leq \frac{\ell}{L+1}$ we have $\widetilde{L} \leq L \leq \ell/\beta - 1$, and thus we return "$L < \ell$". Because we decrease β by a factor 2 in each iteration, the last call satisfies $\beta \geq \frac{\ell}{2(L+1)}$. Hence, the expected running time is $\widetilde{O}((n+M)(L+1)/\ell + 1)$. If $L \geq 1$ then this time bound simplifies to $\widetilde{O}((n+M)L/\ell + 1)$. If $L = 0$, then also $M = 0$, and the time bound becomes $\widetilde{O}(n/\ell + 1)$. In both cases we can bound the expected running time by the claimed $\widetilde{O}((n+M)L/\ell + n/\ell)$, since $\ell \leq n$. ◀

4.2 Approximating the Number of Matching Pairs

Recall that for given strings x, y of length n the number of matching pairs $M = M(x,y)$ can be computed in time $O(n)$, which is linear in the input size. However, later in the paper we will split x into substrings $x_1, \ldots, x_{n/m}$ and y into substrings $y_1, \ldots, y_{n/m}$, each of length m, and we will need estimates of the numbers of matching pairs $M_{ij} = M(x_i, y_j)$. In this setting, the input size is still n (the total length of all strings x_i and y_j) and the output size is $(n/m)^2$ (all numbers M_{ij}), but we are not aware of any algorithm computing the numbers M_{ij} in near-linear time in the input plus output size $\widetilde{O}(n + (n/m)^2)$.[5] Therefore, we devise an approximation algorithm for estimating the number of matching pairs.

▶ **Lemma 6.** *For $x_1, \ldots, x_{n/m}, y_1, \ldots, y_{n/m} \in \Sigma^m$ write $M_{ij} = M(x_i, y_j)$ and $M = \sum_{i,j} M_{ij}$. Given $x_1, \ldots, x_{n/m}, y_1, \ldots, y_{n/m}$ and $q > 0$, we can compute values $\widetilde{M}_{ij}$ that w.h.p. satisfy $M_{ij}/8 - q \leq \widetilde{M}_{ij} \leq 4M_{ij}$, in total expected time $\widetilde{O}(n + M/q)$.*

This yields a near-linear-time constant-factor approximation of all *above-average* M_{ij}: By setting $q := \Theta(\frac{Mm^2}{n^2})$, in expected time $\widetilde{O}(n + (n/m)^2)$ we obtain a constant-factor approximation of all values M_{ij} with $M_{ij} \gg q$.

Proof. The algorithm works as follows.

1. *Graph Construction:* Build a three-layered graph G on vertex set $V(G) = L \cup U \cup R$, where L has a node i for every string x_i, R has a node j for every string y_j, and U has a node (σ, ℓ, r) for any $\sigma \in \Sigma$ and $0 \leq \ell, r \leq \log m$. Put an edge from $i \in L$ to $(\sigma, \ell, r) \in U$ iff $\#_\sigma(x_i) \in [2^\ell, 2^{\ell+1})$. Similarly, put an edge from $j \in R$ to $(\sigma, \ell, r) \in U$ iff $\#_\sigma(y_j) \in [2^r, 2^{r+1})$. Note that all frequencies and thus all edges of this graphs can be computed in total time $\widetilde{O}(n)$. For $i \in L$ and $j \in R$, we denote by $U_{ij} \subseteq U$ their common neighbors. Note that any $(\sigma, \ell, r) \in U_{ij}$ represents all matching pairs of symbol σ in x_i and y_j, and the number of these matching pairs is $\#_\sigma(x_i) \cdot \#_\sigma(y_j) \in [2^{\ell+r}, 2^{\ell+r+2})$.
2. *Subsampling:* We sample a subset $\widetilde{U} \subseteq U$ by removing each node $(\sigma, \ell, r) \in U$ independently with probability $1 - p_{\ell,r}$, where $p_{\ell,r} := \min\{1, 2^{\ell+r+3}/q\}$.
3. *Determine Common Neighbors:* For each $(\sigma, \ell, r) \in \widetilde{U}$ enumerate all pairs of neighbors $i \in L$ and $j \in R$. For each such 2-path, add (σ, ℓ, r) to an initially empty set $\widetilde{U}_{ij}$. This step computes the sets $\widetilde{U}_{ij} := U_{ij} \cap \widetilde{U}$ in time proportional to their total size.
4. *Output:* Return the values $\widetilde{M}_{ij} := \sum_{(\sigma,\ell,r) \in \widetilde{U}_{ij}} 2^{\ell+r}/p_{\ell,r}$.

[5] In fact, one can show conditional lower bounds from Boolean matrix multiplication that rule out near-linear time for computing all M_{ij}'s unless the exponent of matrix multiplication is $\omega = 2$.

Correctness. To analyze this algorithm, we consider the numbers $\overline{M}_{ij} := \sum_{(\sigma,\ell,r)\in U_{ij}} 2^{\ell+r}$. Observe that we have $\overline{M}_{ij} \leq M_{ij} \leq 4\overline{M}_{ij}$, since each $(\sigma,\ell,r) \in U_{ij}$ corresponds to at least $2^{\ell+r}$ and at most $2^{\ell+r+2}$ matching pairs of x_i and y_j. It therefore suffices to show that $\widetilde{M}_{ij}$ is close to $\overline{M}_{ij}$. Using Bernoulli random variables $\mathrm{Ber}(p_{\ell,r})$ to express whether (σ,ℓ,r) survives the subsampling, we write

$$\widetilde{M}_{ij} = \sum_{(\sigma,\ell,r)\in U_{ij}} \frac{2^{\ell+r}}{p_{\ell,r}} \cdot \mathrm{Ber}(p_{\ell,r}).$$

This yields an expected value of $\mathbb{E}[\widetilde{M}_{ij}] = \overline{M}_{ij}$, so by Markov's inequality we obtain $\widetilde{M}_{ij} \leq 4\overline{M}_{ij} \leq 4M_{ij}$ with probability at least 3/4. Since $\widetilde{M}_{ij}$ is a linear combination of independent Bernoulli random variables, we can also easily express its variance as

$$\mathbb{V}[\widetilde{M}_{ij}] = \sum_{(\sigma,\ell,r)\in U_{ij}} \left(2^{\ell+r}/p_{\ell,r}\right)^2 \cdot p_{\ell,r}(1-p_{\ell,r}) = \sum_{(\sigma,\ell,r)\in U_{ij}} 2^{\ell+r} \cdot 2^{\ell+r}\Big(\frac{1}{p_{\ell,r}} - 1\Big).$$

We now use the definition of $p_{\ell,r} := \min\{1, 2^{\ell+r+3}/q\}$ to bound

$$2^{\ell+r}\Big(\frac{1}{p_{\ell,r}} - 1\Big) = 2^{\ell+r}\Big(\max\Big\{1, \frac{q}{2^{\ell+r+3}}\Big\} - 1\Big) = \max\{0, q/8 - 2^{\ell+r}\} \leq q/8.$$

This yields $\mathbb{V}[\widetilde{M}_{ij}] \leq \overline{M}_{ij}q/8$. We now use Chebychev's inequality $\Pr[X < \mathbb{E}[X] - \lambda] \leq \mathbb{V}[X]/\lambda^2$ on $\lambda = 0.5\mathbb{E}[X]$ and $X = \widetilde{M}_{ij}$ to obtain

$$\Pr[\widetilde{M}_{ij} < \overline{M}_{ij}/2] \leq \frac{q}{2\overline{M}_{ij}}.$$

In case $M_{ij} \geq 8q$, we have $\overline{M}_{i,j} \geq M_{ij}/4 \geq 2q$ and hence $\Pr[\widetilde{M}_{ij} \geq M_{ij}/8] \geq \Pr[\widetilde{M}_{ij} \geq \overline{M}_{ij}/2] \geq 3/4$. Otherwise, in case $M_{ij} < 8q$, we can only use the trivial $\widetilde{M}_{ij} \geq 0 > M_{ij}/8 - q$.

Hence, each inequality $\widetilde{M}_{ij} \leq 4M_{ij}$ and $\widetilde{M}_{ij} \geq M_{ij}/8 - q$ individually holds with probability at leat 3/4. Finally, we boost the success probability by repeating the above algorithm $O(\log n)$ times and returning for each i, j the median of all computed values $\widetilde{M}_{ij}$.

Running Time. Steps 1 and 2 can be easily seen to run in time $\widetilde{O}(n)$. Steps 3 and 4 run in time proportional to the total size of all sets $\widetilde{U}_{ij}$, which we claim to be at most $8M/q$ in expectation. Over $O(\log n)$ repetitions, we obtain a total expected running time of $\widetilde{O}(n + M/q)$. (We remark that here we consider a succinct output format, where only the non-zero numbers $\widetilde{M}_{ij}$ are listed; otherwise additional time of $\widetilde{O}((n/m)^2)$ is required to output the numbers $\widetilde{M}_{ij} = 0$.)

It remains to prove the claimed bound of $\mathbb{E}[\sum_{i,j} |\widetilde{U}_{ij}|] \leq 8M/q$. Since $2^{\ell+r}/p_{\ell,r} = \max\{2^{\ell+r}, q/8\} \geq q/8$, from the definition of $\widetilde{M}_{ij} = \sum_{(\sigma,\ell,r)\in\widetilde{U}_{ij}} 2^{\ell+r}/p_{\ell,r}$ we infer $\widetilde{M}_{ij} \geq \frac{q}{8}|\widetilde{U}_{ij}|$. Therefore,

$$\mathbb{E}\Big[\sum_{i,j} |\widetilde{U}_{ij}|\Big] \leq \mathbb{E}\Big[\frac{8}{q}\sum_{i,j} \widetilde{M}_{ij}\Big] = \frac{8}{q}\sum_{i,j} \overline{M}_{ij} \leq \frac{8}{q}\sum_{i,j} M_{ij} = \frac{8M}{q}. \qquad \blacktriangleleft$$

4.3 Single Symbol Approximation Algorithm

For strings x, y that have a large number of matchings pairs $M = M(x,y)$, some symbol must appear often in x and in y. This yields a common subsequence using (several repetitions of) a single alphabet symbol.

▶ **Lemma 7** (Cf. Lemma 6.6.(ii) in [20] or Algorithm 3 in [24]). *For any $x, y \in \Sigma^n$ there exists a symbol $\sigma \in \Sigma$ that appears at least $\frac{M}{2n}$ times in x and in y. Therefore, in time $\widetilde{O}(n)$ we can compute a common subsequence of x, y of length at least $\frac{M}{2n}$. In particular, we can compute a value $\widetilde{L} \leq L$ that satisfies $\widetilde{L} \geq \frac{M}{2n}$.*

Proof. Let k be maximal such that some symbol $\sigma \in \Sigma$ appears at least k times in x and at least k times in y. Let $\Sigma^w := \{\sigma \in \Sigma \mid \#_\sigma(w) \leq k\}$ for $w \in \{x, y\}$. Since no symbol appears more than k times in x and in y, we have $\Sigma^x \cup \Sigma^y = \Sigma$. We can thus bound

$$M = M(x,y) = \sum_{\sigma\in\Sigma} \#_\sigma(x) \cdot \#_\sigma(y) \leq \sum_{\sigma\in\Sigma^x} k \cdot \#_\sigma(y) + \sum_{\sigma\in\Sigma^y} \#_\sigma(x) \cdot k \leq 2kn,$$

since the frequencies $\#_\sigma(x)$ sum up to at most n, and similarly for $\#_\sigma(y)$. It follows that $k \geq \frac{M}{2n}$. Computing k, and a symbol $\sigma \in \Sigma$ attaining k, in time $\widetilde{O}(n)$ is straightforward. ◀

We devise a variant of Lemma 7 in the following setting. For strings $x_1, \ldots, x_{n/m}$, $y_1, \ldots, y_{n/m} \in \Sigma^m$ we write $L_{ij} = L(x_i, y_j)$, $M_{ij} = M(x_i, y_j)$ and $M = \sum_{i,j} M_{ij}$. We want to find for each block (i, j) a frequent symbol in x_i and y_j, or equivalently we want to find a common subsequence of x_i and y_j using a single alphabet symbol. Similarly to Lemma 6, we relax Lemma 7 to obtain a fast running time.

▶ **Lemma 8.** *Given $x_1, \ldots, x_{n/m}, y_1, \ldots, y_{n/m} \in \Sigma^m$ and any $q > 0$, we can compute for each i, j a number $\widetilde{L}_{ij} \leq L_{ij}$ such that w.h.p. $\widetilde{L}_{ij} \geq \frac{M_{ij}-q}{16m}$. The algorithm runs in total expected time $\widetilde{O}(n + M/q)$.*

Proof. We run the same algorithm as in Lemma 6, except that in Step 4 for each i, j with non-empty set $\widetilde{U}_{ij}$ we let $\widetilde{L}_{ij}$ be the maximum of $2^{\min\{\ell,r\}}$ over all $(\sigma, \ell, r) \in \widetilde{U}_{ij}$. For each empty set $\widetilde{U}_{ij}$, we implicitly set $\widetilde{L}_{ij} = 0$, i.e., we output a sparse representation of all non-zero values $\widetilde{L}_{ij}$.

The running time analysis is the same as in Lemma 6.

For the upper bound on $\widetilde{L}_{ij}$, since σ appears at least 2^ℓ times in x_i and at least 2^r times in y_j, there is a common subsequence of x_i and y_j of length at least $\widetilde{L}_{ij}$. Thus, we have $\widetilde{L}_{ij} \leq L_{ij}$.

For the lower bound on $\widetilde{L}_{ij}$, fix i, j and order the tuples $(\sigma, \ell, r) \in U_{ij}$ in ascending order of $2^{\min\{\ell,r\}}$, obtaining an ordering $(\sigma_1, \ell_1, r_1), \ldots, (\sigma_k, \ell_k, r_k)$. For $h \in [k]$ we let $\mathcal{S} := \{(\sigma_1, \ell_1, r_1), \ldots, (\sigma_h, \ell_h, r_h)\}$ and $\mathcal{L} := \{(\sigma_h, \ell_h, r_h), \ldots, (\sigma_k, \ell_k, r_k)\}$. Recall that $\overline{M}_{ij} = \sum_{(\sigma,\ell,r)\in U_{ij}} 2^{\ell+r}$, and observe that we can pick h with

$$\sum_{(\sigma,\ell,r)\in\mathcal{S}} 2^{\ell+r} \geq \overline{M}_{ij}/2 \qquad \text{and} \qquad \sum_{(\sigma,\ell,r)\in\mathcal{L}} 2^{\ell+r} \geq \overline{M}_{ij}/2. \tag{1}$$

Then we have

$$\frac{\overline{M}_{ij}}{2} \leq \sum_{(\sigma,\ell,r)\in\mathcal{S}} 2^{\ell+r} = \sum_{(\sigma,\ell,r)\in\mathcal{S}} 2^{\min\{\ell,r\}} \cdot 2^{\max\{\ell,r\}} \leq 2^{\min\{\ell_h,r_h\}} \sum_{(\sigma,\ell,r)\in\mathcal{S}} 2^{\max\{\ell,r\}}.$$

Note that for any $(\sigma, \ell, r) \in \mathcal{S}$ the symbol σ appears at least $2^{\max\{\ell,r\}}$ times in x_i or in y_j, and thus the sum on the right hand side is at most $2m$. Rearranging, this yields $2^{\min\{\ell_h,r_h\}} \geq \frac{\overline{M}_{ij}}{4m} \geq \frac{M_{ij}}{16m}$, where we used $\overline{M}_{ij} \geq M_{ij}/4$ as in the proof of Lemma 6. In particular, due to our ordering we have for any $(\sigma, \ell, r) \in \mathcal{L}$:

$$2^{\min\{\ell,r\}} \geq 2^{\min\{\ell_h,r_h\}} \geq \frac{M_{ij}}{16m}. \tag{2}$$

Consider the number of nodes in $\mathcal{L}$ surviving the subsampling, i.e., $Z := |\mathcal{L} \cap \widetilde{U}_{ij}|$. If $Z > 0$, then some node in $\mathcal{L}$ survived, and thus by (2) the computed value $\widetilde{L}_{ij}$ is at least $\frac{M_{ij}}{16m}$. It thus remains to analyze $\Pr[Z > 0]$.

In case some $(\sigma, \ell, r) \in \mathcal{L}$ has $p_{\ell,r} = 1$, we have $Z > 0$ with probability 1. Otherwise all $(\sigma, \ell, r) \in \mathcal{L}$ have $p_{\ell,r} < 1$ and thus $p_{\ell,r} = 2^{\ell+r+3}/q$. In this case, we write Z as a sum of independent Bernoulli random variates in the form $Z = \sum_{(\sigma,\ell,r)\in\mathcal{L}} \mathrm{Ber}(p_{\ell,r})$. In particular,

$$\mathbb{E}[Z] = \sum_{(\sigma,\ell,r)\in\mathcal{L}} 2^{\ell+r+3}/q \overset{(1)}{\geq} \frac{4\overline{M}_{ij}}{q} \geq \frac{M_{ij}}{q}.$$

Since Z is a sum of independent $\{0,1\}$-variables, multiplicative Chernoff applies and yields $\Pr[Z < \mathbb{E}[Z]/2] \leq \exp(-\mathbb{E}[Z]/8)$. We thus obtain

$$\Pr[Z > 0] \geq 1 - \Pr\left[Z < \mathbb{E}[Z]/2\right] \geq 1 - \exp\left(-\mathbb{E}[Z]/8\right) \geq 1 - \exp\left(-\frac{M_{ij}}{8q}\right).$$

In case $M_{ij} \geq q$, we obtain $\Pr[Z > 0] \geq 1 - \exp(-1/8) \geq 0.1$, and thus we have $\widetilde{L}_{ij} \geq \frac{M_{ij}}{16m}$ with probability at least 0.1. Otherwise, in case $M_{ij} < q$, we can only use the trivial bound $\widetilde{L}_{ij} \geq 0 > \frac{M_{ij}-q}{16m}$. In any case, we have $\widetilde{L}_{ij} \geq \frac{M_{ij}-q}{16m}$ with probability at least 0.1. Similar to the proof of Lemma 6, we run $O(\log n)$ independent repetitions of this algorithm and return for each i, j the maximum of all computed values $\widetilde{L}_{ij}$, to boost the success probability and finish the proof. ◀

5 Main Algorithm

In this section we prove Theorem 1. First we show that Theorem 9 implies Theorem 1, and then in the remainder of this section we prove Theorem 9.

▶ **Theorem 9** (Main Result, Relaxation). *Given strings x, y of length n and a time budget $T \in [n, n^2]$, in expected time $\widetilde{O}(T)$ we can compute a number $\widetilde{L}$ such that $\widetilde{L} \leq L := L(x, y)$ and w.h.p. $\widetilde{L} \geq \widetilde{\Omega}(LT^{0.4}/n^{0.8})$.*

Recall Theorem 1:

▶ **Theorem 1.** *There is a randomized algorithm that, given strings x, y of length $n \geq 1$ and a time budget $T \in [n, n^2]$, with high probability computes a multiplicative $\widetilde{O}(n^{0.8}/T^{0.4})$-approximation of the length of the LCS of x and y in time $O(T)$.*

Proof of Theorem 1 assuming Theorem 9. Note that the difference between Theorems 1 and 9 is that the latter allows *expected* running time and has an additional slack of logarithmic factors in the running time.

In order to remove the *expected* running time, we abort the algorithm from Theorem 9 after $\widetilde{O}(T)$ time steps. By Markov's inequality, we can choose the hidden constants and logfactors such that the probability of aborting is at most 1/2. We boost the success probability of this adapted algorithm by running $O(\log n)$ independent repetitions and returning the maximum over all computed values $\widetilde{L}$. This yields an $\widetilde{O}(n^{0.8}/T^{0.4})$-approximation with high probability in time $\widetilde{O}(T)$.

To remove the logfactors in the running time, as the first step in our algorithm we subsample the given strings x, y, keeping each symbol independently with probability $p = 1/\mathrm{polylog}(n)$, resulting in subsampled strings $\tilde{x}, \tilde{y}$. Since any common subsequence of $\tilde{x}, \tilde{y}$ is also a common subsequence of x, y, the estimate $\widetilde{L}$ that we compute for $\tilde{x}, \tilde{y}$ satisfies

$\widetilde{L} \leq L(\tilde{x}, \tilde{y}) \leq L(x, y)$. Moreover, if $L(x, y) \geq \text{polylog}(n)$ then by Chernoff bound with high probability we have $L(\tilde{x}, \tilde{y}) = \widetilde{\Omega}(L(x, y))$, so that an $\widetilde{O}(n^{0.8}/T^{0.4})$-approximation on $\tilde{x}, \tilde{y}$ also yields an $\widetilde{O}(n^{0.8}/T^{0.4})$-approximation on x, y. Otherwise, if $L(x, y) \leq \text{polylog}(n)$, then in order to compute a $\widetilde{O}(1)$-approximation it suffices to compute an LCS of length 1, which is just a matching pair and can be found in time $O(n)$ (assuming that the alphabet is $[O(n)]$).

This yields an algorithm that computes a value $\widetilde{L} \leq L$ such that w.h.p. $\widetilde{L} \geq \widetilde{\Omega}(LT^{0.4}/n^{0.8})$. The algorithm runs in time $O(T)$, and this running time bound holds deterministically, i.e., with probability 1. Hence, we proved Theorem 1. ◀

It remains to prove Theorem 9. Our algorithm is a combination of four methods that work well in different regimes of the problem, see Sections 5.1, 5.3, 5.4, and 5.5. We will combine these methods in Section 5.6.

5.1 Algorithm 1: Small L

Algorithm 1 works well if the LCS is short. It yields the following result.

▶ **Theorem 10** (Algorithm 1). *We can compute in expected time $\widetilde{O}(T)$ an estimate $\widetilde{L} \leq L$ that w.h.p. satisfies $\widetilde{L} \geq \min\{L, \sqrt{LT/n}\}$.*

Proof. Our Algorithm 1 works as follows.

1. Run Lemma 7 on x and y.
2. Run Lemma 4 on x and y with $\beta := \max\{1, \frac{M}{2T}\}$.
3. Output the larger of the two common subsequence lengths computed in Steps 1 and 2.

Running Time. Step 1 runs in time $\widetilde{O}(n) = \widetilde{O}(T)$. Step 2 runs in expected time $\widetilde{O}(n+M/\beta)$. Since $\beta \geq \frac{M}{2T}$ we have $M/\beta \leq 2T$, so the expected running time is $\widetilde{O}(n + T) = \widetilde{O}(T)$.

Upper Bound. Steps 1 and 2 compute common subsequences, so the computed estimate $\widetilde{L}$ satisfies $\widetilde{L} \leq L$.

Approximation Guarantee. Note that Step 1 guarantees $\widetilde{L} \geq \frac{M}{2n}$ and Step 2 guarantees w.h.p. $\widetilde{L} \geq L/\beta$. If $M \leq 2T$ then $\beta = 1$ and $\widetilde{L} = L$, so we solved the problem exactly. Otherwise we have $M > 2T$ and $\beta = \frac{M}{2T}$, so Step 2 guarantees w.h.p. $\widetilde{L} \geq 2LT/M$. By multiplying the two guarantees on $\widetilde{L}$ and taking square roots, we obtain w.h.p.

$$\widetilde{L} \geq \sqrt{\frac{M}{2n} \cdot \frac{2LT}{M}} = \sqrt{\frac{LT}{n}}.$$

It follows that w.h.p. $\widetilde{L} \geq \min\{L, \sqrt{LT/n}\}$. ◀

5.2 Block Sequences and Parameter Guessing

This section introduces some general notation and structure for the remaining algorithms.

Block Sequences. We split x into substrings $x_1, \ldots, x_{n/m}$ of length $m = n/\sqrt{T}$. Similarly, we split y into $y_1, \ldots, y_{n/m}$. A pair $(i, j) \in [n/m]^2$, corresponding to the substrings x_i, y_j, is called a *block*. For any block we write $M_{ij} = M(x_i, y_j)$ and $L_{ij} = L(x_i, y_j)$. Moreover, we write $(i, j) < (i', j')$ if and only if $i < i'$ and $j < j'$. A *block sequence* is a set $\mathcal{S} = \{(i_1, j_1), \ldots, (i_k, j_k)\}$ with $\mathcal{S} \subseteq [n/m]^2$ satisfying the monotonicity property $(i_1, j_1) < \ldots <$

(i_k, j_k). In what follows, every algorithm will compute estimates $0 \le \widetilde{L}_{ij} \le L_{ij}$ and then choose a block sequence $\mathcal{S}$ to produce an overall estimate $\widetilde{L} = \sum_{(i,j)\in\mathcal{S}} \widetilde{L}_{ij}$. Note that this guarantees $\widetilde{L} \le L$, as the sum $\sum_{(i,j)\in\mathcal{S}} \widetilde{L}_{ij}$ corresponds to some (block-aligned) common subsequence of x and y. In order to get bounds in the other direction, we need to show that there always exists a block sequence of large LCS sum, i.e., a long "block-aligned common subsequence". This is shown by the following lemma.

▶ **Lemma 11.** *There exists a block sequence $\mathcal{G}$ of size $|\mathcal{G}| = \frac{L\sqrt{T}}{8n}$ such that for any $(i,j) \in \mathcal{G}$ we have $L_{ij} \ge \frac{L}{4\sqrt{T}}$ and $M_{ij} \le \frac{8\mu n}{L\sqrt{T}}$. In particular, we have $\sum_{(i,j)\in\mathcal{G}} L_{ij} \ge \frac{L^2}{32n}$.*

▶ Remark 12. This is analogous to [24, Lemma 8.2], but we improve the size of $\mathcal{G}$.

Proof. Let L^*_{ij} be the *contribution of block* (i,j) to the LCS. More precisely, fix an LCS z of x and y, and write $z = x[a_1]\dots x[a_L] = y[b_1]\dots y[b_L]$ for $(a_1,b_1) < \dots < (a_L,b_L)$. Then for any block (i,j), the number L^*_{ij} counts all indices k with $a_k \in ((i-1)m, im]$ and $b_k \in ((j-1)m, jm]$. Consider the set $\mathcal{A} := \{(i,j) \mid L^*_{ij} > 0\}$ consisting of all contributing blocks. From the monotonicity $(a_1,b_1) < \dots < (a_L,b_L)$ it follows that also the contributing blocks form a monotone sequence, in the sense that for any $(i,j),(i',j') \in \mathcal{A}$ we have $i \le i'$ and $j \le j'$, or $i' \le i$ and $j' \le j$. (However, these inequalities are not necessarily strict, so $\mathcal{A}$ is not necessarily a block sequence.) This monotonicity implies that there are $|\mathcal{A}| \le 2n/m$ contributing blocks. Also note that $\sum_{(i,j)\in\mathcal{A}} L^*_{ij} = L$. Now consider the subset $\mathcal{B} = \{(i,j) \mid L^*_{ij} > \frac{Lm}{4n}\} \subseteq \mathcal{A}$. Note that the remaining blocks in total contribute

$$\sum_{(i,j)\in\mathcal{A}\setminus\mathcal{B}} L^*_{ij} \le |\mathcal{A}| \cdot \frac{Lm}{4n} \le \frac{2n}{m} \cdot \frac{Lm}{4n} = \frac{L}{2},$$

and thus $\mathcal{B}$ contributes $\sum_{(i,j)\in\mathcal{B}} L^*_{ij} \ge L/2$.

We now greedily pick a subset $\mathcal{C} \subseteq \mathcal{B}$ as follows. Pick any $(i,j) \in \mathcal{B}$, add (i,j) to $\mathcal{C}$, and then remove each $(i',j') \in \mathcal{B}$ with $i' = i$ or $j' = j$ from $\mathcal{B}$. Repeat until $\mathcal{B}$ is empty.

By construction, $\mathcal{C}$ is a block sequence and for any $(i,j) \in \mathcal{C}$ we have $L_{ij} \ge \frac{Lm}{4n} = \frac{L}{4\sqrt{T}}$. We claim that $|\mathcal{C}| \ge \frac{L}{4m} = \frac{L\sqrt{T}}{4n}$. To see this, observe that all blocks $(i',j') \in \mathcal{B}$ with $i' = i$ in total contribute at most m, since they describe a subsequence of x_i, which has length m. Similarly, all blocks $(i',j') \in \mathcal{B}$ with $j' = j$ in total contribute at most m. Therefore, one step of the greedy procedure removes a contribution of at most $2m$. Since the total contribution is $\sum_{(i,j)\in\mathcal{B}} L^*_{ij} \ge L/2$, there are at least $\frac{L}{4m} = \frac{L\sqrt{T}}{4n}$ greedy steps. Finally, we consider the number of matching pairs. Since $\mathcal{C}$ is a block sequence, we have $\sum_{(i,j)\in\mathcal{C}} M_{ij} \le \mu$. Thus, on average each $(i,j) \in \mathcal{C}$ has a number of matching pairs of at most $\mu/|\mathcal{C}| = \frac{4\mu n}{L\sqrt{T}}$. By Markov's inequality, at least half of the blocks $(i,j) \in \mathcal{C}$ have $M_{ij} \le \frac{8\mu n}{L\sqrt{T}}$. We denote the set of these blocks by $\mathcal{G} \subseteq \mathcal{C}$. The set $\mathcal{G}$ satisfies all claimed bounds. This finishes the proof. ◀

Parameter Guessing. We analyze our algorithms in terms of n (the length of the strings), T (the running time budget), L (the length of the LCS), as well as λ and μ, defined as

$$\lambda := \sum_{i,j} L_{ij} \qquad \text{and} \qquad \mu := \max_{\text{block seq. } \mathcal{S}} \sum_{(i,j)\in\mathcal{S}} M_{ij},$$

where the maximum goes over all block sequences $\mathcal{S}$. Note that λ is the total LCS length over all blocks and μ is the maximum total number of matching pairs along any block sequence.

The numbers n and T are part of the input, and we can assume to know M, since it can be computed in time $O(n)$. However, in order to set some parameters in our algorithms, it would be convenient to also know L, λ, μ up to constant factors (which seemingly is a contradiction, as our goal is to compute a polynomial-factor approximation of L).

We therefore run our algorithms $O(\log^3 n)$ times, once for each guess $\hat{L} = 2^i$, $\hat{\lambda} = 2^j$, and $\hat{\mu} = 2^k$. Then for at least one call we have $L/2 \leq \hat{L} \leq L$, $\lambda/2 \leq \hat{\lambda} \leq \lambda$, and $\mu/2 \leq \hat{\mu} \leq \mu$, that is, we know L, λ, μ up to constant factors. For this correct guess, we prove that our algorithms have the promised approximation guarantee and running time bound. For the wrong guesses, the approximation guarantee can fail, but we always ensure the upper bound $\widetilde{L} \leq L$, by ensuring that the estimate corresponds to some common subsequence of x and y. Hence, returning the maximum computed value $\widetilde{L}$ over all guesses $\hat{L}, \hat{\lambda}, \hat{\mu}$ yields the promised approximation guarantee. For this reason, in the following we assume to know estimates $\hat{L} \approx L$, $\hat{\lambda} \approx \lambda$, $\hat{\mu} \approx \mu$ up to constant factors; we will only use them to set certain parameters.

We remark that for the wrong guesses, not only the approximation guarantee but also the running time bound can fail, so we need to abort each of the $O(\log^3 n)$ calls after time $\widetilde{O}(T)$.

Diagonals. A *diagonal* is a set of the form $\mathcal{D}_d = \{(i,j) \in [n/m]^2 \mid i - j = d\}$. Each diagonal is a block sequence, so we have $\sum_{(i,j)\in\mathcal{D}_d} M_{ij} \leq \mu$. Note that there are $2n/m - 1 < 2\sqrt{T}$ (non-empty) diagonals. Moreover, we have $\sum_d \sum_{(i,j)\in\mathcal{D}_d} M_{ij} = M$. This yields the inequality

$$M < 2\mu\sqrt{T}. \tag{3}$$

5.3 Algorithm 2: Large L, Large μ

In this section we present Algorithm 2, which works well if μ is large, i.e., if some block sequence has a large total number of matching pairs. The algorithm makes use of the single symbol approximation that we designed in Lemma 8. This yields estimates $0 \leq \widetilde{L}_{ij} \leq L_{ij}$, over which we then perform dynamic programming to determine the maximum of $\sum_{(i,j)\in\mathcal{S}} \widetilde{L}_{ij}$ over all block sequences $\mathcal{S}$. (This is similar to [24, Algorithm 3], but we obtain concentration in a wider regime, see Remark 14 for a comparison.)

▶ **Theorem 13** (Algorithm 2). *We can compute in expected time $\widetilde{O}(T)$ an estimate $\widetilde{L} \leq L$ that w.h.p. satisfies*

$$\widetilde{L} = \Omega\Big(\frac{\mu\sqrt{T}}{n}\Big).$$

Proof. Algorithm 2 works as follows.

1. Run Lemma 8 with $q := \frac{M}{4T}$ to compute values $\widetilde{L}_{ij}$.
2. Perform dynamic programming over $[n/m]^2$ to determine the maximum $\sum_{(i,j)\in\mathcal{S}} \widetilde{L}_{ij}$ over all block sequences $\mathcal{S}$. Output this maximum value $\widetilde{L}$. More precisely:
   - Initialize $D[i,0] = D[0,i] = 0$ for any $0 \leq i \leq n/m$.
   - For $i = 1, \ldots, n/m$ and $j = 1, \ldots, n/m$: $D[i,j] = \max\big\{\widetilde{L}_{ij} + D[i-1, j-1],\ D[i-1, j],\ D[i, j-1]\big\}$.
   - Output $D[n/m, n/m]$.

We analyze this algorithm in the following.

Upper Bound. Since Lemma 8 ensures $\widetilde{L}_{ij} \leq L_{ij}$, the dynamic programming step ensures $\widetilde{L} \leq L$.

Approximation Guarantee. Let $\mathcal{S}$ be a block sequence achieving $\sum_{(i,j)\in\mathcal{S}} M_{ij} = \mu$. Step 2 computes an estimate $\widetilde{L} \geq \sum_{(i,j)\in\mathcal{S}} \widetilde{L}_{ij}$, and Lemma 8 yields w.h.p.

$$\widetilde{L} \geq \sum_{(i,j)\in\mathcal{S}} \widetilde{L}_{ij} \geq \sum_{(i,j)\in\mathcal{S}} \frac{M_{ij} - q}{16m} = \frac{\mu}{16m} - |\mathcal{S}|\frac{q}{16m}.$$

By the monotonicity property of block sequences, we have $|\mathcal{S}| \leq n/m$. Using our definitions of $q = \frac{M}{4T}$ and $m = n/\sqrt{T}$ as well as inequality (3), we obtain

$$|\mathcal{S}|\frac{q}{16m} \leq \frac{qn}{16m^2} = \frac{M}{64n} \leq \frac{\mu\sqrt{T}}{32n}.$$

Plugging this into our bound for $\widetilde{L}$ yields

$$\widetilde{L} \geq \frac{\mu\sqrt{T}}{16n} - \frac{\mu\sqrt{T}}{32n} = \frac{\mu\sqrt{T}}{32n}.$$

Running Time. For Step 1 note that Lemma 8 runs in expected time $\widetilde{O}(n + M/q) = \widetilde{O}(T)$. Step 2 can be easily seen to run in time $O((n/m)^2) = O(T)$ by our choice of $m = n/\sqrt{T}$. This finishes the proof. ◀

▶ Remark 14. Our Algorithm 2 is similar to [24, Algorithm 3], which works as follows. For each block (i, j), their algorithm selects a random symbol σ and uses the minimum of the frequencies $\#_\sigma(x_i), \#_\sigma(y_j)$ as the estimate $\widetilde{L}_{ij}$. It can be shown that this yields $\mathbb{E}[\widetilde{L}_{ij}] = M_{ij}/(2m)$, which is a similar lower bound as provided by Lemma 8, but only in expectation. The summation $\sum_{(i,j)\in\mathcal{S}} \widetilde{L}_{ij}$ over a block sequence $\mathcal{S}$ then allows to apply concentration inequalities to obtain a w.h.p. error guarantee, assuming $\mu \gg m^2$.

However, in the regime $\mu \leq m^2$ the value μ could be dominated by a single block with $M_{ij} \approx \mu$. In this case, we cannot hope to get concentration by summing over many blocks. Thus, picking a random symbol per block does not suffice to obtain a w.h.p. error guarantee.

Since our improved approximation ratio makes it necessary to use Algorithm 2 in the regime $\mu \ll m^2$, their algorithm is not sufficient in our context. Thus, we replace sampling a single symbol by our new Lemma 8.

5.4 Algorithm 3: Large L, Small μ and Large λ

Our next algorithm works well if μ is small (i.e., every block sequence has a small total number of matching pairs) and λ is large (i.e., on average every block has a large LCS).

Let us start with the intuition. The idea is to *pick some diagonal $\mathcal{D}_d$ and run the basic approximation algorithm (Lemma 4) with approximation ratio $\beta = \max\{1, \mu/T\}$ on each block along the diagonal.* Since every diagonal is a block sequence, we have $\sum_{(i,j)\in\mathcal{D}_d} M_{ij} \leq \mu$, which bounds the running time of this algorithm by $\widetilde{O}(n + \sum_{(i,j)\in\mathcal{D}_d} M_{ij}/\beta) = \widetilde{O}(T)$. Moreover, this algorithm produces an estimate $\widetilde{L} \leq L$ that w.h.p. satisfies

$$\widetilde{L} \geq \sum_{(i,j)\in\mathcal{D}_d} L_{ij}/\beta.$$

Since $\sum_d \sum_{(i,j)\in\mathcal{D}_d} L_{ij} = \sum_{i,j} L_{ij} = \lambda$ and there are $O(n/m)$ diagonals, on average a diagonal $\mathcal{D}_d$ satisfies $\sum_{(i,j)\in\mathcal{D}_d} L_{ij} = \Omega(\lambda m/n) = \Omega(\lambda/\sqrt{T})$. If we pick an above-average diagonal, then we obtain an estimate

$$\widetilde{L} \geq \sum_{(i,j)\in\mathcal{D}_d} L_{ij}/\beta = \Omega\Big(\frac{\lambda}{\sqrt{T}\beta}\Big) = \Omega\Big(\min\Big\{\frac{\lambda}{\sqrt{T}}, \frac{\lambda\sqrt{T}}{\mu}\Big\}\Big).$$

If λ is large and μ is small, then this is a good estimate.

The main difficulty in translating this idea to an actual algorithm is how to pick the diagonal. A natural approach is to pick a random diagonal, as then the *expected* LCS sum of the diagonal is sufficiently large. However, in situations where the diagonal sums are highly

unbalanced, so that λ is dominated by very few diagonals that have a very large LCS sum, a random diagonal is unlikely to have an above-average LCS sum. In this situation, a random diagonal works only with negligible probability.

Therefore, we need a sampling process that favors diagonals with large LCS sum. To this end, we first "guess" a value g such that the sum λ is dominated by summands $L_{ij} = \Theta(g)$. We call blocks (i,j) with $L_{ij} = \Omega(g)$ *good*. Next we sample a random good block (i_0, j_0); for this we simply keep sampling random i, j until we find a good block. Finally, we pick the diagonal $\mathcal{D}_d$ containing the "seed" block (i_0, j_0) and run the above algorithm on this diagonal. This sampling procedure favors diagonals with large LCS sum, because such diagonals contain more good blocks (i,j) to start from, and thus we are more likely to pick the "seed" (i_0, j_0) in a diagonal with large LCS sum. This yields the following result.

▶ **Theorem 15** (Algorithm 3). *We can compute in expected time $\widetilde{O}(T)$ an estimate $\widetilde{L} \le L$ that w.h.p. satisfies*

$$\widetilde{L} = \widetilde{\Omega}\Big(\min\Big\{\frac{\lambda}{\sqrt{T}}, \frac{\lambda\sqrt{T}}{\mu}\Big\}\Big).$$

Proof. Note that the theorem statement is trivial if $\lambda \le \sqrt{T}$. Indeed, in time $O(n)$ we can compute $M = M(x,y)$. If $M = 0$ then $L = \lambda = 0$ and we return $\widetilde{L} = 0$. If $M \ge 1$, then we return $\widetilde{L} = 1$. This ensures $\widetilde{L} \le L$, since any matching pair gives a common subsequence of length 1. Moreover, in case $\lambda \le \sqrt{T}$ the returned value $\widetilde{L} = 1$ satisfies the approximation guarantee $\widetilde{L} = \Omega(\lambda/\sqrt{T})$. Therefore, we can assume

$$\lambda > \sqrt{T}. \tag{4}$$

Algorithm 3 repeats the following procedure $O(\log n)$ times to boost its success probability.

1. Repeat the following for g being any power of two with $\max\{1, \hat{\lambda}/(4T)\} \le g \le m$:
   2. *Sampling a good block:* Pick a random set of blocks $\mathcal{R} \subseteq [n/m]^2$ of size $O\big((gT/\hat{\lambda})\log^2 n\big)$. For each block $(i,j) \in \mathcal{R}$, test whether $L_{ij} \ge g$ using our basic decision algorithm (Lemma 5). If no test was successful, then set $\widetilde{L}(g) = 0$ and continue with the next value of g. Otherwise, pick a random successfully tested block (i_0, j_0) and proceed to Step 3.
   3. *Approximating along a diagonal:* Let $\mathcal{D}$ be the diagonal containing the block (i_0, j_0). For each $(i,j) \in \mathcal{D}$: Run our basic approximation algorithm (Lemma 4) with approximation ratio $\beta = \max\{1, \hat{\mu}/T\}$ on x_i, y_j to obtain an estimate $\widetilde{L}_{ij}$. Finally, $\widetilde{L}(g) = \sum_{(i,j)\in\mathcal{D}} \widetilde{L}_{ij}$ is the result of iteration g.
4. Return $\widetilde{L} = \max_g \widetilde{L}(g)$.

Upper Bound. Again it is easy to see that $\widetilde{L} \le L$, since Lemma 4 yields $\widetilde{L}_{ij} \le L_{ij}$.

Approximation Guarantee. Let $\mathcal{B}_g$ be the set of all blocks (i,j) with $g \le L_{ij} \le 2g$.

▷ Claim 16. If $\lambda/2 \le \hat{\lambda} \le \lambda$ then for some power of two g with $\max\{1, \hat{\lambda}/(4T)\} \le g \le m$ we have

$$g \cdot |\mathcal{B}_g| = \Omega(\lambda/\log m). \tag{5}$$

Proof. Write G for the set of all powers of two g with $\max\{1, \hat{\lambda}/(4T)\} \leq g \leq m$. Note that blocks (i,j) with $L_{ij} \leq \frac{\lambda}{2T}$ in total contribute at most $\lambda/2$ to $\lambda = \sum_{i,i} L_{ij}$, since the total number of blocks is $(n/m)^2 = T$. Hence, the blocks with $L_{ij} > \frac{\lambda}{2T}$ contribute at least $\lambda/2$, that is,

$$\frac{\lambda}{2} \leq \sum_{\substack{i,j \\ L_{ij} > \lambda/(2T)}} L_{ij}.$$

Note that the sets $\mathcal{B}_g$ for powers of two $g \geq \max\{1, \lambda/(4T)\} \geq \max\{1, \hat{\lambda}/(4T)\}$ cover all blocks with $L_{ij} > \frac{\lambda}{2T}$. Moreover, the sets $\mathcal{B}_g$ are empty for $g > m$. Therefore, the blocks with $L_{ij} > \frac{\lambda}{2T}$ are covered by the sets $\mathcal{B}_g$ with $g \in G$, that is,

$$\frac{\lambda}{2} \leq \sum_{\substack{i,j \\ L_{ij} > \lambda/(2T)}} L_{ij} \leq \sum_{g \in G} \sum_{(i,j) \in \mathcal{B}_g} L_{ij} \leq \sum_{g \in G} 2g|\mathcal{B}_g|.$$

If for all g appearing in the sum on the right hand side we would have $g \cdot |\mathcal{B}_g| < \lambda/(4 \log m + 4)$ then the right hand side would be less than $\lambda/2$, so we would obtain a contradiction. This proves the claim. ◁

In the following we focus on an iteration of Step 1 in which we pick a value of g as promised by Claim 16.

We call a block (i,j) *good* if $L_{ij} \geq g$, and *bad* otherwise. Note that any $(i,j) \in \mathcal{B}_g$ is good, but not every good block is in $\mathcal{B}_g$. In Step 2, we claim that the set $\mathcal{R}$ w.h.p. contains at least one good block, assuming that our guess $\hat{\lambda}$ is correct up to constant factors. Indeed, since the set $\mathcal{B}_g$ is a subset of the good blocks, the probability that $\Theta((gT/\lambda)\log^2 n)$ sampled blocks do not contain any good block is at most

$$\left(1 - \frac{|\mathcal{B}_g|}{(n/m)^2}\right)^{\Theta((gT/\lambda)\log^2 n)} \overset{(5)}{\leq} \left(1 - \frac{\lambda}{gT \log m}\right)^{\Theta((gT/\lambda)\log^2 n)} \leq \exp(-\Theta(\log n)),$$

which is negligible. For any bad block $(i,j) \in \mathcal{R}$ the test $L_{ij} \geq g$ is unsuccessful, as Lemma 5 has no false positives. For any good block $(i,j) \in \mathcal{R}$ w.h.p. the test is successful, and w.h.p. there is at least one good block in $\mathcal{R}$. It follows that w.h.p. Step 2 finds a good block (i_0, j_0) and proceeds to Step 3. Observe that (i_0, j_0) is chosen uniformly at random from all good blocks.

We call a diagonal *good* if it contains at least $\frac{|\mathcal{B}_g|m}{4n}$ good blocks, and *bad* otherwise. Since there are $< 2n/m$ non-empty diagonals, the number of good blocks contained in bad diagonals is at most $|\mathcal{B}_g|/2$, which is at most half of all good blocks. Therefore, at least half of all good blocks are contained in good diagonals. It follows that the uniformly random good block (i_0, j_0) lies in a good diagonal with probability at least $1/2$.

Hence, with probability at least $1/2 - o(1)$ the diagonal $\mathcal{D}$ considered in Step 3 is good, that is, it contains at least $\frac{|\mathcal{B}_g|m}{4n}$ blocks (i,j) with $L_{ij} \geq g$. Since the approximations $\widetilde{L}_{ij}$ computed in Step 3 w.h.p. satisfy $\widetilde{L}_{ij} \geq L_{ij}/\beta$, we obtain

$$\widetilde{L} \geq \widetilde{L}(g) \geq \frac{|\mathcal{B}_g|m}{4n} \cdot \frac{g}{\beta}.$$

Inequality (5) and the definitions $m = n/\sqrt{T}$ and $\beta = \max\{1, \hat{\mu}/T\}$ now yield

$$\widetilde{L} = \widetilde{\Omega}\Big(\min\Big\{\frac{\lambda}{\sqrt{T}}, \frac{\lambda\sqrt{T}}{\hat{\mu}}\Big\}\Big).$$

If our guess $\hat{\mu} \approx \mu$ is correct up to a constant factor, then this yields the claimed approximation guarantee. Returning the maximum over $O(\log n)$ independent repetitions of this algorithm improves the success probability from $1/2 - o(1)$ to w.h.p.

Running Time. By Lemma 5, the test $L_{ij} \geq g$ runs in expected time $\widetilde{O}((m + M_{ij})L_{ij}/g + m/g) = \widetilde{O}(m^2 L_{ij}/g + m/g)$. Note that in expectation for random i, j we have $\mathbb{E}[L_{ij}] = \lambda/(n/m)^2 = \lambda/T$. Therefore, the expected running time of one test is bounded by $\widetilde{O}\big(\frac{m^2\lambda}{gT} + m/g\big)$. As Step 2 performs $O\big((gT/\hat{\lambda})\log^2 n\big)$ such tests, its expected running time is $\widetilde{O}(m^2 + mT/\lambda)$, assuming that our guess $\hat{\lambda} \approx \lambda$ is correct up to a constant factor. We now use $m^2 = n^2/T \leq T$ from $n \leq T$ and $\lambda \geq \sqrt{T} \geq n/\sqrt{T} = m$ from (4) and $n \leq T$, to bound the expected running time of Step 2 by $\widetilde{O}(T)$.

For Step 3, the expected running time is $\widetilde{O}(n + \sum_{(i,j)\in\mathcal{D}} M_{ij}/\beta)$. Since $\mathcal{D}$ is a block sequence, we have $\sum_{(i,j)\in\mathcal{D}} M_{ij} \leq \mu$. Using $\beta \geq \hat{\mu}/T = \Omega(\mu/T)$ (if our guess $\hat{\mu} \approx \mu$ is correct up to a constant factor) we can bound the expected time by $\widetilde{O}(n + T) = \widetilde{O}(T)$.

Over the $O(\log n)$ iterations of Step 1 and the $O(\log n)$ repetitions for boosting the success probability, the expected running time is still $\widetilde{O}(T)$. ◀

5.5 Algorithm 4: Large L, Small μ, and Small λ

Our next algorithm works well if μ is small (i.e., every block sequence has a small total number of matching pairs), λ is small (i.e., on average every block has a small LCS), and L is large (i.e., there is a long LCS). The goal of this algorithm is to detect a sufficiently large random subset of the block sequence $\mathcal{G}$ from Lemma 11. To this end, we first sample a random set of blocks $\mathcal{R}$ containing each block $(i, j) \in [n/m]^2$ with probability p. Then we use our basic decision algorithm to detect the blocks $(i, j) \in \mathcal{R}$ with $L_{ij} \geq \frac{\hat{L}}{4\sqrt{T}}$, and for these blocks we set $\widetilde{L}_{ij} = \frac{\hat{L}}{4\sqrt{T}}$, while for the remaining blocks we set $\widetilde{L}_{ij} = 0$. Finally, we perform dynamic programming to determine the maximum $\sum_{(i,j)\in\mathcal{S}} \widetilde{L}_{ij}$ over all block sequences $\mathcal{S}$.

Observe that for each block in $\mathcal{G} \cap \mathcal{R}$ this algorithm sets $\widetilde{L}_{ij} = \frac{\hat{L}}{4\sqrt{T}}$, so it detects a random subset of $\mathcal{G}$. We thus obtain a p-fraction of the LCS guaranteed by the block sequence $\mathcal{G}$.

Note that in this algorithm we may focus on blocks with $M_{ij} = O(\frac{\mu n}{L\sqrt{T}})$, since this holds for all blocks in $\mathcal{G}$. Moreover, since λ is small, most blocks outside of $\mathcal{G}$ have small LCS L_{ij}. These bounds on L_{ij} and M_{ij} for the considered blocks allow us to bound the running time of the basic decision algorithm. We elaborate this algorithm in the following theorem.

▶ **Theorem 17** (Algorithm 4). *We can compute in expected time $\widetilde{O}(T)$ an estimate $\widetilde{L} \leq L$ that w.h.p. satisfies*

$$\widetilde{L} = \Omega\Big(\min\Big\{\frac{L^3}{n^2}, \frac{L^3 T}{\lambda n^2}, \frac{L^4 T}{\lambda\mu n^2}\Big\}\Big), \quad \textit{assuming that} \quad \frac{L^2 T^{0.5}}{n^2}, \frac{L^2 T^{1.5}}{\lambda n^2}, \frac{L^3 T^{1.5}}{\lambda\mu n^2} = n^{\Omega(1)}.$$

Proof. Algorithm 4 works as follows.

1. Run Lemma 6 with $q := \frac{M}{T}$ to compute values $\widetilde{M}_{ij}$. Initialize $\widetilde{L}_{ij} = 0$ for all i, j.
2. Run the preprocessing of the basic decision algorithm (Lemma 5) on each string y_j.
3. Sample a set $\mathcal{R} \subseteq [n/m]^2$ by including each block (i, j) independently with probability

$$p := \min\Big\{\frac{\hat{L}}{n}, \frac{\hat{L}T}{\hat{\lambda}n}, \frac{\hat{L}^2 T}{\hat{\lambda}\hat{\mu}n}\Big\}.$$

4. For each $(i,j) \in \mathcal{R}$ with $\widetilde{M}_{ij} \leq 64\hat{\mu}n/(\hat{L}\sqrt{T})$: Run the query of the basic decision algorithm (Lemma 5) to test whether $L_{ij} \geq \frac{\hat{L}}{4\sqrt{T}}$. If this test is successful then set $\widetilde{L}_{ij} := \frac{\hat{L}}{4\sqrt{T}}$.
5. Perform dynamic programming over $[n/m]^2$ to determine the maximum $\sum_{(i,j)\in\mathcal{S}} \widetilde{L}_{ij}$ over all block sequences $\mathcal{S}$. Output this maximum value $\widetilde{L}$.

Upper Bound. Since Lemma 5 has no false positives, we ensure $\widetilde{L}_{ij} \leq L_{ij}$ and thus $\widetilde{L} \leq L$.

Approximation Guarantee. The values $\widetilde{M}_{ij}$ computed in Step 1 w.h.p. satisfy $M_{ij}/8 - q \leq \widetilde{M}_{ij} \leq 4M_{ij}$. For all blocks $(i,j) \in \mathcal{G}$ we have $M_{ij} \leq \frac{8\mu n}{L\sqrt{T}}$ (by Lemma 11) and thus w.h.p. $\widetilde{M}_{ij} \leq \frac{32\mu n}{L\sqrt{T}}$. We may assume that our guesses $\hat{L}, \hat{\mu}$ satisfy $L/2 \leq \hat{L} \leq L$ and $\mu/2 \leq \hat{\mu} \leq \mu$; then we obtain $\widetilde{M}_{ij} \leq \frac{64\hat{\mu}n}{\hat{L}\sqrt{T}}$.

Therefore, each block in $\mathcal{G} \cap \mathcal{R}$ satisfies the property checked in Step 4, that is, for each such block we run the basic decision algorithm. Since for each $(i,j) \in \mathcal{G}$ we have $L_{ij} \geq \frac{L}{4\sqrt{T}} \geq \frac{\hat{L}}{4\sqrt{T}}$, in Step 4 for each block in $\mathcal{G} \cap \mathcal{R}$ w.h.p. we obtain an estimate $\widetilde{L}_{ij} = \frac{\hat{L}}{4\sqrt{T}}$. Since $\mathcal{G}$ is a block sequence, also $\mathcal{G} \cap \mathcal{R}$ is a block sequence, and thus the dynamic programming in Step 5 returns an estimate of

$$\widetilde{L} \geq \sum_{(i,j)\in\mathcal{G}\cap\mathcal{R}} \widetilde{L}_{ij} = |\mathcal{G} \cap \mathcal{R}| \cdot \frac{\hat{L}}{4\sqrt{T}}.$$

Note that the size $|\mathcal{G} \cap \mathcal{R}|$ is distributed as a binomial random variable $\mathrm{Bin}(|\mathcal{G}|, p)$, with expectation $p|\mathcal{G}|$. Assuming that our guesses $\hat{L}, \hat{\lambda}, \hat{\mu}$ are correct up to constant factors, we have

$$p|\mathcal{G}| = \Omega\Big(\min\Big\{\frac{L}{n}, \frac{LT}{\lambda n}, \frac{L^2T}{\lambda\mu n}\Big\} \cdot \frac{L\sqrt{T}}{n}\Big) = \Omega\Big(\min\Big(\frac{L^2T^{0.5}}{n^2}, \frac{L^2T^{1.5}}{\lambda n^2}, \frac{L^3T^{1.5}}{\lambda\mu n^2}\Big)\Big) = n^{\Omega(1)},$$

by the assumption in the theorem statement. By Chernoff bound, we have

$$\Pr[|\mathcal{G} \cap \mathcal{R}| < p|\mathcal{G}|/2] \leq \exp(-p|\mathcal{G}|/8) = \exp(-n^{\Omega(1)}),$$

and thus w.h.p. we have $|\mathcal{G} \cap \mathcal{R}| \geq p|\mathcal{G}|/2$. Plugging this into our lower bound for $\widetilde{L}$ yields w.h.p.

$$\widetilde{L} \geq \frac{p|\mathcal{G}|\hat{L}}{8\sqrt{T}} = \Omega\Big(\min\Big\{\frac{L^3}{n^2}, \frac{L^3T}{\lambda n^2}, \frac{L^4T}{\lambda\mu n^2}\Big\}\Big),$$

assuming that our guesses $\hat{L}, \hat{\lambda}, \hat{\mu}$ are correct up to constant factors. This shows the claimed lower bound.

Running Time. The expected running time of Step 1 is $\widetilde{O}(n + M/q) = \widetilde{O}(T)$ since $q = \frac{M}{T}$. Step 2 runs in time $\widetilde{O}(\sum_j |y_j|) = \widetilde{O}(n)$. Steps 3 and 5 take time $O((n/m)^2) = O(T)$. In the remainder we show that Step 4 also runs in expected time $\widetilde{O}(T)$, assuming that our guesses $\hat{L}, \hat{\lambda}, \hat{\mu}$ are correct up to constant factors. Recall that w.h.p. $M_{ij}/8 - q \leq \widetilde{M}_{ij} \leq 4M_{ij}$; we condition on this event in the following.[6] Then $\widetilde{M}_{ij} = O(\frac{\mu n}{L\sqrt{T}})$ implies $M_{ij} = O(q + \frac{\mu n}{L\sqrt{T}})$.

[6] In the error event we bound the running time of Step 4 by $O(n^2)$. This has a negligible contribution to the expected running time of Step 4.

Using inequality (3) we have $q = \frac{M}{T} \leq \frac{2\mu}{\sqrt{T}} \leq \frac{2\mu n}{L\sqrt{T}}$, so $M_{ij} = O(\frac{\mu n}{L\sqrt{T}})$. Since only blocks (i,j) with $\widetilde{M}_{ij} = O(\frac{\mu n}{L\sqrt{T}})$ are tested, each invocation of the basic approximation algorithm in Step 4 runs in expected time $\widetilde{O}((|x_i| + M_{ij})L_{ij}\sqrt{T}/L + |x_i|\sqrt{T}/L) = \widetilde{O}((m + \frac{\mu n}{L\sqrt{T}})L_{ij}\sqrt{T}/L + m\sqrt{T}/L)$. Since each block is tested with probability at most p, Step 4 has an expected running time of

$$O\Big(\sum_{i,j} p \cdot \Big(\Big(m + \frac{\mu n}{L\sqrt{T}}\Big)L_{ij}\frac{\sqrt{T}}{L} + \frac{m\sqrt{T}}{L}\Big)\Big) = O\Big(\Big(\frac{n}{\sqrt{T}} + \frac{\mu n}{L\sqrt{T}}\Big)\frac{p\lambda\sqrt{T}}{L} + \frac{pnT}{L}\Big)$$
$$= \widetilde{O}\Big(\frac{p\lambda n}{L} + \frac{p\lambda\mu n}{L^2} + \frac{pnT}{L}\Big).$$

Note that our choice of $p = \Theta(\min\{\frac{L}{n}, \frac{LT}{\lambda n}, \frac{L^2T}{\lambda\mu n}\})$ ensures that this running time is $\widetilde{O}(T)$. ◀

5.6 Combining the Algorithms

We conclude the proof of Theorem 9 by verifying that for any input at least one of Algorithms 1-4 computes an estimate $\widetilde{L} = \widetilde{\Omega}(LT^{0.4}/n^{0.8})$. The proof of this claim is a case distinction that is deferred to the full version of this paper.

References

1. Amir Abboud and Arturs Backurs. Towards hardness of approximation for polynomial time problems. In *ITCS*, volume 67 of *LIPIcs*, pages 11:1–11:26, 2017.
2. Amir Abboud, Arturs Backurs, and Virginia Vassilevska Williams. Tight hardness results for LCS and other sequence similarity measures. In *FOCS*, pages 59–78. IEEE, 2015.
3. Amir Abboud and Karl Bringmann. Tighter connections between Formula-SAT and shaving logs. In *ICALP*, volume 107 of *LIPIcs*, pages 8:1–8:18, 2018.
4. Amir Abboud, Thomas Dueholm Hansen, Virginia Vassilevska Williams, and Ryan Williams. Simulating branching programs with edit distance and friends: or: a polylog shaved is a lower bound made. In *STOC*, pages 375–388. ACM, 2016.
5. Amir Abboud and Aviad Rubinstein. Fast and deterministic constant factor approximation algorithms for LCS imply new circuit lower bounds. In *ITCS*, volume 94 of *LIPIcs*, pages 35:1–35:14, 2018.
6. Alfred V. Aho, Daniel S. Hirschberg, and Jeffrey D. Ullman. Bounds on the complexity of the longest common subsequence problem. *J. ACM*, 23(1):1–12, 1976.
7. Alexandr Andoni, Robert Krauthgamer, and Krzysztof Onak. Polylogarithmic approximation for edit distance and the asymmetric query complexity. In *FOCS*, pages 377–386. IEEE, 2010.
8. Alexandr Andoni and Negev Shekel Nosatzki. Edit distance in near-linear time: it's a constant factor. In *FOCS*, pages 990–1001. IEEE, 2020.
9. Alexandr Andoni and Krzysztof Onak. Approximating edit distance in near-linear time. *SIAM J. Comput.*, 41(6):1635–1648, 2012.
10. Alberto Apostolico. Improving the worst-case performance of the Hunt-Szymanski strategy for the longest common subsequence of two strings. *Inf. Process. Lett.*, 23(2):63–69, 1986.
11. Alberto Apostolico and Concettina Guerra. The longest common subsequence problem revisited. *Algorithmica*, 2:316–336, 1987.
12. Arturs Backurs and Piotr Indyk. Edit distance cannot be computed in strongly subquadratic time (unless SETH is false). In *STOC*, pages 51–58. ACM, 2015.
13. Ziv Bar-Yossef, T. S. Jayram, Robert Krauthgamer, and Ravi Kumar. Approximating edit distance efficiently. In *FOCS*, pages 550–559. IEEE, 2004.
14. Tugkan Batu, Funda Ergün, Joe Kilian, Avner Magen, Sofya Raskhodnikova, Ronitt Rubinfeld, and Rahul Sami. A sublinear algorithm for weakly approximating edit distance. In *STOC*, pages 316–324. ACM, 2003.

15 Tugkan Batu, Funda Ergün, and Süleyman Cenk Sahinalp. Oblivious string embeddings and edit distance approximations. In *SODA*, pages 792–801. ACM, 2006.

16 Lasse Bergroth, Harri Hakonen, and Timo Raita. A survey of longest common subsequence algorithms. In *SPIRE*, pages 39–48. IEEE, 2000.

17 Joshua Brakensiek and Aviad Rubinstein. Constant-factor approximation of near-linear edit distance in near-linear time. In *STOC*, pages 685–698. ACM, 2020.

18 Karl Bringmann and Tobias Friedrich. Exact and efficient generation of geometric random variates and random graphs. In *ICALP*, volume 7965 of *LNCS*, pages 267–278, 2013.

19 Karl Bringmann and Marvin Künnemann. Quadratic conditional lower bounds for string problems and dynamic time warping. In *FOCS*, pages 79–97. IEEE, 2015.

20 Karl Bringmann and Marvin Künnemann. Multivariate fine-grained complexity of longest common subsequence. In *SODA*, pages 1216–1235. SIAM, 2018.

21 Diptarka Chakraborty, Debarati Das, Elazar Goldenberg, Michal Koucký, and Michael E. Saks. Approximating edit distance within constant factor in truly sub-quadratic time. In *FOCS*, pages 979–990. IEEE, 2018.

22 David Eppstein, Zvi Galil, Raffaele Giancarlo, and Giuseppe F. Italiano. Sparse dynamic programming I: linear cost functions. *J. ACM*, 39(3):519–545, 1992.

23 Elazar Goldenberg, Robert Krauthgamer, and Barna Saha. Sublinear algorithms for gap edit distance. In *FOCS*, pages 1101–1120. IEEE, 2019.

24 MohammadTaghi Hajiaghayi, Masoud Seddighin, Saeed Seddighin, and Xiaorui Sun. Approximating LCS in linear time: Beating the $\sqrt{n}$ barrier. In *SODA*, pages 1181–1200. SIAM, 2019.

25 MohammadTaghi Hajiaghayi, Masoud Seddighin, Saeed Seddighin, and Xiaorui Sun. Approximating LCS in linear time: Beating the $\sqrt{\text{n}}$ barrier. *CoRR*, abs/2003.07285, 2020. `arXiv:2003.07285`.

26 Daniel S. Hirschberg. Algorithms for the longest common subsequence problem. *J. ACM*, 24(4):664–675, 1977.

27 James W. Hunt and Thomas G. Szymanski. A fast algorithm for computing longest subsequences. *Commun. ACM*, 20(5):350–353, 1977.

28 Costas S. Iliopoulos and Mohammad Sohel Rahman. A new efficient algorithm for computing the longest common subsequence. *Theory Comput. Syst.*, 45(2):355–371, 2009.

29 Michal Koucký and Michael E. Saks. Constant factor approximations to edit distance on far input pairs in nearly linear time. In *STOC*, pages 699–712. ACM, 2020.

30 William J. Masek and Mike Paterson. A faster algorithm computing string edit distances. *J. Comput. Syst. Sci.*, 20(1):18–31, 1980.

31 Eugene W. Myers. An $O(ND)$ difference algorithm and its variations. *Algorithmica*, 1(2):251–266, 1986.

32 Narao Nakatsu, Yahiko Kambayashi, and Shuzo Yajima. A longest common subsequence algorithm suitable for similar text strings. *Acta Informatica*, 18:171–179, 1982.

33 Aviad Rubinstein, Saeed Seddighin, Zhao Song, and Xiaorui Sun. Approximation algorithms for LCS and LIS with truly improved running times. In *FOCS*, pages 1121–1145. IEEE, 2019.

34 Aviad Rubinstein and Zhao Song. Reducing approximate longest common subsequence to approximate edit distance. In *SODA*, pages 1591–1600. SIAM, 2020.

35 Robert A. Wagner and Michael J. Fischer. The string-to-string correction problem. *J. ACM*, 21(1):168–173, 1974.

36 Sun Wu, Udi Manber, Gene Myers, and Webb Miller. An $O(NP)$ sequence comparison algorithm. *Inf. Process. Lett.*, 35(6):317–323, 1990.

Current Algorithms for Detecting Subgraphs of Bounded Treewidth Are Probably Optimal

Karl Bringmann
Saarland University, Saarland Informatics Campus, Saarbrücken, Germany
Max Planck Institute for Informatics, Saarland Informatics Campus, Saarbrücken, Germany

Jasper Slusallek
Saarland University, Saarland Informatics Campus, Saarbrücken, Germany

Abstract

The Subgraph Isomorphism problem is of considerable importance in computer science. We examine the problem when the pattern graph H is of bounded treewidth, as occurs in a variety of applications. This problem has a well-known algorithm via color-coding that runs in time $O(n^{\mathrm{tw}(H)+1})$ [Alon, Yuster, Zwick'95], where n is the number of vertices of the host graph G. While there are pattern graphs known for which Subgraph Isomorphism can be solved in an improved running time of $O(n^{\mathrm{tw}(H)+1-\varepsilon})$ or even faster (e.g. for k-cliques), it is not known whether such improvements are possible for all patterns. The only known lower bound rules out time $n^{o(\mathrm{tw}(H)/\log(\mathrm{tw}(H)))}$ for any class of patterns of unbounded treewidth assuming the Exponential Time Hypothesis [Marx'07].

In this paper, we demonstrate the existence of maximally hard pattern graphs H that require time $n^{\mathrm{tw}(H)+1-o(1)}$. Specifically, under the Strong Exponential Time Hypothesis (SETH), a standard assumption from fine-grained complexity theory, we prove the following asymptotic statement for large treewidth t:

> For any $\varepsilon > 0$ there exists $t \geq 3$ and a pattern graph H of treewidth t such that Subgraph Isomorphism on pattern H has no algorithm running in time $O(n^{t+1-\varepsilon})$.

Under the more recent 3-uniform Hyperclique hypothesis, we even obtain tight lower bounds for each specific treewidth $t \geq 3$:

> For any $t \geq 3$ there exists a pattern graph H of treewidth t such that for any $\varepsilon > 0$ Subgraph Isomorphism on pattern H has no algorithm running in time $O(n^{t+1-\varepsilon})$.

In addition to these main results, we explore **(1)** colored and uncolored problem variants (and why they are equivalent for most cases), **(2)** Subgraph Isomorphism for $\mathrm{tw} < 3$, **(3)** Subgraph Isomorphism parameterized by pathwidth instead of treewidth, and **(4)** a weighted variant that we call Exact Weight Subgraph Isomorphism, for which we examine pseudo-polynomial time algorithms. For many of these settings we obtain similarly tight upper and lower bounds.

2012 ACM Subject Classification Theory of computation → Design and analysis of algorithms; Theory of computation → Computational complexity and cryptography

Keywords and phrases subgraph isomorphism, treewidth, fine-grained complexity, hyperclique

Digital Object Identifier 10.4230/LIPIcs.ICALP.2021.40

Category Track A: Algorithms, Complexity and Games

Related Version *Full Version*: `https://arxiv.org/abs/2105.05062`

Funding *Karl Bringmann*: This work is part of the project TIPEA that has received funding from the European Research Council (ERC) under the European Unions Horizon 2020 research and innovation programme (grant agreement No. 850979).

1 Introduction

The Subgraph Isomorphism problem is commonly defined as follows: Given a graph H on k vertices, and a graph G on n vertices, is there a (not necessarily induced) subgraph of G which is isomorphic to H?

Subgraph Isomorphism generalizes many problems of independent interest, such as the k-path and k-clique problems. The problem is also of considerable interest when H is less structured, with applications to discovering patterns in graphs that, for example, arise from biological processes such as gene transcription or food networks, from social interaction, from electronic circuits, from neural networks [42], from chemical compounds [47] or from control flow in programs [19]. In some fields, the problem is sometimes referred to as the search for "network motifs", i.e. subgraphs that appear more often than would normally be expected.

In its general form, the problem is NP-hard. We are interested in solving the problem when the pattern graph H is "tree-like" or "path-like", i.e. when the treewidth $\mathrm{tw}(H)$ or the pathwidth $\mathrm{pw}(H)$ of H is bounded. Such pattern graphs of low treewidth or pathwidth often arise in practice when considering the structure of chemical compounds, the control flow of programs, syntactic relations in natural language, or many other graphs from practical applications (see e.g. [14, 16]). On the theoretical side, many restricted classes of graphs have bounded treewidth, see also [15]. Restricting NP-hard problems to graphs of bounded tree- and pathwidth often yields polynomial-time algorithms, and Subgraph Isomorphism is no exception. Most notably, the classic Color-Coding algorithm by Alon, Yuster and Zwick [9] solves the problem by a Las Vegas algorithm in expected time $O(n^{\mathrm{tw}(H)+1}g(k))$, or by a deterministic algorithm in time $\widetilde{O}(n^{\mathrm{tw}(H)+1}g(k))$, where g is a computable function (and $\widetilde{O}(\cdot)$ is used to suppress factors that are polylogarithmic in the input size). In other words, if the pattern graph H has treewidth bounded by some constant, the problem is fixed-parameter tractable when parameterized by k. The Color-Coding algorithm is also relevant for practical purposes: Recently, it has received an efficient implementation, which tested well against state-of-the-art programs for Subgraph Isomorphism [39].

Many researchers wondered whether the Color-Coding algorithm can be improved. This question has been studied in many different directions, including the following:

- Marx [40] showed that no algorithm solves the Subgraph Isomorphism problem in time $O(n^{o(\mathrm{tw}(H)/\log(\mathrm{tw}(H)))}g(k))$ unless the Exponential Time Hypothesis (ETH) fails, and this even holds when restricted to any class of pattern graphs of unbounded treewidth.
- A series of work has improved the computable function g, see e.g. [10, 29, 44].
- For many special pattern graphs faster algorithms have been found; the most famous example is the k-Clique problem, which can be solved in time $O(n^{k\omega/3}g(k))$ [43][1].

In this paper, we use a different angle to approach the question whether Color-Coding can be improved. We ask whether there exist "hard" pattern graphs:

Do there exist pattern graphs H for which Subgraph Isomorphism cannot be solved in time $O(n^{\mathrm{tw}(H)+1-\varepsilon})$ for any constant $\varepsilon > 0$?

To the best of our knowledge, this question has not been previously studied. As our main result, we (conditionally) give a positive answer to this question. More precisely, we show that for every $t \geq 3$ there exists a pattern graph H with $\mathrm{tw}(H) = t$ for which Subgraph Isomorphism cannot be solved in time $O(n^{\mathrm{tw}(H)+1-\varepsilon})$ for any constant $\varepsilon > 0$, assuming the 3-uniform k-Hyperclique hypothesis; see Section 1.3 for details on this hypothesis. We also show a slightly weaker statement under the Strong Exponential Time Hypothesis. This conditionally shows that the Color-Coding algorithm by Alon, Yuster and Zwick cannot be significantly improved while still working for all pattern graphs.

[1] This bound assumes that k is divisible by 3; there are similar results for general k [27].

For the case of $\mathrm{tw}(H) = 2$, an algorithm of Curticapean, Dell and Marx [24] can be adapted such that it solves Subgraph Isomorphism in time $\widetilde{O}(n^\omega g(k))$. We unify this with the algorithm of Alon, Yuster and Zwick by showing that both time bounds can be achieved within a simple framework. In particular, we use so-called k-wise matrix products, an operation which was introduced in its general form in [31] and studied further in [38].

We also study the Subgraph Isomorphism problem when the pathwidth of H is bounded, and specialize our framework to show slight improvements in running time compared to the case of bounded treewidth. Here, we use rectangular matrix products, for which faster-than-naive algorithms are known [30].

In further results, our focus is on the weighted variant Exact Weight Subgraph Isomorphism, where the subgraph must also have total weight equal to zero. In this work, we consider both the node-weighted and the edge-weighted variant of this problem, for both bounded treewidth and bounded pathwidth, allowing the maximum absolute weight W to appear in the running time (i.e. the pseudopolynomial-time setting). We show that our algorithms for the unweighted case can be adapted to the weighted case. We also speed up the weighted algorithms by using the fact that fast convolution (or rather, sumset computation), a folklore technique that lies at the core of many fast algorithms for problems with weights (e.g. [20, 18, 35, 34, 17, 12] and [23, exercise 30.1.7]), can easily be adapted to work with rectangular matrices and tensors. We furthermore show tight conditional lower bounds in many cases. Last but not least, we show that our algorithms can be slightly improved for the case of node-weighted instances for which either the pathwidth of H is bounded, or H is a tree. These algorithms also rely on fast rectangular matrix products.

1.1 Related Work

Additional to the conditional lower bound of $O(n^{o(\mathrm{tw}(H)/\log(\mathrm{tw}(H)))}g(k))$ by Marx [40], there is an unconditional lower bound of $O(n^{\kappa(H)})$ for the size of any AC^0-circuit, for some graph parameter $\kappa(H) = \Omega(\mathrm{tw}(H)/\log(\mathrm{tw}(H)))$, which holds even when considering the average case [37]. Interestingly, the factor of $1/\log(\mathrm{tw}(H))$ does not seem to be an artefact of the proof: There is an AC^0-circuit of size $O(n^{o(\mathrm{tw}(H))}g(k))$ that solves the problem on certain unbounded-treewidth classes in the average case [45].

In a different direction, Dalirrooyfard et al. [25] design various reductions from k-Clique to Subgraph Isomorphism, among other results. They also present results on the detection of induced subgraphs (we focus on non-induced subgraphs).

For the weighted variant of Subgraph Isomorphism, lower bounds under the k-Sum hypothesis for stars, paths, cycles and some other pattern graphs are presented in [5]. Edge-weighted triangle detection has a by-now classic $O(n^{3-\varepsilon})$ lower bound under both the 3Sum hypothesis and the APSP hypothesis [7]. On the other hand, in [6], it is proven that finding node-weighted k-cliques can be done almost as quickly as finding unweighted k-cliques. We are not aware of any results on the Exact Weight Subgraph Isomorphism problem when W may appear in the running time (i.e. a pseudopolynomial-time algorithm), which is what we focus on here.

In our work, we pose no restrictions on the host graph G. For an extensive classification of Subgraph Isomorphism with respect to various parameters of both G and H, see [41].

1.2 Hardness Assumptions

The most standard hypothesis from fine-grained complexity theory is the Strong Exponential Time Hypothesis (SETH) [32], which postulates that for any $\varepsilon > 0$ there exists $k \geq 3$ such that k-Sat on n variables cannot be solved in time $O^*(2^{(1-\varepsilon)n})$.

More recent is the Hyperclique hypothesis. In the h-uniform k-Hyperclique problem, for a given h-uniform hypergraph we want to decide whether there exist a set of k vertices such that every size-h subset of these vertices forms a hyperedge. For any $k > 3$, the 3-uniform k-Hyperclique hypothesis postulates that this problem cannot be solved in time $O(n^{k-\varepsilon})$ for any $\varepsilon > 0$. This hypothesis has also been formulated when replacing 3 with any $h < k$, getting progressively more believable with larger h. For a more in-depth discussion of the believability of this hypothesis we refer to [38, Section 7].

Note that we will also use the h-uniform Hyperclique hypothesis for various h, which is simply the conjecture that the h-uniform k-Hyperclique hypothesis is true for all $k > h$.

Related to this is the k-Clique conjecture, which postulates that the k-Clique problem (which is the 2-uniform k-Hyperclique problem) cannot be solved in time $O(n^{\omega k/3-\varepsilon})$ for any constant $\varepsilon > 0$, where $\omega < 2.373$ [36] is the exponent of matrix multiplication.

1.3 Our Results

Unweighted Subgraph Isomorphism with Bounded Treewidth. First, consider the case of the unweighted Subgraph Isomorphism problem for bounded-treewidth pattern graphs H. As was said, and as we will re-prove with a unified algorithm later, this problem has an algorithm running in time $\widetilde{O}(n^{\mathrm{tw}(H)+1})$ for $\mathrm{tw}(H) \geq 3$. We show tight conditional lower bounds by proving the following obstacles to faster algorithms, which use the k-clique hypothesis and the h-uniform k-hyperclique hypothesis. Note that when we say, for some x, that an algorithm has running time $O(n^{x-\varepsilon})$, what we mean is that the algorithm runs in time $O(n^{x-\varepsilon})$ for some constant $\varepsilon > 0$.

▶ **Theorem 1.** *The following statements are true.*

1. *For each $t \geq 3$ and each $3 \leq h \leq t$, there exists a connected, bipartite pattern graph $\mathcal{H}_{t,h}$ of treewidth t such that there cannot be an algorithm solving the Subgraph Isomorphism problem on pattern graph $\mathcal{H}_{t,h}$ in time $O(n^{t+1-\varepsilon})$ unless the h-uniform $h(t+1)$-hyperclique hypothesis fails.*
2. *For each $t \geq 2$ and each $h \geq 3$, there exists a connected, bipartite pattern graph $\mathcal{H}_{t,h}$ of treewidth t such that there cannot be an algorithm solving the Subgraph Isomorphism problem on pattern graph $\mathcal{H}_{t,h}$ in time $O(n^{t-\varepsilon})$ unless the h-uniform ht-hyperclique hypothesis fails.*
3. *For each $t \geq 2$, there exists a connected, bipartite pattern graph $\mathcal{H}_t$ of treewidth t such that there cannot be an algorithm solving the Subgraph Isomorphism problem on pattern graph $\mathcal{H}_t$ in time $O(n^{(t+1)\omega/3-\varepsilon})$ unless the $(t+1)$-Clique hypothesis fails.*

Indeed, with the very same reduction, we also get an obstacle from SETH. However, the lower bound it provides is not as tight as the above, and in the case of the second part does not work for each target treewidth t.

▶ **Theorem 2.** *Assuming SETH, the following two statements are true.*

1. *For any $t \geq 3$ and any $\varepsilon > 0$ there exists a pattern graph $\mathcal{H}_{t,\varepsilon}$ of treewidth t such that there cannot be an algorithm solving all instances of Subgraph Isomorphism with pattern graph $\mathcal{H}_{t,\varepsilon}$ in time $O(n^{t-\varepsilon})$.*
2. *For any $\varepsilon > 0$ there exists a $t \geq 3$ and a pattern graph $\mathcal{H}_\varepsilon$ of treewidth t such that there cannot be an algorithm solving all instances of Subgraph Isomorphism with pattern graph $\mathcal{H}_\varepsilon$ in time $O(n^{t+1-\varepsilon})$.*

On the algorithmic side, we present an algorithm that achieves matching running times (as listed in Theorem 3 below). As was said, the results in the following theorem are not new. Part 1 was shown via Color-Coding in [9] and part 2 follows from techniques in [24].

We unify these two results by providing a single, relatively simple algorithmic technique achieving both, based on k-wise matrix products. These techniques are later expanded to also work for the weighted version, where they then achieve new results. In the following, $\omega < 2.373$ [36] is the exponent of matrix multiplication.

▶ **Theorem 3.** *There are algorithms which, given an arbitrary instance $\phi = (H, G)$ of* Subgraph Isomorphism *where H has treewidth $\mathrm{tw}(H)$, solve ϕ in*

1. *time $\widetilde{O}(n^{\mathrm{tw}(H)+1}g(k))$ when $\mathrm{tw}(H) \geq 3$,*
2. *time $\widetilde{O}(n^{\omega}g(k))$ when $\mathrm{tw}(H) = 2$, and*
3. *time $\widetilde{O}(n^{2}g(k))$ when $\mathrm{tw}(H) = 1$,*

where $k := |V(H)|, n := |V(G)|$ and g is a computable function.

Semi-Equivalence of Hyperclique and Subgraph Isomorphism. In the full version of the paper, we also discuss how our results not only show a reduction from Hyperclique to Subgraph Isomorphism with bounded treewidth, but also in the other direction. For this, we show that calculating the boolean k-wise matrix products, which is the bottleneck in our algorithm for bounded-treewidth Subgraph Isomorphism, is actually equivalent to the k-uniform $(k+1)$-Hypergraph problem. Hence we have a reduction in the second direction. This gives an interesting intuition for why the Hyperclique hypothesis is the "correct" conjecture to prove conditional hardness of Subgraph Isomorphism for bounded treewidth.

We remark that this does not lead to a full equivalence of these problems because the uniformity (i.e. the size of hyperedges) of the Hyperclique problem we reduce from in the first reduction is much smaller than the size of the hypercliques we search for. Hence we only have a reduction from a Hyperclique instance with small edge uniformity to Subgraph Isomorphism, and a reduction from Subgraph Isomorphism to Hyperclique instances with large edge uniformity.

Weighted Subgraph Isomorphism with Bounded Treewidth. Now consider the weighted version of Subgraph Isomorphism for bounded-treewidth graphs H. Recall that the weighted version can be either node- or edge-weighted and is defined such that the weights in the solution subgraph must have total weight zero. A trivial dynamic programming algorithm on the tree decomposition achieves a running time of $\widetilde{O}(n^{\mathrm{tw}(H)+1} \cdot W \log W)$ for $\mathrm{tw}(H) \geq 3$.

Note that these results show conditional lower bounds even when the maximum weight is restricted to $W = \Theta(n^{\gamma})$, for any constant $\gamma > 0$.

▶ **Theorem 4.** *For both the node- and edge weighted variant of the problems, the following statements are true.*

1. *For each $t \geq 3$, each $\gamma \in \mathbb{R}^+$ and each $3 \leq h \leq t$, there exists a connected, bipartite graph $\mathcal{H}_{t,h,\gamma}$ of treewidth t such that there cannot be an algorithm solving the Exact Weight Subgraph Isomorphism problem on pattern graph $\mathcal{H}_{t,h,\gamma}$ for instances with maximum weight $W = \Theta(n^{\gamma})$ in time $O(n^{t+1-\varepsilon}W)$, unless the h-uniform Hyperclique hypothesis fails.*
2. *For each $t \geq 1$, each $\gamma \in \mathbb{R}^+$ and each $h \geq 3$, there exists a connected, bipartite graph $\mathcal{H}_{t,h,\gamma}$ of treewidth t such that there cannot be an algorithm solving the Exact Weight Subgraph Isomorphism problem on pattern graph $\mathcal{H}_{t,h,\gamma}$ for instances with maximum weight $W = \Theta(n^{\gamma})$ in time $O(n^{t-\varepsilon}W)$, unless the h-uniform Hyperclique hypothesis fails.*
3. *For each $t \geq 1$ and each $\gamma \in \mathbb{R}^+$, there exists a connected, bipartite graph $\mathcal{H}_{t,\gamma}$ of treewidth t such that there cannot be an algorithm solving the Exact Weight Subgraph Isomorphism problem on pattern graph $\mathcal{H}_{t,\gamma}$ for instances with maximum weight $W = \Theta(n^{\gamma})$ in time $O(n^{(t+1)\omega/3-\varepsilon}W^{\omega/3})$, unless the Clique hypothesis fails.*

Similar lower bounds also hold when trying to reduce the exponent of W instead of n. Meaning there is also no algorithm of running time $O(n^{t+1}W^{1-\varepsilon})$ in part 1, etc.

On the algorithmic side, we present an algorithm that achieves matching running times for $\mathrm{tw}(H) \geq 3$, and almost matching running times for $\mathrm{tw}(H) = 1, 2$. Note that in terms of exponents, the first algorithm below is not better than the naive one with running time $O(n^{\mathrm{tw}+1}W \log W)$. However, it avoids a factor of $\log W$ in the largest term, and instead appends it to a smaller term, so in a way it presents an improvement of $\log W$ in the running time. Specifically, we show

▶ **Theorem 5.** *There are algorithms which, given an arbitrary instance $\phi = (H, G, w)$ of the* Exact Weight Subgraph Isomorphism *problem where H has treewidth $\mathrm{tw}(H)$, solve ϕ in*
1. *time $\widetilde{O}((n^{\mathrm{tw}(H)+1}W + n^{\mathrm{tw}(H)}W \log W)g(k))$ when $\mathrm{tw}(H) \geq 3$,*
2. *time $\widetilde{O}((n^{\omega}W + n^2 W \log W)g(k))$ when $\mathrm{tw}(H) = 2$, or*
3. *time $\widetilde{O}((n^2 W + nW \log W)g(k))$ when $\mathrm{tw}(H) = 1$,*

where $n := |V(G)|, k := |V(H)|$, g is a computable function, and W is the maximum absolute weight in the image of w.

Comparing these upper bounds with the lower bounds from Theorem 5, we have a tight lower bound for the weighted case with $\mathrm{tw}(H) \geq 3$. For weighted $\mathrm{tw}(H) = 2$, we have a lower bound which is tight except for the exponent of $\omega/3$ to W; it is unclear whether this can be strengthened. The lower bound for weighted graphs with $\mathrm{tw}(H) = 1$ is obviously not tight: We have an upper bound of $\widetilde{O}(n^2 W + nW \log W)$, but our lower bounds only states that it requires time $O(n^{1-o(1)}W^{1-o(1)})$ and $O(n^{2\omega/3-o(1)}W^{\omega/3-o(1)})$. Tighter lower bounds for this case remain an important open problem.

Unweighted Subgraph Isomorphism with Bounded Pathwidth. So far we have only looked at the case of bounded treewidth. However, similar results hold for the case of bounded pathwidth. Let us start with the unweighted Subgraph Isomorphism problem.

Note that we do not get any lower bounds for the current setting. This is because we prove all our lower bounds by showing an equivalence of the standard Subgraph Isomorphism problem to a colored variant (see also Section 1.4), and then proving a lower bound for the colored version. We do not know how to prove such an equivalence for the current setting, therefore we do not get lower bounds in this case; we leave this as an open problem.

Since a path decomposition is always also a tree decomposition, we trivially get upper bounds as in Theorem 3 (when replacing treewidth by pathwidth). However, we can do better by using rectangular matrix multiplication to speed up the computation. For $z \in \mathbb{R}^+$, let $\omega(z)$ be the smallest real number such that multiplying a $n \times n$ matrix with a $n \times n^z$ matrix can be done in time $O(n^{\omega(z)})$[2]. We prove the following upper bounds.

▶ **Theorem 6.** *There are algorithms which, given an arbitrary instance $\phi = (H, G)$ of* Subgraph Isomorphism *where H has pathwidth p, solve ϕ in*
1. *time $\widetilde{O}(n^{\omega(p-1)}g(k))$ when $p \geq 2$, and*
2. *time $\widetilde{O}(n^2 g(k))$ when $p = 1$*

where $k := |V(H)|$ and $n := |V(G)|$.

[2] Le Gall [30] has shown that there are fast algorithms for rectangular matrix multiplication based on the Coppersmith-Winograd method [22, 36] used for square matrix multiplication. Among other values, he shows $\omega(0.31) = 2$, $\omega(2) < 3.26$, $\omega(3) < 4.2$, $\omega(4) < 5.18$ and $\omega(5) < 6.16$. See [30] for an extensive table of such values.

We certainly have $p \leq \omega(p-1) < p+1$, so these results represent only a minor improvement, which is nonetheless important because it "beats" the lower bound for treewidth. Hence the lower bound for pathwidth cannot be the same as for treewidth.

Weighted Subgraph Isomorphism with Bounded Pathwidth. We also analyze the bounded-pathwidth pattern graph version of Weighted Subgraph Isomorphism. Specifically, we get the following lower bound.

▶ **Theorem 7** (Theorem 4 for pathwidth)**.** *Parts 2 and 3 of Theorem 4 also hold when replacing the treewidth t by the pathwidth p. Part 1 does not hold.*

And on the algorithmic side, we can again use rectangular matrix multiplication to improve on the algorithms from the case of bounded treewidth. Specifically, we get:

▶ **Theorem 8.** *There are algorithms which, given an arbitrary instance $\phi = (H, G, w)$ of the Exact Weight Subgraph Isomorphism problem, solve ϕ in*

1. *time $\widetilde{O}((n^{\omega(\mathrm{pw}(H)-1)}W + n^{\mathrm{pw}(H)}W \log W)g(k))$ when $\mathrm{pw}(H) \geq 2$,*
2. *time $\widetilde{O}((n^2W + nW \log W)g(k))$ when $\mathrm{pw}(H) = 1$*

where $n := |V(G)|, k := |V(H)|$, and W is the maximum absolute weight in the image of w.

For $\mathrm{pw}(H) \geq 3$, the lower bounds are therefore obviously not tight (at least for current algorithms), unless significant advances in matrix multiplication techniques are made. For $\mathrm{pw}(H) = 1, 2$, the situation is the same as with treewidth, see the discussion of Theorem 5.

Improvements to Special Cases of Weighted Subgraph Isomorphism. It is natural to think that the exponents $\frac{\omega}{3}$ to W in the lower bounds of Theorems 4 and 7 are only artefacts of the reduction, and that with more advanced methods, this exponent can be improved to 1. However, the following two theorems show that this notion is false for $\mathrm{tw}(H) = 1$ and $\mathrm{pw}(H) = 1, 2$, at least when considering the node-weighted case. Indeed, for $\mathrm{tw}(H) = 1$ (or $\mathrm{pw}(H) = 1$) and $W = n$, these bounds are tight, so further general improvements on the exponent are impossible.

Specifically, Theorems 9 and 10 show the following improvements of the algorithms from Theorems 5 and 8 for small tree- or pathwidth. Let $\mathrm{MM}(n, n, x)$ be the time in which a $n \times n$ matrix can be multiplied with with a $n \times x$ matrix.

▶ **Theorem 9.** *There is an algorithm which, given an arbitrary instance $\phi = (H, G, w)$ of the node-weighted Exact Weight Subgraph Isomorphism problem where H is a tree, solves ϕ in time $\widetilde{O}((\mathrm{MM}(n, n, W) + nW \log W)g(k))$.*

▶ **Theorem 10.** *There is an algorithms which, given an arbitrary instance $\phi = (H, G, w)$ of the node-weighted Exact Weight Subgraph Isomorphism problem, solves ϕ in time $\widetilde{O}(\mathrm{MM}(n, n, n^{\mathrm{pw}(H)-1}W)g(k))$.*

For $W = O(n^\gamma)$, the running time of Theorem 9 is $\widetilde{O}(n^{\omega(\gamma)}\,\mathrm{poly}(k))$. This implies several interesting facts. One such fact is that due to the known convergence $\lim_{\gamma\to\infty} \omega(\gamma) - \gamma = 1$ [21], we have that for node-weighted trees, there cannot be a lower bound of $O(n^{(1+\varepsilon-o(1)}W)$ for any $\varepsilon > 0$ that holds for any constant $\gamma > 0$. A similar result holds for bounded-pathwidth graphs. Other implications are discussed in the full version.

1.4 Equivalence of the Colored and Uncolored Problems

All mentioned algorithms and conditional lower bounds are shown for the restricted problem of Colored Subgraph Isomorphism, where the nodes of G and H are colored with $|V(H)|$ colors and the isomorphism must preserve colors, as also studied in [40]. In the full version of the paper, we prove that the standard and the colored variant of Subgraph Isomorphism can be solved in essentially the same running time in almost all cases. Specifically, we show the following lemma.

▶ **Lemma 11.** *Let ρ be any graph parameter.*

1. *If there is a $T(n,k,\rho(H))$ time algorithm for Colored Subgraph Isomorphism, then there is a $\widetilde{O}(T(kn,k,\rho(H))g(k))$ time algorithm for Subgraph Isomorphism, where g is some computable function.*
2. *If there is a $T(n,k,\rho(H),W)$ time algorithm for Exact Weight Colored Subgraph Isomorphism, then there is a $\widetilde{O}(T(kn,k,\rho(H),W)g(k))$ time algorithm for Exact Weight Subgraph Isomorphism, where g is some computable function.*
3. *Let $\mathrm{tw}(H) \geq 2$. If there is a $T(n,k,\mathrm{tw}(H))$ time algorithm for Subgraph Isomorphism, then there is a $O(T(\mathrm{poly}(k)n,\mathrm{poly}(k),\mathrm{tw}(H)) + \mathrm{poly}(k)n^2)$ time algorithm for Colored Subgraph Isomorphism.*
4. *If there is a $T(n,k,\rho(H),W)$ time algorithm for Exact Weight Subgraph Isomorphism, then there is a $O(T(2n,2k,\rho(H),2^kW) + \mathrm{poly}(k)n^2)$ time algorithm for Exact Weight Colored Subgraph Isomorphism.*

This lemma enables us to prove results for (Exact Weight) Subgraph Isomorphism while only talking about the more structured colored variants of the problem.

2 Technical Overview of Our Main Results

We now give proof sketches of our main lower bound results. We do not give proofs for the upper bound results due to their technicality. The full version of this paper contains all proofs with full detail.

2.1 Lower Bound for Subgraph Isomorphism

Our main result is the existence of the hard pattern graphs for bounded-treewidth Subgraph Isomorphism. We now prove their existence for treewidth at least 3 under the Hyperclique hypothesis, i.e. part 1 of Theorem 1.

The exact statement we prove is that for each $t \geq 3$ and each $3 \leq h \leq t$, there exists a pattern graph of treewidth t such that Subgraph Isomorphism cannot be solved in time $O(n^{t+1-\varepsilon})$ on that pattern graph unless the h-uniform $h(t+1)$-hyperclique hypothesis fails. Note that the proof actually shows this for the colored variant of Subgraph Isomorphism, after which we can use Lemma 11 to transfer the lower bound to the uncolored problem.

Proof sketch. See Figure 1 for a sketch of the reduction. Let $t \geq 3$ and $3 \leq h \leq t$ be given and assume that Subgraph Isomorphism can be solved in time $O(n^{t+1-\varepsilon})$ on pattern graphs of treewidth t. We show that the h-uniform $h(t+1)$-hyperclique hypothesis fails.

Construction of H. We construct a pattern graph H as a bipartite graph with vertex set $A \cup B$ as follows. Writing $[c] := \{1,\dots,c\}$, we set $A := [t+1]$ and $B := \binom{A \times [h]}{h}$. We connect a vertex $b = ((a_1,j_1),..,(a_h,j_h))$ in B to a vertex a in A if $a = a_\ell$ for some ℓ. Set $k := |A| + |B|$.

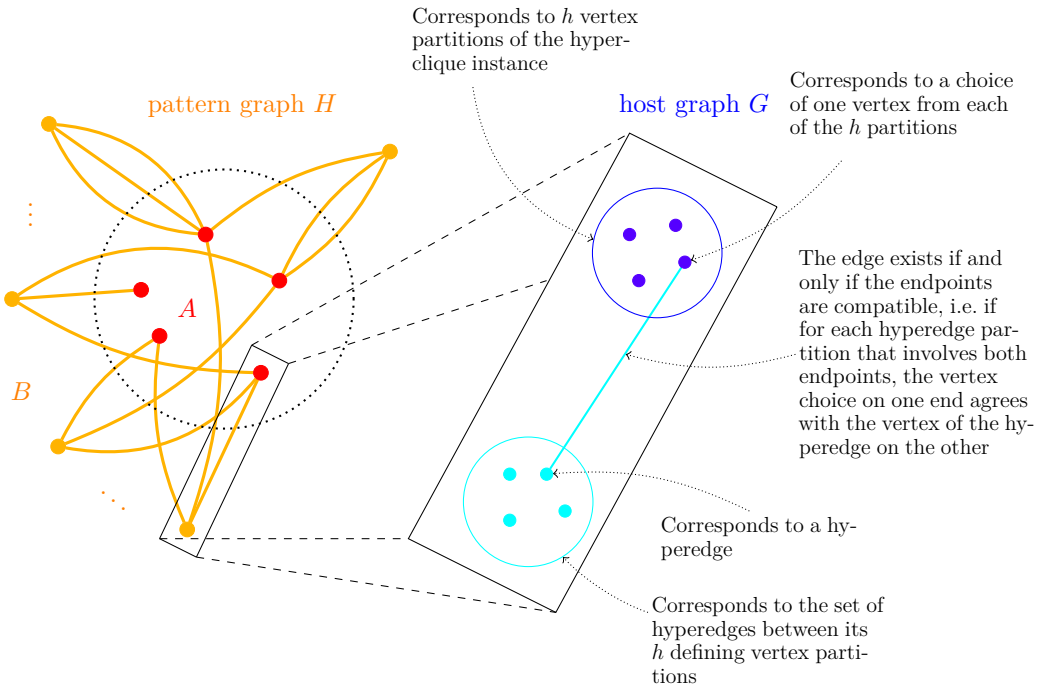


Figure 1 A sketch of the reduction we use to prove part 1 of Theorem 1 where $h = 3$ and $t = 4$. Note that this is only a partial sketch of the pattern graph. We use multiedges to signify that for the endpoints $a \in A$ and $b = ((a_1, j_1), \ldots, (a_h, j_h)) \in B$ there exists more than one ℓ such that $a = a_\ell$.

We show that this pattern has a treewidth of t, via a well-known characterization of treewidth as a graph-theoretic game: A graph F has treewidth $\leq t$ if and only if $t + 1$ cops can catch[3] a robber on F [46]. To show the bound on the treewidth of H, initially place a cop on each vertex of A. No matter on which vertex of B the robber starts, they are surrounded by cops. Since every vertex in B has $h \leq t < t + 1$ neighbors in A, there must exist some cop which is not adjacent to the robber, so this cop can catch the robber in a single step. This concludes the proof that the pattern graph H has treewidth t.

Construction of G. Now given a $h(t + 1)$-partite hypergraph H', i.e. an instance of the h-uniform k'-Hyperclique problem for $k' := h(t + 1)$, we write the vertex set of H' as $U_{1,1} \cup \ldots \cup U_{1,h} \cup \ldots \cup U_{t+1,1} \cup \ldots \cup U_{t+1,h}$. Let N_H be the number of vertices vertices in each partition and $n_H = O(N_H)$ the number of vertices overall.

We construct a k-partite graph G as follows. For a in A we set $V_a := U_{a,1} \times \ldots \times U_{a,h}$. For $b = ((a_1, j_1), .., (a_h, j_h))$ in B, we set $V_b := E(H') \cap (U_{a_1,j_1} \times \ldots \times U_{a_h,j_h})$. This describes the k parts of the k-partite vertex set $V(G)$. Note that each part has size at most $N_G := N_H^h$. Now we construct the edges. For any a in A and $b = ((a_1, j_1), .., (a_h, j_h))$ in B with (a, b) in $E(H)$, consider an arbitrary $u = (u_1, .., u_h)$ in V_a and $u' = (u'_1, ..., u'_h)$ in V_b. We say that u and u' are "compatible" if for every ℓ with $a_\ell = a$ we have $u'_\ell = u_{j_\ell}$; in this case we connect u and u' by an edge. This finishes the construction of G.

[3] The game works as follows: The $k + 1$ cops select their starting vertices in the graph. Then the robber may choose their starting vertex. The cops can always see the robber and adapt their strategy accordingly. Similarly, the robber can see the cops. The game now proceeds in steps, where in each step, one of the cops chooses an arbitrary destination vertex and takes off via helicopter in the direction of that vertex. While the cop is travelling, the robber sees where they will land and may now move arbitrarily along edges of the graph, as long as they do not pass through stationary cops. When the robber has finished moving, the cop lands. The cops win if and only if they are guaranteed to catch the robber after a finite number of moves, and lose otherwise.

Correctness. Note that any colored subgraph isomorphism of H in G chooses vertices v_a in V_a for all a in A. This corresponds to choosing vertices $u_{i,j}$ in $U_{i,j}$ for all i in $[t+1]$, Moreover, the edges of a $h(t+1)$-hyperclique are in one-to-one correspondence with the set B. Since for each b in B the colored subgraph isomorphism of H in G needs to choose a vertex v_b in V_b, which corresponds to an edge between certain vertices $u_{i,j}$, we indeed check that the chosen vertices $u_{i,j}$ form an h-uniform $h(t+1)$-hyperclique.

Running Time. Trivially, the construction time and output size are $O(N_H^{2h})$ (actually, it is slightly better, but this is not important in this proof sketch), and G has $n = O(N_G) = O(N_H^h)$ vertices. Now if we can solve Subgraph Isomorphism in time $O(n^{t+1-\varepsilon})$, we solve the h-uniform $h(t+1)$-Hyperclique instance in time $O(N_H^{2h} + (N_H^h)^{t+1-\varepsilon}) = O(N_H^{h(t+1)-h\varepsilon}) = O(N_H^{h(t+1)-\varepsilon'}) = O(n_H^{h(t+1)-\varepsilon'})$. ◀

This shows part 1 of Theorem 1. Part 2 can be shown by the almost the exact same proof, except that now we choose the size of A to be t instead of $t+1$, and we start with an ht-hyperclique instead of an $h(t+1)$-hyperclique. It can be seen that in this case, the pattern graph still has treewidth t. The third part of the theorem can be seen by simply taking an instance of $(t+1)$-Clique and subdividing the edges in the obvious way to make the graph bipartite.

Let us also quickly mention how the proof of the slightly weaker bounds under SETH, i.e. Theorem 2, works. The split-and-list technique from [48] allows one to reduce the Satisfiability problem to Hyperclique. Using this technique, the following result was shown in [38, Lemma 9.1].

▶ **Lemma 12** ([38]). *Assuming SETH, for any $\varepsilon > 0$ there exists $h \geq 3$ such that for all $k > h$, the h-uniform k-Hyperclique problem is not in time $O(n^{k-\varepsilon})$.*

The SETH result now follows by using essentially the same reduction as above, but we prefix it by the reduction from SAT to Hyperclique.

2.2 Lower Bound for Exact Weight Subgraph Isomorphism

We also give lower bounds for the exact weight variant of the Subgraph Isomorphism problem. In particular, we prove the existence of hard pattern graphs for the bounded-treewidth Exact Weight Subgraph Isomorphism problem for any polynomial weight bound. We give this result for any treewidth which is at least 3, and under the Hyperclique hypothesis. This is part 1 of Theorem 4.

The exact statement we prove is that for each $t \geq 3$, $\gamma \in \mathbb{R}^+$ and $3 \leq h \leq t$, there exists a pattern graph of treewidth t such that Exact Weight Subgraph Isomorphism with maximum weight $W = \Theta(n^\gamma)$ cannot be solved in time $O(n^{t+1-\varepsilon}W)$ unless the h-uniform Hyperclique hypothesis fails. Again, we show this statement for the colored problem and transfer the lower bound via Lemma 11.

To do this, we will encode part of a hyperclique instance in the edges of the Exact Weight Subgraph Isomorphism problem, and the rest of the instance in the weights. To do the latter, we need to encode certain equality constraints only via weights. This can be done using so-called k-average free sets[4], which we define below.

[4] These k-average-free sets are a tool which are very useful for weighted problems, especially when they have additive elements. Such problems include k-sum, Subset Sum, Bin Packing, various scheduling problems, Tree Partitioning, Max-Cut, Maximum/Minimum Bisection, a Dominating Set variant with capacities, and similar [3, 4, 6, 11, 28, 33, 26]. Other uses of k-average-free sets in computer science include constructions in extremal graph theory, see e.g. [1, 2, 8].

▶ **Definition 13** (k-average free sets). *A set $S \subseteq \mathbb{Z}$ is called k-average-free if, for any $s_1, \dots, s_{k'+1} \in S$ with $k' \leq k$, we have $s_1 + \dots + s_{k'} = k' \cdot s_{k'+1}$ if and only if $s_1 = \dots = s_{k'+1}$. In other words, the average of $s_1, \dots, s_{k'} \in S$ is in S if and only if all s_i are equal.*

We use the following construction for k-average free sets, originally proven in [13], modified into a more useful version in [6] and formulated in this form in [3].

▶ **Lemma 14.** *There exists a universal constant $c > 0$ such that, for all constants $\varepsilon \in (0,1)$ and $k \geq 2$, a k-average-free set S of size n with $S \subseteq [0, k^{c/\varepsilon} n^{1+\varepsilon}]$ can be constructed in time* $\mathrm{poly}(n)$.

Let us now prove the statement about Exact Weight Subgraph Isomorphism. We will construct an instance that is node-weighted, however this can easily be converted into an edge-weighted version by moving the weight of each vertex to all of its incident edges.

Proof sketch. Let $t \geq 3$, $3 \leq h \leq t$ and $\gamma \in \mathbb{R}$ be given and assume that Exact Weight Subgraph Isomorphism can be solved in time $O(n^{t+1-\varepsilon}W)$ on instances where the pattern graph has treewidth t and all weights are bounded by $W = \Theta(n^\gamma)$. We show that the h-uniform k-hyperclique hypothesis fails for some large enough k.

Construction of H. We construct a pattern graph H as a graph with vertex set $(A_1 \cup A_2) \cup B$ as follows. We set $A_1 := [t+1]$, $A_2 := [r]$ (for some r large enough) and $B := \binom{(A_1 \cup A_2) \times [h]}{h}$. We connect a vertex $b = ((a_1, j_1), \dots, (a_h, j_h))$ in B to a vertex a in A_1 (not in A_2) if $a = a_\ell$ for some ℓ. Set $k := |A_1| + |A_2| + |B|$. By almost the same proof as in the unweighted version, it can be shown that this pattern H has treewidth t.

Grouping partitions. Now let an instance of the h-uniform k'-Hyperclique problem be given, and write the vertex set of H' as $U_1 \cup \dots \cup U_k$. Let N_H be the number of vertices vertices in each partition and $n_H = O(N_H)$ the number of vertices overall.

We construct the k-partite graph G with at most some number N_G of vertices in each partition as follows. We will encode a β-fraction of the Hyperclique instance in the weights of the final Exact Weight Subgraph Isomorphism instance, and a $(1-\beta)$-fraction in the edges, for some β chosen appropriately. To do this, we will choose β such that $\frac{\beta k'}{hr}, \frac{(1-\beta)k'}{h(t+1)} \in \mathbb{N}$ and then group the sets $U_1, \dots, U_{\beta k'}$ into hr groups and the sets $U_{\beta k'+1}, \dots, U_{k'}$ into $h(t+1)$ groups. Specifically, for each $(x,y) \in [r] \times [h]$, we create the set $U^1_{x,y} = U_{(xr+y-1)\frac{\beta k'}{hr}+1} \times \dots \times U_{(xr+y)\frac{\beta k'}{hr}}$, and for each $(x,y) \in [t+1] \times [h]$, we create the set $U^2_{x,y} = U_{\beta k' + (x(t+1)+y-1)\frac{(1-\beta)k'}{h(t+1)}+1} \times \dots \times U_{\beta k' + (x(t+1)+y)\frac{(1-\beta)k'}{h(t+1)}}$.

Vertices of G. Now for each a in A_1 we set $V_a := U^1_{a,1} \times \dots \times U^1_{a,h}$ and for each a in A_2 we set $V_a := U^2_{a,1} \times \dots \times U^2_{a,h}$. Finally, for each $b = ((a_1, j_1), \dots, (a_h, j_h))$ in B, where for each ℓ we have $a_\ell \in A_{i_\ell}$, we set $V_b := E(H') \cap (U^{i_1}_{a_1,j_1} \times \dots \times U^{i_h}_{a_h,j_h})$. This describes the k parts of the k-partite vertex set $V(G)$. We choose r large enough so that the maximum size of each part is $N_G := N_H^{(1-\beta)k'/(t+1)}$.

Edges of G Now we construct the edges and weights of the graph. Let us start with the edges. The construction here is basically the same as the construction of the edges in the unweighted proof in the last section. For each a in A_2 and $b = ((a_1, j_1), \dots, (a_h, j_h))$ in B with (a,b) in $E(H)$, consider an arbitrary $u = (u_1, \dots, u_h)$ in V_a and $u' = (u'_1, \dots, u'_h)$ in V_b. We say that u and u' are "compatible" if for every ℓ with $a_\ell = a$ we have $u'_\ell = u_{j_\ell}$; in this case we connect u and u' by an edge. This finishes the construction of the edges of G.

Weights of G. Now we construct the weights. We want to encode the same edge constraints as we just encoded for A_2, but now for A_1, and we have to use weights instead of edges. To do this, we use $|B|$-average free sets via the construction of Lemma 14. We simplify the usage in this shortened proof to avoid dealing with too many variables. We use the lemma to obtain in polynomial time (which we will treat as negligible here) a $|B|$-average free set S of size $N_H^{\beta k'/(hr)}$ such that $S \subseteq [0, C]$, where $C \approx O(N_H^{\beta k'/(hr)})$ (up to a factor of $(1+\varepsilon)$ in the exponent, but we will ignore this here for simplicity). From this, we can construct an arbitrary bijection $\varrho_S : [N_H^{\beta k'/(hr)}] \to S$.

To simplify our construction, we specify a target weight T (instead of the default target zero). We can easily get rid of this again later by subtracting T from the weights of all vertices of some set of the partition. The binary representation of T consists of hr blocks of $\lceil 2|B|C \rceil$ bits, indexed by pairs $(i,j) \in [r] \times [h]$, each containing the binary representation of $|B|C$. The block (i,j) represents the group $U^1_{i,j}$. The size of the blocks is large enough to prevent overflow between the blocks. Note that the maximum weight W now satisfies $\log_2(W) = \Theta(hr \log_2(2|B|C))$ and hence $W \approx O(N_H^{\beta k'})$.

Let us now actually specify the weights of the vertices, beginning with the vertices in V_a for $a \in A_1$. For each $(i,j) \in [r] \times [h]$, we relabel the elements of each $U^1_{i,j}$ as $\{1, \ldots, N_H^{\beta k'/(hr)}\}$. Now we define the weight of the vertex $V_a \ni u = (u_1, \ldots, u_h)$ to have, for each $i \in [h]$, the value $|B|C - |N(a)| \cdot \varrho_S(u_i)$ in the block (a,i) of its binary representation. Now we move on to the vertices in V_b for $b = ((a_1, j_1), \ldots, (a_h, j_h))$. We define the weight of the vertex $V_b \ni u' = (u'_1, \ldots, u'_h)$ to have, for each $i \in [h]$ such that $a_i \in A_1$, the value $\varrho_S(u'_i)$ in the block (a_i, j_i) of its binary representation.

All blocks and vertices which have not been assigned a weight yet are assigned a value of zero. This concludes the construction of G.

Correctness. Note that any colored subgraph isomorphism of H in G chooses vertices v_a in V_a for all a in $A_1 \cup A_2$. This corresponds to choosing vertices u_i in U_i for each $i \in [k']$. Moreover, the edges of a k'-hyperclique are in one-to-one correspondence with the set B. We simply need to show that the choice of hyperclique vertices induced by the choice of vertices in $A_1 \cup A_2$ agrees with the choice of hyperclique edges induced by the choice of vertices in B. For the vertices in A_2, this is easily seen to be ensured by the edges. For the vertices in A_1, we need to prove that the weights encode the same constraint. This, however, is simply the definition of a $|B|$-average free set: Consider the block (i,j) (where $(i,j) \in [r] \times [h]$) in the binary representation of the total weight of the subgraph. Suppose that for $i \in A_1$ the vertex $V_i \ni u = (u_1, \ldots, u_h)$ was selected, and that for each $B \ni b = ((a_1, j_1), \ldots, (a_h, j_h))$ with $\exists \ell : (a_\ell, j_\ell) = (i,j)$ the vertex $V_b \ni u' = (u'_1, \ldots, u'_h)$ was selected. Then by construction, the total value in the block (i,j) is the value $|B|C - |N(i)|\varrho_S(u_j)$ (where $N(i)$ is the neighbourhood of $i \in A_1$), plus the value $\varrho_S(u'_\ell)$ for all b as above. Note that the latter term has exactly $|N(i)|$ summands, hence in order for the value in the block to be equal to $|B|C$ as specified by the target weight, we must have that the value of $\varrho_S(u_j)$ is equal to the value of each of the $\varrho_S(u'_\ell)$ by the definition of $|B|$-average free sets. Since ϱ_S is a bijection, this ensures that the choice of hyperclique vertices in the sets appearing in the Cartesian product defining $U^1_{i,j}$ – i.e. $U_{(ir+j-1)\frac{betak'}{hr}+1}, \ldots, U_{(ir+j)\frac{\beta k'}{hr}}$ – agree with the choice of hyperclique edges. This is true for all i,j and hence for each U_ℓ for $\ell \in [k']$.

The other direction is easy to see via a similar, simpler argument. This concludes the correctness proof.

Running Time. It can be seen that the running time of this reduction is $O(N_H^{2h})$, up to the running time of the algorithm for the construction of the $|B|$-average free set, which we will ignore here for sake of simplicity. Now suppose we can solve Exact Weight

Subgraph Isomorphism in time $O(n^{t+1-\varepsilon}W)$. We use the reduction above to convert a Hyperclique instance with $n_H = O(N_H)$ nodes to an Exact Weight Subgraph Isomorphism instance where $W \approx \Theta(N_H^{\beta k'})$ and $n = O(N_H^{(1-\beta)k'/(t+1)})$. Choosing β carefully, we get $W = \Theta(N_H^{\gamma})$; note that we are ignoring some intricacies in the choice of β that arise when you consider the running time of the algorithm that constructs the $|B|$-average free set – the details are available in the full version of this paper. Now via the algorithm for Exact Weight Subgraph Isomorphism, we can solve this instance and hence the original Hyperclique problem in time $O(N_H^{2h}+N_H^{((1-\beta)k'/(t+1))(t+1-\varepsilon)}\cdot N_H^{\beta k'}) = O(N_H^{k'-\varepsilon'}) = O(n_H^{k'-\varepsilon'})$. ◀

It is easy to see that the same proof also rules out algorithms running in time $O(n^{t+1}W^{1-\varepsilon})$. Similar as with the proof for the unweighted problem, basically the same techniques can be used to prove the other parts of Theorem 4.

3 Open Problems

In this paper we discussed many different variants of the Subgraph Isomorphism problem. For some of these variants we leave gaps, which gives rise to several open problems:

1. Can the algorithms for weighted trees be improved? We have shown that some improvements can be made for node-weighted trees (see Theorem 9), but are these optimal? What about edge-weighted trees?
2. Are there fast algorithms for unweighted Subgraph Isomorphism on graphs of bounded pathwidth that do not use rectangular matrix multiplication? Can the gap between exponent $\omega(p-1)$ and exponent p be closed? Similar questions apply to the weighted case; see Theorems 6 and 8.
3. Relatedly, are there good lower bounds for unweighted Subgraph Isomorphism on graphs of bounded pathwidth? Recall that Lemma 11 does not allow us to transfer our lower bounds for the colored case to the uncolored case.

We conclude with some more general open problems:

1. Do our algorithms and lower bounds also work for other types of graph homomorphisms, and for counting the number of solutions? Techniques from [24] seem applicable.
2. In this work we demonstrated the existence of maximally hard patterns for which Subgraph Isomorphism requires time $n^{\mathrm{tw}(H)+1-o(1)}$. Can we classify which (classes of) patterns are maximally hard?
3. Changing our focus from hard patterns to easy patterns, we can ask: do classes of patterns of unbounded treewidth exist for which Subgraph Isomorphism can be solved in time $n^{o(\mathrm{tw}(H))}$? Recall that a conditional lower bound rules out $n^{o(\mathrm{tw}(H)/\log \mathrm{tw}(H))}$ [40].

References

1 Amir Abboud and Greg Bodwin. The 4/3 additive spanner exponent is tight. *Journal of the ACM (JACM)*, 64(4):1–20, 2017.

2 Amir Abboud, Greg Bodwin, and Seth Pettie. A hierarchy of lower bounds for sublinear additive spanners. *SIAM Journal on Computing*, 47(6):2203–2236, 2018.

3 Amir Abboud, Karl Bringmann, Danny Hermelin, and Dvir Shabtay. SETH-based lower bounds for Subset Sum and Bicriteria Path. In *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 41–57. SIAM, 2019.

4 Amir Abboud, Karl Bringmann, Danny Hermelin, and Dvir Shabtay. Scheduling lower bounds via AND Subset Sum. *arXiv preprint*, 2020. `arXiv:2003.07113`.

5 Amir Abboud and Kevin Lewi. Exact weight subgraphs and the k-SUM conjecture. In *International Colloquium on Automata, Languages, and Programming*, pages 1–12. Springer, 2013.

6 Amir Abboud, Kevin Lewi, and Ryan Williams. Losing weight by gaining edges. In *European Symposium on Algorithms*, pages 1–12. Springer, 2014.

7 Amir Abboud, Virginia Vassilevska Williams, and Huacheng Yu. Matching triangles and basing hardness on an extremely popular conjecture. *SIAM Journal on Computing*, 47(3):1098–1122, 2018.

8 Noga Alon. Testing subgraphs in large graphs. *Random Structures & Algorithms*, 21(3-4):359–370, 2002.

9 Noga Alon, Raphael Yuster, and Uri Zwick. Color-coding. *Journal of the ACM (JACM)*, 42(4):844–856, 1995.

10 Omid Amini, Fedor V Fomin, and Saket Saurabh. Counting subgraphs via homomorphisms. In *International Colloquium on Automata, Languages, and Programming*, pages 71–82. Springer, 2009.

11 Zhao An, Qilong Feng, Iyad Kanj, and Ge Xia. The complexity of tree partitioning. In *Workshop on Algorithms and Data Structures*, pages 37–48. Springer, 2017.

12 MohammadHossein Bateni, MohammadTaghi Hajiaghayi, Saeed Seddighin, and Cliff Stein. Fast algorithms for knapsack via convolution and prediction. In *Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing*, pages 1269–1282, 2018.

13 Felix A Behrend. On sets of integers which contain no three terms in arithmetical progression. *Proceedings of the National Academy of Sciences of the United States of America*, 32(12):331, 1946.

14 Hans L Bodlaender. A tourist guide through treewidth. *Acta cybernetica*, 11(1-2):1, 1994.

15 Hans L Bodlaender. A partial k-arboretum of graphs with bounded treewidth. *Theoretical computer science*, 209(1-2):1–45, 1998.

16 Hans L Bodlaender. Discovering treewidth. In *International Conference on Current Trends in Theory and Practice of Computer Science*, pages 1–16. Springer, 2005.

17 David Bremner, Timothy M Chan, Erik D Demaine, Jeff Erickson, Ferran Hurtado, John Iacono, Stefan Langerman, and Perouz Taslakian. Necklaces, convolutions, and x+ y. In *European Symposium on Algorithms*, pages 160–171. Springer, 2006.

18 Karl Bringmann. A near-linear pseudopolynomial time algorithm for Subset Sum. In *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1073–1084. SIAM, 2017.

19 Danilo Bruschi, Lorenzo Martignoni, and Mattia Monga. Detecting self-mutating malware using control-flow graph matching. In *International conference on detection of intrusions and malware, and vulnerability assessment*, pages 129–143. Springer, 2006.

20 Timothy M Chan and Moshe Lewenstein. Clustered integer 3SUM via additive combinatorics. In *Proceedings of the forty-seventh annual ACM symposium on Theory of computing*, pages 31–40, 2015.

21 Don Coppersmith. Rapid multiplication of rectangular matrices. *SIAM Journal on Computing*, 11(3):467–471, 1982.

22 Don Coppersmith and Shmuel Winograd. Matrix multiplication via arithmetic progressions. In *Proceedings of the nineteenth annual ACM symposium on Theory of computing*, pages 1–6, 1987.

23 Thomas H Cormen, Charles E Leiserson, Ronald L Rivest, and Clifford Stein. *Introduction to algorithms*. MIT press, 2009.

24 Radu Curticapean, Holger Dell, and Dániel Marx. Homomorphisms are a good basis for counting small subgraphs. In *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing*, pages 210–223, 2017.

25 Mina Dalirrooyfard, Thuy Duong Vuong, and Virginia Vassilevska Williams. Graph pattern detection: Hardness for all induced patterns and faster non-induced cycles. In *Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing*, pages 1167–1178, 2019.

26 Bartłomiej Dudek, Paweł Gawrychowski, and Tatiana Starikovskaya. All non-trivial variants of 3-LDT are equivalent. In *Proceedings of the 52nd Annual ACM SIGACT Symposium on Theory of Computing*, pages 974–981, 2020.

27 Friedrich Eisenbrand and Fabrizio Grandoni. On the complexity of fixed parameter clique and dominating set. *Theoretical Computer Science*, 326(1-3):57–67, 2004.

28 Fedor V Fomin, Petr A Golovach, Daniel Lokshtanov, and Saket Saurabh. Almost optimal lower bounds for problems parameterized by clique-width. *SIAM Journal on Computing*, 43(5):1541–1563, 2014.

29 Fedor V Fomin, Daniel Lokshtanov, Venkatesh Raman, Saket Saurabh, and BV Raghavendra Rao. Faster algorithms for finding and counting subgraphs. *Journal of Computer and System Sciences*, 78(3):698–706, 2012.

30 François Le Gall and Florent Urrutia. Improved rectangular matrix multiplication using powers of the Coppersmith-Winograd tensor. In *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1029–1046. SIAM, 2018.

31 Edinah K Gnang, Ahmed Elgammal, and Vladimir Retakh. A spectral theory for tensors. In *Annales de la Faculté des sciences de Toulouse: Mathématiques*, volume 20, pages 801–841, 2011.

32 Russell Impagliazzo and Ramamohan Paturi. On the complexity of k-SAT. *Journal of Computer and System Sciences*, 62(2):367–375, 2001.

33 Klaus Jansen, Stefan Kratsch, Dániel Marx, and Ildikó Schlotter. Bin packing with fixed number of bins revisited. *Journal of Computer and System Sciences*, 79(1):39–49, 2013.

34 Konstantinos Koiliaris and Chao Xu. A faster pseudopolynomial time algorithm for Subset Sum. In *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1062–1072. SIAM, 2017.

35 Marvin Künnemann, Ramamohan Paturi, and Stefan Schneider. On the fine-grained complexity of one-dimensional dynamic programming. In *44th International Colloquium on Automata, Languages, and Programming (ICALP 2017)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2017.

36 François Le Gall. Powers of tensors and fast matrix multiplication. In *Proceedings of the 39th international symposium on symbolic and algebraic computation*, pages 296–303, 2014.

37 Yuan Li, Alexander Razborov, and Benjamin Rossman. On the AC^0 complexity of Subgraph Isomorphism. *SIAM Journal on Computing*, 46(3):936–971, 2017.

38 Andrea Lincoln, Virginia Vassilevska Williams, and Ryan Williams. Tight hardness for shortest cycles and paths in sparse graphs. In *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1236–1252. SIAM, 2018.

39 Josef Malík, Ondřej Suchỳ, and Tomáš Valla. Efficient implementation of Color Coding algorithm for Subgraph Isomorphism problem. In *International Symposium on Experimental Algorithms*, pages 283–299. Springer, 2019.

40 Dániel Marx. Can you beat treewidth? In *48th Annual IEEE Symposium on Foundations of Computer Science (FOCS'07)*, pages 169–179. IEEE, 2007.

41 Dániel Marx and Michal Pilipczuk. Everything you always wanted to know about the parameterized complexity of Subgraph Isomorphism (but were afraid to ask). In Ernst W. Mayr and Natacha Portier, editors, *31st International Symposium on Theoretical Aspects of Computer Science (STACS 2014)*, volume 25 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 542–553, Dagstuhl, Germany, 2014. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. doi:10.4230/LIPIcs.STACS.2014.542.

42 Ron Milo, Shai Shen-Orr, Shalev Itzkovitz, Nadav Kashtan, Dmitri Chklovskii, and Uri Alon. Network motifs: simple building blocks of complex networks. *Science*, 298(5594):824–827, 2002.

43 Jaroslav Nešetřil and Svatopluk Poljak. On the complexity of the subgraph problem. *Commentationes Mathematicae Universitatis Carolinae*, 26(2):415–419, 1985.

44 Kevin Pratt. Waring rank, parameterized and exact algorithms. In *2019 IEEE 60th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 806–823. IEEE, 2019.

45 Gregory Rosenthal. Beating treewidth for average-case subgraph isomorphism. In *14th International Symposium on Parameterized and Exact Computation (IPEC 2019)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2019.

46 Paul D Seymour and Robin Thomas. Graph searching and a min-max theorem for tree-width. *Journal of Combinatorial Theory, Series B*, 58(1):22–33, 1993.

47 Edward H Sussenguth. A graph-theoretic algorithm for matching chemical structures. *Journal of Chemical Documentation*, 5(1):36–43, 1965.

48 Ryan Williams. A new algorithm for optimal 2-constraint satisfaction and its implications. *Theoretical Computer Science*, 348(2-3):357–365, 2005.

Fast n-Fold Boolean Convolution via Additive Combinatorics

Karl Bringmann
Saarland University, Saarland Informatics Campus, Saarbrücken, Germany
Max Planck Institute for Informatics, Saarland Informatics Campus, Saarbrücken, Germany

Vasileios Nakos
Saarland University, Saarland Informatics Campus, Saarbrücken, Germany
Max Planck Institute for Informatics, Saarland Informatics Campus, Saarbrücken, Germany

Abstract

We consider the problem of computing the Boolean convolution (with wraparound) of n vectors of dimension m, or, equivalently, the problem of computing the sumset $A_1 + A_2 + \ldots + A_n$ for $A_1, \ldots, A_n \subseteq \mathbb{Z}_m$. Boolean convolution formalizes the frequent task of combining two subproblems, where the whole problem has a solution of size k if for some i the first subproblem has a solution of size i and the second subproblem has a solution of size $k - i$. Our problem formalizes a natural generalization, namely combining solutions of n subproblems subject to a modular constraint. This simultaneously generalises Modular Subset Sum and Boolean Convolution (Sumset Computation). Although nearly optimal algorithms are known for special cases of this problem, not even tiny improvements are known for the general case.

We almost resolve the computational complexity of this problem, shaving essentially a factor of n from the running time of previous algorithms. Specifically, we present a *deterministic* algorithm running in *almost* linear time with respect to the input plus output size k. We also present a *Las Vegas* algorithm running in *nearly* linear expected time with respect to the input plus output size k. Previously, no deterministic or randomized $o(nk)$ algorithm was known.

At the heart of our approach lies a careful usage of Kneser's theorem from Additive Combinatorics, and a new deterministic almost linear output-sensitive algorithm for non-negative sparse convolution. In total, our work builds a solid toolbox that could be of independent interest.

2012 ACM Subject Classification Theory of computation → Design and analysis of algorithms

Keywords and phrases convolution, sumset computation, modular subset sum, output sensitive

Digital Object Identifier 10.4230/LIPIcs.ICALP.2021.41

Category Track A: Algorithms, Complexity and Games

Related Version *Full Version*: `https://arxiv.org/abs/2105.03968`

Funding This work is part of the project TIPEA that has received funding from the European Research Council (ERC) under the European Unions Horizon 2020 research and innovation programme (grant agreement No. 850979).

1 Introduction

In this paper we study n-fold variants of the following fundamental 2-fold problems.

1.1 2-Fold Case

Boolean Convolution and Sumset Computation. In Boolean convolution we are given vectors $A, B \in \{0,1\}^m$ and the task is to compute the vector $C = A \circledast B \in \{0,1\}^m$ defined by $C[k] = \bigvee_i A[i] \wedge B[k-i]$. This formalizes a situation in which we split a computational problem into two subproblems, so that in total there is a solution of size k if and only if for some i there is a solution of the left subproblem of size i and there is a solution of the right subproblem of size $k - i$. This is a natural task that frequently arises in algorithm design. There are two variants of this problem: *Without wraparound* the $\bigvee_i$-quantifier goes

over $0 \leq i \leq k$; *with wraparound* the quantifier goes over all $i \in [m]$ and the entry $B[k-i]$ means $B[(k-i) \bmod m]$. Algorithmically the two variants are equivalent, and throughout this paper we study the latter variant.

An equivalent problem is *sumset computation*: Given sets $A, B \subseteq \mathbb{Z}_m$, compute their sumset $A + B$, which denotes the set of all sums $a + b$ modulo m with $a \in A, b \in B$. This corresponds to Boolean convolution with wraparound[1].

Standard Convolution and Polynomial Multiplication. In (standard) convolution we are given vectors $A, B \in \mathbb{R}^m$ and the task is to compute the vector $C = A \star B \in \mathbb{R}^m$ with $C[k] = \sum_i A[i] \cdot B[k-i]$. For instance, if $A[i]$ and $B[i]$ count the number of size-i solutions of the left and right subproblem, then $C[k]$ counts the number of size-k solutions of the whole problem. Again one can consider variants with or without wraparound. A typical restriction are *non-negative* entries, which is well-motivated in case that A, B, C represent numbers of solutions.

This problem is equivalent to *polynomial multiplication*: Given the coefficients of polynomials $P(X) = \sum_{i=0}^{m} A[i] \cdot X^i$ and $Q(X) = \sum_{i=0}^{m} B[i] \cdot X^i$, compute the coefficients of their product $P \cdot Q$.

State of the Art. Using Fast Fourier Transform (FFT), all of the above problems can be solved in time $O(m \log m)$. A long line of work has considered these problems in a sparse setting, called sparse convolution or sparse polynomial multiplication, see, e.g., [23, 16, 25, 21, 28, 5, 15, 26, 24, 18, 11]. Here the task is to compute the convolution of two sparse vectors much faster than performing FFT, ideally in near-linear time in terms of the input plus output size (i.e., the number of non-zero entries of the input and output vectors). Near-linear in the input plus output size running time was achieved for vectors with non-negative entries by Cole and Hariharan [16] and for general vectors in [24], see also [18] for additional $\log m$ factors improvements. Very recently, a Monte Carlo $O(k \log k)$-time algorithm has been achieved in [11] for non-negative convolution, where k is the input plus output size. Sparse convolution techniques are crucially used in [3, 4, 2, 15, 8, 12], and are also relevant to the study of sparse wildcard matching, a fundamental string problem [14, 16].

However, all known algorithms for these sparse problems are randomized, and thus an open problem is to *close the gap between deterministic and randomized algorithms*. This was explicitly posed as an open problem in [15, Remark 8.2].

Our Contribution to the 2-Fold Case. We present a deterministic algorithm for convolution of non-negative vectors (and thus also for Boolean convolution) running in time $k \cdot m^{o(1)}$, where k is the input plus output size. This matches up to the $m^{o(1)}$ term the best known algorithms in the randomize case [16]. Our algorithm heavily builds upon an algorithm by Chan and Lewenstein [15], which operates under the additional assumption that a small superset of the non-negative terms is known in advance. We remove their assumption by gradually building the sumset using calls to their algorithm.

[1] By removing the modulo operation and thus working over $\mathbb{Z}$ we can also pose a problem variant corresponding to Boolean convolution without wraparound. Again, algorithmically these variants are equivalent, since for any $A, B \subseteq \{0, 1, \ldots, m-1\}$, on the one hand computing $A + B$ over $\mathbb{Z}$ and taking the result modulo m yields $A + B$ over $\mathbb{Z}_m$, and on the other hand computing $A + B$ over $\mathbb{Z}_{2m}$ yields $A + B$ over $\mathbb{Z}$.

▶ **Theorem 1** (Deterministic Non-Negative Sparse Convolution, Section 6). *Denote by $\|x\|_0$ the number of non-zero entries of a vector x. Given vectors $A, B \in \mathbb{R}^m_{\geq 0}$, we can compute their convolution $A \star B$ (with wraparound) in time $\|A \star B\|_0 \cdot m^{o(1)}$ by a deterministic algorithm. More precisely, the running time is $\|A \star B\|_0 \cdot 2^{O(\sqrt{\log \|A \star B\|_0 \log\log m})}$.*

Observe that $\|A\|_0, \|B\|_0 \leq \|A \star B\|_0$, and thus rather than bounding the running time in terms of the input plus output size $\|A\|_0 + \|B\|_0 + \|A \star B\|_0$, it suffices to bound the running time in terms of only the output size $\|A \star B\|_0$. Moreover, note that since $\|A \star B\|_0 \leq m$ the above running time is bounded by $m^{1+o(1)}$. As an additional bonus, our approach gives a quite simple $\|A \star B\|_0 \cdot \operatorname{polylog}(m)$-time Las Vegas algorithm for the 2-fold case of non-negative sparse convolution, see Theorem 17.

We leave it as an open problem whether similarly efficient deterministic algorithms exist under the presence of negative entries.

1.2 n-Fold Case

The focus of this paper is on n-fold generalizations of the above problems. Indeed, in typical applications we do not only split a problem into two subproblems, but these subproblems are recursively split into further subproblems. If the recursion tree has n leaves, we therefore want to compute Boolean convolutions of the form $A_1 \circledast \ldots \circledast A_n$ for vectors $A_1, \ldots, A_n$.

Note that now the "gold standard" would be linear running time in terms of the total input plus output size $k = \|A_1\|_0 + \ldots + \|A_n\|_0 + \|A_1 \circledast \ldots \circledast A_n\|_0$. Note that in contrast to the 2-fold case, the size of the output is incomparable to the size of the input.

A Special Case: Modular Subset Sum. As an example, consider the Modular Subset Sum problem, where we are given $x_1, \ldots, x_n \in \mathbb{Z}_m$ and a target t, and the task is to decide whether for some subset $I \subseteq [n]$ we have $\sum_{i \in I} x_i \equiv t \pmod m$. Observe that the sumset $\{0, x_1\} + \ldots + \{0, x_n\} \subseteq \mathbb{Z}_m$ denotes the set of all attainable subset sums modulo m, and thus Modular Subset Sum can be solved by a direct application of n-fold sumset computation, which is equivalent to n-fold Boolean convolution (with wraparound).

The state of the art for Modular Subset Sum is as follows. A standard dynamic programming approach solves the problem in time $O(n \cdot m)$. After the first improvements by Koiliaris and Xu [20], Axiotis et al. [8] designed an algorithm running in time $O((n+m)\operatorname{polylog}(n+m))$, which was further simplified, sped up and made deterministic in [7, 13]. Those running times match a conditional lower bound based on the Strong Exponential Time Hypothesis [1, 8]. Moreover, all of the above algorithms can be analyzed to run in time $O(k \operatorname{polylog} m)$, where k is the total input plus output size [8].

In other words, for the special case $|A_1| = \ldots = |A_n| = 2$ of n-fold sumset computation near-optimal algorithms are known. Furthemore, the techniques crucially exploit the fact that all sets A_i have constant cardinality. The goal of this paper is to investigate the general case without any restrictions on $|A_i|$. Can one move beyond the problem-specific techniques in [20, 8, 7, 13] which seem to apply solely to Modular Subset Sum?

Naive Approach. As it is already known how to compute $A \star B$ (and thus $A \circledast B$) in time near-linear in the output size [16, 24, 11], is there an easy generalization to compute the n-fold Boolean convolution $A_1 \circledast \ldots \circledast A_n$? Naively, if we compute the n-fold convolution in a linear fashion as $(((A_1 \circledast A_2) \circledast A_3) \star \ldots \circledast A_{n-1}) \circledast A_n$, then each intermediate convolution has input plus output size at most k, so using [16] we can bound the total expected running time by $O(nk \operatorname{polylog} m)$. Unfortunately, this running time analysis is tight. The issue is that up to $\widetilde{\Omega}(n)$ intermediate results may have size $\Omega(k)$.

The same is true if we compute the n-fold convolution in a bottom-up tree-like fashion as $((A_1 \circledast A_2) \circledast (A_3 \circledast A_4)) \circledast \ldots$, as shown by the following example. Pick $b = \lfloor \frac{\log m}{\log \log m} \rfloor$ and $\ell = \lceil \log_b(m) \rceil = \Theta(b)$, and set A_i to the indicator vector of $b^{i \bmod \ell} \cdot \{0, 1, 2, \ldots, b-1\}$. Then the Boolean convolution of any ℓ consecutive A_i's is the all-ones vector and thus has size m, and this holds for $\Omega(n/\ell) = \Omega\big(n \frac{\log \log m}{\log m}\big)$ intermediate convolutions. On the other hand, the input size is $O(n \frac{\log m}{\log \log m})$ and the output size is $O(m)$.

Analyzing the time only in terms of n, m, the naive approach yields time $O(nm \operatorname{polylog} m)$. A simple algorithm using $n-1$ FFTs also yields time $O(nm \log m)$. Before this work it was open whether n-fold Boolean convolution can be solved in time close to linear in $n+m$, and close to linear in k, or whether the additional factor $\widetilde{\Theta}(n)$ of the naive approach is necessary.

n-Fold Boolean Convolution versus n-Fold Convolution. We note that n-fold Boolean convolution is quite different from n-fold convolution, and we focus on the former in this paper. The reason is that n-fold convolution results in exponentially large entries. Indeed, assuming that $A_1, \ldots, A_n$ are non-negative integer vectors, each with at least two non-zero entries, one can check that $\|A_1 \star \ldots \star A_n\|_1 = \|A_1\|_1 \cdot \ldots \cdot \|A_n\|_1 \geq 2^n$ (here $\|x\|_1 = \sum_i |x[i]|$), and thus at least one output entry requires $\Omega(n)$ bits to represent exactly. Possible ways to handle this situation are (1) to let k be the total number of input plus output *bits*, (2) assume that entries come from a finite field, or (3) relax to approximation. We leave these as open problems and focus on Boolean convolution in this paper.

Our Contribution to n-Fold Boolean Convolution. We show that the multiplicative factor n in the naive running times $O(nk \operatorname{polylog} m)$ and $O(s + nm \log m)$ is not necessary (here s is the size of the input). Specifically, our approach yields two new results for n-fold Boolean convolution: a randomized Las Vegas algorithm running in expected time $O(k \cdot \operatorname{polylog} m)$, and a deterministic algorithm running in time $k \cdot m^{o(1)}$. Morally, we show that one can convolve n Boolean vectors in a much better way than doing $n-1$ FFTs. In particular, in terms of m, n and the size of the input s, the known algorithms would run in time $\widetilde{O}(s + mn)$, whereas our approach yields time $\widetilde{O}(s+m)$. Thus, in instances where the size of the input does not dominate (as in Modular Subset Sum where $s = 2n$) our approach yields a substantial improvement.

Our algorithm falls in a line of research that tries to apply results from Additive Combinatorics in algorithm design, such as [17, 15, 6, 9, 22, 12]. Quite interestingly, this is the first time that such a connection has produced an (almost) optimal result. Previous algorithms [17, 15, 6, 9, 22, 12] had less clean running time bounds and are thus likely to be suboptimal, partly because of the Additive Combinatorics machinery used.

We now state our results more formally. The main result of this paper is the following.

▶ **Theorem 2** (n-Fold Boolean Convolution)**.** *Given vectors $A_1, A_2, \ldots, A_n \in \{0,1\}^m$ we can compute their Boolean convolution with wrap-around $A_1 \circledast A_2 \circledast \ldots \circledast A_n$*
(1) *by a randomized Las Vegas algorithm in $O(k \cdot \operatorname{polylog}(mk))$ expected time, or*
(2) *by a deterministic algorithm in $k \cdot 2^{O(\sqrt{\log k \cdot \log\log m})}$ time*
Here, $k := \|A_1\|_0 + \ldots + \|A_n\|_0 + \|A_1 \circledast A_2 \circledast \ldots \circledast A_n\|_0$ is the total input plus output size.

▶ Remark 3. It might seem confusing that for very small k, specifically for $k \leq \log^{O(1)} m$, our deterministic time is faster than our randomized time. However, as we will discuss later, it is easy to solve the problem deterministically in time $k^{O(1)}$. In fact our time bounds are $\min\{k^3, k \cdot \operatorname{polylog}(mk))\}$ expected time, and $\min\left\{k^3, k \cdot 2^{O(\sqrt{\log k \cdot \log\log m})} \cdot \operatorname{polylog}(mk)\right\}$ deterministically; the latter can be simplified to the expression in Theorem 2.

In order to employ Additive Combinatorics machinery, it will be convenient to phrase the problem in terms of sets and sumsets, making the connection more clear. To this end, we replace every vector $A_i \in \{0,1\}^m$ by a set $A'_i \subseteq \mathbb{Z}_m$ such that $x \in A'_i$ if and only the x-th entry of A_i is 1. Then, it can easily be seen that the Boolean convolution $A_1 \circledast A_2 \circledast \ldots \circledast A_n$ is equivalent to computing the sumset $A'_1 + A'_2 + \ldots + A'_n$. Written in a more Additive-Combinatorics-friendly way, our main result can be rephrased in the following way.

▶ **Theorem 4** (Theorem 2 restated, Section 5). *Given sets $A_1, \ldots A_n \subseteq \mathbb{Z}_m$, we can compute their sumset $A_1 + A_2 + \ldots + A_n$*
(1) *by a randomized Las Vegas algorithm in $O(k \cdot \text{polylog}(mk))$ expected time, or*
(2) *by a deterministic algorithm in $k \cdot 2^{O(\sqrt{\log k \cdot \log\log m})}$ time.*
Here, $k := |A_1| + \ldots + |A_n| + |A_1 + \ldots + A_n|$ is the total input plus output size.

We remark that further improvements over Theorem 1 would directly improve Theorems 2 and 4. In particular, our factor $m^{o(1)} = 2^{O(\sqrt{\log m \log\log m})}$ stems entirely from the application of Theorem 1, and thus indirectly from a tool called the FFT Lemma [15] that we use to prove Theorem 1.

We also remark that Theorem 4 is formulated for sumsets over $\mathbb{Z}_m$, but by setting m sufficiently large (like $1 + \sum_i \max(A_i)$) we can also compute sumsets $A_1 + \ldots + A_n \subseteq \mathbb{Z}$ over the integers in time close to the input plus output size. However, this is a much simpler result that can also be achieved by elementary means, without any Additive Combinatorics.

2 Preliminaries and Technical Toolkit

For any positive integer m, we let $\mathbb{Z}_m$ be the group of residues modulo m. For two sets $A, B \subseteq \mathbb{Z}_m$, we define $A + B := \{x \mid \exists a \in A, b \in B \colon a + b = x\}$. Unless explicitly stated otherwise, all sumsets throughout the paper are computed in the underlying group $\mathbb{Z}_m$, i.e., $A + B \subseteq \mathbb{Z}_m$. We also write $A \bmod q := \{a \bmod q \mid a \in A\}$.

Throughout the paper we use the notation of sumset computation instead of the equivalent Boolean convolution.

2.1 Randomized Sumset Computation

Cole and Hariharan's sparse convolution algorithm [16] implies that the sumset $A + B$ can be computed in Las Vegas time $O(|A + B| \cdot \log^2 m + \text{poly}(\log m))$. Very recently, this was improved to $O(|A + B| \cdot \log |A + B| + \text{poly}(\log m))$ [11] with a Monte Carlo algorithm.

▶ **Theorem 5** (Randomized Sumset Computation, [16], see also Section 6). *Given sets $A, B \subseteq \mathbb{Z}_m$, their sumset $A + B$ can be computed in expected time $O(|A + B| \,\text{poly}(\log m))$.*

2.2 The Symmetry Group and its Properties

▶ **Definition 6** (Symmetry group of a set). *Let $A \subseteq \mathbb{Z}_m$. We define the symmetry group of A as $Sym(A) = \{h \in \mathbb{Z}_m \mid A + \{h\} = A\}$.*

It is easy to check that $Sym(A)$ satisfies the group properties with respect to addition, and thus $Sym(A)$ is a subgroup of $\mathbb{Z}_m$. In particular, we have $Sym(A) = d \cdot \mathbb{Z}_{m/d}$, where d is the minimum non-zero element of $Sym(A)$ (to see this, note that the minimum non-zero element of a cyclic subgroup is also a generator of it).

One can check that $Sym(A) \subseteq Sym(A+B)$ holds for any sets $A, B \subseteq \mathbb{Z}_m$. This property will be of great importance to us. Moreover, for any non-empty set A and any $x \in A$ we have $Sym(A) \subseteq A + \{-x\}$. This holds since any $h \in Sym(A)$ maps x to some $x' \in A$, which means $x' = x + h \pmod m$, hence $h = x' - x \pmod m$. In particular, the symmetry group of a non-empty set A has size at most $|A|$.

We show that the symmetry group can be computed in linear time.

▶ **Theorem 7** (Computing the Symmetry Group, Section 7)**.** *Given a sorted non-empty set $A \subseteq \mathbb{Z}_m$, we can compute $Sym(A)$ in time $O(|A|)$.*

2.3 Kneser's Theorem

The following theorem lies at the core of our algorithms.

▶ **Theorem 8** (Kneser's Theorem, see, e.g., Theorem 5.5 in [27])**.** *Let $A, B \subseteq \mathbb{Z}_m$ be non-empty. Then*

$$|A + B| \geq \min\{|A| + |B| - |Sym(A + B)|, m\}.$$

We will use the following simple corollary.

▶ **Corollary 9.** *Let $A, B \subseteq \mathbb{Z}_m$ be non-empty. If $|A+B| < |A|+|B|-1$ then $|Sym(A+B)| > 1$.*

Proof. For $m = 1$ it cannot happen that $|A + B| < |A| + |B| - 1$, so assume $m \geq 2$.

If $|A + B| = m$, then $A + B = \mathbb{Z}_m$. This implies $Sym(A + B) = \mathbb{Z}_m$ and thus $|Sym(A + B)| = m > 1$. Otherwise, if $|A + B| < m$, we can simplify the bound obtained from Kneser's Theorem to

$$|A + B| \geq |A| + |B| - |Sym(A + B)|.$$

Together with $|A + B| < |A| + |B| - 1$, this implies $|Sym(A + B)| > 1$. ◀

3 Overview and Comparison with Previous Approaches

We start by giving a rough overview of our algorithm, leaving out several details. Our improvements are obtained by delving deeper into the additive structure of sumset computation over $\mathbb{Z}_m$ than previous work. Our algorithms compute the sumset $A_1 + \ldots + A_n$ in a bottom-up tree-like fashion as $((A_1 + A_2) + (A_3 + A_4)) + \ldots$. For any two sets X, Y for which we compute $X + Y$ during the execution of this algorithm, we check whether $|X+Y| < |X|+|Y|-1$. If this is the case, Kneser's Theorem (specifically Corollary 9) implies that $X + Y$ has a non-trivial symmetry group, and hence $A_1 + \ldots + A_n$ has a non-trivial symmetry group. A non-trivial symmetry group of a set $Z = X + Y \subseteq \mathbb{Z}_m$ implies that the set is *periodic*: there exists a divisor d of m and a set $Z' \subseteq \{0, \ldots, d-1\}$ such that $Z = Z' + d \cdot \mathbb{Z}_{m/d} = Z' + \{0, d, 2d, \ldots, m-d\}$, i.e., Z is a rotation (by multiples of d) of a subset of $\{0, \ldots, d-1\}$. This allows us to reduce to the smaller universe $\mathbb{Z}_d$, which is progress (it might seem from this discussion that we require a factorization of m, but this is not the case: if we reduce to a smaller universe $\mathbb{Z}_d$, then d is a divisor of m that can be easily read off the sumset $Z = X + Y$, by computing the symmetry group $Sym(X + Y)$ and taking its smallest non-zero element). It remains to argue about the situation in which every computed sumset satisfies $|X + Y| \geq |X| + |Y| - 1$. Using this inequality, we can control at any intermediate step of the algorithm the total size of all sumsets computed so far. When the computation arrives at the root, the running time that we spent on computing these sumsets is almost linear in the input plus output size.

Why Previous Approaches Cannot Solve the Generalized Problem. A natural question to ask is whether previous algorithms for Subset Sum or Modular Subset Sum were also able to tackle the more general problem of n-fold sumset computation. The techniques underlying the algorithms for Subset Sum in [10, 20, 19] are inherently non-modular, and hence cannot facilitate n-fold sumset computation problem over the group $\mathbb{Z}_m$. More relevant is the Modular Subset Sum problem, which is a standard variant of Subset Sum, where one works over $\mathbb{Z}_m$ rather than $\mathbb{Z}$. This problem has seen two interesting developments in the last few years.

The deterministic algorithm of Koiliaris and Xu [20] uses multiple interesting problem-specific tricks for Modular Subset Sum, but it is unclear how to generalize them to n-fold sumset computation. In fact, their algorithm can be viewed as a reduction from Modular Subset Sum to $\min\{\sqrt{n}, m^{1/4}\}$-fold sumset computation, which they then solve by the straightforward repeated Fast Fourier Transform. Hence, also for the general case of n-fold sumset computation their approach does not seem to yield time $o(nm)$.

All known algorithms for Modular Subset Sum [8, 7, 13] compute the set of attainable subset sums $\mathcal{S}(A) = \{0, a_1\} + \ldots + \{0, a_n\} \subseteq \mathbb{Z}_m$ for $A = \{a_1, \ldots, a_n\}$. The main idea is to compute $\mathcal{S}(A)$ from $\mathcal{S}(A \setminus \{a\})$ by forming the vector $1_{a+\mathcal{S}(A\setminus\{a\})} - 1_{\mathcal{S}(A\setminus\{a\})}$. It can be easily seen that this vector consists of an equal number of positive and negative entries, and the positive entries correspond to the "new" sums $\mathcal{S}(A) \setminus \mathcal{S}(A \setminus \{a\})$. Using hashing-based arguments or appropriate data structures for string manipulation, they show how to recover the support of the aforementioned vector in near-linear output-sensitive time. A possibility to generalize this approach to n-fold sumset computation $A_1 + \ldots + A_n$ would be to consider the vector $\sum_{a\in A_n}(1_{a+A_1+\ldots+A_{n-1}} - 1_{A_1+\ldots+A_{n-1}})$. However, measuring this vector would incur time $\Omega(|A_n|)$, and thus an immediate generalization of their approach would at least pay a factor $\max_i |A_i|$ on top of the output size.

Symmetry Manifestations in Higher Dimensions. It would be interesting to understand whether the symmetry considerations of our algorithm manifest themselves in other abelian groups, most notably in $\mathbb{Z}_m^D$. The characterization of subgroups over $\mathbb{Z}_m^D$ with $D > 1$ is less convenient for our purposes than the characterization in the one-dimensional case, so it seems that a different treatment and notion of progress is needed in that case. We leave this to potential future work. Even if one does worry about the n-fold case and concentrates in the simplest case of $n = 2$, i.e. 2-fold d-dimensional sparse convolution, the best algorithm we are aware of solves the problem with a multiplicative 2^d multiplicative factor on top of output size. We leave as an open question the problem of avoiding the exponential dependence of 2-fold d-dimensional sparse convolution.

4 Warmup: n-Fold Sumset Computation over Prime Universe

As a warmup, we consider universe $\mathbb{Z}_m = \mathbb{Z}_p$ for prime p. For simplicity, we analyze our algorithm only in terms of the input size and the universe size p, that is, we defer the output-sensitive analysis to the general algorithm in Section 5.

Suppose we are given sets $A_1, \ldots, A_n \subseteq \mathbb{Z}_p$. We may assume that n is a power of 2, since otherwise we can add an appropriate number of sets $A_i = \{0\}$ without affecting the sumset. Consider Algorithm 1. We compute $A_1 + \ldots + A_n$ in a tree-like bottom-up fashion, by first computing $A_1 + A_2, A_3 + A_4, \ldots$, then computing $A_1 + A_2 + A_3 + A_4, \ldots$, and so on. The intermediate sets in round r are called $X_{r,1}, \ldots, X_{r,n/2^r}$. The termination criterion is that the sets that we computed so far in the current round r have total size significantly more

Algorithm 1 Computing the n-fold sumset $A_1 + \ldots + A_n$ over $\mathbb{Z}_p$ for prime p.

```
1: procedure nFoldSumsetInPrimeUniverse(A_1, A_2, ..., A_n, p)
2:                          ▷ n is a power of 2; p is prime; non-empty sets A_1, ..., A_n ⊆ Z_p
3:     X_{0,i} ← A_i, for all i ∈ [n]
4:     for r = 1 to log n do
5:         for i = 1 to n/2^r do
6:             X_{r,i} ← X_{r-1,2i-1} + X_{r-1,2i}        ▷ sumset computation via Theorem 1
7:             if Σ_{j≤i} |X_{r,j}| > p + i - 1 then
8:                 return Z_p
9:     return X_{log n,1}
```

than p, more precisely, $\sum_{j\leq i} |X_{r,j}| > p + i - 1$. If this criterion is satisfied, then we return the complete universe $\mathbb{Z}_p$. If the termination criterion is never satisfied, then in the end we return $X_{\log n,1}$.

It remains to analyze correctness and running time of this algorithm. To analyze correctness of the termination criterion, we need the following lemma.

▶ **Lemma 10.** *Let p be a prime, and let $A_1, A_2, \ldots, A_n \subseteq \mathbb{Z}_p$ be non-empty. If $\sum_{j=1}^{n} |A_j| \geq p + n - 1$, then $A_1 + A_2 + \ldots + A_n = \mathbb{Z}_p$.*

Proof. Suppose that the symmetry group has size $|Sym(A_1 + \ldots + A_n)| > 1$. Since $\mathbb{Z}_p$ has no non-trivial subgroups, this yields $Sym(A_1 + \ldots + A_n) = \mathbb{Z}_p$. Since $|Sym(A)| \leq |A|$ holds for any set A, we obtain $A_1 + \ldots + A_n = \mathbb{Z}_p$.

It remains to consider the case $|Sym(A_1 + \ldots + A_n)| = 1$. Since $Sym(A) \subseteq Sym(A + B)$ holds for any sets A, B, it follows that $|Sym(A_1 + \ldots + A_i)| = 1$ for all $1 \leq i \leq n$. We now inductively prove that $|A_1 + \ldots + A_i| \geq \min\{\sum_{j=1}^{i} |A_j| - i + 1, p\}$, from which the corollary follows. The induction base for $i = 1$ is trivial. For $i > 1$, we use Kneser's theorem on $A := A_1 + \ldots + A_{i-1}$ and $B := A_i$ to obtain

$$|A_1 + \ldots + A_i| \geq \min\left\{|A_1 + \ldots + A_{i-1}| + |A_i| - |Sym(A_1 + \ldots + A_i)|, p\right\}.$$

Plugging in $|Sym(A_1 + \ldots + A_i)| = 1$ and the induction hypothesis on $|A_1 + \ldots + A_{i-1}|$, and simplifying $\min\{\min\{a, p\} + b, p\}$ to $\min\{a + b, p\}$, yields

$$|A_1 + \ldots + A_i| \geq \min\left\{\left(\sum_{j=1}^{i-1} |A_j| - (i-1) + 1\right) + |A_i| - 1, p\right\} = \min\left\{\sum_{j=1}^{i} |A_j| - i + 1, p\right\},$$

which finishes the inductive proof.[2] ◀

▶ **Lemma 11** (Analysis of Algorithm 1)**.** *Given non-empty sets $A_1, \ldots, A_n \subseteq \mathbb{Z}_p$, where p is prime and n is a power of 2, Algorithm 1 correctly computes $A_1 + \ldots + A_n$ and runs in deterministic time $O((p+n)^{1+o(1)} + \sum_{i=1}^{n} |A_i|)$.*

Proof. If the termination criterion $\sum_{j\leq i} |X_{r,j}| > p + i - 1$ is satisfied, then Lemma 10 implies that $X_{r,1} + \ldots + X_{r,i} = \mathbb{Z}_p$, and hence $A_1 + \ldots + A_n = \mathbb{Z}_p$, so we correctly return $\mathbb{Z}_p$. Otherwise we reach the last line of Algorithm 1, and we correctly computed $X_{\log n,1} = A_1 + \ldots + A_n$. This shows correctness.

[2] We remark that for this lemma it would be sufficient to use the Cauchy-Davenport theorem (see, e.g., [27, Theorem 5.4]) instead of Kneser's theorem. Only for the generalization to non-prime m we need the more general theorem by Kneser.

To analyze the running time, let (r^*, i^*) be the values of r and i at the end of the execution of the algorithm. In particular, if $r^* = \log n$ we have $i^* = 1$. By our use of Theorem 1, the total running time of the algorithm is

$$\sum_{r<r^*} \sum_{i=1}^{n/2^r} |X_{r,i}| \cdot p^{o(1)} + \sum_{i \le i^*} |X_{r^*,i}| \cdot p^{o(1)}.$$

We use the fact that the termination criterion was not satisfied before step (r^*, i^*) to obtain:

$$\sum_{i=1}^{n/2^r} |X_{r,i}| \le p + \frac{n}{2^r} - 1 \qquad \text{for any } r < r^*,$$
$$\sum_{i<i^*} |X_{r^*,i}| \le p + \frac{n}{2^{r^*}} - 1.$$

Moreover, we have $|X_{r^*,i^*}| \le p$. Combining these observations allows us to further bound the running time by

$$\left(\sum_{r<r^*} \left(p + \frac{n}{2^r} - 1\right) + \left(p + \frac{n}{2^{r^*}} - 1\right) + p \right) \cdot p^{o(1)} = (p \log n + n) \cdot p^{o(1)} = (p+n)^{1+o(1)}. \blacktriangleleft$$

5 Algorithm for n-Fold Sumset Computation

This section proves Theorem 4. The main idea is that whenever we detect a non-trivial symmetry group we reduce to a problem over a smaller universe $\mathbb{Z}_d$, for a divisor d of m.

Consider Algorithm 2. Suppose we are given sets $A_1, \ldots, A_n \subseteq \mathbb{Z}_p$. We may assume that n is a power of 2, since otherwise we can add an appropriate number of sets $A_i = \{0\}$ without affecting the sumset. We maintain a guess s of the outputsize $|A_1 + \ldots + A_n|$. Specifically, s loops over all powers of 2 starting from $2^0 = 1$, and the algorithm returns the correct result once we reach the first iteration with $s \ge |A_1 + \ldots + A_n|$. Thus, in iteration s we know that the output size is more than $s/2$, and our primary goal is to test whether $|A_1 + \ldots + A_n| \le s$. If this is true then we want to compute the set $A_1 + \ldots + A_n$. We compute $A_1 + \ldots + A_n$ in a tree-like bottom-up fashion, by first computing $A_1 + A_2, A_3 + A_4, \ldots$, then computing $A_1 + A_2 + A_3 + A_4, \ldots$, and so on. The intermediate sets in round r are called $X_{r,1}, \ldots, X_{r,n/2^r}$. Our two main ideas now are as follows.

First, due to the presence of non-trivial subgroups in $\mathbb{Z}_m$ when m is not a prime, an intermediate set $X_{r,i}$ can have a non-trivial symmetry group $Sym(X_{r,i})$. As a criterion for a non-trivial symmetry group we test whether $|X_{r,i}| < |X_{r-1,2i}| + |X_{r-1,2i+1}| - 1$ (cf. Corollary 9 of Kneser's Theorem). Once we have found a non-trivial symmetry group $Sym(X_{r,i})$, then also $Sym(A_1 + \ldots + A_n) \supseteq Sym(X_{r,i})$ is non-trivial, and thus the output set $A_1 + \ldots + A_n$ is periodic, with period length $d = m/|Sym(X_{r,i})|$. It therefore suffices to compute $A_1 + \ldots + A_n$ modulo d. Hence, we reduce to a problem over a smaller universe $\mathbb{Z}_d$. This case is handled in lines 8-12. Note that d may not be the smallest period length for $A_1 + A_2 + \ldots + A_n$, but since we only need to reduce the problem size, any period suffices for us.

Second, if the criterion $|X_{r,i}| < |X_{r-1,2i}| + |X_{r-1,2i+1}| - 1$ is never satisfied, then we can use it to bound the output size. Specifically, we obtain a lower bound for $|X_{\log n,1}|$ in terms of the total intermediate size $\sum_j |X_{r,j}|$. In particular, if the total intermediate size is much larger than s, then also the output size is more than s. However, we cannot move to the next guess $2s$ yet, since we do not know whether the criterion $|X_{r,i}| < |X_{r-1,2i}| + |X_{r-1,2i+1}| - 1$ will

Algorithm 2 Computing the n-fold sumset $A_1 + \ldots + A_n$ over $\mathbb{Z}_m$ for general m.

```
1:  procedure nFoldSumset(A_1, ..., A_n, m)
2:                                  ▷ n is a power of 2; non-empty A_1, ..., A_n ⊆ Z_m
3:      for s = 1, 2, 4, ..., 2^⌈log m⌉ do
4:          X_{0,i} ← A_i, for all i ∈ [n]
5:          for r = 1 to log n do
6:              for i = 1 to n/2^r do
7:                  X_{r,i} ← X_{r-1,2i-1} + X_{r-1,2i}      ▷ sumset computation via Theorem 1 or 5
8:                  if |X_{r,i}| < |X_{r-1,2i-1}| + |X_{r-1,2i}| - 1 then
9:                      Compute Sym(X_{r,i})    ▷ symmetry group computation via Theorem 7
10:                     d ← m/|Sym(X_{r,i})|                ▷ Sym(X_{r,i}) = d · Z_{m/d}
11:                     A'_i ← A_i mod d, for all i ∈ [n]
12:                     return nFoldSumset(A'_1, ..., A'_n, d) + d · {0, 1, 2, ..., m/d - 1}
13:                 if Σ_{j≤i} |X_{r,j}| ≥ s + n/2^r then
14:                     X_{r,j} ← {0}, for all i < j ≤ n/2^r
15:                     break
16:         if |X_{log n,1}| ≤ s then
17:             return X_{log n,1}
```

be satisfied in future rounds $r' > r$. Nevertheless, we argue that once we have intermediate set size $\sum_{j \le i} |X_{r,j}| \gg s$, then we can ignore the remaining sets $X_{r,j}$, $j > i$, by setting them to $\{0\}$, cf. lines 13-15. This allows us to bound the total size of all intermediate sets to be linear in the input plus output size.

We next prove correctness and then analyze the running time of Algorithm 2.

▶ **Lemma 12** (Correctness of Algorithm 2). *Given non-empty sets $A_1, \ldots, A_n \subseteq \mathbb{Z}_m$, where n is a power of 2, Algorithm 2 correctly computes $A_1 + \ldots + A_n$.*

Proof. Note that without lines 13-15, we would compute the sumset in a straightforward bottom-up tree-like fashion as $((A_1 + A_2) + (A_3 + A_4)) + \ldots$, and thus the intermediate set $X_{r,i}$ would be equal to $A_x + A_{x+1} + \ldots + A_y$ for $x = (i-1)2^r + 1$ and $y = i2^r$. In the additional lines 13-15, we set some intermediate sets $X_{r,i}$ to $\{0\}$. Thus, we may lose some summands, but any intermediate set $X_{r,i}$ still corresponds to the sumset of a subset of its summands $A_x, A_{x+1}, \ldots, A_y$. More precisely, the set $X_{r,i}$ satisfies $X_{r,i} = A_{z_1} + A_{z_2} + \ldots + A_{z_\ell}$ for some $\{z_1, \ldots, z_\ell\} \subseteq \{x, x+1, \ldots, y\}$, with the understanding that $X_{r,i} = \{0\}$ if $\ell = 0$. (This property holds initially in line 4 and it continues to hold when we set $X_{r,i}$ in lines 7 and 14.) In particular, we always have

$$Sym(X_{r,i}) = Sym(A_{z_1} + A_{z_2} + \ldots + A_{z_\ell}) \subseteq Sym(A_1 + \ldots + A_n). \tag{1}$$

Moreover, we also infer

$$|X_{r,i}| = |A_{z_1} + A_{z_2} + \ldots + A_{z_\ell}| \le |A_1 + \ldots + A_n|. \tag{2}$$

We shall perform induction on the universe size m. For the base case $m = 1$, the result is obvious. For larger m, we consider the following two cases.

Case 1: *At some point in the execution, the criterion $|X_{r,i}| < |X_{r-1,2i-1}| + |X_{r-1,2i}| - 1$ in line 8 is satisfied.* Then by Corollary 9, $Sym(X_{r,i})$ is non-trivial and hence $Sym(A_1 + \ldots + A_n) \supseteq Sym(X_{r,i})$ is also non-trivial. We make use of the fact that all subgroups of

$\mathbb{Z}_m$ are of the form $d \cdot \mathbb{Z}_{m/d}$, where d divides m. In particular, $Sym(X_{r,i}) = d \cdot \mathbb{Z}_{m/d}$ for $d := m/|Sym(X_{r,i})|$. This means that $A_1 + \ldots + A_n$ is cyclic with period length d. It follows that for $A'_i := A_i \bmod d$ we have (using the induction hypothesis on d)

$$A_1 + \ldots + A_n = \textsc{nFoldSumset}(A'_1, \ldots, A'_n, d) + d \cdot \{0, 1, \ldots, \tfrac{m}{d} - 1\}.$$

This shows correctness of lines 8-12.

Case 2: *Lines 8-12 are never executed.* That is, for each computed set $X_{r,i}$ in line 7 we have

$$|X_{r,i}| \geq |X_{r-1,2i-1}| + |X_{r-1,2i}| - 1. \tag{3}$$

We use inequality (3) to analyze line 13. Fix any $s \in \{1, 2, 4, \ldots, 2^{\lceil \log m \rceil}\}$, and consider iteration s.

▷ Claim 13. In iteration s, we have $|X_{\log n,1}| > s$ if and only if $|A_1 + \ldots + A_n| > s$. Moreover, if $|X_{\log n,1}| \leq s$ then $X_{\log n,1} = A_1 + \ldots + A_n$.

Comparing this claim with lines 16-17, we see that if our guess s for the output size is too small, i.e., $|A_1 + \ldots + A_n| > s$, then the algorithm proceeds with the next larger guess. Otherwise, the algorithm correctly computes $X_{\log n,1} = A_1 + \ldots + A_n$ and returns this set. It remains to prove the claim.

Proof. The "only if" part follows from the bound $|X_{\log n,1}| \leq |A_1 + \ldots + A_n|$ by (2).

For the "if" part, we consider two cases:

Case A: If the criterion $\sum_{j \leq i} |X_{r,j}| \geq s + n/2^r$ in line 13 is never satisfied, then the algorithm computes $X_{\log n,1} = A_1 + \ldots + A_n$ in a straightforward manner, and thus $|X_{\log n,1}| = |A_1 + \ldots + A_n|$.

Case B: If the criterion $\sum_{j \leq i} |X_{r,j}| \geq s + n/2^r$ in line 13 is satisfied in some iteration r, then the following bound shows that it will also be satisfied in iteration $r + 1$ for some value of i:

$$\sum_{j=1}^{n/2^{r+1}} |X_{r+1,j}| \overset{(3)}{\geq} \sum_{j=1}^{n/2^r} |X_{r,j}| - \frac{n}{2^{r+1}} \geq \left(s + \frac{n}{2^r}\right) - \frac{n}{2^{r+1}} = s + \frac{n}{2^{r+1}}.$$

This nearly proves that the criterion is satisfied in iteration $r + 1$, but it ignores that some of the sets $X_{r+1,j}$ could be set to $\{0\}$ by lines 13-15. However, when this happens then by the criterion in line 13 we nevertheless have $\sum_{j=1}^{n/2^{r+1}} |X_{r+1,j}| \geq s + n/2^{r+1}$.

Therefore, if the criterion in line 13 is satisfied in some iteration r, then it is also satisfied for $r = \log n$, which yields $|X_{\log n,1}| \geq s + 1 > s$.

In either case, we obtain $|X_{\log n,1}| > s$ if $|A_1 + \ldots + A_n| > s$. This proves the equivalence.

For the second claim, note that $|X_{\log n,1}| \leq s$ only happens in Case A, and in this case we showed that $X_{\log n,1} = A_1 + \ldots + A_n$. ◁

In summary, if at any point during the course of the algorithm the criterion $|X_{r,i}| < |X_{r-1,2i-1}| + |X_{r-1,2i}| - 1$ in line 8 is satisfied (Case 1), then we have found a non-trivial symmetry group, and we can move to a problem over a smaller universe $\mathbb{Z}_d$, where $d < m$

is a divisor of m. Correctness then follows by induction on m. If this never happens (Case 2), then the algorithm behaves as follows. We have an increasing guess s for the output size $|A_1 + \ldots + A_n|$. When this guess is too small, at some point the criterion $\sum_{j \le i} |X_{r,j}| \ge s + n/2^r$ in line 13 is satisfied, from which point on we set some of the sets $X_{r,i}$ to $\{0\}$, but we ensure that we end up with $|X_{\log n,1}| > s$. This allows us to conclude that our guess s was too small, so we increase it. When our guess s reaches the smallest power of 2 that is at least $|A_1 + \ldots + A_n|$, then the algorithm correctly computes $X_{\log n,1} = A_1 + \ldots + A_n$ and returns this set. ◀

▶ **Lemma 14** (Running Time of Algorithm 2). *Let $k := |A_1| + \ldots + |A_n| + |A_1 + \ldots + A_n|$ be the total input plus output size. Depending on whether we use Theorem 1 or Theorem 5 for sumset computation, Algorithm 2 is*

1. *deterministic and runs in time $k \cdot 2^{O(\sqrt{\log k \log\log m})} \cdot \log m$, or*
2. *randomized and runs in expected time $O(k \cdot \mathrm{polylog}(mk))$.*

Proof. Let $T(k, m)$ be the running time of our algorithm. Note that we have at most one call to a recursive subproblem in line 12, incurring time $T(k, d)$, where d is a divisor of m and thus $d \le m/2$.

Let s^* be the smallest power of 2 that is at least $|A_1 + \ldots + A_n|$. Similarly as in the proof of correctness, we see that the algorithm only performs iterations s from 1 to at most s^*, since we return the correct output in iteration s^*, unless we call a recursive subproblem before that.

We bound the running time in iteration s as follows. For any iteration r, let $X_{r,i(r)}$ be the last set that we computed in line 7. (That is, after computing $X_{r,i(r)}$ we either move to a recursive call, or we set all remaining sets $X_{r,j} \leftarrow \{0\}$, for any $j > i(r)$.) Note that $\sum_{j<i(r)} |X_{r,j}| < s + n/2^r$, since otherwise we would have set $X_{r,i(r)} \leftarrow \{0\}$ and not computed it in line 7. Moreover, $|X_{r,i(r)}| \le |A_1 + \ldots + A_n| \le k$ by (2). By Theorem 1, computing $X_{r,i}$ takes time

$$|X_{r,i}| \cdot 2^{O(\sqrt{\log |X_{r,i}| \log\log m})} \le |X_{r,i}| \cdot 2^{O(\sqrt{\log k \log\log m})}.$$

Therefore, the total time spent in iteration r is bounded by

$$\left(s + \frac{n}{2^r} + k\right) \cdot 2^{O(\sqrt{\log k \log\log m})} \le k \cdot 2^{O(\sqrt{\log k \log\log m})},$$

for any $1 \le s \le s^* = O(k)$. Summing over all iterations r adds a factor $\log n \le \log k \le 2^{O(\sqrt{\log k})}$, which can be ignored. Summing over all iterations s adds a factor $\log s^* = O(\log k)$, which can also be ignored. Adding the potential recursive call, the total running time is

$$T(k, m) \le k \cdot 2^{O(\sqrt{\log k \log\log m})} + T(k, m/2).$$

This solves to total time $k \cdot 2^{O(\sqrt{\log k \log\log m})} \cdot \log(m)$.

The analysis of the randomized variant is analogous. ◀

Proof of Theorem 4. Algorithm 2 almost proves the theorem, except that the deterministic running time shown in Lemma 14 is $k \cdot 2^{O(\sqrt{\log k \log\log m})} \cdot \log m$ instead of the promised $k \cdot 2^{O(\sqrt{\log k \log\log m})}$. Note that the former can be bounded by the latter unless $k \le \log^c m$, for some absolute constant c. In the case $k \le \log^c m$ we switch to a different algorithm. Specifically, we simply compute $((A_1 + A_2) + A_3) + \ldots + A_n$ in a linear fashion, in each step using a naive sumset computation that computes $A + B$ in time $\widetilde{O}(|A| \cdot |B|)$. Since

each intermediate result has size at most k, each sumset computation takes time $O(k^2)$. Since $n \leq k$, in total this simple algorithm runs in time $O(k^3)$. Finally, since $k \leq \log^c m$ we can bound $O(k^3) \leq 2^{O(\sqrt{\log k \log\log m})}$. This shows the promised running time also in case $k \leq \log^c m$. We obtain the promised guarantees even if we do not know k, by running both algorithms in parallel until the first one finishes. ◀

6 Output-sensitive Sumset Computation

Recall that in sumset computation we are given sets $A, B \subseteq \mathbb{Z}_m$ and the task is to compute $A + B$. In this section we present a deterministic algorithm for sumset computation. We also show a generalization to convolution of non-negative vectors, proving Theorem 1.

Chan and Lewenstein [15] designed very efficient algorithms for sumset computation in a specialized setting, in which the input additionally contains a set T promised to be a superset of $A + B$. Their running time is close to linear in $|T|$. Specifically, they proved the following lemma.

▶ **Lemma 15** (FFT Lemma from [15]). *Given sets $A, B \subseteq \{0, 1, \ldots, m-1\}$ and given a set T which is known to be a superset of $A + B$, we can compute $A + B$ (over $\mathbb{Z}$)*
(1) *by a randomized Las Vegas algorithm in $O(|T| \operatorname{polylog} m)$ expected time, or*
(2) *by a deterministic algorithm in $|T| \cdot 2^{O(\sqrt{\log |T| \log\log m})}$ time*
The running time bounds are taken from [15, Section 8].

Here we show a trick that yields the same time bounds in the standard setting (without the additional set T). We note that it makes no significant difference whether we compute $A + B$ over $\mathbb{Z}_m$ or over $\mathbb{Z}$, as discussed also in the introduction. We choose to work over $\mathbb{Z}_m$, for consistency with the rest of this paper.

▶ **Lemma 16.** *Given sets $A, B \subseteq \mathbb{Z}_m$, we can compute $A + B$ (over $\mathbb{Z}_m$)*
(1) *by a randomized Las Vegas algorithm in $O(|A + B| \operatorname{polylog} m)$ expected time, or*
(2) *by a deterministic algorithm in $|A + B| \cdot 2^{O(\sqrt{\log |A+B| \log\log m})}$ time.*

Note that bullet point (1) reproves Theorem 5 by Cole and Hariharan [16], and bullet point (2) answers an open problem by Chan and Lewenstein [15].

Proof. First note that we can assume m to be a power of 2. Indeed, if m is not a power of 2, we let m' be the smallest power of 2 greater than $2m$. Given $A, B \subseteq \{0, 1, \ldots, m-1\}$ we compute $A + B$ over $\mathbb{Z}_{m'}$ and take the resulting set modulo m to obtain $A + B$ over $\mathbb{Z}_m$. This assumption is not necessary, but shall make the exposition cleaner, avoiding using the ceil and floor functions.

So assume that m is a power of 2, and set $m' := m/2$. Let $A' := A \bmod m'$ and $B' := B \bmod m'$ and recursively compute $S := A' + B'$ over $\mathbb{Z}_{m'}$. Then we have $S = (A+B) \bmod m'$. Thus, since $\max(A) + \max(B) < 2m \leq 4m'$, the set $T := S + \{0, m', 2m', 3m'\}$ covers $A + B$. In other words, T is a superset of $A + B$ over $\mathbb{Z}$. We can thus use the FFT Lemma to compute $A + B$ over $\mathbb{Z}$. Reducing the resulting set modulo m yields $A + B$ over $\mathbb{Z}_m$. This leads to the recursive Algorithm 3.

Since we can bound $|T| \leq 4|S| \leq 4|A + B|$, the expected running time of one recursive step is $O(|A + B| \cdot \operatorname{polylog} m)$, and there are $O(\log m)$ recursive steps. This yields the claimed randomized running time. For the deterministic variant we obtain running time $|A + B| \cdot 2^{O(\sqrt{\log |A+B| \log\log m})} \cdot \operatorname{polylog} m$ from the FFT Lemma, times an additional $\log m$ factor due to the recursion.

Algorithm 3 A deterministic algorithm for computing the sumset $A + B$ over $\mathbb{Z}_m$.

```
1: procedure DeterministicSumset(A, B, m)
2:                ▷ m is a power of 2; non-empty A, B ⊆ Z_m; computes A + B over Z_m
3:     m' := m/2
4:     S ← DeterministicSumset(A mod m', B mod m', m')
5:     T ← S + {0, m', 2m', 3m'} over Z                    ▷ T ⊇ A + B over Z
6:     Compute R := A + B over Z via the FFT Lemma using additional input T
7:     return R mod m
```

Now, to get rid of the additional polylog(m) factor and obtain the promised guarantee, we shall observe the following. If $|A + B| \geq \log m$, then this running time is bounded by the claimed $|A + B| \cdot 2^{O(\sqrt{\log |A+B| \log\log m})}$. If $|A + B| < \log m$, then the naive approach which computes $A + B$ in time $\tilde{O}(|A| \cdot |B|) = \tilde{O}(|A + B|^2) = 2^{O(\sqrt{\log |A+B| \log\log m})}$. Running both algorithms in parallel until the first one finishes yields the claimed bound. ◀

A similar result also holds for convolution of non-negative vectors. We denote by $\|x\|_0$ the number of non-zero entries of a vector x.

▶ **Lemma 17.** *Given vectors $A, B \in \mathbb{R}^m_{\geq 0}$, we can compute their convolution $A \star B$ (with wraparound)*
(1) *by a randomized Las Vegas algorithm in $O(\|A \star B\|_0 \operatorname{polylog} m)$ expected time, or*
(2) *by a deterministic algorithm in $\|A \star B\|_0 \cdot 2^{O(\sqrt{\log \|A\star B\|_0 \log\log m})}$ time.*
Again bullet point (1) reproves a result by Cole and Hariharan [16], and bullet point (2) proves Theorem 1.

Proof. Denote by I and J the indicator vectors of the non-zero entries of A and B, respectively. Observe that $|I + J| = \|A \star B\|_0$. We can thus compute $I + J$ in expected time $O(\|A \star B\|_0 \operatorname{polylog} m)$ by Lemma 16. We now make use of a variant of the FFT Lemma from [15, Remark 8.2], stating that if we know a superset $T \supseteq I + J$ then we can compute $A \star B$ in expected time $O(|T| \operatorname{polylog} m)$. Using this for $T = I + J$ yields expected time $O(\|A \star B\|_0 \operatorname{polylog} m)$, or time $\|A \star B\|_0 \cdot 2^{O(\sqrt{\log \|A\star B\|_0 \log\log m})} \cdot \operatorname{poly}(\log m)$ for the deterministic variant. Now, we can get rid of the polylog(m) factors in the deterministic variant using the same argument as the one in Lemma 16. ◀

7 Computing the Symmetry Group

In this section, we show how to compute the symmetry group $Sym(A)$ for any given non-empty set $A \subseteq \mathbb{Z}_m$ in time $O(|A|)$, proving Theorem 7. Let $n := |A|$ and denote by $a_1 < a_2 < \ldots < a_n$ the elements of A. For simplicity of notation, we set

$$a_{n+1} := a_1, \quad a_{n+2} := a_2, \quad \ldots, \quad a_{2n} := a_n.$$

Note that for our applications of this Theorem, a_i correspond to residue classes modulo m. We construct a string P (the pattern) of length n by setting for any $1 \leq i \leq n$:

$$P_i := (a_{i+1} - a_i) \bmod m.$$

Similarly, we construct a string T (the text) of length $2n - 1$ by setting for any $1 \leq i \leq 2n - 1$:

$$T_i := (a_{i+1} - a_i) \bmod m.$$

Note that the text is constructed by repeating the pattern twice and removing the last letter.

We say that there is a match of pattern P in text T at position i if $P_j = T_{i-1+j}$ holds for any $1 \leq j \leq n$. The following lemma shows that the matches of P in T are in one-to-one correspondence with the symmetry group $Sym(A)$. Since all matches of P in T can be computed in time $O(n)$ by the classic Knuth-Morris-Pratt pattern matching algorithm, this finishes the proof of Theorem 7.

▶ **Lemma 18.** *If there is a match of P in T at position i, then $a_i - a_1 \in Sym(A)$. Moreover, for any $x \in Sym(A)$, we have $x = a_i - a_1$ for some $1 \leq i \leq n$ and there is a match of P in T at position i.*

Proof. Note that there is a match at position i if and only if for all $1 \leq j \leq n$

$$a_{j+1} - a_j = a_{i+j} - a_{i+j-1} \pmod{m}.$$

Summing this equation in a telescoping sum over all $j \in \{1, \ldots, \ell-1\}$, for any fixed $1 \leq \ell \leq n$, yields

$$a_\ell - a_1 = a_{i+\ell-1} - a_i \pmod{m},$$

or, equivalently,

$$a_\ell + (a_i - a_1) = a_{i+\ell-1} \pmod{m}.$$

This establishes $a_i - a_1 \in Sym(A)$.

For the second part, recall that $Sym(A) \subseteq A - \{a_1\}$, as discussed in Section 2.2. Therefore for any $x \in Sym(A)$ we have $x = a_i - a_1$ for some i. Now consider the values $a'_j := (a_j - x) \bmod m$ for $1 \leq j \leq n$. Observe that the sequence $a'_1, \ldots, a'_n$ is monotonically increasing up to some point, where the modulo operation reduces by an additional $-m$, and then is again monotonically increasing. In particular, for some $1 \leq r \leq n$ we have

$$a'_r < a'_{r+1} < \ldots < a'_{n-1} < a'_n < a'_1 < a'_2 < \ldots < a'_{r-2} < a'_{r-1}.$$

Since $x \in Sym(A)$ and $Sym(A)$ is a group, also $-x \in Sym(A)$, and thus $a'_j \in A$ for all j. Hence, the n different values a'_j must correspond to the elements of A. It follows that $a'_{r-1+j} = a_j$ for all $1 \leq j \leq n$.

Observing that $a'_i = a_i - x = a_i - (a_i - a_1) = a_1 \pmod{m}$, we see that $r = i$. In other words, we have for all $1 \leq j \leq n$

$$a_{i-1+j} - (a_i - a_1) = a_j \pmod{m}.$$

Subtracting this equation for j from this equation for $j+1$ yields, for any $1 \leq j \leq n$,

$$a_{i+j} - a_{i+j-1} = a_{j+1} - a_j \pmod{m}.$$

As noted in the beginning of this proof, this means that there is a match of P in T at position i. ◀

References

1. Amir Abboud, Karl Bringmann, Danny Hermelin, and Dvir Shabtay. Seth-based lower bounds for subset sum and bicriteria path. In *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 41–57. SIAM, 2019.
2. Amihood Amir, Ayelet Butman, and Ely Porat. On the relationship between histogram indexing and block-mass indexing. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 372(2016):20130132, 2014.
3. Amihood Amir, Oren Kapah, and Ely Porat. Deterministic length reduction: Fast convolution in sparse data and applications. In *Combinatorial Pattern Matching, 18th Annual Symposium, CPM 2007, London, Canada, July 9-11, 2007, Proceedings*, pages 183–194, 2007.
4. Amihood Amir, Oren Kapah, Ely Porat, and Amir Rothschild. Polynomials: a new tool for length reduction in binary discrete convolutions. *CoRR*, 2014.
5. Andrew Arnold and Daniel S Roche. Output-sensitive algorithms for sumset and sparse polynomial multiplication. In *Proceedings of the 2015 ACM on International Symposium on Symbolic and Algebraic Computation*, pages 29–36. ACM, 2015.
6. Per Austrin, Petteri Kaski, Mikko Koivisto, and Jesper Nederlof. Dense Subset Sum may be the hardest. In *Proc. of the 33rd Symposium on Theoretical Aspects of Computer Science (STACS)*, pages 13:1–13:14, 2016.
7. Kyriakos Axiotis, Arturs Backurs, Karl Bringmann, Ce Jin, Vasileios Nakos, Christos Tzamos, and Hongxun Wu. Fast and simple modular subset sum. In *Symposium on Simplicity in Algorithms (SOSA)*, pages 57–67. SIAM, 2021.
8. Kyriakos Axiotis, Arturs Backurs, Ce Jin, Christos Tzamos, and Hongxun Wu. Fast modular subset sum using linear sketching. In *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 58–69. SIAM, 2019.
9. Nikhil Bansal, Shashwat Garg, Jesper Nederlof, and Nikhil Vyas. Faster space-efficient algorithms for Subset Sum, k-Sum and related problems. In *Proc. of the 49th Annual ACM SIGACT Symposium on Theory of Computing (STOC)*, pages 198–209, 2017.
10. Karl Bringmann. A near-linear pseudopolynomial time algorithm for Subset Sum. In *Proc. of of the 28th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1073–1084, 2017.
11. Karl Bringmann, Nick Fischer, and Vasileios Nakos. Sparse nonnegative convolution is equivalent to dense nonnegative convolution. In *STOC 2021 (to appear)*. SIAM, 2021.
12. Karl Bringmann and Vasileios Nakos. Top-k-convolution and the quest for near-linear output-sensitive subset sum. In *Proccedings of the 52nd Annual ACM SIGACT Symposium on Theory of Computing (STOC)*, pages 982–995, 2020.
13. Jean Cardinal and John Iacono. Modular subset sum, dynamic strings, and zero-sum sets. In *Symposium on Simplicity in Algorithms (SOSA)*, pages 45–56. SIAM, 2021.
14. David E. Cardoze and Leonard J. Schulman. Pattern matching for spatial point sets. In *39th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 156–165, 1998.
15. Timothy M. Chan and Moshe Lewenstein. Clustered integer 3SUM via additive combinatorics. In *Proc. of the 47th Annual ACM Symposium on Theory of Computing (STOC)*, pages 31–40, 2015.
16. Richard Cole and Ramesh Hariharan. Verifying candidate matches in sparse and wildcard matching. In *Proceedings of the 34th Annual ACM Symposium on Theory of Computing (STOC)*, pages 592–601. ACM, 2002.
17. Zvi Galil and Oded Margalit. An almost linear-time algorithm for the dense Subset-Sum problem. *SIAM J. Comput.*, 20(6):1157–1189, 1991.
18. Pascal Giorgi, Bruno Grenet, and Armelle Perret du Cray. Essentially optimal sparse polynomial multiplication. In *Proceeding of the 45th International Symposium on Symbolic and Algebraic Computation (ISSAC)*, pages 202–209. ACM, 2020.

19 Ce Jin and Hongxun Wu. A simple near-linear pseudopolynomial time randomized algorithm for subset sum. In *2nd Symposium on Simplicity in Algorithms, SOSA@SODA 2019*, volume 69 of *OASICS*, pages 17:1–17:6, 2019.

20 Konstantinos Koiliaris and Chao Xu. A faster pseudopolynomial time algorithm for Subset Sum. In *Proc. of the 28th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1062–1072, 2017.

21 Michael Monagan and Roman Pearce. Parallel sparse polynomial multiplication using heaps. In *Proceedings of the 2009 International Symposium on Symbolic and Algebraic Computation (ISSAC)*, pages 263–270. ACM, 2009.

22 Marcin Mucha, Karol Węgrzycki, and Michał Włodarczyk. A subquadratic approximation scheme for partition. In *Proceedings of the 30th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 70–88, 2019.

23 Shanmugavelayutham Muthukrishnan. New results and open problems related to non-standard stringology. In *Proceedings of the 6th Annual Symposium on Combinatorial Pattern Matching (CPM)*, volume 937, pages 298–317. Springer, 1995.

24 Vasileios Nakos. Nearly optimal sparse polynomial multiplication. *IEEE Trans. Inf. Theory*, 66(11):7231–7236, 2020.

25 Daniel S Roche. Adaptive polynomial multiplication. *Proc. Milestones in Computer Algebra (MICA'08)*, pages 65–72, 2008.

26 Daniel S. Roche. What can (and can't) we do with sparse polynomials? In *Proceedings of the 2018 ACM International Symposium on Symbolic and Algebraic Computation (ISSAC)*, pages 25–30, 2018.

27 Terence Tao and Van H. Vu. *Additive Combinatorics*, volume 105. Cambridge University Press, 2006.

28 Joris Van Der Hoeven and Grégoire Lecerf. On the complexity of multivariate blockwise polynomial multiplication. In *Proceedings of the 37th International Symposium on Symbolic and Algebraic Computation (ISSAC)*, pages 211–218. ACM, 2012.

Additive Approximation Schemes for Load Balancing Problems

Moritz Buchem
Maastricht University, Maastricht, The Netherlands

Lars Rohwedder
EPFL, Lausanne, Switzerland

Tjark Vredeveld
Maastricht University, Maastricht, The Netherlands

Andreas Wiese
University of Chile, Santiago, Chile

Abstract

We formalize the concept of *additive approximation schemes* and apply it to load balancing problems on identical machines. Additive approximation schemes compute a solution with an absolute error in the objective of at most ϵh for some suitable parameter h and any given $\epsilon > 0$. We consider the problem of assigning jobs to identical machines with respect to common load balancing objectives like *makespan minimization*, the *Santa Claus* problem (on identical machines), and the *envy-minimizing Santa Claus* problem. For these settings we present additive approximation schemes for $h = p_{\max}$, the maximum processing time of the jobs.

Our technical contribution is two-fold. First, we introduce a new relaxation based on integrally assigning slots to machines and *fractionally* assigning jobs to the slots. We refer to this relaxation as the *slot-MILP*. While it has a linear number of integral variables, we identify structural properties of (near-)optimal solutions, which allow us to compute those in polynomial time. The second technical contribution is a local-search algorithm which rounds any given solution to the slot-MILP, introducing an additive error on the machine loads of at most $\epsilon \cdot p_{\max}$.

2012 ACM Subject Classification Theory of computation → Scheduling algorithms

Keywords and phrases Load balancing, Approximation schemes, Parallel machine scheduling

Digital Object Identifier 10.4230/LIPIcs.ICALP.2021.42

Category Track A: Algorithms, Complexity and Games

Related Version *Full Version*: `https://arxiv.org/abs/2007.09333` [6]

Funding *Lars Rohwedder*: Swiss National Science Foundation project 200021-184656.
Andreas Wiese: Partially supported by FONDECYT Regular grant 1200173.

Acknowledgements We wish to thank José Verschae, Alexandra Lassota and Klaus Jansen for helpful discussions.

1 Introduction

Typically, when constructing an approximation algorithm, one provides a multiplicative approximation ratio ρ such that the respective algorithm finds a solution of value at most (or at least, in case of maximization problems) $\rho \cdot \mathrm{OPT}$, where OPT is the optimal solution value. Then, an approximation scheme is a family with an algorithm for each ratio $\rho = 1 + \epsilon$ with $\epsilon > 0$ (or $\rho = 1 - \epsilon$ for maximization problems).

In this paper, we study approximation algorithms and schemes for which we measure their performance as the absolute difference between the value of their computed solution and OPT. We measure this difference with respect to some problem depending quantity h (which might be much smaller than OPT).

▶ **Definition 1.** *For a given optimization problem, consider a quantity defined by $h(I)$ for each instance I. An algorithm is an* additive approximation algorithm w.r.t. *h if for any instance I it computes in polynomial time a solution with value $A(I)$ satisfying $|A(I) - \mathrm{OPT}(I)| \leq h(I)$. An* additive approximation scheme w.r.t. *h is a family of algorithms containing an additive approximation algorithm w.r.t $\epsilon \cdot h$ for each $\epsilon > 0$.*

Studying additive approximation algorithms is particularly interesting in the following two scenarios.

1. If $h(I) \ll (\rho - 1)\mathrm{OPT}(I)$ for some $\rho > 1$ then an additive approximation algorithm w.r.t. h gives a much stronger guarantee than a (multiplicative) ρ-approximation algorithm. In particular, if $h(I) \ll \mathrm{OPT}(I)$ an additive approximation schemes then gives a much stronger guarantee than a PTAS.
2. When there cannot exist a PTAS, or even any multiplicative guarantee for a given problem, additive approximation algorithms give an alternative notion for approximating it. A notable example is the case when it is NP-hard to decide whether $\mathrm{OPT} = 0$, as then no multiplicative approximation guarantee can be obtained.

While a lot of research has focused on traditional multiplicative approximation guarantees, additive approximation guarantees are relatively unexplored in the literature. Notable exceptions include Vizing's algorithm that finds an edge coloring with at most $\Delta + 1$ colors, where Δ is the maximum degree of a graph [36], which is hence an additive approximation for $h \equiv 1$. Also, for bin packing there is an algorithm known that uses at most $\mathrm{OPT} + O(\log \mathrm{OPT})$ bins [17] (improving earlier results in [25] and [33]) which is thus an additive approximation for $h \equiv O(\log \mathrm{OPT})$.

In this paper, we present additive approximation schemes for scheduling and load balancing problems which are among the classical problems in the literature on approximation algorithms, starting with the seminal work of Graham [14]. In these problems, n jobs need to be processed by m machines such that each job is completely processed by one single machine. Each job j has a given processing time p_j and the load of a machine i is the sum of the processing times of the jobs assigned to i. The goal is to find a schedule (represented by an assignment of jobs to the machines) that optimizes some objective function over the machine loads. Since it is strongly NP-hard to decide whether there is a schedule that assigns the same load to each machine (see [13]), most non-trivial load balancing problems of this form are also strongly NP-hard. This observation has led to extensive research on approximation algorithms. In the following, we consider three variations of load balancing problems.

Objective functions. Our first objective function is to minimize the maximum machine load, i.e., to minimize the *makespan*. This is the one of the most classical scheduling problems on parallel machines and has led to the first approximation algorithms [14,15]. Sahni [34] showed that the problem admits an FPTAS for constant number of machines and Hochbaum and Shmoys [19] found a PTAS if the number of machines is part of the input. Since then, there has been lively research in improving the running time, e.g., to an EPTAS [1, 8, 18, 20, 21].

The second objective function that we consider yields the *max-min allocation* problem [7]. Here, the goal is to maximize the load of the least loaded machine. Bansal and Sviridenko [5] called it the Santa Claus problem when they studied it in the restricted assignment setting. This objective is considered to measure the fairness of the allocation. The case of identical machines was also considered by Woeginger [37] who presents a PTAS.

As a third and final objective function, we consider to minimize the *maximum envy*, which is defined as the maximum load minus the minimum load. This objective has been considered by Lipton et. al [30]. While in the Santa Claus problem fairness is measured

by the minimum load of a machine, in this setting fairness is considered by the difference between the maximum and minimum load. Note that it is strongly NP-hard to decide whether or not the envy is 0. Therefore, unless P = NP, there cannot exist any polynomial time approximation algorithm with any (multiplicative) performance guarantee.

For all three variants there is a simple greedy algorithm, which assigns the jobs iteratively to the respective least loaded machine, and which gives an additive error of $p_{\max} := \max_{j \in J} p_j$. Such an additive approximation w.r.t. $p_{\max}$ exists even for unrelated machines, i.e., when the processing time of a job depends on the machine [29]. Note that this guarantee is incomparable to the error of ϵOPT of a PTAS. In particular, in the regime where $p_{\max} = o(\mathrm{OPT})$ the greedy algorithm is still the best algorithm we know.

Our contribution. In this paper, we present additive approximation schemes w.r.t. $p_{\max}$ for the three load balancing problems defined above on identical machines. For the makespan and Santa Claus objective this gives a significant improvement over the greedy algorithm mentioned above while also dominating the guarantees of the known PTASes[1]; for minimizing the maximum envy this demonstrates how additive approximation schemes can lead to non-trivial guarantees when no multiplicative guarantees are possible.

Since $p_{\max}$ can be much smaller than OPT, the main approach of the known (multiplicative) PTASes does not work when aiming for an additive approximation guarantee of $\epsilon p_{\max}$. In those results, a job j is typically considered as small if $p_j \leq \epsilon\mathrm{OPT}$ and otherwise as large. Then there is only a constant number of large jobs on each machine in the optimal solution which allows methods based on enumeration or integer programming in constant dimension (after rounding or grouping the job sizes according to, e.g., powers of $(1+\epsilon)$). The small jobs are finally assigned greedily. However, if $p_{\max} < \epsilon\mathrm{OPT}$ then these algorithms would simply assign all jobs greedily which yields an additive error of up to $p_{\max} > \epsilon \cdot p_{\max}$. Given this, it seems natural to define a job j to be large if $p_j > \epsilon \cdot p_{\max}$; however, then it is not guaranteed that there are only a constant number of large jobs on each machine and thus the aforementioned techniques are not applicable anymore. Moreover, we cannot even afford to round all job sizes to powers of $1+\epsilon$ since this might yield a total error of $\epsilon \sum_j p_j$, which can be much larger than $\epsilon \cdot p_{\max}$. Therefore, there is need for new (non-trivial) machinery.

To this end, we present a new fractional relaxation for this general class of load balancing problems, which we call the *slot-MILP*. This slot-MILP can be interpreted as a strengthened variant of the assignment-LP. The assignment-LP is a relaxation in which each job is assigned separately (fractionally) to a machine. In the slot-MILP we first group jobs of similar sizes, but we do not round the job sizes (unlike most previous PTASes). In particular, different jobs belonging to the same group may have different job sizes. In addition to the constraints of the assignment-LP we require an *integral* number of jobs of each group to be assigned to each machine (which can be implemented using a linear number of integer variables). Our relaxation can be thought of as assigning slots (for the groups of jobs) *integrally* to machines and then assigning the jobs *fractionally* to the slots.

Due to the many integer variables, it is not obvious how to solve the slot-MILP efficiently. This contrasts our approach to many other approximation algorithms which are based on purely fractional relaxations or on MILP-relaxations with only few integral variables which can be solved with Lenstra's algorithm (e.g., as done in some EPTASes for minimizing the

[1] While for makespan minimization $p_{\max} \leq \mathrm{OPT}$ always holds, for Santa Claus this can be assumed essentially w.l.o.g., since the optimum does not change if job sizes are capped at OPT and the latter can be guessed by binary search.

makespan [20, 21]). Instead, we manage to solve it using non-trivial structural properties combined with dynamic programming. While the additive integrality gap of the assignment-LP can be as large as $p_{\max}$, for the slot-MILP this gap is only $\epsilon \cdot p_{\max}$. We show this using a rounding procedure inspired by a local search method for the restricted assignment problem [23, 24, 35]. The local search algorithm repeatedly moves jobs between machines, eventually converging to a good solution. Although in the restricted assignment problem no polynomial running time bound is known for the local search procedure, in our case we obtain such a bound for our local search.

We remark that our slot-MILP is stronger than the known configuration-LP, e.g., since for minimizing the makespan on identical machines, the latter has a multiplicative integrality gap of at least $1+\frac{1}{1023}$ [28] and hence an additive integrality gap of at least $\mathrm{OPT}\cdot\frac{1}{1023} \geq p_{\max}\cdot\frac{1}{1023}$.

Other related work. The case of small values of $p_{\max}$ has also been considered from a parameterized point of view: If all processing times are integers, then it is possible to obtain a running time that is fixed-parameter tractable (FPT) for the parameter $k = p_{\max}$ [26, 27, 31]. In other words, there is an algorithm that finds an exact solution in time $f(k) \cdot |I|^{O(1)}$ for some computable function f.

Other variants of load balancing problems on identical machines have been considered by Alon et al. [1]. They identify some conditions on the objective function, e.g. makespan minimization and the Santa Claus problem, so that the load balancing or machine scheduling problem admits an (E)PTAS. The Santa Claus problem has been considered in the restricted assignment setting in which each job is only allowed to be processed on a subset of the machines, starting with [5]. In a series of papers [3, 4, 9, 12, 16], the approximation ratio was further and further improved, and the currently best known result is a polynomial time $(4+\epsilon)$-approximation algorithm [9, 10] and an upper bound of $3+\frac{21}{26} \approx 3.808$ for the integrality gap of the configuration-LP [9] (which can be solved in polynomial time to any desired accuracy).

For the bin packing problem, Jansen et al. [22] present an additive approximation algorithm w.r.t $h \equiv 1$ in time exponential in the optimal number of bins plus a polynomial in the number of items to be packed. Ophelders et al. [32] showed that a simple local search algorithm for the so-called Equitable Hamiltonian Cycle finds a solution that is at most 1 away from the optimal solution value. Alon et al. [2] present an additive approximation algorithm w.r.t. $h = \epsilon n^2$ for every $\epsilon > 0$ for the edge deletion problem to obtain a graph with a monotone property. This is hence an additive approximation scheme for $h = n^2$ (which is in fact an upper bound on the minimum number of edges to be deleted).

2 Slot-MILP

We introduce an alternative relaxation for a general class of load balancing problems on identical machines. We first formally define this class of load balancing problems as the *target load balancing problem.*

▶ **Definition 2.** *In the* target load balancing problem *we are given a set of jobs $\mathcal{J}$ with a processing time p_j for each $j \in \mathcal{J}$ and a set of machines $\mathcal{M}$ with values ℓ, u. The goal is to assign each job $j \in \mathcal{J}$ to a machine $i \in \mathcal{M}$ such that every machine load satisfies the target load interval $[\ell, u]$.*

This generalizes the load balancing settings mentioned earlier, e.g., for makespan minimization we choose $\ell = 0$ and $u = T$, where T is a guess on the optimal makespan.

Let $\epsilon > 0$ and assume w.l.o.g. that $1/\epsilon \in \mathbb{N}$. Our task is to either assert that there is no solution for the given instance or to find a solution in which the load of each machine i is in the interval $[\ell - \epsilon \cdot p_{\max}, u + \epsilon \cdot p_{\max}]$ with $p_{\max} := \max_{j \in J} p_j$. First we partition the jobs into sets $\mathcal{J}_1, \ldots, \mathcal{J}_{1/\epsilon}$, where for $k = 1, ..., 1/\epsilon$ the set $\mathcal{J}_k$ contains all jobs $j \in \mathcal{J}$ with $p_j \in ((k-1)\epsilon \cdot p_{\max}, k\epsilon \cdot p_{\max}]$. We define a new relaxation for this problem in which for each machine i and each $k = 1, ..., 1/\epsilon$ we specify integrally how many jobs from $\mathcal{J}_k$ are assigned to i (one may imagine that this defines slots for jobs from $\mathcal{J}_k$ on i). Then the jobs from $\mathcal{J}_k$ are assigned fractionally to these slots.

$$\begin{aligned} \sum_{i \in \mathcal{M}} x_{i,j} &= 1 && \forall j \in \mathcal{J} \\ \ell \leq \sum_{j \in \mathcal{J}} p_j x_{i,j} &\leq u && \forall i \in \mathcal{M} \\ \sum_{j \in \mathcal{J}_k} x_{i,j} &= y_{i,k} && \forall i \in \mathcal{M}, \forall k \in \{1, \ldots, 1/\epsilon\} \\ x_{i,j} &\geq 0 && \forall j \in \mathcal{J}, i \in \mathcal{M} \\ y_{i,k} &\in \mathbb{N}_0 && \forall i \in \mathcal{M}, k \in \{1, \ldots, 1/\epsilon\} \end{aligned} \tag{1}$$

We refer to this relaxation as the slot-MILP. The integer variables define exactly how many jobs of a type are assigned to a machine but do not imply a specific load based on rounded processing times. The load of a machine is based on an assignment that satisfies the distribution of slots among the machines.

Since the slot-MILP contains $1/\epsilon \cdot |\mathcal{M}|$ integral variables, it is not clear how to solve it in polynomial time. Nevertheless, we present two methods of efficiently solving the slot-MILP. The first method gives an exact solution while the second method gives a solution that slightly violates the target load intervals. Afterwards, we show how to round a fractional solution of the slot-MILP to an integral solution, while violating the load interval $[\ell, u]$ for each machine $i \in \mathcal{M}$ by at most $\epsilon \cdot p_{\max}$.

2.1 Exact solution method for the relaxation

We make use of a structural property to find an exact solution to the slot-MILP. Note that in this case an exact solution is one that satisfies (1). This structure allows us to guess the values of the integral variables in polynomial time and then the remaining problem is only a linear program.

Given a solution (x, y), we write y_i for the $(1/\epsilon)$-tuple $(y_{i,1}, \ldots, y_{i,1/\epsilon})$. Using similar arguments to [11] we show that there exist solutions in which there are not too many different vectors y_i.

▶ **Lemma 3.** *There is a solution (x, y) to the slot-MILP such that for all $i, i' \in \mathcal{M}$ with $y_i \equiv y_{i'} \bmod 2$ it follows that $y_i = y_{i'}$.*

Proof. Let (x, y) be a solution to the slot-MILP and assume that x is the solution which minimizes

$$\sum_{i \in \mathcal{M}} \sum_{k=1}^{1/\epsilon} \|y_{i,k}\|_2. \tag{2}$$

Now suppose toward contradiction that there are i_1, i_2 with $y_{i_1} \equiv y_{i_2} \bmod 2$, but $y_{i_1} \neq y_{i_2}$. We construct a new solution x', which has a lower value of (2). We set $x'_{i,j} = x_{i,j}$ for all $i \notin \{i_1, i_2\}$ and $x'_{i_1,j} = x'_{i_2,j} = (x_{i_1,j} + x_{i_2,j})/2$. In other words, we evenly distribute all jobs between i_1 and i_2. Let us first check that the solution remains feasible. Let $j \in \mathcal{J}$. Then

$$\begin{aligned}\sum_{i\in\mathcal{M}} x'_{i,j} &= x'_{i_1,j} + x'_{i_2,j} + \sum_{i\notin\{i_1,i_2\}} x'_{i,j} \\ &= \frac{x_{i_1,j} + x_{i_2,j}}{2} + \frac{x_{i_1,j} + x_{i_2,j}}{2} + \sum_{i\notin\{i_1,i_2\}} x_{i,j} \\ &= \sum_{i\in\mathcal{M}} x_{i,j} = 1.\end{aligned}$$

For all machines $i \notin \{i_1, i_2\}$ the load does not change and, hence, the load of machine i remains within $[\ell, u]$. For i_1 and i_2, we argue

$$\begin{aligned}\sum_{j\in\mathcal{J}} p_j x'_{i_1,j} = \sum_{j\in\mathcal{J}} p_j x'_{i_2,j} &= \sum_{j\in\mathcal{J}} p_j \frac{x_{i_1,j} + x_{i_2,j}}{2} \\ &= \frac{1}{2}\sum_{j\in\mathcal{J}} p_j x_{i_1,j} + \frac{1}{2}\sum_{j\in\mathcal{J}} p_j x_{i_2,j} \\ &\leq \frac{u}{2} + \frac{u}{2} = u\end{aligned}$$

and

$$\begin{aligned}\sum_{j\in\mathcal{J}} p_j x'_{i_1,j} = \sum_{j\in\mathcal{J}} p_j x'_{i_2,j} &= \sum_{j\in\mathcal{J}} p_j \frac{x_{i_1,j} + x_{i_2,j}}{2} \\ &= \frac{1}{2}\sum_{j\in\mathcal{J}} p_j x_{i_1,j} + \frac{1}{2}\sum_{j\in\mathcal{J}} p_j x_{i_2,j} \\ &\geq \frac{\ell}{2} + \frac{\ell}{2} = \ell.\end{aligned}$$

Hence, the solution remains optimal. As for the integrality constraints, again the machines $i \notin \{i_1, i_2\}$ do not change. Let $k \in \{1, \ldots, 1/\epsilon\}$. Since $y_{i_1,k} \equiv y_{i_2,k}$, we have that $y_{i_1,k} + y_{i_2,k}$ is even. It follows that

$$\sum_{j\in\mathcal{J}_k} x'_{i_1,j} = \sum_{j\in\mathcal{J}_k} x'_{i_2,j} = \frac{y_{i_1} + y_{i_2}}{2}$$

is integral. Now it remains to show that (2) has decreased. Notice that by triangle inequality

$$\|y'_{i_1,k}\|_2 + \|y'_{i_2,k}\|_2 = 2\left\|\frac{y_{i_1,k} + y_{i_2,k}}{2}\right\|_2 \leq \|y_{i_1,k}\|_2 + \|y_{i_2,k}\|_2$$

and strict inequality holds when $y_{i_1,k} \neq y_{i_2,k}$. Since this is the case for at least one k and all machines $i \notin \{i_1, i_2\}$ do not change, we have that (2) has decreased. A contradiction. ◀

Using Lemma 3 we can solve the slot-MILP in polynomial time.

▶ **Lemma 4.** *We can solve the slot-MILP in time* $m^{O(2^{1/\epsilon})} \cdot n^{O(1/\epsilon \cdot 2^{1/\epsilon})}$.

Proof. Lemma 3 implies that there are only $2^{1/\epsilon}$ many machine types denoted by the $1/\epsilon$-tuples y_i. This allows us to guess all values of $y_{i,k}$ (up to permutations of machines) of the optimal solution as follows. For each of the $2^{1/\epsilon}$ types we guess (1) the number of machines

having this type and (2) for each $k \in \{1, ..., 1/\epsilon\}$ we guess the value of $y_{i,k}$ for each machine $i \in \mathcal{M}$ of this type. Note that the machines are identical and hence it suffices to guess the number of machines of each type, rather than guessing which exact machine is of which type. The total number of guesses is bounded by $m^{O(2^{1/\epsilon})} \cdot n^{O(1/\epsilon \cdot 2^{1/\epsilon})}$. Then the remaining problem is only a linear program since all integral variables of the slot-MILP are already fixed. If our guess was correct then the LP must have a feasible solution. ◀

2.2 Faster (approximate) solution to the relaxation

The solution based on Lemma 3 can be found in double exponential time with respect to the number of job types $1/\epsilon$ and is an exact solution to the slot-MILP. Using a different (slightly more complicated) structural property one can find an additive δ-approximate solution to the slot-MILP in single exponential time with respect to $1/\epsilon$ and polynomial in $1/\delta$, i.e., even with $\delta := 1/n^{O(1)}$ we obtain polynomial running time. Here, δ-approximate means that we find a solution to a weaker version of slot-MILP with lower and upper bounds $\ell' = \ell - \delta \cdot p_{\max}$ and $u' = u + \delta \cdot p_{\max}$ for the load of every machine i. We refer to this weaker MILP the slot-MILP'.

The algorithm is based on a different structural property than the one proved in Lemma 3. Given a solution (x, y) of the slot-MILP, for each machine $i \in \mathcal{M}$ and each $k \in \{1, \dots, 1/\epsilon\}$, we denote by $z_{i,k}$ the average size of the jobs type k on machine i defined by

$$z_{i,k} \cdot y_{i,k} = \sum_{j \in \mathcal{J}_k} p_j x_{i,j}.$$

In the case that $y_{i,k} = 0$ this allows us to freely choose the value of $z_{i,k}$ which is important for the structural property in the following lemma. We prove that there is always a solution to the slot-MILP and an ordering of the machines such that for each $k \in \{1, \dots, 1/\epsilon\}$ the values $z_{i,k}$ are non-decreasing and on each prefix of length ℓ of the machines the total size of the slots for the jobs in $\mathcal{J}_k$ is at least as large as the $y_{\sigma(1),k} + \cdots + y_{\sigma(\ell),k}$ smallest jobs in $\mathcal{J}_k$. For each integer n' let $\mathcal{J}_k^{\min}(n') \subseteq \mathcal{J}_k$ be the n' smallest jobs in $\mathcal{J}_k$.

▶ **Lemma 5.** *There is an optimal solution (x, y) for the slot-MILP, a corresponding vector $\{z_{i,k}\}_{i \in \mathcal{M}, k \in \{1,\dots,1/\epsilon\}}$, and an ordering $\sigma : \{1, ..., |\mathcal{M}|\} \to \mathcal{M}$ such that*

$$\sum_{\ell'=1}^{\ell} y_{\sigma(\ell'),k} z_{\sigma(\ell'),k} \geq \sum_{j \in \mathcal{J}_k^{\min}(y_{\sigma(1),k} + \cdots + y_{\sigma(\ell),k})} p_j \qquad \begin{matrix} \forall k \in \{1, \dots, 1/\epsilon\} \\ \forall \ell \in \{1, ..., |\mathcal{M}|\} \end{matrix} \tag{3}$$

$$\sum_{i \in \mathcal{M}} y_{i,k} z_{i,k} = \sum_{j \in \mathcal{J}_k} p_j \qquad \forall k \in \{1, \dots, 1/\epsilon\} \tag{4}$$

$$z_{\sigma(\ell),k} \leq z_{\sigma(\ell+1),k} \qquad \begin{matrix} \forall k \in \{1, \dots, 1/\epsilon\} \\ \forall \ell \in \{1, ..., |\mathcal{M}| - 1\}. \end{matrix} \tag{5}$$

Proof. Condition (3) and (4) follow directly from feasibility of the solution.

To show condition (5), let x, y be a solution with corresponding average load vector $\{z_{i,k}\}_{i \in \mathcal{M}, k \in 1,\dots,1/\epsilon\}}$, where the values $z_{i,k}$ when $y_{i,k} = 0$ are chosen appropriately. Let $\hat{z}_1 \leq \cdots \leq \hat{z}_{\bar{n}}$ be an ordering of the $\bar{n} = |\{(i,k) : y_{ik} > 0\}|$ values $z_{i,k}$ for all $i \in \mathcal{M}, k \in \{1, \dots, 1/\epsilon\}$ with $y_{ik} > 0$. Assume that (x, y) is the solution maximizing the following potential function

$$\sum_{i=1}^{\bar{n}} n^{2(m/\epsilon - i)} \hat{z}_i. \tag{6}$$

We will now show that in this case we can iteratively find an ordering of machines such that condition (5) holds and otherwise get a contradiction with respect to the potential function. Let i be the machine minimizing $\sum_{k=1}^{1/\epsilon} z_{i,k}$. All other machines i' must satisfy one of the following two cases: (1) $z_{i,k} \leq z_{i',k}$ for all $k \in \{1, ..., 1/\epsilon\}$ or (2) $z_{i,k} > z_{i',k}$ for some k. If (1) holds for all machines i' we relabel machine i as machine 1. Otherwise, let $i' \neq i$ be a machine such that for some k

$$z_{i,k} > z_{i',k}. \tag{7}$$

Then, as i minimizes $\sum_{k=1}^{1/\epsilon} z_{i,k}$ we know that there must exist $\overline{k} \neq k$ with

$$z_{i,\overline{k}} < z_{i',\overline{k}}. \tag{8}$$

As we can freely choose the value of $z_{\overline{i},k'}$, whenever $y_{\overline{i},k'} = 0$, we know that $y_{i,k}, y_{i,\overline{k}}, y_{i',k}, y_{i',\overline{k}} > 0$. We now gradually exchange jobs of $\mathcal{J}_k$ and $\mathcal{J}_{\overline{k}}$ between i and i' without changing the total load on either of the machines. Indeed, there must be some $j, j' \in \mathcal{J}_k$ with $x_{i,j} > 0$, $x_{i',j'} > 0$, and $p_j > p_{j'}$. Conversely, there are $\overline{j}, \overline{j}' \in \mathcal{J}_{\overline{k}}$ with $x_{i,\overline{j}} > 0$, $x_{i',\overline{j}'} > 0$, and $p_{\overline{j}} < p_{\overline{j}'}$. For some $\delta, \overline{\delta} > 0$ we now augment the solution in the following way.

$$\begin{aligned} x_{i,j'} &\leftarrow x_{i,j'} + \delta & \qquad x_{i,\overline{j}'} &\leftarrow x_{i,\overline{j}'} + \overline{\delta} \\ x_{i,j} &\leftarrow x_{i,j} - \delta & \qquad x_{i,\overline{j}} &\leftarrow x_{i,\overline{j}} - \overline{\delta} \\ x_{i',j'} &\leftarrow x_{i',j'} - \delta & \qquad x_{i',\overline{j}'} &\leftarrow x_{i',\overline{j}'} - \overline{\delta} \\ x_{i',j} &\leftarrow x_{i',j} + \delta & \qquad x_{i',\overline{j}} &\leftarrow x_{i',\overline{j}} + \overline{\delta} \end{aligned}$$

It is easy to see that for δ and $\overline{\delta}$ sufficiently small each variable remains non-negative. Moreover, each job remains fully assigned and the number of jobs of $\mathcal{J}_k$ and $\mathcal{J}_{\overline{k}}$ assigned to i and i' remains the same.

By setting $\overline{\delta} = \delta(p_j - p_{j'})/(p_{\overline{j}'} - p_{\overline{j}})$ the load over each of the two machines stays the same. Furthermore, as $p_j > p_{j'}$ and $p_{\overline{j}'} > p_{\overline{j}}$ we have that $\delta, \overline{\delta} > 0$. We choose δ maximal such that all x variables remain non-negative and the inequalities (7) and (8) still hold or turn to equality. This means that we decreased $z_{i,k}$ by $\frac{\delta(p_j - p'_j)}{y_{i,k}}$ and $z_{i',\overline{k}}$ by $\frac{\delta(p_j - p_{j'})}{y_{i,\overline{k}}}$. At the same time we increased $z_{i,\overline{k}}$ by $\frac{\delta(p_j - p_{j'})}{y_{i,\overline{k}}}$ and $z_{i',k}$ by $\frac{\delta(p_j - p'_j)}{y_{i',k}}$. Since $z_{i',k}$ and $z_{i,\overline{k}}$ (the respective smaller z-variables for i and i' that we change) increase by at least $\frac{\delta(p_j - p'_j)}{n}$ and $z_{i,k}$ and $z_{i',\overline{k}}$ decrease by at most $\delta(p_j - p'_j)$, we have that (6) increases. This gives a contradiction.

As we can repeat this argument iteratively assuming that machines $\{1, \ldots, i_0\}$ are correctly sorted for some $i_0 \in \{1, \ldots, m\}$, we have that there exists a solution (x, y) with vector $\{z_{i,k}\}_{i \in \mathcal{M}, k \in 1, \ldots, 1/\epsilon\}}$ such that condition (5) holds. ◀

We introduce a dynamic program that uses the property from Lemma 5. Intuitively, our DP guesses the machines in the ordering σ one after the other. When it guesses the next machine i, it guesses for each $k \in \{1, \ldots, 1/\epsilon\}$ the value $z_{i,k}$ and the number of jobs $y_{i,k}$ from $\mathcal{J}_k$ on machine i. In order to bound the running time we need to consider rounded values of $z_{i,k}$. Therefore, the DP ensures that the conditions (3) and (5) on the vectors y, z from Lemma 5 are satisfied and that condition (4) as well as the upper and lower target load bounds are only violated to a small extent. The following lemma shows that this is sufficient in order to compute an approximate solution to the slot-MILP based on the vectors y, z.

▶ **Lemma 6.** *Suppose that we are given an ordering* $\sigma : \{1, ..., |\mathcal{M}|\} \to \mathcal{M}$ *and vectors* $\{y_{i,k}, z_{i,k}\}_{i\in\mathcal{M},k\in\{1,...,1/\epsilon\}}$ *such that conditions* (3) *and* (5) *hold. Moreover, assume that for each* $i \in \mathcal{M}$ *it holds that*

$$\ell \leq \sum_{k=1}^{1/\epsilon} y_{i,k} z_{i,k} \leq u + \delta p_{\max} \tag{9}$$

and for each $k \in \{1, \cdots, 1/\epsilon\}$ *we have that condition* (4) *is slightly violated as follows*

$$\sum_{j\in\mathcal{J}_k} p_j \leq \sum_{i\in\mathcal{M}} y_{i,k} z_{i,k} \leq \sum_{j\in\mathcal{J}_k} p_j + \delta\epsilon \cdot p_{\max}. \tag{10}$$

Then we can compute a vector $\{x_{i,j}\}_{i\in\mathcal{M},j\in\mathcal{J}}$ *such that* (x, y) *is a solution to slot-MILP' in time* $O(mn^2)$.

Proof. We first show that there exists an assignment vector $\{x_{i,j}\}_{i\in\mathcal{M},j\in\mathcal{J}}$ satisfying

$$\sum_{j\in\mathcal{J}_k} p_j x_{i,j} \leq y_{i,k} z_{i,k} \tag{11}$$

for all $i \in \mathcal{M}$ and $k \in \{1, \ldots, 1/\epsilon\}$. In order to do so we use condition (3) of Lemma 5. We find this assignment independently for all k. We start by assigning $\mathcal{J}_k^{\min}(y_{1,k})$ (completely) to machine 1, then $\mathcal{J}_k^{\min}(y_{1,k} + y_{2,k}) \setminus \mathcal{J}_k^{\min}(y_{1,k})$ to machine 2, etc. This assignment does not necessary have the desired property (11). Hence, we repair the property iteratively for $i = 2, \ldots, m$. Machine 1 clearly satisfies (11) because of (3)). Let $i \in \{2, \ldots, m-1\}$ such that all machines $1, \ldots, i$ satisfy (11). In each iteration i we do not touch any of the machines $i+1, \ldots, m$. Hence, when repairing machine i we may assume that machines $1, \ldots, i$ contain only $\mathcal{J}_k^{\min}(y_{1,k} + \cdots + y_{i,k})$. If machine i satisfies (11) we are done and continue with $i+1$. Otherwise, we know that there is a job j with $p_j > z_{i,k}$ and $x_{i,j} > 0$. Moreover, because of condition (3) we have

$$\sum_{i'=1}^{i} \sum_{j\in\mathcal{J}_k^{\min}(y_{1,k}+\cdots+y_{i,k})} p_j x_{i',j} = \sum_{j\in\mathcal{J}_k^{\min}(y_{1,k}+\cdots+y_{i,k})} p_j \leq y_{1,k} z_{1,k} + \cdots + y_{i,k} z_{i,k}. \tag{12}$$

Since i violates (11) there must be some $i' < i$ satisfying (11) with strict inequality. In particular, there is a job j' with $p_{j'} < z_{i',k} \leq z_{i,k} < p_j$ and $x_{i',j'} > 0$. We now choose an $\alpha > 0$ and exchange j' and j between i and i' as follows

$$x_{i,j'} \leftarrow x_{i,j'} + \alpha \qquad x_{i',j'} \leftarrow x_{i',j'} - \alpha$$
$$x_{i,j} \leftarrow x_{i,j'} - \alpha \qquad x_{i',j'} \leftarrow x_{i',j'} + \alpha$$

Clearly, the solution remains feasible. We choose α maximal such that either i' satisfies (11) with equality, i satisfies (11), $x_{i,j} = 0$, or $x_{i',j'} = 0$. The choice of α makes sure that each pair j, j' that can be exchanged like this will only be exchanged once. This procedure is repeated until i satisfies (11). As the procedure is repeated for all i and possibly has to check all pairs of every job type in each exchange we have a running time of $O(mn^2)$.

Next, we claim that for all i and k, we have that (13) holds, that is,

$$\sum_{j\in\mathcal{J}_k} p_j x_{i,j} \geq y_{i,k} z_{i,k} - \delta\epsilon \cdot p_{\max}. \tag{13}$$

To prove this claim, assume by contradiction that for some machine i' (13) does not hold. Then by (11) and condition (4), we have that

$$\sum_{j\in\mathcal{J}_k}\sum_{i\in\mathcal{M}} x_{i,j}p_j < \sum_{i\in\mathcal{M}} y_{i,k}z_{i,k} - \delta\epsilon\cdot p_{\max} < \sum_{j\in\mathcal{J}_k} p_j. \tag{14}$$

This contradicts the fact that all jobs are fully assigned and thus $\sum_{j\in\mathcal{J}_k} p_j x_{i,j} = \sum_{j\in\mathcal{J}_k} p_j$. Hence, we have that

$$\sum_{k=1}^{1/\epsilon}\sum_{k\in\mathcal{J}_k} p_j x_{ij} \geq \sum_{k=1}^{1/\epsilon} y_{i,k}z_{i,k} - \delta p_{\max} \geq \ell - \delta p_{\max}, \tag{15}$$

where the last inequality follows by condition (9). ◀

The goal of our DP is to compute vectors $\{y_{i,k}, z_{i,k}\}_{i\in\mathcal{M},k\in\{1,\dots,1/\epsilon\}}$ that satisfy the conditions due to Lemma 6. The key insight is now that when we consider the next machine i' in the ordering, we do not need to remember all vectors $\{y_{i,k}, z_{i,k}\}_{i\in\mathcal{M},k\in\{1,\dots,1/\epsilon\}}$ for all previously considered machines i, but it suffices to remember the number of previously assigned jobs from each set $\mathcal{J}_k$, the current left hand side of inequality (3) for each k and the vector $\{z_{i'',k}\}_{k\in\{1,\dots,1/\epsilon\}}$ of the previously considered machine i''. At each iteration the DP then guesses the vectors $\{y_{i,k}, z_{i,k}\}_{i\in\mathcal{M},k\in\{1,\dots,1/\epsilon\}}$ such that the new solution consisting of the guess for machine i and the remembered solution for the previous machines satisfies the conditions stated in Lemma 7. If none of the guesses satisfies these conditions the DP cell corresponding to this iteration remains empty.

To give a more detailed description, we introduce a DP-table with one cell for each combination of

- a value $i \in \{0, ..., m\}$ indicating the number of machines that have already been considered,
- a vector $\{z_{i,k}\}_{k\in\{1,\dots,1/\epsilon\}}$ where $z_{i,k} \in \left\{0, \frac{\delta\epsilon}{n}p_{\max}, \frac{2\delta\epsilon}{n}p_{\max}, ..., p_{\max}\right\}$ for $k \in \{1, \dots, 1/\epsilon\}$. The vector $z_{i,k}$ corresponds to the average loads on the currently considered machine,
- a vector $\{y_{i,k}\}_{k\in\{1,\dots,1/\epsilon\}}$ where $y_{i,k} \in \{0, 1, 2, ..., n_k\}$ for each $k \in \{1, \dots, 1/\epsilon\}$. The vector $y_{i,k}$ corresponds to the number of jobs on the currently considered machine,
- for each $k \in \{1, ..., 1/\epsilon\}$
  - a value $n'_k \in \{0, ..., |\mathcal{J}_k|\}$ indicating the number of slots for jobs of $\mathcal{J}_k$ that have already been assigned to machines,
  - a value S_k with $S_k \in \left\{0, \frac{\delta\epsilon}{n}p_{\max}, \frac{2\delta\epsilon}{n}p_{\max}, ..., np_{\max}\right\}$ which corresponds to the value $\sum_{i'=1}^{i} y_{i',k}z_{i',k}$.

Each cell corresponds to the subproblem of checking whether there is a solution using machines $\{1, \dots, i\}$ such that machine i is assigned $y_{i,k}$ jobs of each type k with average load $z_{i,k}$ and for each type k a total number of n'_k slots is assigned of a total volume of S_k. Due to the dimension of the values corresponding to a DP-cell, the dimension of the DP table is given by $m^2 \cdot (\frac{n}{\delta\epsilon})^{O(1/\epsilon)}$.

When considering cell

$$\left(i, \{z_{i,k}\}_{k\in\{1,\dots,1/\epsilon\}}, \{y_{i,k}\}_{k\in\{1,\dots,1/\epsilon\}}, \{n'_k\}_{k\in\{1,\dots,1/\epsilon\}}, \{S_k\}_{k\in\{1,\dots,1/\epsilon\}}\right)$$

the DP checks whether for some $\{\tilde{z}_{i-1,k}\}_{k\in\{1,\dots,1/\epsilon\}}$ and $\{\tilde{y}_{i-1,k}\}_{k\in\{1,\dots,1/\epsilon\}}$ the entry of the DP is true in cell

$$\left(i-1, \{\tilde{z}_{i-1,k}\}_{k\in\{1,\dots,1/\epsilon\}}, \{\tilde{y}_{i-1,k}\}_{k\in\{1,\dots,1/\epsilon\}}, \{\tilde{n}'_k\}_{k\in\{1,\dots,1/\epsilon\}}, \left\{\tilde{S}_k\right\}_{k\in\{1,\dots,1/\epsilon\}}\right),$$

where $\tilde{n}'_k = n'_k - y_{i,k}$ for each $k \in \{1, \dots, 1/\epsilon\}$, and $\tilde{S}_k = S_k - y_{i,k} z_{i,k}$ for each $k \in \{1, \dots, 1/\epsilon\}$. Then we need to check if the following conditions are true for all $k \in \{1, ..., 1/\epsilon\}$:

$$S_k \leq \delta\epsilon \cdot p_{\max} + \sum_{j \in \mathcal{J}_k} p_j \tag{16}$$

$$z_{i,k} \geq \tilde{z}_{i-1,k}. \tag{17}$$

If these conditions are true, then there exists a solution corresponding to the considered DP cell. Filling each cell takes $(\frac{n}{\delta\epsilon})^{O(1/\epsilon)}$.

Finally, for each possible value of $\{z_{m,k}\}_{k \in \{1,\dots,1/\epsilon\}}$ we check whether there exists a solution for the DP cell

$$\left(m, \{z_{m,k}\}_{k \in \{1,\dots,1/\epsilon\}}, \{y_{m,k}\}_{k \in \{1,\dots,1/\epsilon\}}, \{n'_k\}_{k \in \{1,\dots,1/\epsilon\}}, \{S_k\}_{k \in \{1,\dots,1/\epsilon\}}\right),$$

where all jobs are assigned and for every $k \in \{1, \dots, 1/\epsilon\}$ we have that

$$\sum_{j \in \mathcal{J}_k} p_j \leq S_k \leq \sum_{j \in \mathcal{J}_k} p_j + \delta\epsilon \cdot p_{\max}.$$

If this is the case we use standard backward recursion to find vectors $\{z_{i,k}\}_{k \in \{1,\dots,1/\epsilon\}}$ and $\{y_{i,k}\}_{k \in \{1,\dots,1/\epsilon\}}$ for all $i \in \{1, ..., m\}$ and an assignment of machine types to machine indices and use Lemma 6 to obtain a solution (x, y) to slot-MILP'. If there is no such solution we assert that there is no solution to the original relaxation, i.e., to slot-MILP.

▶ **Lemma 7.** *For each $\delta > 0$ there is an algorithm with a running time of $m^2(\frac{n}{\delta\epsilon})^{O(1/\epsilon)}$ which either finds a δ-approximate solution to the slot-MILP (and thus a feasible solution to slot-MILP') or asserts that the slot-MILP is infeasible.*

Proof. The running time follows from the dimension of the DP table and the time it takes to validate a specific DP cell. This amounts to a running time of $m^2(\frac{n}{\delta\epsilon})^{O(1/\epsilon)}$.

For the correctness of the DP we need two observations: (1) due to conditions (16) and (17) and the way we check whether a solution corresponding to a DP cell exists we have that there exists a solution for machine i if and only if there is a solution for machine $i-1$. Hence, we can indeed find a solution via backward recursion and (2) if for some $\{z_{m,k}\}_{k \in \{1,\dots,1/\epsilon\}}$ and $\{y_{m,k}\}_{k \in \{1,\dots,1/\epsilon\}}$ there is a solution for the cell

$$\left(m, \{z_{m,k}\}_{k \in \{1,\dots,1/\epsilon\}}, \{y_{m,k}\}_{k \in \{1,\dots,1/\epsilon\}}, \{n'_k\}_{k \in \{1,\dots,1/\epsilon\}}, \{S_k\}_{k \in \{1,\dots,1/\epsilon\}}\right),$$

then we can apply Lemma 6 to find a solution to slot-MILP'. If there is no such solution, we know that due to our rounding of the z-values there is also no solution satisfying Lemma 5. This implies that there is no solution to the slot-MILP. ◀

3 Rounding the relaxation

We assume that we are given an exact solution to the slot-MILP via the algorithm due to Lemma 4 or an approximate solution via the algorithm due to Lemma 7. In this section, we describe an algorithm with a running time of $n^{O(1)}$ that computes an integral solution to the slot-MILP (or slot-MILP') which for each machine $i \in \mathcal{M}$ violates the target load intervals by at most $\epsilon \cdot p_{\max}$. For a solution to the slot-MILP this implies that $\sum_{j \in \mathcal{J}} p_j x_{i,j} \in$

$[\ell-\epsilon\cdot p_{\max}, u+\epsilon\cdot p_{\max}]$. For a solution to slot-MILP' this implies that the target load violation is given by the error made due to the approximate solution and due to the rounding, i.e., after rounding the solution it holds that $\sum_{j\in\mathcal{J}} p_j x_{i,j} \in [\ell-\delta\cdot p_{\max}-\epsilon\cdot p_{\max}, u+\delta\cdot p_{\max}+\epsilon\cdot p_{\max}]$. In the following we describe the rounding procedure based on an exact solution to the slot-MILP as the same arguments hold for an approximate solution.

We imagine that each machine $i \in \mathcal{M}$ has $y_{i,k}$ slots for the jobs in $\mathcal{J}_k$, for each $k \in \{1, ..., 1/\epsilon\}$. We say that these slots are of *type* k. Notice that $\sum_{i\in\mathcal{M}} y_{i,k} = |\mathcal{J}_k|$. We compute an initial solution by assigning each job $j \in \mathcal{J}_k$ to an arbitrary slot of type k. In this solution there might be a machine i whose load is not in $[\ell - \epsilon \cdot p_{\max}, u + \epsilon \cdot p_{\max}]$, i.e., the load is too small or too large. We present a local search algorithm that repeatedly swaps pairs of jobs from the same set $\mathcal{J}_k$ such that eventually each machine $i \in \mathcal{M}$ has a load in $[\ell - \epsilon \cdot p_{\max}, u + \epsilon \cdot p_{\max}]$ while maintaining the number of jobs from each set $\mathcal{J}_k$ on each machine.

3.1 Local search

We describe how to perform one iteration of the local search algorithm. Each iteration aims at finding a pair of jobs that can be swapped. Let $\mathcal{M}_1$ be the set of machines $i \in \mathcal{M}$ that have a load strictly greater than $u + \epsilon \cdot p_{\max}$. Consider a $k \in \{1, ..., 1/\epsilon\}$ such that a job $j \in \mathcal{J}_k$ is assigned to a machine $i \in \mathcal{M}_1$. We would like to exchange j for a smaller job $j' \in \mathcal{J}_k$ that is assigned to a machine $i' \notin \mathcal{M}_1$. Thus, consider all jobs $j' \in \mathcal{J}_k$ with $p_{j'} < p_j$ which are assigned to a machine $i' \notin \mathcal{M}_1$. If the load of i' is at most u then we exchange j and j' which completes the swap. We try to perform such a swap for each $k \in \{1, ..., 1/\epsilon\}$ such that a job $j \in \mathcal{J}_k$ is assigned to a machine in $\mathcal{M}_1$. If we did not perform a swap then let $\mathcal{M}_2$ denote the set of machines $i' \in \mathcal{M}$ having a job j' which we tried to swap with a job j on a machine $i \in \mathcal{M}_1$, i.e., $\mathcal{M}_2$ contains all machines $i' \in \mathcal{M} \setminus \mathcal{M}_1$ for which there exists a machine $i \in \mathcal{M}_1$ and a $k \in \{1, ..., 1/\epsilon\}$ such that there is a job $j \in \mathcal{J}_k$ assigned to i and a job $j' \in \mathcal{J}_k$ assigned to i' with $p_{j'} < p_j$. Observe that each machine $i' \in \mathcal{M}_2$ has a load of more than u.

Now we repeat this procedure: Suppose that we constructed sets of machines $\mathcal{M}_1, ..., \mathcal{M}_\ell$. For each $k \in \{1, ..., 1/\epsilon\}$ such that there is a job $j \in \mathcal{J}_k$ assigned to a machine $i \in \mathcal{M}_\ell$ consider all jobs $j' \in \mathcal{J}_k$ with $p_{j'} < p_j$, which are assigned to a machine $i' \notin \mathcal{M}_1 \cup \ldots \cup \mathcal{M}_\ell$. If the load on one such machine i' is at most u, then we exchange j and j' which completes the swap. In particular, we do not reuse the constructed sets $\mathcal{M}_1, \ldots, \mathcal{M}_\ell$ for the next swap but we forget these sets before the next swap starts. Otherwise, if each considered machine i' has a load strictly more than u we construct a set $\mathcal{M}_{\ell+1}$ consisting of all these machines i' and continue in the current iteration.

Suppose that at the beginning of a swap there is no machine $i \in \mathcal{M}$ that has a load strictly greater than $u + \epsilon \cdot p_{\max}$. Then, a second stage of the local search algorithm takes place. We take the current solution and perform an analogous procedure in order to ensure that each machine $i \in \mathcal{M}$ has a load of at least $\ell - \epsilon \cdot p_{\max}$. Initially define $\mathcal{M}_1$ to be the set of all machines $i \in \mathcal{M}$ with a load strictly less than $\ell - \epsilon \cdot p_{\max}$. Suppose that we constructed sets of machines $\mathcal{M}_1, ..., \mathcal{M}_\ell$. For each $k \in \{1, ..., 1/\epsilon\}$ such that there is a job $j \in \mathcal{J}_k$ assigned to a machine $i \in \mathcal{M}_\ell$ consider all jobs $j' \in \mathcal{J}_k$ with $p_{j'} > p_j$, which are assigned to a machine $i' \notin \mathcal{M}_1 \cup \ldots \cup \mathcal{M}_\ell$. If the load on one such machine i' is at least ℓ, then we exchange j and j' which completes the swap. Otherwise, if each such machine i' has a load of strictly less than ℓ we construct a set $\mathcal{M}_{\ell+1}$ consisting of all these machines i' and continue. As we never increase the load of a machine with load at least $\ell - \epsilon \cdot p_{\max}$ we do not introduce new violations of the upper bound on the load. The algorithm terminates if the load of each machine $i \in \mathcal{M}$ is within $[\ell - \epsilon \cdot p_{\max}, u + \epsilon \cdot p_{\max}]$.

3.2 Correctness and running time

We show now that the algorithm terminates in $n^{O(1)}$ time. Then, by construction it outputs a solution in which each machine $i \in \mathcal{M}$ has a load in the interval $[\ell - \epsilon \cdot p_{\max}, u + \epsilon \cdot p_{\max}]$. In the following we prove this for the first stage of the local search, i.e., when at least one machine has as a load higher than $u + \epsilon p_{\max}$. We first show that in each iteration of the first stage we can find a pair of jobs j, j' to swap. Using a similar argument the same can be shown for the second stage.

▶ **Lemma 8.** *In each iteration the algorithm finds two jobs j, j' that it swaps and finding such a pair can be done in time $O(n^2)$.*

Proof. Suppose towards contradiction that the algorithm does not find two jobs $j, j' \in \mathcal{J}_k$ such that j is assigned to a machine $i \in \mathcal{M}_\ell$ with load more than $u + \epsilon \cdot p_{\max}$ and job j' is assigned to a machine $i' \notin \mathcal{M}_1 \cup \ldots \cup \mathcal{M}_\ell$ and $p'_j < p_j$. This means that the machines in $\mathcal{M}_1 \cup \ldots \cup \mathcal{M}_\ell$ are assigned the smallest jobs of type k while each machine having load at least u. Hence, even a fractional assignment of jobs cannot reduce the total load on these machines. Hence, in a fractional assignment at least one machine must have a load greater than u. This gives a contradiction. When finding a pair of jobs to swap each pair of type k is considered exactly once as each job is assigned fully to a machine. As the existence of a pair was shown for an arbitrary k this gives a worst case running time of $O(n^2)$. ◀

Next, we show that the first stage of the algorithm always terminates after at most $O(n^3)$ swaps. As Lemma 8 states that each swap can be done in time $O(n^2)$, this shows that the first stage finishes in time $n^{O(1)}$. To this end, we give an alternative formulation of the first stage of the algorithm as a repeated breadth-first search (BFS). We construct a weighted, directed graph. It contains one special vertex, the source s, and one vertex for each slot, that is $|\mathcal{J}| + 1$ vertices in total. Each non-source vertex is associated with a machine and a size class. The slots of the same machine form a clique: There is an edge from each slot to the other with weight 0. Furthermore, there is an edge of weight 1 from slot v to w, when (1) v and w are not on the same machine, (2) v and w belong to the same size class, and (3) v is currently assigned a larger job than w. Additionally, there is an edge of weight 0 from the source to every slot on a machine with load more than $u + \epsilon \cdot p_{\max}$. The algorithm performs a BFS on the graph above starting in s. Once it reaches a machine with load at most u, it selects the edge (v, w) over which the machine was reached and swaps the jobs assigned to the slots v and w. This is continued until every machine i is assigned a load at most $u + \epsilon \cdot p_{\max}$. The following lemma shows that the distance between s and other slots does not decrease by a swap.

▶ **Lemma 9.** *The distance from s to any slot does not decrease by a swap in the first stage of the algorithm.*

Proof. Note that when we make a swap, we actually reverse the direction of an edge. To get the main idea of the proof, consider an edge (v, w) that is "swapped". As the search is a BFS, we know that the distance in the original graph to w is equal to the shortest path to v plus 1. When we make the swap, this shortest path is eliminated, whereas all other paths from s to w remain. Therefore, the distance from s to w will not decrease. Furthermore, the distance from s to v did not change due to the swap.

Formally, we observe how the graph changes when swapping two jobs. Throughout the proof the distance $d(w)$ of a slot w is the distance from s to w before the swap is executed. Clearly, removing any edges from the graph cannot decrease the distances of vertices. Edges

between slots of the same machine do not change and no new edges can be added from the source, since during the execution of the algorithm a machine with load at most $u + \epsilon \cdot p_{\max}$ will never exceed $u + \epsilon \cdot p_{\max}$. Hence, it suffices to look at the changes in edges of weight 1. Although such an edge (v, w) might be added to the graph, we will show that this happens only when $d(w) \leq d(v) + 1$. Adding this edges cannot decrease any distances, since the first part of a shortest path using (v, w) could always be replaced by a path to w without this edge.

Now we have to check that these are the only changes made to the graph. Let v, w be the slots in which we exchange the jobs. The size of the job in v decreases; the size of the job in w increases. We only need to look at the incoming and outgoing edges of v and w, since all other edges remain the same.

Consider the incoming edges of v. Since the size of the job in v decreases, there could be new incoming edges. Let (w', v) be an edge of weight 1 that is added. This means the job in slot w' has a larger size than the job on v. Either w' is a slot on the same machine as w or (w', w) is in the graph before the swap. The former case implies that $d(w') = d(w) = d(v)+1$. In the latter case we have $d(v) = d(w) - 1 \leq d(w')$. No outgoing edge from v can be added, since the size of v's job decreases.

Now consider w. Since the size of its job increases, no incoming edge can be added. As for the outgoing edges, let (w, w') be an outgoing edge added by the swap. Then either v and w' are slots on the same machine or (v, w') was is in the graph before the swap. In the former case, $d(w') = d(v) = d(w) - 1$. In the latter case, $d(w') \leq d(v) + 1 = d(w)$. ◀

Using Lemmas 8 and 10 we can bound the maximum number of swaps necessary in the first stage of the algorithm.

▶ **Lemma 10.** *The first stage of the algorithm terminates after at most $O(n^3)$ swaps.*

Proof. Let $j_1, \ldots, j_n$ be the jobs $\mathcal{J}$ in increasing order of size. We claim that the potential

$$\sum_{i=1}^{n} i \cdot d(j_i)$$

increases with every swap. Here $d(j_i)$ denotes the distance from s to the slot to which j_i is assigned. Since the function is integral and bounded by n^3, the claim follows. Let j_k, j_h be the jobs that are swapped. Assume that $k < h$, i.e., $p_{j_k} < p_{j_h}$. Let d, d' be the distance functions before and after the swap. Based on Lemma 9 we have $d'(j_i) \geq d(j_i)$ for all $i \notin \{k, h\}$, $d'(j_h) \geq d(j_k)$ and $d'(j_k) \geq d(j_h)$, since these jobs swapped their slots. It follows that

$$\begin{aligned} k \cdot d'(j_k) + h \cdot d'(j_h) &\geq k \cdot d(j_h) + h \cdot d(j_k) \\ &= k \cdot d(j_h) + (h - k) \cdot \underbrace{d(j_k)}_{> d(j_h)} + k \cdot d(j_k) > h \cdot d(j_h) + k \cdot d(j_k). \end{aligned}$$ ◀

In order to complete the running time analysis we consider the second stage. Again, we construct a graph on $|\mathcal{J}| + 1$ vertices with the difference that we add an edge from s to every slot on a machine with load less than $\ell - \epsilon p_{\max}$ and an edge (v, w) between two slots of the same job type associated to different machines if v is currently assigned a smaller job than w. Following the same ideas as the proofs for the first stage, the running time of the second stage can be shown. For the detailed proofs for the second stage of the algorithm we refer to [6].

▶ **Lemma 11.** *The distance from s to any slot does not decrease by a swap in the second stage of the algorithm.*

▶ **Lemma 12.** *The second stage of the algorithm terminates after at most $O(n^3)$ swaps.*

Together, Lemmas 10 and 12 give the running time of the local search procedure.

▶ **Lemma 13.** *Given a solution to the slot-MILP, in time $n^{O(1)}$ we can compute an integral solution to slot-MILP such that $\sum_{j\in\mathcal{J}} p_j x_{i,j} \in [\ell - \epsilon \cdot p_{\max}, u + \epsilon \cdot p_{\max}]$ for each machine $i \in \mathcal{M}$.*

The main theorem follows from Lemmas 7 and 13.

▶ **Theorem 14.** *There is an algorithm for the target load balancing problem with a running time of $m^2 n^{O(1/\epsilon)}$ that computes a solution in which the load of each machine $i \in \mathcal{M}$ is in $[\ell - \epsilon \cdot p_{\max}, u + \epsilon \cdot p_{\max}]$, or asserts that there is no feasible solution.*

4 Applications

We can use Theorem 14 to obtain additive approximation schemes for *makespan minimization*, the *Santa Claus* problem and the *envy-minimizing Santa Claus* problem on identical machines. The idea is to guess the target load intervals up to multiples of $\epsilon \cdot p_{\max}$ and then applying the algorithm due to Theorem 14.

For $P||C_{\max}$ and the *Santa Claus* problem the guessing procedure can be done in time $O(1/\epsilon)$. For $P||C_{\max}$ we set $\ell = 0$ and guess u as a multiple of $\epsilon \cdot p_{\max}$ within the interval $[\frac{1}{m}\sum_{j=1}^{n} p_j, \frac{1}{m}\sum_{j=1}^{n} p_j + p_{\max}]$.

▶ **Corollary 15.** *There is an algorithm for $P||C_{\max}$ with a running time of $m^2 n^{O(1/\epsilon)}$ that computes a solution with makespan at most $OPT + \epsilon \cdot p_{\max}$.*

For the *Santa Claus* problem we set $u = \sum_{j=1}^{n} p_j$ and guess ℓ within the interval $[\frac{1}{m}\sum_{j=1}^{n} p_j - p_{\max}, \frac{1}{m}\sum_{j=1}^{n} p_j]$.

▶ **Corollary 16.** *There is an algorithm for the Santa Claus problem on identical machines with a running time of $m^2 n^{O(1/\epsilon)}$ that computes a solution in which each machine has a load of at least $OPT - \epsilon \cdot p_{\max}$.*

For the *envy-minimizing* Santa Claus problem we need to guess both ℓ and u simultaneously from the intervals above with a total number of $O(1/\epsilon^2)$ guesses.

▶ **Corollary 17.** *There is an algorithm for envy-minimization on identical machines with a running time of $m^2 n^{O(1/\epsilon)}$ which computes a solution with envy at most $OPT + \epsilon \cdot p_{\max}$.*

5 Conclusion

In this paper, we formalized the concept of additive approximation schemes and presented such algorithms for makespan minimization, the Santa Claus problem, and minimizing the maximum envy on identical machines with respect to the parameter $p_{\max}$. For the former two problems, additive approximation schemes are particularly interesting when $p_{\max} \ll \mathrm{OPT}$. For the latter, a multiplicative approximation guarantee is not possible for polynomial time algorithms (unless P=NP); therefore, an additive approximation scheme is a suitable alternative way to obtain non-trivial approximation guarantees.

For $P||C_{\max}$ and the Santa Claus problem on identical machines there is an EPTAS [2,21]. We leave as an open question to find additive approximation schemes for these problems with a running time of this form or to rule out that such schemes exists. Note that using similar techniques as in [34], one can easily show that all three versions admit even an additive FPTAS in case that m is a constant.

References

1 N. Alon, Y. Azar, G. J. Woeginger, and T. Yadid. Approximation schemes for scheduling on parallel machines. *Journal of Scheduling*, 1:55–66, 1998.

2 N. Alon, A. Shapira, and B. Sudakov. Additive approximation for edge-deletion problems. *Annals of Mathematics*, 170:37–411, 2009.

3 C. Annamalai, C. Kalaitzis, and O. Svensson. Combinatorial algorithm for restricted max-min fair allocation. *ACM Transactions on Algorithms*, 13(3):37:1–37:28, 2017.

4 A. Asadpour, U. Feige, and A. Saberi. Santa claus meets hypergraph matchings. *ACM Transactions on Algorithms*, 8:24:1–24:9, 2012.

5 N. Bansal and M. Sviridenko. The santa claus problem. In *Proceedings of the 38th Annual ACM Symposium on Theory of Computing, STOC*, pages 31–40, 2006.

6 M. Buchem, L. Rohwedder, T. Vredeveld, and A. Wiese. Additive approximation schemes for load balancing problems. *CoRR*, abs/2007.09333, 2020. `arXiv:2007.09333`.

7 D. Chakrabarty. Max-min allocation. In M.-Y. Kao, editor, *Encyclopedia of Algorithms*, pages 1244–1247. Springer US, Boston, MA, 2016.

8 L. Chen, K. Jansen, and G. Zhang. On the optimality of exact and approximation algorithms for scheduling problems. *Journal of Computer and System Sciences*, 96:1–32, 2018.

9 S.-W. Cheng and Y. Mao. Restricted max-min allocation: Approximation and integrality gap. In *Proceedings of the 46th International Colloquium on Automata, Languages, and Programming, ICALP*, pages 38:1–38:13, 2019.

10 S. Davies, T. Rothvoss, and Y. Zhang. A tale of santa claus, hypergraphs and matroids. In *Proceedings of the 2020 ACM-SIAM Symposium on Discrete Algorithms, SODA*, pages 2748–2757, 2020.

11 F. Eisenbrand and G. Shmonin. Carathéodory bounds for integer cones. *Operations Research Letters*, 34(5):564–568, 2006.

12 U. Feige. On allocations that maximize fairness. In *Proceedings of the 19th Annual ACM-SIAM Symposium on Discrete Algorithms, SODA*, pages 287–293. SIAM, 2008.

13 M. R. Garey and D. S. Johnson. "strong" NP-completeness results: motivation, examples, and implications. *Journal of the ACM*, 25:499–508, 1978.

14 R. L. Graham. Bounds for certain multiprocessing anomalies. *Bell System Technical Journal*, 45:1563–1581, 1966.

15 R. L. Graham. Bounds on multiprocessing timing anomalies. *SIAM Journal on Applied Mathematics*, 17:416–429, 1969.

16 B. Haeupler, B. Saha, and A. Srinivasan. New constructive aspects of the lovász local lemma. *Journal of the ACM*, 58, 2011.

17 R. Hoberg and T. Rothvoss. A logarithmic additive integrality gap for bin packing. In *Proceedings of the 28th Annual ACM-SIAM Symposium on Discrete Algorithms, SODA*, pages 2616–2625. SIAM, 2017.

18 D. S. Hochbaum. Various notions of approximations: Good, better, best and more. *Approximation algorithms for NP-hard problems*, 1997.

19 D. S. Hochbaum and D. B. Shmoys. Using dual approximation algorithms for scheduling problems: Theoretical and practical results. *Journal of the ACM*, 34:144–162, 1987.

20 K. Jansen. An EPTAS for scheduling jobs on uniform processors: Using an MILP relaxation with a constant number of integral variables. *SIAM Journal on Discrete Mathematics*, 24:457–485, 2010.

21 K. Jansen, K.-M. Klein, and J. Verschae. Closing the gap for makespan scheduling via sparsification techniques. *Mathematics of Operations Research*, 45:1371–1392, 2020.

22 K. Jansen, S. Kratsch, D. Marx, and I. Schlotter. Bin packing with fixed number of bins revisited. *Journal of Computer and System Sciences*, 79:39–49, 2013.

23 K. Jansen and L. Rohwedder. On the configuration-lp of the restricted assignment problem. In *Proceedings of the 28th Annual ACM-SIAM Symposium on Discrete Algorithms, SODA*, pages 2670–2678, 2017.

24 K. Jansen and L. Rohwedder. A quasi-polynomial approximation for the restricted assignment problem. In *Proceedings of the 19th International Conference on Integer Programming and Combinatorial Optimization, IPCO*, pages 305–316, 2017.

25 N. Karmarkar and R. M. Karp. An efficient approximation scheme for the one-dimensional bin-packing problem. In *Proceedings of the 23rd Annual Symposium on Foundations of Computer Science, FOCS*, pages 312–320. IEEE Computer Society, 1982.

26 D. Knop and M. Koutecký. Scheduling meets n-fold integer programming. *Journal of Scheduling*, 21(5):493–503, 2018.

27 D. Knop, M. Koutecký, and M. Mnich. Combinatorial n-fold integer programming and applications. *Mathematical Programming*, 184(1):1–34, 2020.

28 A. Kurpisz, M. Mastrolilli, C. Mathieu, T. Mömke, V. Verdugo, and A. Wiese. Semidefinite and linear programming integrality gaps for scheduling identical machines. *Math. Program.*, 172(1-2):231–248, 2018.

29 J. K. Lenstra, D. B. Shmoys, and E. Tardos. Approximation algorithms for scheduling unrelated parallel machines. *Mathematical Programming*, 46:259–271, 1990.

30 R. J. Lipton, E. Markakis, E. Mossel, and A. Saberi. On approximately fair allocations of indivisible goods. In *Proceedings of the 5th ACM Conference on Electronic Commerce, EC*, pages 125–131, 2004.

31 M. Mnich and A. Wiese. Scheduling and fixed-parameter tractability. *Mathematical Programming*, 154(1-2):533–562, 2015.

32 T. Ophelders, R. Lambers, F. C. R. Spieksma, and T. Vredeveld. A note on equitable hamiltonian cycles. *Discrete Applied Mathematics*, page forthcoming, 2021.

33 T. Rothvoss. Better bin packing approximations via discrepancy theory. *SIAM Journal on Computing*, 45(3):930–946, 2016.

34 S. K. Sahni. Algorithms for scheduling independent tasks. *Journal of the ACM*, 23(1):116–127, 1976.

35 O. Svensson. Santa claus schedules jobs on unrelated machines. *SIAM Journal on Computing*, 41(5):1318–1341, 2012.

36 V. G. Vizing. On an estimate of the chromatic class of a p-graph (in russian). *Diskret. Analiz*, 3:25–30, 1964.

37 G. J. Woeginger. A polynomial-time approximation scheme for maximizing the minimum machine completion time. *Operations Research Letters*, 20:149–154, 1997.

Genome Assembly, from Practice to Theory: Safe, Complete and Linear-Time

Massimo Cairo
Department of Computer Science, University of Helsinki, Finland

Romeo Rizzi
Department of Computer Science, University of Verona, Italy

Alexandru I. Tomescu
Department of Computer Science, University of Helsinki, Finland

Elia C. Zirondelli
Department of Mathematics, University of Trento, Italy
Department of Computer Science, University of Verona, Italy

Abstract

Genome assembly asks to reconstruct an unknown string from many shorter substrings of it. Even though it is one of the key problems in Bioinformatics, it is generally lacking major theoretical advances. Its hardness stems both from practical issues (size and errors of real data), and from the fact that problem formulations inherently admit multiple solutions. Given these, at their core, most state-of-the-art assemblers are based on finding non-branching paths (*unitigs*) in an assembly graph. While such paths constitute only partial assemblies, they are likely to be correct. More precisely, if one defines a genome assembly solution as a *closed arc-covering walk* of the graph, then unitigs appear in all solutions, being thus *safe* partial solutions. Until recently, it was open what are *all* the safe walks of an assembly graph. Tomescu and Medvedev (RECOMB 2016) characterized all such safe walks (*omnitigs*), thus giving the first safe and *complete* genome assembly algorithm. Even though omnitig finding was later improved to quadratic time, it remained open whether the crucial linear-time feature of finding unitigs can be attained with omnitigs.

We answer this question affirmatively, by describing a surprising $O(m)$-time algorithm to *identify* all maximal omnitigs of a graph with n nodes and m arcs, notwithstanding the existence of families of graphs with $\Theta(mn)$ total maximal omnitig size. This is based on the discovery of a family of walks (*macrotigs*) with the property that all the non-trivial omnitigs are univocal extensions of subwalks of a macrotig. This has two consequences: (1) A *linear-time output-sensitive* algorithm enumerating all maximal omnitigs. (2) A *compact* $O(m)$ *representation* of all maximal omnitigs, which allows, e.g., for $O(m)$-time computation of various statistics on them. Our results close a long-standing theoretical question inspired by practical genome assemblers, originating with the use of unitigs in 1995. We envision our results to be at the core of a reverse transfer from theory to practical and *complete* genome assembly programs, as has been the case for other key Bioinformatics problems.

2012 ACM Subject Classification Mathematics of computing → Paths and connectivity problems; Theory of computation → Graph algorithms analysis; Applied computing → Computational biology

Keywords and phrases Graph algorithm, strong connectivity, reachability under failures

Digital Object Identifier 10.4230/LIPIcs.ICALP.2021.43

Category Track A: Algorithms, Complexity and Games

Related Version *Full Version*: `https://arxiv.org/abs/2002.10498` [12]

Funding This work was partially funded by the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreement No. 851093, SAFEBIO) and by the Academy of Finland (grants No. 322595, 328877).

Acknowledgements We thank Sebastian Schmidt for useful comments, including the observation that the bound on the total length of all maximal macrotigs can be improved to $O(n)$ (from $O(m)$ initially), Shahbaz Khan and Bastien Cazaux for helpful discussions and comments.

1 Introduction

Theoretical and practical background of genome assembly. Genome assembly is one of the flagship problems in Bioinformatics, along with other problems originating in – or highly motivated by – this field, such as edit distance computation, reconstructing and comparing phylogenetic trees, text indexing and compression. In genome assembly, we are given a collection of strings (or *reads*) and we need to reconstruct the unknown string (the genome) from which they originate. This is motivated by sequencing technologies that are able to read either "short" strings (100-250 length, Illumina technology), or "long" strings (10.000-50.000 length, Pacific Biosciences or Oxford Nanopore technologies) in huge amounts from the genomic sequence(s) in a sample. For example, the SARS-CoV-2 genome was obtained in [58] from short reads using the MEGAHIT assembler [39].

Other leading Bioinformatics problems have seen significant theoretical progress in major Computer Science venues, culminating (just to name a few) with both positive results, see e.g. [17, 57] for phylogeny problems, [22, 6, 34] for text indexing, [23, 7, 35] for text compression, and negative results, see e.g. [4, 1, 5, 21] for string matching problems. However, the genome assembly problem is generally lacking major theoretical advances.

One reason for this stems from practice: the huge amount of data (e.g. the 3.1 Billion characters long human genome is read 50 times over) which impedes slower than linear-time algorithms, errors of the sequencing technologies (up to 15% for long reads), and various biases when reading certain genomic regions [47]. Another reason stems from theory: historically, finding an optimal genome assembly solution is considered NP-hard under several formulations [49, 33, 32, 43, 46, 29, 48], but, more fundamentally, even if one outputs a 3.1 Billion characters long string, this is likely incorrect, since problem formulations inherently admit a large number of solutions of such length [36].

Given all these setbacks, most state-of-the-art assemblers, including e.g. MEGAHIT [39] (for short reads), or wtdbg2 [52] (for long reads), generally employ a very simple and *linear-time* strategy, dating back to 1995 [32]. They start by building an assembly graph encoding the overlaps of the reads, such as a *de Bruijn graph* [50] or an *overlap graph* [45] (graphs are directed in this paper). After some simplifications to this graph to remove practical artifacts such as errors, at their core they find strings labeling paths whose internal nodes have in-degree and out-degree equal to 1 (called *unitigs*), approach dating back to 1995 [32]. That is, they do not output *entire* genome assemblies, but only shorter strings that are likely to be present in the sequenced genome, since unitigs do not branch at internal nodes.

Safe and complete algorithms: A theoretical framing of practical genome assembly. With the aim of enhancing the widely-used practical approach of assembling just unitigs – as those walks considered to be present in any possible assembly solution – a result in a major Bioinformatics venue [55] asked *what is the "limit" of the correctly reconstructible information from an assembly graph.* Moreover, is all such reconstructible information still obtainable in *linear time*, as in the case of the popular unitigs? Variants of this question also appeared in [27, 8, 46, 53, 37, 9], while other works already considered simple linear-time generalizations of unitigs [51, 44, 30, 36], without knowing if the "assembly limit" is reached.

To make this question precise, [55] introduced the following *safe and complete* framework. Given a notion of solution to a problem (e.g. a type of walk in a graph), a partial solution (e.g. some shorter walk in the graph) is called *safe* if it appears (e.g. is a subwalk) in all solutions. An algorithm reporting only safe partial solutions is called a *safe algorithm*. A safe algorithm reporting *all* safe partial solutions is called *safe and complete*. A safe and complete algorithm outputs all and only what is likely part of the unknown object to be reconstructed, *synthesizing all solutions from the point of view of correctness.* Safety generalizes the existing notion of *persistency*: a *single* node or edge was called *persistent* if it appears in all solutions [28, 16, 13], for example persistent edges for maximum bipartite matchings [16]. It also has roots in other Bioinformatics works [56, 14, 24, 59] considering the aligned symbols appearing in all optimal (and sub-optimal) alignments of two strings.

There are many theoretical formulations of genome assembly as an optimization problem, e.g. a shortest common superstring of all the reads [49, 33, 32], or some type of shortest walk covering all nodes or arcs of the assembly graph [51, 43, 44, 31, 29, 48, 46]. However, it is widely acknowledged [46, 48, 42, 47, 41, 36] that, apart from some being NP-hard, these formulations are lacking in several aspects, for example they collapse repeated regions of a genome. At present, given the complexity of the problem, there is no definitive notion of a "good" genome assembly solution. Therefore, [55] considered as genome assembly solution *any* closed arc-covering walk of a graph, where *arc-covering* means that it passes through each arc *at least* once. The main benefit of considering *any* arc-covering walk is that safe walks for them are safe also for any possible restriction of such covering walks (e.g. by some additional optimality criterion[1]). Put otherwise, safe walks for *all* arc-covering walks are more likely to be correct than safe walks for some peculiar type of arc-covering walks.

Prior results on safety in closed arc-covering walks. It is immediate to see that unitigs are safe walks for closed arc-covering walks. A first safe generalization of unitigs consisted of those paths whose internal nodes have only out-degree equal to 1 (with no restriction on their in-degree) [51]. Further, these safe paths have been generalized in [44, 30, 36] to those partitionable into a prefix whose nodes have in-degree equal to 1, and a suffix whose nodes have out-degree equal to 1. *All* safe walks for closed arc-covering walks were characterized by [55, 54] as being exactly those that are *omnitigs*, see Definition 1, Figure 1, and Theorem 8. This leads to the first *safe and complete* genome assembly algorithm (obtained thus 20 years after unitigs were first considered), outputting all *maximal* omnitigs in polynomial time (maximal omnitigs are those which are not sub-walks of other omnitigs).

▶ **Definition 1** (Omnitig). *Let $W = e_0 \dots e_\ell$ be a walk. We say that a non-empty path P is a j-i* forbidden path *for W, for some $1 \leq i \leq j \leq \ell$, if the first arc of P has the same tail as e_j and is different from e_j, and the last arc of P has the same head as e_{i-1} and is different from e_{i-1}. We say that W is an* omnitig *if for no $1 \leq i \leq j \leq \ell$ there exists a j-i forbidden path for W.*

Furthermore, through experiments on "perfect" human read datasets, [55] also showed that strings labeling omnitigs are about 60% longer on average than unitigs, and contain about 60% more biological content on average. Thus, once other issues of real data (e.g. errors)

[1] For example, closed arc-covering walks are a common relaxation of the fundamental notions of closed Eulerian walk (we now pass through each arc *at least once*), and of closed Chinese postman walk (i.e. a closed arc-covering walk *of minimum length*) [26], which were mentioned in [46] as unsatisfactory models of genome assembly.

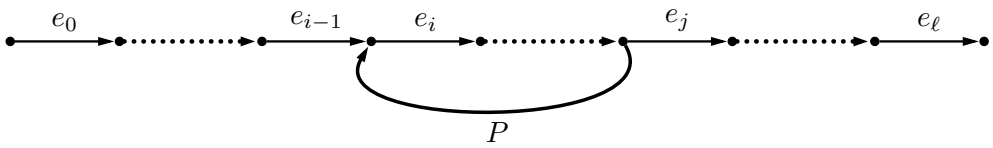


Figure 1 Walk $e_0 \dots e_\ell$ is *not* an omnitig because there is a forbidden path P.

are added to the problem formulation, omnitigs (and the safe walks for such extended models) have the potential to significantly improve the quality of genome assembly results. Nevertheless, for this to be possible, one first needs the best possible results for omnitigs (given e.g. the sheer size of the read datasets), and a full comprehension of them, otherwise, such extensions are hard to solve efficiently.

Cairo et al. [11] recently proved that the length of all maximal omnitigs of any graph with n nodes and m arcs is $O(nm)$, and proposed an $O(nm)$-time algorithm enumerating all maximal omnitigs. This was also proven to be optimal, in the sense that they constructed families of graphs where the total length of all maximal omnitigs is $\Theta(nm)$. However, it was left open if it is necessary to pay $O(nm)$ even when the total length of the output is smaller. Moreover, that algorithm cannot break this barrier, because e.g. $O(m)$-time traversals have to be done for $O(n)$ cases.

Our results. Our main result is an $O(m)$-size representation of all maximal omnitigs[2], based on a careful structural decomposition of the omnitigs of a graph. This is surprising, given that there are families of graphs with $\Theta(nm)$ total length of maximal omnitigs [11].

▶ **Theorem 2.** *Given a strongly connected graph G with n nodes and m arcs, there exists a $O(m)$-size representation of all maximal omnitigs, consisting of a set $\mathcal{M}$ of walks (*maximal macrotigs*) of total length $O(n)$ and a set $\mathcal{F}$ of arcs, such that every maximal omnitig is the univocal extension[3] of either a subwalk of a walk in $\mathcal{M}$, or of an arc in $\mathcal{F}$.*

Moreover, $\mathcal{M}$, $\mathcal{F}$, and the endpoints of macrotig subwalks univocally extending to maximal omnitigs can be computed in time $O(m)$.

Since the univocal extension $U(W)$ of a walk W can be trivially computed in time linear in the length of $U(W)$, we immediately get the linear-time output sensitive algorithm:

▶ **Corollary 3.** *Given a strongly connected graph G, it is possible to enumerate all maximal omnitigs of G in time linear in their total length.*

We obtain Theorem 2 using two interesting ingredients. The first is a novel graph structure (*macronodes*), obtained after a *compression* operation of "easy" nodes and arcs (Section 4). The second is a connection to a recent result by Georgiadis et al. [25] showing that it is possible to answer in $O(1)$-time strong connectivity queries under a *single* arc removal, after linear-time preprocessing (notice that a forbidden path is defined w.r.t. *two* arcs to avoid).

Theorem 2 has additional practical implications. First, omnitigs are also representable in the same (linear) size as the commonly used unitigs. Second, maximal macrotigs enable various $O(m)$-time operations on maximal omnitigs (without listing them explicitly), by pre-computing the univocal extensions from any node, needed in Theorem 2. For example, given that the number of maximal omnitigs is $O(m)$ [11], this implies the following result:

[2] Note that the total length of the maximal omnitigs is at least m, since every arc is an omnitig.
[3] The *univocal extension* $U(W)$ of a walk W is obtained by appending to W the longest path whose nodes (except for the last one) have out-degree 1, and prepending to W the longest path whose nodes (except for the first one) have in-degree 1; see Section 2 for the formal definition.

▶ **Corollary 4.** *Given a strongly connected graph G with m arcs, it is possible to compute the lengths of all maximal omnitig in total time $O(m)$.*

Corollary 4 leads to a linear-time computation of various statistics about maximal omnitigs, such as minimum, maximum, and average length (useful e.g. in [15]). One can also use this to filter out subfamilies of them (e.g. those of length smaller and/or larger than a given value) before enumerating them explicitly.

Significance of our results. This paper shows that *all* the strings that can be correctly assembled from a graph can be obtained in output-sensitive linear time, a time feasible for being implemented in practical genome assemblers. It closes the issue of finding safe walks for a fundamental model of genome assembly (*any* closed arc-covering walk), a long-standing theoretical question and originating with the use of unitigs in 1995 [32].

This theoretical question is crucial also from the practical point of view: assembly graphs have the number of nodes and arcs in the order of millions, and yet the total length of the maximal omnitigs is almost linear in the size of the graph. For example, the compressed (see Section 4) de Bruijn graph of human chromosome 10 (length 135 million) has 467 thousand arcs [11, Table 1], and the length of all maximal omnitigs (i.e. their total number of arcs, not their total string length) is 893 thousand. Moreover, even though this chromosome is only about 4% of the full human genome, the running time of the *quadratic* algorithm of [11] on its compressed de Bruijn graph is about 30 minutes.

We envision a reverse transfer from theory to practical and *complete* genome assembly programs, as in other Bioinformatics problems. For example, trivially, safe walks for all closed arc-covering walks are also safe for more specific types of arc-covering walks. Moreover, while a genome solution defined as a single closed arc-covering walk does not incorporate several practical issues of real data, in a follow-up work [10] we show that omnitigs are the basis of more advanced models handling many practical aspects. For example, to allow more types of genomes to be assembled, one can define an assembly solution as a *set* of closed walks that together cover all arcs [2], which is the case in *metagenomic* sequencing of bacteria. For linear chromosomes (as in eukaryotes such as human), or when modeling missing sequencing coverage, one can analogously consider one, or many, such *open* walks [54, 55]. Safe walks for all these models are subsets of omnitigs [2, 10]. Moreover, when modeling sequencing errors, or mutations present e.g. only in the mother copy of a chromosome (and not in the father's copy), one can require some arcs not to be covered by a solution walk, or even to be "invisible" from the point of view safety. Finding safe walks for such models is also based on first finding omnitigs-like walks [10].

Notice that such separation between theoretical formulations and their practical embodiments is common for many classical problems in Bioinformatics. For example, computing edit distance is often replaced with computing edit distance under affine gap costs [18], or enhanced with various heuristics as in the well-known BLAST aligner [3]. Also text indexes such as the FM-index [22] are extended in popular read mapping tools (e.g. [40, 38]) with many heuristics handling errors and mutations in the reads.

Finally, our results show that safe partial solutions enjoy interesting combinatorial properties, further promoting the persistency and safety frameworks. For real-world problems admitting multiple solutions, safe and complete algorithms are more pragmatic than the classical approach of outputting an arbitrary optimal solution. They are also more efficient than enumerating all, or only the first k-best, solutions [19, 20], because they already *synthesize all that can be correctly reconstructed from the input data.*

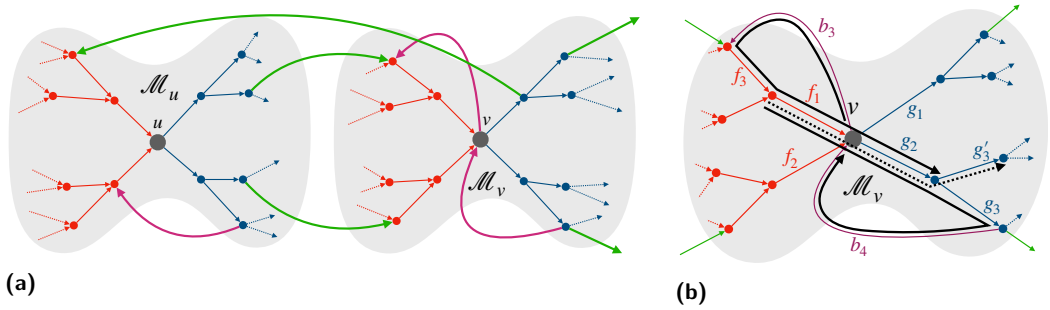


Figure 2 Figure 2a: Given a bivalent node v, the macronode $\mathcal{M}_v$ is the subgraph of G induced by the nodes reaching v with a split-free path (in red), and the nodes reachable from v with a join-free path (in blue). These two types of nodes induce the two trees of the macronode. By definition, every arc with endpoints in different macronodes is bivalent (in green, denoted *cross-bivalent arcs*). The remaining bivalent arcs have endpoints in the same macronode (in purple, denoted *self-bivalent arcs*). Figure 2b: The only omnitig traversing the bivalent node v is f_1g_2; e.g., by the X-intersection Property neither f_2g_2 is an omnitig ($b_3f_3f_1$ is a forbidden path) nor f_1g_1 is an omnitig ($g_2g_3b_4$ is a forbidden path). Extending the micro-omnitig f_1g_2 to the right we notice that $f_1g_2g_3$ is an omnitig and by the Y-intersection Property $f_1g_2g_3'$ is not an omnitig (g_3b_4 is a forbidden path). Hence, the only maximal right-micro omnitig is $f_1g_2g_3b_4$, and the only maximal left-micro omnitig is $b_3f_3f_1g_2$. Merging the two on f_1g_2, we obtain the maximal microtig $b_3f_3f_1g_2g_3b_4$.

2 Overview of the proofs

We highlight here our key structural and algorithmic contributions, and give more formal details in Sections 4 and 5. We start with the minimum terminology needed to understand this section, and defer the rest of the notation to Section 3.

Terminology. Functions $t(\cdot)$ and $h(\cdot)$ denote the *tail* node and the *head* node, respectively, of an arc or walk. We classify the nodes and arcs of a strongly connected graph as follows (see Figure 2a): (i) A node v is a *join node* if its *in-degree* $d^-(v)$ satisfies $d^-(v) > 1$, and a *join-free node* otherwise. An arc f is called a *join arc* if $h(f)$ is a join node, and a *join-free arc* otherwise. (ii) A node v is a *split node* if its *out-degree* $d^+(v)$ satisfies $d^+(v) > 1$, and a *split-free node* otherwise. An arc g is called a *split arc* if $t(g)$ is a split node, and a *split-free arc* otherwise. (iii) A node or arc is called *bivalent* if it is both join and split, and it is called *biunivocal* if it is both split-free and join-free. A walk W is *split-free* (resp., *join-free*) if all its arcs are split-free (resp., join-free). Given a walk W, its *univocal extension* $U(W)$ is defined as W^-WW^+, where W^- is the longest join-free path to $t(W)$ and W^+ is the longest split-free path from $h(W)$ (observe that they are uniquely defined).

Structure. The main structural insight of this paper is that omnitigs enjoy surprisingly limited freedom, in the sense that any omnitig can be seen as a concatenation of walks in a very specific set. In order to give the simplest exposition, we first simplify the graph by contracting biunivocal nodes and arcs. The nodes of the resulting graph can now be partitioned into *macronodes* (see Figure 2a and Definition 12), where each macronode $\mathcal{M}_v$ is uniquely identified by a bivalent node v (its *center*). We can now split the problem by first finding omnitigs inside each macronode, and then characterizing the ways in which omnitigs from different macronodes can combine.

We discover a key combinatorial property of how omnitigs can be extended: there are at most two ways that any omnitig can traverse a macronode center (see also Figure 2b):

▶ **Theorem 5** (X-intersection Property). *Let v be a bivalent node. Let $f_1 \neq f_2$ be join arcs with $h(f_1) = h(f_2) = v$; let $g_1 \neq g_2$ be split arcs with $t(g_1) = t(g_2) = v$.*

(i) *If f_1g_1 and f_2g_2 are omnitigs, then $d^+(v) = d^-(v) = 2$.*

(ii) *If f_1g_1 is an omnitig, then there are no omnitigs f_1g' with $g' \neq g_1$, nor $f'g_1$ with $f' \neq f_1$.*

In order to prove the X-intersection Property, we prove an even more fundamental property: once an omnitig traverses a macronode center, for any node it meets after the center node, there is at most one way of continuing from that node (Y-intersection Property), see Figure 2b. The basic intuition is that if there is more than one possibilities, then strong connectivity creates forbidden paths.

Given an omnitig fg traversing the bivalent node v, we define the *maximal right-micro omnitig* as the longest extension fgW in the macronode $\mathcal{M}_v$ (see Figure 2b and Definition 14). The *maximal left-micro omnitig* is the symmetrical omnitig Wfg. By Theorem 5, there are at most two maximal right-micro omnitigs and two maximal left-micro omnitigs. The merging of a maximal left- and right-micro omnitig on fg is called a *maximal microtig* (see Figure 2b and Definition 14; notice that a microtig is not necessarily an omnitig). These *at most two* maximal microtigs represent "forced tracks" to be followed by omnitigs crossing v.

We now describe how omnitigs can advance from one macronode to another. We prove that any arc having endpoints in different macronodes is a bivalent arc, and moreover, for every maximal microtig ending with a bivalent arc b, there is at most one maximal microtig starting with b. As such, when an omnitig track exits a macronode, there is at most one way of connecting it with an omnitig track from another macronode. It is natural to merge all omnitig tracks (i.e. maximal microtigs) on all bivalent arcs between *different* macronodes, and thus obtain *maximal macrotigs* (Definition 17 and Figure 5). The total size of all maximal macrotigs is $O(n)$ (Theorem 20), and they are a representation of all maximal omnitigs, except for those that are univocal extensions of the arcs of $\mathcal{F}$, see below and Theorem 21.

Algorithms. Our algorithms first build the set $\mathcal{M}$ of maximal macrotigs, and then identify maximal omnitigs inside them. The set $\mathcal{F}$ of arcs univocally extending to the remaining maximal omnitigs will be the set of bivalent arcs not appearing in $\mathcal{M}$ (Theorem 21).

Crucial to the algorithms is an *extension* primitive deciding what new arc (if any) to choose when extending an omnitig (recall that the X- and Y-intersection Properties limits the *number* of such arcs to one). Suppose we have an omnitig fW, with f a join arc, and we need to decide if it can be extended with an arc g out-going from $h(W)$. Naturally, this extension can be found by checking that there is no forbidden path from $t(g) = h(W)$. However, this forbidden path can potentially end in any node of W. Up to this point, [54, 55, 11] need to do an entire $O(m)$ graph traversal to check if any node of W is reachable by a forbidden path. We prove here a new key property:

▶ **Theorem 6** (Extension Property). *Let fW be an omnitig in G, where f is a join arc. Then fWg is an omnitig if and only if g is the only arc with $t(g) = h(W)$ such that there exists a path from $h(g)$ to $h(f)$ in $G \setminus f$.*

Thus, for each arc g with $t(g) = h(W)$, we can do a *single* reachability query under one arc removal: "does $h(g)$ reach $h(f)$ in $G \setminus f$?" Since the target of the reachability query is also the head of the arc excluded f, then we can apply an immediate consequence of [25]:

▶ **Theorem 7** ([25]). *Let G be a strongly connected graph with n nodes and m arcs. After $O(m)$-time preprocessing, one can build an $O(n)$-space data structure that, given a node w and an arc f, tests in $O(1)$ worst-case time if there is a path from w to $h(f)$ in $G \setminus f$.*

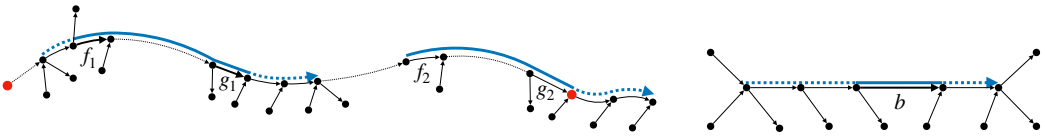


Figure 3 Any maximal omnitig is identified (in solid blue) either by a macrotig interval (from a join arc f to a split arc g; left), or by a bivalent arc b not appearing in any macrotig (right). The full maximal omnitig is obtained by univocal extension (dotted blue), extension which may go outside of the maximal macrotig.

Using the Extension Property and Theorem 7, we can thus pay $O(1)$ time to check each out-outgoing arc g, before discovering the one (if any) with which to extend fW. In the full version of this paper we describe how to transform the graph to have constant degree, so that we pay $O(1)$ per node. This transformation also requires slight changes to the maximal omnitig enumeration algorithm to maintain the linear-time output sensitive complexity. We use the Extension Property when building the left- and right-maximal micro omnitigs, and when identifying maximal omnitigs inside macrotigs, as follows.

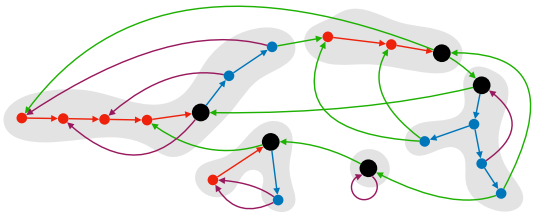
Once we have the set $\mathcal{M}$ of maximal macrotigs, we scan each macrotig with two pointers, a left one always on a join arc f, and a right one always on a split arc g (see Figure 3 and Algorithm 2). Both pointers move from left to right in such a way that the subwalk between them is always an omnitig. The subwalk is grown to the right by moving the right pointer as long as it remains an omnitig (checked with the Extension Property). When growing to the right is no longer possible, the omnitig is shrunk from the left by moving the left pointer. This technique runs in time linear to the total length of the maximal macrotigs, namely $O(n)$. In Figure 4 we work out all these notions on a concrete example.

Comparison with previous techniques. The algorithm of [55] exhaustively extends an omnitig with every edge outgoing from its head, as long as the resulting walk remained an omnitig, and did not use any insights on the structure of omnitigs. The $O(nm)$-time algorithm of [11] was obtained using two structural results: there can be only one left-maximal omnitig ending with a split arc (which we do not use here, since we prove deeper insights on the structure of omnitigs, e.g. the X- and Y-intersection Properties) and the existence of an acyclic order between split arcs connected by "simple" omnitigs. In [11], these allow computation to be memoized when recursively computing the left-maximal omnitig ending with a given split arc. The two-pointer technique was used also in [2] for a related problem, to test the safety of intervals of an *entire solution*. Our surprising discovery of macrotigs allow for a "small search space" of total size to $O(n)$, and eliminate the need of recursion, while the Extension Property enables the use of [25], thus the pay of $O(1)$ per omnitig extension, instead of $O(m)$ as in [54, 55, 11].

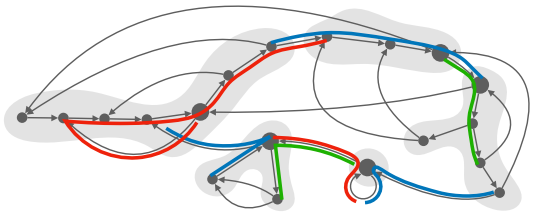
3 Basics

In this paper, a *graph* is a tuple $G = (V, E)$, where V is a finite set of *nodes*, E is a finite multi-set of ordered pairs of nodes called *arcs*. Parallel arcs and self-loops are allowed. For an arc $e \in E(G)$, we denote $G \setminus e = (V, E \setminus \{e\})$. The *reverse graph* G^R of G is obtained by reversing the direction of every arc. In the rest of this paper, we assume a fixed strongly connected graph $G = (V, E)$, with $|V| = n$ and $|E| = m \geq n$.

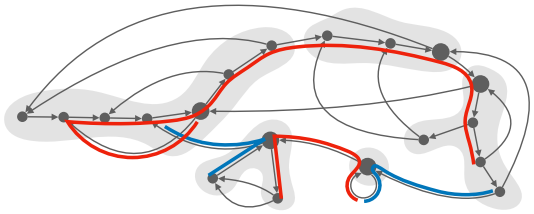
A *walk* in G is a sequence $W = (v_0, e_1, v_1, e_2, \ldots, v_{\ell-1}, e_\ell, v_\ell)$, $\ell \geq 0$, where $v_0, v_1, \ldots, v_\ell \in V$, and each e_i is an arc from v_{i-1} to v_i. Sometimes we drop the nodes $v_0, \ldots, v_\ell$ of W, and write W more compactly as $e_1 \ldots e_\ell$. If an arc e appears in W, we write $e \in W$. We say



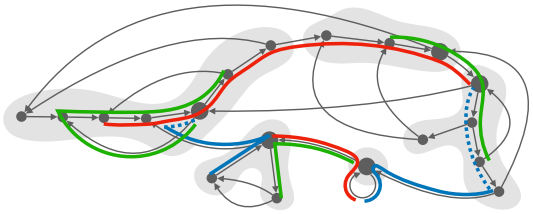
(a) Nodes and arcs color-coded as in Figure 2a.



(b) Maximal microtigs.



(c) Maximal macrotigs.



(d) The maximal omnitigs obtained from maximal macrotigs (univocal extensions are dotted). All other maximal omnitigs are univocal extensions of the bivalent arcs not appearing in macrotigs.

Figure 4 A concrete example of the main notions of this paper. In Figures 4b–4d walks have different colors for visual distinguishability.

that W goes *from* $t(W) = v_0$ *to* $h(W) = v_\ell$, has *length* ℓ, contains $v_1, \ldots, v_{\ell-1}$ as *internal nodes*, *starts with* e_1, *ends with* e_ℓ, and contains $e_2, \ldots, e_{\ell-1}$ as *internal arcs*. A walk W is called *empty* if it has length zero, and *non-empty* otherwise. There exists exactly one empty walk $\epsilon_v = (v)$ for every node $v \in V$, and $t(\epsilon_v) = h(\epsilon_v) = v$. A walk W is called *closed* if it is non-empty and $t(W) = h(W)$, otherwise it is *open*. The concatenation of walks W and W' (with $h(W) = t(W')$) is denoted WW'. A walk $W = (v_0, e_1, v_1, \ldots, e_\ell, v_\ell)$ is called a *path* when the nodes $v_0, v_1, \ldots, v_\ell$ are all distinct, with the exception that $v_\ell = v_0$ is allowed (in which case we have either a closed or an empty path). To simplify notation, we may denote a walk $W = (v_0, e_1, v_1, \ldots, e_\ell, v_\ell)$ as a sequence of arcs, i.e. $W = e_1 \ldots e_\ell$. Subwalks of open walks are defined in the standard manner. For a closed walk $W = e_0 \ldots e_{\ell-1}$, we say that $W' = e'_0 \ldots e'_j$ is a subwalk of W if there exists $i \in \{0, \ldots, \ell - 1\}$ such that for every $k \in \{0, \ldots, j\}$ it holds that $e'_k = e_{(i+k) \bmod \ell}$.

A closed *arc-covering* walk (i.e. passing through every arc at least once) exists if and only if the graph is strongly connected. We are interested in the (safe) walks that are subwalks of all closed arc-covering walks:

▶ **Theorem 8** ([55]). *Let G be a strongly connected graph different from a closed path. Then a walk W is a subwalk of all closed arc-covering walks of G if and only if W is an* omnitig.

Observe that W is an omnitig in G if and only if W^R is an omnitig in G^R. Moreover, any subwalk of an omnitig is an omnitig. For every arc e, its univocal extension $U(e)$ is an omnitig. A walk W satisfying a property $\mathcal{P}$ is *right-maximal* (resp., *left-maximal*) if there is no walk We (resp., eW) satisfying $\mathcal{P}$. A walk satisfying $\mathcal{P}$ is *maximal* if it is left- and right-maximal w.r.t. $\mathcal{P}$. Notice that if G is a closed path, then every *walk* of G is an omnitig. As such, it is relevant to find the maximal omnitigs of G only when G is different from a closed path. Thus, in the rest of this paper our strongly connected graph G is considered to be different from a closed path, even when we do not mention it explicitly.

4 Macronodes and macrotigs

We summarize here the key results about macronodes and macrotigs, and refer the reader to the full version of this paper for the full details and proofs. Unless otherwise stated, we assume that the input graph is *compressed*, in the sense that it has no biunivocal nodes and arcs. In some algorithms we will also require that the graph has constant in- and out-degree. In the full version of this paper we show how these properties can be guaranteed, by transforming any strongly connected graph G with m arcs, in time $O(m)$, into a compressed graph of constant degree and with $O(m)$ nodes and arcs. Observe that in a compressed graph all arcs are split, join or bivalent. Moreover, the following properties hold.

▶ **Observation 9.** *Let G be a compressed graph. Let f and g be a join and a split arc, respectively, in G. The following holds:*

(i) *if fWg is a walk, then W has a node which is a bivalent node;*

(ii) *if gWf is a walk, then gWf contains a bivalent arc.*

▶ **Lemma 10.** *Every maximal omnitig of a compressed graph contains both a join arc and a split arc. Moreover, it has a bivalent arc or an internal bivalent node.*

▶ **Lemma 11.** *Let u be a bivalent node. No omnitig contains u twice as an internal node.*

Macronodes. We introduce a natural partition of the nodes of a compressed graph; each class of such a partition (i.e. a *macronode*) contains precisely one bivalent node. We identify each class with the unique bivalent node they contain. All other nodes belonging to the same class are those that either reach the bivalent node with a join-free path or those that are reached by the bivalent node with a split-free path (recall Figure 2a).

▶ **Definition 12** (Macronode). *Let v be a bivalent node of G. Consider the following sets:*

- $R^+(v) := \{u \in V(G) : \exists$ *a join-free path from v to $u\}$;*
- $R^-(v) := \{u \in V(G) : \exists$ *a split-free path from u to $v\}$.*

The subgraph $\mathcal{M}_v$ induced by $R^+(v) \cup R^-(v)$ is called the macronode centered in v.

▶ **Lemma 13.** *In a compressed graph G, the following properties hold:*

i) *The set $\{V(\mathcal{M}_v) : v$ is a bivalent node of $G\}$ is a partition of $V(G)$.*

ii) *In a macronode $\mathcal{M}_v$, $R^+(v)$ and $R^-(v)$ induce two trees with common root v, but oriented in opposite directions. Except for the common root, the two trees are node-disjoint, all nodes in $R^-(v)$ being join nodes and all nodes in $R^+(v)$ being split nodes.*

iii) *The only arcs with endpoints in two different macronodes are bivalent arcs.*

To analyze how omnitigs can traverse a macronode and the degrees of freedom they have in choosing their directions within the macronode, we introduce the following definitions. *Central-micro omnitigs* are the smallest omnitigs that cross the center of a macronode. *Left- and right-micro omnitigs* start from a central-micro omnitig and proceed to the periphery of a macronode. Finally, we combine left- and right-micro omnitigs into microtigs (which are not necessarily omnitigs themselves); recall Figure 2b.

▶ **Definition 14** (Micro omnitigs, microtigs). *Let f be a join arc and g be a split arc, such that fg is an omnitig.*

- *The omnitig fg is called a* central-micro omnitig.
- *An omnitig fgW (Wfg, resp.) that does not contain a bivalent arc as an internal arc is called a* right-micro omnitig *(respectively,* left-micro omnitig*).*
- *A walk $W = W_1fgW_2$, where W_1fg and fgW_2 are, respectively, a left-micro omnitig, and a right-micro omnitig, is called a* microtig.

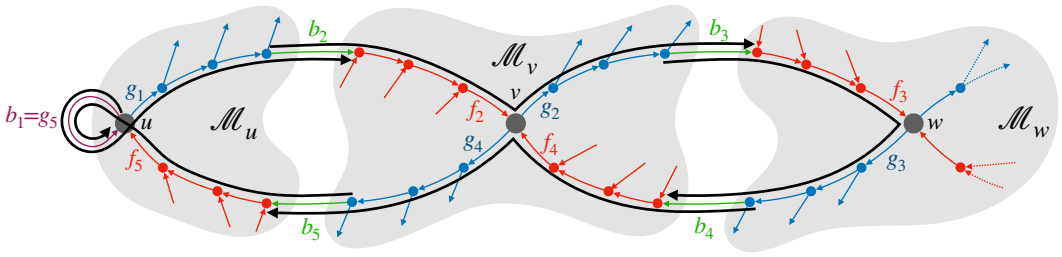


Figure 5 Three macronodes $\mathcal{M}_u, \mathcal{M}_v, \mathcal{M}_w$ (as gray areas) with arcs color-coded as in Figure 2a. Black walks mark their five maximal microtigs: $b_1 g_1 \dots b_2$, $b_i \dots f_i g_i \dots b_{i+1}$ ($i \in \{2,3,4\}$), $b_5 \dots f_5 g_5$ ($g_5 = b_1$). The maximal macrotig M is obtained by overlapping them on the cross-bivalent arcs b_2, b_3, b_4, b_5, i.e. $M = b_1 \dots b_2 \dots b_3 \dots b_4 \dots b_5 \dots b_1$. Notice that no arc is contained twice in M, with the exception of the self-bivalent arc b_1, appearing as the first and last arc of M.

▶ **Theorem 15** (Maximal microtigs). *The maximal microtigs of any strongly connected graph G with n nodes, m arcs, and arbitrary degree have total length $O(n)$, and can be computed in time $O(m)$.*

Macrotigs. Macronodes are connected with each other by bivalent arcs (Lemma 13), but merging microtigs on all possible bivalent arcs may create too complicated structures. However, this can be avoided by a simple classification of bivalent arcs: those that connect a macronode with itself (*self-bivalent*) and those that connect two different macronodes (*cross-bivalent*), recall Figure 2a.

▶ **Definition 16** (Self-bivalent and cross-bivalent arcs). *A bivalent arc b is called a* self-bivalent arc *if $U(b)$ goes from a bivalent node to itself. Otherwise it is called a* cross-bivalent arc.

A macrotig is now obtained by merging those microtigs from different macronodes which overlap only on a cross-bivalent arc, see also Figure 5.

▶ **Definition 17** (Macrotig). *Let W be any walk. W is called a* macrotig *if*

1. *W is an microtig, or*
2. *By writing $W = W_0 b_1 W_1 b_2 \dots b_{k-1} W_{k-1} b_k W_k$, where $b_1, \dots, b_k$ are all the internal bivalent arcs of W, the following conditions hold:*
   - **a.** *the arcs $b_1, \dots, b_k$ are all cross-bivalent arcs, and*
   - **b.** *$W_0 b_1$, $b_1 W_1 b_2$, …, $b_{k-1} W_{k-1} b_k$, $b_k W_k$ are all microtigs.*

Our structural results (including Lemmas 18 and 19 below) show that we can construct all *maximal* macrotigs by repeatedly joining microtigs overlapping on cross-bivalent arcs, as long as possible, and obtain Theorem 20.

▶ **Lemma 18.** *For any macrotig W there exists a unique maximal macrotig containing W.*

▶ **Lemma 19.** *Let W be a macrotig and let e be an arc of W. If e is self-bivalent, then e appears at most twice in W (as first or as last arc of W). Otherwise, e appears only once.*

▶ **Theorem 20** (Maximal macrotigs). *The maximal macrotigs of any strongly connected graph G with n nodes, m arcs, and arbitrary degree have total length $O(n)$, and can be computed in time $O(m)$.*

5 Maximal omnitig representation and enumeration

In the algorithms of this section we assume that the graph has constant degree. In the full version of this paper we explain how to handle the non-constant degree case.

We begin by proving the first part of Theorem 2. Theorem 20 guarantees that the total length of maximal macrotigs is $O(n)$. Thus, it remains to prove the following lemma, since since any macrotig is a subwalk of a maximal macrotig (Lemma 18).

▶ **Theorem 21** (Maximal omnitig representation)**.** *Let W be a maximal omnitig.*

- **i)** *If W contains an internal bivalent node, then W is of the form $U(fW'g)$, where f is the first join arc of W and $g \neq f$ is the last split arc of W, and W' is a possibly empty walk. Moreover, $fW'g$ is a macrotig.*
- **ii)** *Otherwise, W is of the form $U(b)$, where b is a bivalent arc, and b does not belong to any macrotig.*

Proof. To prove i), let u be an internal bivalent node of W, and let f_u and g_u be, respectively, the join arc and the split arc of W with $h(f_u) = u = t(g_u)$; both such f_u and g_u exist, since u is an internal node of W. Therefore, since W contains at least f_u and g_u, let f and g be, respectively the first join arc and the last split arc of W. Observe that f is either f_u or it appears before f_u in W; likewise, g is either g_u or it appears after g_u in W. Thus, f comes before g, and we can write $W = W^- fW'gW^+$, where W' is the subwalk of W, possibly empty, from $h(f)$ to $t(g)$. Therefore, by the maximality of W, we have $W = W^- fW'gW^+ = U(fW'g)$.

To prove that the subwalk $fW'g$ of W is a macrotig, we prove by induction that *any* walk of the form $fW'g$, where f is a join arc and g is a split arc, is a macrotig. The induction is on the length of W'.

Case 1: W' contains no internal bivalent arcs. Since $fW'g$ contains a bivalent node (Observation 9), it is of the form $fW'g = W_1' f' g' W_2'$, with $h(f') = t(g') = u$ bivalent node. Notice that $W_1' f' g' W_2'$ is an microtig and thus it is a macrotig, by definition.

Case 2: $fW'g$ contains an internal bivalent arc b, i.e. $fW'g = W_1' b W_2'$, with W_1', W_2' non empty. By induction, $W_1'b$ and bW_2' are macrotigs and both contain a bivalent node as internal node. Suppose b is a self-bivalent arc, then both $W_1'b$ and bW_2' would contain the same bivalent node u as internal node, contradicting Lemma 11. Thus, b is a cross-bivalent arc and $W_1' b W_2'$ is also a macrotig, by definition.

For ii), notice that if W contains no internal bivalent node then it contains a unique bivalent arc b, by Lemma 10 and Observation 9. Thus, by the maximality of W, it holds that $W = U(b)$. It remains to prove that there is no macrotig containing b.

Suppose for a contradiction that there is a maximal left-micro omnitig M containing b. By definition, M is of the form $bW_M f_M g_M$. Notice that Wg_M is an omnitig, because M is an omnitig and the arcs of W before b are join-free, so Wg_M can have no forbidden path. This contradicts the fact that W is maximal.

Symmetrically, we have that there is no maximal right-micro omnitig containing b. Thus, by definition, b appears in no microtig, and thus in no macrotig. ◀

▶ Remark 22. The number of maximal omnitigs containing an internal bivalent node is $O(n)$. This follows by Theorem 21(i), by maximality, and by the fact that the total length of maximal macrotigs is $O(n)$ (Theorem 20).

Algorithm 1 Function IsOmnitigRightExtension.

Function IsOmnitigRightExtension(G, f, g)
  Input : The compressed graph G. A join arc f and a split arc g such that there exists a walk fWg where fW is an omnitig.
  Output : Whether fWg is also an omnitig.
  $S \leftarrow \{g' \in E(G) \mid t(g') = t(g)$ and there is a path from $h(g')$ to $h(f)$ in $G \setminus f\}$
  Return : **True** if $S = \{g\}$ and **False** otherwise

Next, we are going to prove the second, algorithmic, part of Theorem 2. By Theorem 20 we can compute the maximal macrotigs of G in time $O(m)$. We can trivially obtain in $O(m)$ time the set $\mathcal{F}$ of arcs not appearing in the maximal macrotigs. It remains to show how to obtain the subwalks of the maximal macrotigs univocally extending to maximal omnitigs.

We first prove an auxiliary lemma needed for the proof of the Extension Property (Theorem 6).

▶ **Lemma 23.** *Let fW be an omnitig, where f is a join arc. Let P be a path from $t(P) = h(W)$ to a node in W, such that the last arc of P is not an arc of fW. Then no internal node of P is a node of W.*

Proof. Consider P_W the longest suffix of P, such that no internal node of P_W is a node of W. If $P_W = P$, the lemma trivially holds. Let now $W = (u_0, e_1, u_1, e_2, \ldots, e_k, u_k)$. Let $u_i = t(P_W)$ and $u_j = h(P_W)$. If $i \geq j$, then P_W is a forbidden path for fW,a contradiction. Hence, assume $i < j < k$. Let $f'WQ$ be a closed path. Consider the walk $Z = P_W e_{j+1} \ldots e_k Q$. Notice that $e_{i+1} \notin Z$ and $f \notin Z$. Thus Z can transformed in a forbidden path for fW, from u_i to $h(f)$. ◀

▶ **Theorem 6** (Extension Property). *Let fW be an omnitig in G, where f is a join arc. Then fWg is an omnitig if and only if g is the only arc with $t(g) = h(W)$ such that there exists a path from $h(g)$ to $h(f)$ in $G \setminus f$.*

Proof. To prove the existence of an arc g, which satisfies the condition, consider any closed path Pf' in G, where f' is an arbitrary sibling join arc of f. Notice that W is a prefix of Pf', since fW is an omnitig, since otherwise one can easily find a forbidden path for the omnitig fW as a subpath of Pf', from the head of the very first arc of Pf' that is not in W to $h(f')$. Therefore, let g be the the first arc of Pf' after the prefix W, in such a way that the suffix of Pf' starting from $h(g)$ is a path to $h(f)$ in $G \setminus f$. Moreover, assume g is a split arc, otherwise the statement trivially holds.

First, assume that there is a g' sibling split arc of g and a path P from $h(g')$ to $h(f)$ in $G \setminus f$. We prove that there exists a forbidden path for fWg. Let P_W be the prefix of P ending in the first occurrence of a node in W (i.e., no node of P_W belongs to W, except for $h(P_W)$). Notice that $g'P_W$ is a forbidden path for the omnitig fWg (it is possible, but not necessary, that $h(P_W) = h(f)$).

Second, take any forbidden path P for the omnitig fWg. We prove that there exists a g' sibling split arc of g and a path from $h(g)$ to $h(f)$ in $G \setminus f$. Notice that $t(P) = h(W) = t(g)$, otherwise P would be a forbidden path for fW. As such, P starts with a split arc $g' \neq g$ and, by Lemma 23, P does not contain f. Thus, the suffix of P from $h(g')$ is a path in $G \setminus f$ from $h(g')$ to $h(f)$. ◀

Algorithm 2 Computing all maximal omnitigs.

Input : The compressed strongly connected graph G of constant degree.
Output : All maximal omnitigs of G.

▷Assume that AllMaximalMacrotigs(G) returns all the maximal macrotigs in G.
▷Recall that $U(W)$ is the univocal extension of the walk W.

1 $B \leftarrow \{b \mid b$ bivalent arc that does not occur in any $W \in$ AllMaximalMacrotigs$(G)\}$
2 **foreach** $b \in B$ **do output** $U(b)$
3 **foreach** $f^*Xg^* \in$ AllMaximalMacrotigs(G) **do**
  ▷With the notation $X[f..g]$, we refer to the subwalk of f^*Xg^* starting with the occurrence of f in f^*X (unique by Lemma 19) and ending with the occurrence of g in Xg^* (unique by Lemma 19).
4 $f \leftarrow f^*$, $g \leftarrow$ **nil**, $g' \leftarrow$ first split arc in Xg^*
5 **while** $g' \neq$ **nil** **do**
6 **while** $g' \neq$ **nil and** IsOmnitigRightExtension(f, g') **do**
    ▷Grow $X[f..g]$ to the right as long as possible
7 $g \leftarrow g'$
8 $g' \leftarrow$ next split arc in Xg^* after g
  ▷$X[f..g]$ cannot be grown to the right anymore
9 **output** $U(X[f..g])$
10 **while** $g' \neq$ **nil and not** IsOmnitigRightExtension(f, g') **do**
    ▷Shrink $X[f..g]$ from the left until it can be grown to the right again
11 $f \leftarrow$ next join arc in f^*X after f

To describe the algorithm that identifies all maximal omnitigs (Algorithm 2), we first introduce an auxiliary procedure (Algorithm 1), which uses the Extension Property (Theorem 7) and Theorem 6 to find the unique possible extension of an omnitig.

▶ **Corollary 24.** *Algorithm 1 is correct. Assuming that the graph has constant degree, we can preprocess it in time $O(m + n)$ time, so that Algorithm 1 runs in constant time.*

Maximal omnitigs are identified with a two-pointer scan of maximal macrotigs (Algorithm 2): a left pointer always on a join arc f and a right pointer always on a split arc g, recall Figure 3. For completeness, Algorithm 2 also outputs the maximal omnitigs.

▶ **Theorem 25** (Maximal omnitig enumeration)**.** *Algorithm 2 is correct and, if the compressed graph has constant degree, it runs in time linear in the size of the graph and of its output.*

Proof. By Theorem 21, any maximal omnitig W is either of the form $U(fW'g)$ (where $fW'g$ is a macrotig, and thus also a subwalk of a maximal macrotig, by Lemma 18), or of the form $W = U(b)$, where b is a bivalent arc not appearing in any macrotig. In the latter case, such omnitigs are outputted in Line 2. In the former case, it remains to prove that the external *while* cycle outputs all the maximal omnitigs of the form $U(fW'g)$ where $fW'g$ is contained in a maximal macrotig f^*Xg^*. At the beginning of the first iteration, $W = U(X[f..g'])$ is left-maximal since $f = f^*$. The first internal *while* cycle ensures that $W = U(X[f..g])$ is also right-maximal, at which point it is printed in output. Then, the second internal *while* cycle ensures that $W = U(X[f..g'])$ is a left-maximal omnitig, and the external cycle repeats.

For the running time analysis, note that AllMaximalMacrotigs(G) runs in $O(m)$ time, by Theorem 20. Each iteration of the *foreach* cycle takes time linear in the total size of the maximal macrotig X and of its output (by Corollary 24), and that the total size of all maximal macrotigs is linear, by Theorem 20. ◀

References

1 Amir Abboud, Arturs Backurs, and Virginia Vassilevska Williams. Tight hardness results for LCS and other sequence similarity measures. In Venkatesan Guruswami, editor, *IEEE 56th Annual Symposium on Foundations of Computer Science, FOCS 2015, Berkeley, CA, USA, 17-20 October, 2015*, pages 59–78. IEEE Computer Society, 2015. doi:10.1109/FOCS.2015.14.

2 Nidia Obscura Acosta, Veli Mäkinen, and Alexandru I. Tomescu. A safe and complete algorithm for metagenomic assembly. *Algorithms for Molecular Biology*, 13(1):3:1–3:12, 2018. doi:10.1186/s13015-018-0122-7.

3 Stephen F Altschul, Warren Gish, Webb Miller, Eugene W Myers, and David J Lipman. Basic local alignment search tool. *Journal of molecular biology*, 215(3):403–410, 1990.

4 Arturs Backurs and Piotr Indyk. Edit distance cannot be computed in strongly subquadratic time (unless SETH is false). In Rocco A. Servedio and Ronitt Rubinfeld, editors, *Proceedings of the Forty-Seventh Annual ACM on Symposium on Theory of Computing, STOC 2015, Portland, OR, USA, June 14-17, 2015*, pages 51–58. ACM, 2015. doi:10.1145/2746539.2746612.

5 Arturs Backurs and Piotr Indyk. Which regular expression patterns are hard to match? In Irit Dinur, editor, *IEEE 57th Annual Symposium on Foundations of Computer Science, FOCS 2016, 9-11 October 2016, Hyatt Regency, New Brunswick, New Jersey, USA*, pages 457–466. IEEE Computer Society, 2016. doi:10.1109/FOCS.2016.56.

6 Djamal Belazzougui. Linear time construction of compressed text indices in compact space. In David B. Shmoys, editor, *Symposium on Theory of Computing, STOC 2014, New York, NY, USA, May 31 - June 03, 2014*, pages 148–193. ACM, 2014. doi:10.1145/2591796.2591885.

7 Djamal Belazzougui and Simon J. Puglisi. Range predecessor and lempel-ziv parsing. In Robert Krauthgamer, editor, *Proceedings of the Twenty-Seventh Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2016, Arlington, VA, USA, January 10-12, 2016*, pages 2053–2071. SIAM, 2016. doi:10.1137/1.9781611974331.ch143.

8 Sébastien Boisvert, François Laviolette, and Jacques Corbeil. Ray: simultaneous assembly of reads from a mix of high-throughput sequencing technologies. *Journal of computational biology*, 17(11):1519–1533, 2010.

9 G. Bresler, M. Bresler, and D. Tse. Optimal Assembly for High Throughput Shotgun Sequencing. *BMC Bioinformatics*, 14(Suppl 5):S18, 2013.

10 Massimo Cairo, Shahbaz Khan, Romeo Rizzi, Sebastian S. Schmidt, Alexandru I. Tomescu, and Elia C. Zirondelli. Genome assembly, a universal theoretical framework: unifying and generalizing the safe and complete algorithms. *CoRR*, abs/2011.12635, 2020. arXiv:2011.12635.

11 Massimo Cairo, Paul Medvedev, Nidia Obscura Acosta, Romeo Rizzi, and Alexandru I. Tomescu. An Optimal $O(nm)$ Algorithm for Enumerating All Walks Common to All Closed Edge-covering Walks of a Graph. *ACM Trans. Algorithms*, 15(4):48:1–48:17, 2019. doi:10.1145/3341731.

12 Massimo Cairo, Romeo Rizzi, Alexandru I Tomescu, and Elia C Zirondelli. Genome assembly, from practice to theory: safe, complete and linear-time. *arXiv preprint*, 2020. arXiv:2002.10498.

13 Katarína Cechlárová. Persistency in the assignment and transportation problems. *Mat. Meth. OR*, 47(2):243–254, 1998. doi:10.1007/BF01194399.

14 Kun-Mao Chao, Ross C. Hardison, and Webb Miller. Locating well-conserved regions within a pairwise alignment. *CABIOS*, 9(4):387–396, 1993. doi:10.1093/bioinformatics/9.4.387.

15 Rayan Chikhi and Paul Medvedev. Informed and automated k-mer size selection for genome assembly. *Bioinformatics*, 30(1):31–37, June 2013. doi:10.1093/bioinformatics/btt310.

16 Marie Costa. Persistency in maximum cardinality bipartite matchings. *Oper. Res. Lett.*, 15(3):143–9, 1994. doi:10.1016/0167-6377(94)90049-3.

17 Bartłomiej Dudek and Paweł Gawrychowski. Computing quartet distance is equivalent to counting 4-cycles. In *Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing*, STOC 2019, pages 733–743, New York, NY, USA, 2019. Association for Computing Machinery. doi:10.1145/3313276.3316390.

18 Richard Durbin, Sean R Eddy, Anders Krogh, and Graeme Mitchison. *Biological sequence analysis: probabilistic models of proteins and nucleic acids*. Cambridge university press, 1998.

19 David Eppstein. K-best enumeration. *Bulletin of the EATCS*, 115, 2015. URL: http://eatcs.org/beatcs/index.php/beatcs/article/view/322.

20 David Eppstein. k-best enumeration. In Ming-Yang Kao, editor, *Encyclopedia of Algorithms*, pages 1003–1006. Springer, New York, NY, 2016. doi:10.1007/978-1-4939-2864-4_733.

21 Massimo Equi, Roberto Grossi, Veli Mäkinen, and Alexandru I. Tomescu. On the complexity of string matching for graphs. In Christel Baier, Ioannis Chatzigiannakis, Paola Flocchini, and Stefano Leonardi, editors, *46th International Colloquium on Automata, Languages, and Programming, ICALP 2019, July 9-12, 2019, Patras, Greece*, volume 132 of *LIPIcs*, pages 55:1–55:15. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019. doi:10.4230/LIPIcs.ICALP.2019.55.

22 Paolo Ferragina and Giovanni Manzini. Opportunistic data structures with applications. In *41st Annual Symposium on Foundations of Computer Science, FOCS 2000, 12-14 November 2000, Redondo Beach, California, USA*, pages 390–398. IEEE Computer Society, 2000. doi:10.1109/SFCS.2000.892127.

23 Paolo Ferragina, Igor Nitto, and Rossano Venturini. On the bit-complexity of lempel-ziv compression. In Claire Mathieu, editor, *Proceedings of the Twentieth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2009, New York, NY, USA, January 4-6, 2009*, pages 768–777. SIAM, 2009. doi:10.1137/1.9781611973068.

24 A Friemann and S Schmitz. A new approach for displaying identities and differences among aligned amino acid sequences. *Comput Appl Biosci*, 8(3):261–265, June 1992.

25 Loukas Georgiadis, Giuseppe F Italiano, and Nikos Parotsidis. Strong connectivity in directed graphs under failures, with applications. In *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1880–1899. SIAM, 2017.

26 Meigu Guan. Graphic programming using odd and even points. *Chinese Math.*, 1:237–277, 1962.

27 A. Guénoche. Can we recover a sequence, just knowing all its subsequences of given length? *Computer Applications in the Biosciences*, 8(6):569–574, 1992. URL: http://dblp.uni-trier.de/db/journals/bioinformatics/bioinformatics8.html#Guenoche92.

28 P. L. Hammer, P. Hansen, and B. Simeone. Vertices belonging to all or to no maximum stable sets of a graph. *SIAM Journal on Algebraic Discrete Methods*, 3(4):511–522, 1982. doi:10.1137/0603052.

29 Iu, V. L. Florent'ev, A. A. Khorlin, K. R. Khrapko, and V. V. Shik. Determination of the nucleotide sequence of DNA using hybridization with oligonucleotides. A new method. *Doklady Akademii nauk SSSR*, 303(6):1508–1511, 1988. URL: http://view.ncbi.nlm.nih.gov/pubmed/3250844.

30 Benjamin Grant Jackson. *Parallel methods for short read assembly*. PhD thesis, Iowa State University, 2009.

31 Evgeny Kapun and Fedor Tsarev. De Bruijn superwalk with multiplicities problem is NP-hard. *BMC Bioinformatics*, 14(Suppl 5):S7, 2013.

32 John D. Kececioglu and Eugene W. Myers. Combinatorial algorithms for DNA sequence assembly. *Algorithmica*, 13(1/2):7–51, 1995.

33 John Dimitri Kececioglu. *Exact and approximation algorithms for DNA sequence reconstruction*. PhD thesis, University of Arizona, Tucson, AZ, USA, 1992.

34 Dominik Kempa and Tomasz Kociumaka. String synchronizing sets: sublinear-time BWT construction and optimal LCE data structure. In Moses Charikar and Edith Cohen, editors, *Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing, STOC 2019, Phoenix, AZ, USA, June 23-26, 2019*, pages 756–767. ACM, 2019. doi:10.1145/3313276.3316368.

35 Dominik Kempa and Nicola Prezza. At the roots of dictionary compression: string attractors. In Ilias Diakonikolas, David Kempe, and Monika Henzinger, editors, *Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2018, Los Angeles, CA, USA, June 25-29, 2018*, pages 827–840. ACM, 2018. doi:10.1145/3188745.3188814.

36 Carl Kingsford, Michael C Schatz, and Mihai Pop. Assembly complexity of prokaryotic genomes using short reads. *BMC Bioinformatics*, 11(1):21, 2010.

37 Ka-Kit Lam, Asif Khalak, and David Tse. Near-optimal assembly for shotgun sequencing with noisy reads. *BMC Bioinform.*, 15(S-9):S4, 2014. doi:10.1186/1471-2105-15-S9-S4.

38 Ben Langmead and Steven L Salzberg. Fast gapped-read alignment with bowtie 2. *Nature Methods*, 9(4):357, 2012.

39 Dinghua Li, Chi-Man Liu, Ruibang Luo, Kunihiko Sadakane, and Tak-Wah Lam. Megahit: an ultra-fast single-node solution for large and complex metagenomics assembly via succinct de bruijn graph. *Bioinformatics*, 31(10):1674–1676, 2015.

40 Heng Li and Richard Durbin. Fast and accurate short read alignment with Burrows–Wheeler transform. *Bioinformatics*, 25(14):1754–1760, 2009.

41 Veli Mäkinen, Djamal Belazzougui, Fabio Cunial, and Alexandru I. Tomescu. *Genome-Scale Algorithm Design: Biological Sequence Analysis in the Era of High-Throughput Sequencing*. Cambridge University Press, 2015. doi:10.1017/CBO9781139940023.

42 Paul Medvedev. Modeling biological problems in computer science: a case study in genome assembly. *Briefings in bioinformatics*, 20(4):1376–1383, 2019.

43 Paul Medvedev and Michael Brudno. Maximum likelihood genome assembly. *Journal of computational biology*, 16(8):1101–1116, 2009.

44 Paul Medvedev, Konstantinos Georgiou, Gene Myers, and Michael Brudno. Computability of models for sequence assembly. In *WABI*, pages 289–301, 2007.

45 Eugene W. Myers. The fragment assembly string graph. In *ECCB/JBI*, page 85, 2005.

46 Niranjan Nagarajan and Mihai Pop. Parametric complexity of sequence assembly: theory and applications to next generation sequencing. *Journal of computational biology*, 16(7):897–908, 2009.

47 Niranjan Nagarajan and Mihai Pop. Sequence assembly demystified. *Nature Reviews Genetics*, 14(3):157–167, 2013.

48 Giuseppe Narzisi, Bud Mishra, and Michael C Schatz. On algorithmic complexity of biomolecular sequence assembly problem. In *Algorithms for Computational Biology*, pages 183–195. Springer, 2014.

49 Hannu Peltola, Hans Söderlund, Jorma Tarhio, and Esko Ukkonen. Algorithms for some string matching problems arising in molecular genetics. In *IFIP Congress*, pages 59–64, 1983.

50 P. A. Pevzner. l-Tuple DNA sequencing: computer analysis. *Journal of Biomolecular Structure & Dynamics*, 7(1):63–73, 1989.

51 Pavel A. Pevzner, Haixu Tang, and Michael S. Waterman. An Eulerian path approach to DNA fragment assembly. *Proceedings of the National Academy of Sciences*, 98(17):9748–9753, 2001.

52 Jue Ruan and Heng Li. Fast and accurate long-read assembly with wtdbg2. *Nature Methods*, 17(2):155–158, 2020. doi:10.1038/s41592-019-0669-3.

53 Ilan Shomorony, Samuel H. Kim, Thomas A. Courtade, and David N. C. Tse. Information-optimal genome assembly via sparse read-overlap graphs. *Bioinform.*, 32(17):494–502, 2016. doi:10.1093/bioinformatics/btw450.

54 Alexandru I. Tomescu and Paul Medvedev. Safe and Complete Contig Assembly Via Omnitigs. In Mona Singh, editor, *Research in Computational Molecular Biology - 20th Annual Conference, RECOMB 2016, Santa Monica, CA, USA, April 17-21, 2016, Proceedings*, volume 9649 of *Lecture Notes in Computer Science*, pages 152–163. Springer, 2016. doi:10.1007/978-3-319-31957-5_11.

55 Alexandru I. Tomescu and Paul Medvedev. Safe and complete contig assembly through omnitigs. *Journal of Computational Biology*, 24(6):590–602, 2017.

56 Martin Vingron and Patrick Argos. Determination of reliable regions in protein sequence alignments. *Prot. Engin.*, 3(7):565–569, 1990. doi:10.1093/protein/3.7.565.

57 Virginia Vassilevska Williams, Joshua R. Wang, Richard Ryan Williams, and Huacheng Yu. Finding four-node subgraphs in triangle time. In Piotr Indyk, editor, *Proceedings of the Twenty-Sixth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2015, San Diego, CA, USA, January 4-6, 2015*, pages 1671–1680. SIAM, 2015. doi:10.1137/1.9781611973730.111.

58 Fan Wu, Su Zhao, Bin Yu, Yan-Mei Chen, Wen Wang, Zhi-Gang Song, Yi Hu, Zhao-Wu Tao, Jun-Hua Tian, Yuan-Yuan Pei, Ming-Li Yuan, Yu-Ling Zhang, Fa-Hui Dai, Yi Liu, Qi-Min Wang, Jiao-Jiao Zheng, Lin Xu, Edward C. Holmes, and Yong-Zhen Zhang. A new coronavirus associated with human respiratory disease in china. *Nature*, 579(7798):265–269, 2020. doi:10.1038/s41586-020-2008-3.

59 M Zuker. Suboptimal sequence alignment in molecular biology. alignment with error analysis. *J Mol Biol*, 221(2):403–420, September 1991.

Lifting for Constant-Depth Circuits and Applications to MCSP

Marco Carmosino
Department of Computer Science, Boston University, MA, USA

Kenneth Hoover[1]
Department of Computer Science, University of California, San Diego, CA, USA

Russell Impagliazzo
Department of Computer Science, University of California, San Diego, CA, USA

Valentine Kabanets
School of Computing Science, Simon Fraser University, Burnaby, Canada

Antonina Kolokolova
Department of Computer Science, Memorial University of Newfoundland, St. John's, Canada

Abstract

Lifting arguments show that the complexity of a function in one model is essentially that of a related function (often the composition of the original function with a small function called a *gadget*) in a more powerful model. Lifting has been used to prove strong lower bounds in communication complexity, proof complexity, circuit complexity and many other areas.

We present a lifting construction for constant depth unbounded fan-in circuits. Given a function f, we construct a function g, so that the depth $d+1$ circuit complexity of g, with a certain restriction on bottom fan-in, is controlled by the depth d circuit complexity of f, with the same restriction. The function g is defined as f composed with a parity function. With some quantitative losses, average-case and general depth-d circuit complexity can be reduced to circuit complexity with this bottom fan-in restriction. As a consequence, an algorithm to approximate the depth d (for any $d > 3$) circuit complexity of given (truth tables of) Boolean functions yields an algorithm for approximating the depth 3 circuit complexity of functions, i.e., there are quasi-polynomial time mapping reductions between various gap-versions of AC^0-MCSP. Our lifting results rely on a *blockwise* switching lemma that may be of independent interest.

We also show some barriers on improving the efficiency of our reductions: such improvements would yield either surprisingly efficient algorithms for MCSP or stronger than known AC^0 circuit lower bounds.

2012 ACM Subject Classification Theory of computation → Problems, reductions and completeness; Theory of computation → Circuit complexity

Keywords and phrases circuit complexity, constant-depth circuits, lifting theorems, Minimum Circuit Size Problem, reductions, Switching Lemma

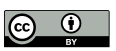
Digital Object Identifier 10.4230/LIPIcs.ICALP.2021.44

Category Track A: Algorithms, Complexity and Games

Funding *Marco Carmosino*: Work supported by NSF Computing Innovation Fellowship.
Kenneth Hoover: Work supported by the Simons Foundation and NSF grant CCF-1909634.
Russell Impagliazzo: Work supported by the Simons Foundation and NSF grant CCF-1909634.
Valentine Kabanets: Work supported in part by an NSERC Discovery grant.
Antonina Kolokolova: Work supported in part by an NSERC Discovery grant.

Acknowledgements We thank the anonymous reviewers for their helpful and insightful comments.

[1] Corresponding author



1 Introduction

Determining the circuit complexity of a given Boolean function (presented by its truth table) is a fundamental algorithmic problem with a long history. One can view the whole area of circuit synthesis as consisting of variants of this problem. Historically, this is an apparently intractable search problem that motivated early thought on the P vs. NP problem; both Karp [21] and Levin [30] were thinking about this problem. It is also a fundamental problem from the point of view of computational complexity, where its complexity in different models has been related to cryptography, learning theory, derandomization, and circuit lower bounds.

Its decision version, known as the Minimum Circuit Size Problem (MCSP), asks to decide for a given truth table of a function f and a parameter s whether there is a circuit with at most s gates computing the function f. The high-level overview is that while any algorithmic improvement for MCSP would have dramatic consequences, thus making MCSP very likely to be extremely difficult, there are also many barriers to showing that MCSP is NP-complete [20, 24, 17, 5, 16, 4, 15, 29].

Intuitively, both algorithms for MCSP and completeness results for MCSP would require a better understanding of circuits, and in particular, would seem to be connected to proving lower bounds for circuit complexity. We could then hope to understand variants of MCSP where we restrict the class of circuits considered. However, MCSP remains mysterious even for the best understood circuit classes, such as constant depth unbounded fan-in circuits, where strong lower bounds are known. One aspect that adds to the mystery is that there is no known general relation between the minimum circuit size problems for various circuit classes, even when one class is stronger than the other, or even when they are almost identical in computational power.

Recently, Ilango [18] made break-through progress by showing NP-completeness for the constant-depth formula size version of MCSP under randomized quasi-polynomial time Turing reductions. He used a *lifting* argument to reduce the depth d formula complexity of a function f to the depth $d+1$ formula complexity of a related function f'. *Lifting arguments* show that the complexity of a function in one model is essentially that of a related function (often the composition of the original function with a small function called a *gadget*) in a more powerful model. Lifting has been used to prove strong lower bounds in communication complexity, proof complexity, circuit complexity and many other areas [26, 12, 25, 10]. As in the work of Ilango, lifting results can also be viewed as a *reduction* from the problem of computing the complexity of functions in the first model to that in the second model.

This intriguing result also raises a number of questions. In particular, what is the strength of lifting arguments as reductions between MCSP variants? Are the restrictions in Ilango's argument (formulas rather than circuits, randomization, quasi-polynomial time, and Turing rather than mapping) essential, or can at least some of these be eliminated? How general is the lifting technique, i.e., could there be natural reductions between these problems that do not use lifting? Our work is a first step towards answering some of these questions.

1.1 Our Results

To answer these questions, we present a lifting construction for constant depth unbounded fan-in circuits that given a function f, constructs a function g, so that the depth $d+1$ complexity of g is controlled by the depth d complexity of f. Our construction is very simple: g is f composed with a suitable sized parity function. Given that there are strong constant depth lower bounds using the properties of the parity function, the form of our construction is quite natural (and can be traced back to the classical Andreev function construction [6]).

However, converting this intuition into a formal lifting theorem is delicate. First, tight lifting for this construction seems to require that we look at circuits with restricted bottom gate fan-in, rather than general circuits. Even with this restriction, we need to get both lower bounds and upper bounds on the complexity of the constructed function g, that are very close together. So we need to take extra care for both parts of the argument. In particular, for the upper bound, it is not sufficient to compose the circuit for f with an optimal depth 2 circuit for the parity; instead, we show that the bottom level of the circuit for f applied to parities has a relatively small depth 2 circuit. For the lower bound, we need a variant of the classical Switching Lemma [13], because we cannot afford to have even a small probability that an input to f is removed by the restriction. Our Switching Lemma variant (Lemma 10) is actually both as easy or easier to prove than the standard one, and implies the standard switching lemma, so could have independent pedagogical value.

Our main lifting theorem (Theorem 18) shows that if a Boolean function $f\colon \{0,1\}^n \to \{0,1\}$ is computable by a size s depth $d+1$ circuit with bottom gate fan-in at most $\log s$ (we term such circuits to be of depth $d + 1/2$), then the new "lifted" function $g = f \circ \oplus_\ell$ (f composed with parity on $\ell < n$ inputs) will have its depth $((d+1) + 1/2)$ circuit complexity sandwiched between $s^{1/2}$ and s^6.

We apply this lifting theorem to give reductions between approximately computing the constant-depth circuit complexity of a given function at depth d and any higher constant depth d'. Because our lifting theorems control the circuit complexity only approximately, we phrase these reductions as reductions between gap-versions of MCSP; these are promise problems to distinguish between functions of circuit complexity at most s_{yes} and of circuit complexity greater than s_{no}, for some parameters $s_{yes} < s_{no}$.

We first observe that the restriction on circuits we have in our lifting theorem does not change their average-case complexity substantially. This gives a reduction from small depth "tolerant" gap-MCSP to tolerant gap-MCSP for larger depths.

Also, by giving a non-trivial size/bottom fan-in trade-off for constant depth circuits, loosely based on the size-width trade-offs in proof complexity [8], we show that hardness for approximation of MCSP for weakly exponential sizes at one depth can be translated to similar hardness for higher depths.

Since the input to MCSP is the entire truth table of the function, the operation we give yields a quasi-polynomial, rather than polynomial time reduction, between MCSP for smaller depths and larger. Under a suitable assumption about the difficulty of MCSP, we show that any natural (in the sense of [20]) polynomial time reduction would either implicitly contain a stronger lifting theorem or yield a subexponential-time algorithm for small-depth MCSP.

Finally, we note that our lifting results are a step towards proving the NP-completeness of AC^0_d-MCSP for any constant depth $d \geq 3$: one just needs to prove that depth 3 circuit complexity is NP-hard to approximate to within the factors needed by our lifting theorems.

1.2 Related Work

The most closely related to our work is the recent result by Ilango [18] that there is a quasi-polytime randomized Turing reduction from the AC^0_d minimum formula size problem to the AC^0_{d+1} minimum formula size problem. The main theorem used for this reduction is a kind of lifting theorem, where for any f and a sufficiently hard to compute g relative to f, the AC^0_{d+1} formula complexity of $f \wedge g$ is close to the AC^0_d formula complexity of f plus the AC^0_{d+1} formula complexity of g. With this, they were able to show that AC^0 minimum formula size problem is NP-hard under such reductions. Their proof crucially relied on the model being *formulas*, whereas our work gives similar depth-increasing reductions for *circuits*, along with some barriers to improving these reductions.

Khot and Saket [22] proved essentially optimal hardness of approximation for DNF-GapMCSP, under the assumption that NP $\not\subseteq$ QuasiP, via a connection shown by Feldman [11] between the gap problem and a hypercube covering problem. Under this same assumption, Hirahara, Oliveira, and Santhanam [15] showed that DNF $\circ$ MOD$_m$-GapMCSP was hard for $O(\log n)$ gaps. Both of these gap problems are known to be easy for $O(n)$ gaps [9, 15]. These results rely on the existing geometric characterization of the classes in question, the former being a union of subcubes, the latter a union of affine subspaces. For general fixed-depth circuits, we no longer have these characterizations; the bottom levels, being CNFs or DNFs, can represent any subset of the hypercube.

Allender and Hirahara [4] show that under modest cryptographic assumptions (the existence of a one-way function), GapMCSP for general circuits and $N^{1-o(1)}$ gaps is NP-intermediate. Their arguments depend heavily on being able to compose functions to large depths. See a recent survey by Allender [1] for more on MCSP-related results.

Remainder of the paper. We give basic definitions and prove our blockwise switching lemma in Section 2. We prove our lifting theorem for the worst-case AC^0 circuit complexity in Section 3, and for the average-case AC^0 circuit complexity in Section 4. We discuss the barriers to improving our lifting reductions in Section 5, and state some open questions in Section 6.

2 Preliminaries

2.1 General

▶ **Definition 1** (Boolean function composition). *The composition of boolean functions* $f : \{0,1\}^n \to \{0,1\}$ *and* $g : \{0,1\}^m \to \{0,1\}$ *is the function* $f \circ g : \{0,1\}^{nm} \to \{0,1\}$ *obtained by dividing the input* z *into* n *consecutive blocks* $z_1, \ldots, z_n$ *of length* m*, applying* g *to each block, and then computing* $f(g(z_1), g(z_2), \ldots, g(z_n))$.

▶ **Definition 2** (Time $t(n)$ many-one reductions). *We say that* $A \leq_m^t B$ *if there is a time* $t = t(n)$*-computable function* $g\colon \{0,1\}^n \to \{0,1\}^{n'}$ *such that* $x \in A \iff f(x) \in B$*. We denote the polynomial* $t(n) = n^{O(1)}$ *by* poly*, and quasipolynomial* $t(n) = n^{(\log n)^{O(1)}}$ *by* qpoly.

2.2 Complexity Measures

We denote the set of n-input Boolean functions by $\mathcal{F}_n$, and the set of all Boolean functions by $\mathcal{F}$. A *complexity measure* is a function $\Lambda : \mathcal{F} \to \mathbb{N}$ that maps Boolean functions to the natural numbers, quantifying some aspect of their complexity relative to a concrete computational model. The most basic such quantity is number of gates in a circuit C, denoted by $|C|$. We assume every circuit has at least one gate for each input bit.

▶ **Definition 3** (Constant-Depth Alternating Circuits: AC^0_d). *The* depth-d alternating circuit complexity *of* f*, denoted* $\mathsf{AC}^0_d(f)$*, is the minimum number of gates in any unbounded-fanin circuit computing* f *where gates are from the set* $\{\wedge, \vee, \neg\}$*, layers alternate, and the depth is at most* d.

▶ **Definition 4** (Bounded Bottom-Fanin Alternating Circuits: $\mathsf{AC}^0_{d+1/2}$). *The* bounded bottom fan-in depth-d alternating circuit complexity *of* f*, denoted* $\mathsf{AC}^0_{d+1/2}(f)$*, is the minimum number* s *of gates in any depth-*$d+1$ *circuit computing* f *with bottom fan-in at most* $\log s$*. Equivalently, it is the minimum over all depth-*$d+1$ *circuits* C *computing* f *of* $\max\{|C|, 2^{w_C}\}$*, where* w_C *denotes the bottom fan-in of* C.

We measure average-case complexity by expanding what counts as "computing" f.

▶ **Definition 5** (Tolerance Operator: $\widehat{\Lambda}$). *Let Λ be a complexity measure. Denote by* $\mathrm{dist}(f,\epsilon)$ *the Hamming ball of radius $\epsilon \cdot 2^n$ around the truth-table of f. The* ϵ-relaxation *of Λ is* $\widehat{\Lambda}[\epsilon] = \min_{f' \in \mathrm{dist}(f,\epsilon)} \Lambda(f')$.

2.3 Meta-Complexity Problems

Every meta-complexity problem is defined relative to a complexity measure.

▶ **Definition 6** (MCSP). *For a complexity measure Λ, Λ-$\mathtt{MCSP} = \{(f,s) \mid \Lambda(f) \leq s\}$.*

▶ **Definition 7** (GapMCSP). *For a complexity measure Λ, Λ-$\mathtt{GapMCSP}_n[s_{yes}, s_{no}]$ is the following promise problem, where f_n denotes an n-variate Boolean function:*

$$\mathcal{Y} = \{f_n \mid \Lambda(f_n) \leq s_{yes}\} \quad \text{and} \quad \mathcal{N} = \{f_n \mid \Lambda(f_n) > s_{no}\}$$

▶ **Definition 8** (Tolerant GapMCSP). *For a complexity measure Λ, $\widehat{\Lambda}[\epsilon_1,\epsilon_2]$-$\mathtt{GapMCSP}_n[s_{yes}, s_{no}]$ is the following promise problem:*

$$\mathcal{Y} = \{f_n \mid (\widehat{\Lambda}[\epsilon_1])(f_n) \leq s_{yes}\} \quad \text{and} \quad \mathcal{N} = \{f_n \mid (\widehat{\Lambda}[\epsilon_2])(f_n) > s_{no}\}$$

2.4 Blockwise Switching Lemma

We will need a strengthening of Håstad's Switching Lemma [13] for the case of *structured* random restrictions that leave exactly one variable unset in every block of variables.

▶ **Definition 9** (Blockwise Restrictions, $\mathcal{B}_n^\ell$). *A binary string of length $n \cdot \ell$ can naturally be divided into n consecutive "blocks" of ℓ bits each. Variables $\{y_{i,j} : i \in [n],\ j \in [\ell]\}$ index into these strings. Denote by $\mathcal{B}_n^\ell$ the set of all restrictions ρ that place* ***exactly one*** $\star$ *in each block of an n-block, ℓ-block-size string. Formally, we have $\rho : [n] \times [k] \to \{0,1,\star\}$ and $\forall i \in [n]\ \exists! j \in [k]$ such that $\rho(i,j) = \star$.*

▶ **Lemma 10** (Blockwise Switching Lemma). *Let φ be a k-CNF on $n \cdot \ell$ variables. For any $s \geq 0$, $\Pr_{\rho \sim \mathcal{B}_n^\ell}[\varphi \restriction_\rho$ cannot be expressed as an 2^s-term s-DNF$] \leq \left(\frac{8k}{\ell}\right)^s$.*

▶ Remark 11. While the proof of Lemma 10 is actually slightly simpler than that of the standard Switching Lemma [27, 7], this Blockwise Switching Lemma implies the standard Switching Lemma (as stated in [7]). A uniformly random subset of pn out of n variables can be chosen as follows: Randomly uniformly permute the n variables, then partition them into pn consecutive disjoint blocks of size $1/p$ each, and, finally, randomly uniformly choose exactly one variable from each of the pn blocks. For each fixed permutation of n variables, Lemma 10 applies with $\ell = 1/p$. We get that the probability that a given k-CNF fails to simplify to an s-DNF when hit with a random restriction that leaves exactly pn variables unset is upper-bounded by $(8pk)^s$.

▶ Remark 12. Our blockwise restrictions are different from Håstad's blockwise random restrictions used in the context of the AC^0 depth hierarchy theorem [13] (later improved to the average-case depth hierarchy theorem in [14]). Håstad's blockwise restrictions were designed to preserve the structure of Sipser's function; ours will allow us to recover a circuit for f from a higher-depth circuit for the composition $f \circ \oplus_\ell$ of f with parities over disjoint blocks of ℓ variables, for any Boolean function f. Because of this, the two random restriction distributions are very different, and the Switching Lemmas that result are quantitatively quite different.

We prove the Switching Lemma (Lemma 10) below, via a modification of the "compression"-based proof of the Switching Lemma due to Razborov [27, 7].

Canonical Decision Trees & Notation

We write assignments to a set of variables $\{x_i | i \in [n]\}$ as functions $\alpha : [n] \to \{0,1\}$. A k-CNF $\varphi(x_1, \ldots, x_n)$ is a conjunction of m clauses, where each clause is a disjunction over at most k literals. A **Decision Tree** is a binary tree where nodes are labeled by variables $x_1, \ldots, x_n$, and leaves and edges are labeled by constants $\{0,1\}$.

To evaluate a Decision Tree on an assignment α, begin at the root, labeled by some x_i. Move down the edge labeled by $\alpha(i)$. Repeat until you arrive at a leaf and report the constant labeling that leaf as the value of the tree.

Given a k-CNF $\varphi(x_1, \ldots, x_n) = C_1 \wedge \cdots \wedge C_m$ we can create a ***Canonical* Decision Tree**. Fix a lexical ordering on variables and use it to sort and de-duplicate clauses; let $i \in [m]$ index the clauses of φ in this sorted order. We define $\mathrm{CDT}(\varphi)$ recursively:

> Transform $C_1 \in \varphi$ to a depth $\leq k$ tree T querying all variables of C_1 in lexical order. For each branch b of T, follow b to induce a partial assignment α_b; set $\varphi_b \leftarrow$ Simplify φ/α_b. If φ_b is empty, terminate b with leaf labeled 1; if φ_b is falsified, terminate b with leaf labeled 0; otherwise, if φ_b is undetermined, extend b with $\mathrm{CDT}(\varphi_b)$.

A restriction is a *partial assignment*: a map $\rho : [n] \to \{0, 1, \star\}$. The result of applying a restriction to a Boolean function f is written $f \restriction_\rho$ where we substitute each occurrence of x_i by $\rho(i)$ for every $\rho(i) \neq \star$. We will need to define restrictions that *extend* other restrictions. Let $\mathcal{E}_D(\rho)$ denote the set of restrictions that are identical to ρ, except for replacing D star locations with constants. Let $\mathcal{E}(\rho)$ denote the set of restrictions that replace *all* $\star$-locations of ρ with constants. We will be concerned with the blockwise restrictions of Definition 9.

Coding and Decoding Large-Depth Restrictions

Suppose all we know about ρ is that it produces a large-depth canonical decision tree when applied to φ. We can witness this with some "long" path σ through the tree. Our code will consist of a restriction $\tilde{\rho}_c$ that extends ρ and a short bitstring "hint" that allows us to implicitly navigate "down" a long path of the CDT and guess ρ by un-setting variables of ρ_c.

Algorithm 1 `ENC`.

```
Let σ be a long path (≥ depth D) through T;
foreach clause along σ, C_i^σ do
    foreach variable η_ij appearing in C_i^σ do
        record the following hint: begin
            η_ij as an index into C_i^σ (log k bits);
            Assignment to η_ij along σ (single bit);
            Is this the last variable queried in C_i^σ? (single bit);
    Record τ_i as an assignment to η_i that falsifies C_i^σ;
ρ_c = ρ ∘ τ_1 ∘ ··· ∘ τ_D
return ρ̃_c ← ρ_c completed to a full assignment uniformly at random, all hints
```

▷ Claim 13 (Decoding from `ENC` output). Suppose $\rho \in \mathcal{B}_n^\ell$ fails to simplify a particular k-CNF φ, so that $\mathrm{CDT}(\varphi \restriction_\rho) \geq D$. Then, $\Pr_{\tilde{\rho}_c \sim \mathtt{ENC}(\rho)} [\mathtt{DEC}(\tilde{\rho}_c) = \rho] \geq \left(\frac{1}{\ell}\right)^{(n-D)}$.

Proof of Claim 13. Fix $\rho \in \mathcal{B}_n^\ell$ and suppose $T = \mathrm{CDT}(\varphi \restriction_\rho)$ has depth $\geq D$. Let σ be a witnessing path of length at least D through T. We'll require some notation; denote by C_i^σ the ith clause traversed along the path σ, in the sense that the recursive CDT construction

Algorithm 2 DEC.

Initialize: $\rho_1 \leftarrow \rho_c = \rho \circ \tau_1 \circ \cdots \circ \tau_D$ and $i \leftarrow 1$
for $i = 1$ *to* D **do**
  Simplify $\varphi_i \restriction_{\rho_i}$;
  Find first falsified clause of $\varphi_i = C_i^\sigma$;
  Read hint to find τ_i and σ_i (stop-bit tells you when to stop).;
  $\rho_{i+1} \leftarrow \rho_i$ with τ_i replaced by σ_i (so $\rho_{i+1} = \rho \circ \sigma_1 \circ \sigma_{i-1} \circ \sigma_i \circ \tau_{i+1} \ldots \tau_D$)
return ρ_D *with* $\sigma_1 \circ \ldots \sigma_D$ *unset, and* $\star$*'s* guessed *uniformly at random for all other blocks*

worked on clause C to produce that section of the decision tree. Note, this may well be smaller than m, due to simplifications applied during construction of the CDT. Further, let σ_i be the section of σ that traverses C_i^σ and let η_i be the variables queried along σ_i. We can think of σ_i as a sequence of assignments to these variables.

Now, consider the operation of DEC on $\tilde{\rho}_c \sim \mathtt{ENC}(\rho)$. First observe that no clauses of φ were falsified by ρ alone, by our assumption that $\mathrm{CDT}(\varphi \restriction_\rho) \geq D$ – a falsified clause would give a depth-1 decision tree with a single 0 leaf. Therefore, any falsified clause is due to variables set by some τ_i or a randomly set variable.

Because the CDT is constructed in lexical-clause-order and ENC follows this order, the *first* falsified clause of $\varphi \restriction_{\tilde{\rho}_c}$ must be C_1^σ. We wish to recover which variables τ_1 set; the trick is that now we know they must reside in a uniquely identified clause of at most k variables. So, we spend $\log k$ bits of the hint per variable to name *which* variables of C_1^σ were along σ and thus set in τ_1.

Iterating this argument, we see that lexical ordering of the canonical decision tree ensures recovery of D $\star$ locations of ρ. So, after running the main loop of DEC on $\tilde{\rho}_c$ we have a candidate that matches ρ exactly in D blocks. For each remaining block, DEC will simply guess at random which variable in the block was a $\star$ in ρ. Each block has ℓ bits, so we have a $(1/\ell)$ chance of guessing correctly – that is, in agreement with the original location of the $\star$ in ρ. The number of blocks that must be guessed (instead of recovered using deterministic decoding, DEC) is $(n - D)$. Every guess must be correct to successfully decode ρ. This gives the claimed probability of decoding. ◁

Given instead a *random* completion ρ_r of blockwise restriction ρ and a *random* hint h_r, can any algorithm decode ρ? We can upper bound this probability.

▷ Claim 14 (Decoding from random information). For any algorithm $\mathcal{A}$, for every blockwise restriction ρ, $\Pr_{\rho_r \sim \mathcal{E}(\rho)}[\mathcal{A}(\rho_r, h_r) = \rho] \leq \left(\frac{1}{\ell}\right)^n$.

Proof of Claim 14. The hint h_r is clearly useless, because it is a random string. Furthermore, the random variables ρ_r and ρ are conditionally independent, given that ρ_r is a randomly sampled completion of ρ. This means that observing ρ_r provides *no information* regarding the $\star$-locations of ρ. Therefore, no algorithm can do better than to randomly guess which location, in each block, was a star, for every block of the received ρ_r. There are n blocks of ℓ bits each and every guess must be correct, for the overall probability $(1/\ell)^n$. ◁

Completing the proof

We are now ready to prove Lemma 10, re-stated below in a more general form.

▶ **Lemma 15** (Blockwise Switching Lemma). *Let φ be a k-CNF. Pick ρ from $\mathcal{B}_n^\ell$ uniformly at random. Then* $\Pr[\mathrm{CDT}(\varphi \restriction_\rho) \geq D] \leq \left(\frac{8k}{\ell}\right)^D$.

Proof. We can lower-bound the probability of decoding from random completion ρ_r: if we are lucky enough that the randomly sampled completion agrees with ρ_c in the "special" blocks set by ENC, then we can significantly narrow down the number of blocks whose $\star$ must be guessed at random! That is, the non-trivial probability of recovery for DEC can be exploited. Formally,

$$\begin{aligned}\left(\frac{1}{\ell}\right)^n &\geq \Pr[\mathtt{DEC}(\rho_r, h_r) = \rho] && \text{(by Claim 14)}\\ &\geq \Pr[\mathrm{CDT}(\varphi \restriction_\rho) \geq D] \times \Pr[h_r = h] \times \Pr[\rho_r \text{ extends } \rho_c] \times \Pr[\mathtt{DEC} \text{ decodes } \star\text{'s}]\end{aligned}$$

Taking each event in turn:

1. $|h| = D(\log(k) + 2)$ so there are $(4k)^D$ possible strings. Flipping h_r uniformly at random, $\Pr[h_r = h] = (4k)^{-D}$.
2. To extend ρ_c, the randomly chosen ρ_r must agree in D locations. One of these settings is correct, so $\Pr[\rho_r \text{ extends } \rho_c] = 2^{-D}$.
3. Given a correct hint and randomly completed ρ_c, the probability of DEC recovering ρ is $\left(\frac{1}{\ell}\right)^{(n-D)}$ by Claim 13.

Plugging in, we get

$$\left(\frac{1}{\ell}\right)^n \geq \Pr[\mathrm{CDT}(\varphi \restriction_\rho) \geq D] \times (4k)^{-D} \times 2^{-D} \times \left(\frac{1}{\ell}\right)^{(n-D)}.$$

The proof of the lemma follows. ◀

3 Constant-Depth GapMCSP Reductions

The focus of this section is "hardness lifting" for circuits of depth $(d+1)$ to circuits of depth $(d+2)$, and its applications to GapMCSP for the respective classes. Theorem 18 shows how to lift hardness for bounded-fan-in AC^0 circuits from depth d to depth $d+1$ (also bounded fan-in). Here, a function of a not much larger size yet higher depth is constructed by replacing input variables of the original function by disjoint relatively small parities. This theorem is then applied to reduce GapMCSP for AC^0_d circuits to GapMCSP for AC^0_{d+1} circuits.

The reduction proceeds in three steps, with the middle step potentially repeated multiple times for a larger depth increase. The first step converts unbounded bottom fan-in circuits of depth $(d+1)$ to bounded (by log of the circuit size) fan-in circuits of the same depth, at the cost of increasing the size from s to $2^{O(\sqrt{n \log n \log s})}$; see Corollary 17. This rebalancing only needs to be done once.

The second step, which relies on the hardness lifting theorem, is the quasi-polynomial time reduction from GapMCSP for bounded bottom fan-in circuits of depth $(d+1)$, to GapMCSP for bounded bottom fan-in circuits of depth $(d+2)$. The quasi-polynomial running time of this reduction comes from the blow-up in the size of the output truth table of the new function. Then, we show that for this setting GapMCSP for depth $d+1$ circuits reduces to GapMCSP for depth $d+2$ circuits (both bounded bottom fan-in), with a small loss in the gap size. See Theorem 19 for the exact statement.

The last step is a polytime reduction from GapMCSP for bounded bottom fan-in circuits of depth $(d+2)$, to GapMCSP for unbounded bottom fan-in circuits of the same depth; see Theorem 21.

3.1 Depth $d+1$ to $d+1/2$

▶ **Lemma 16** (Fanin vs Size Tradeoff). *For any $d \geq 3$, let C be any depth-d size-s circuit over n inputs. Then, for any $w \geq 1$, there is an equivalent depth-d circuit C' with bottom fan-in at most w, and the size at most $s^{(4n\log n)/w}$.*

Proof. Assume WLOG that all the bottom gates of C are disjunctions. We will recursively define a decision tree T such that each leaf ℓ is associated with a restriction ρ_ℓ resulting in $C\restriction_{\rho_\ell}$ having bottom fan-in at most w.

Initially, T consists of a single leaf node corresponding to the empty restriction. While there is a leaf v in T corresponding to a restriction ρ such that $C\restriction_\rho$ has some nonempty set S of bottom gates of fan-in greater than w, do the following. Let $t = |S| \leq s$. Let z be the literal that occurs in the most gates of S. Since there are more than tw literal occurrences among the gates in S and there are $2n$ literals, z must appear in more than $(tw)/(2n)$ bottom gates. Branch on z, with the left child v_1 of v corresponding to $z=1$, and the right child v_0 to $z=0$. Note that the restriction corresponding to v_1 satisfies all bottom gates containing z, and the restriction corresponding to v_0 reduces their fan-in by 1.

Every left branching we take in the decision tree results in t shrinking by more than a factor $\left(1-\frac{w}{2n}\right)$. So after k left branchings, there are fewer than $t\left(1-\frac{w}{2n}\right)^k$ large fan-in gates left. Setting $k = (2n/w)\ln s$, we have that after k left branchings there are no large fan-in gates left. If $k \geq n/2$ (i.e., $w \leq 4\ln s$), then we can use the trivial upper bound 2^n on the size of T; note that, in this case, $2^n \leq 2^{4n\cdot(\ln s)/w}$, as required.

Otherwise, for $k < n/2$, we can upper-bound the size of T as follows. Since each branch of T is of length at most n, and it may contain at most k left branchings, we get that the size of T is at most

$$\sum_{r=0}^{k}\binom{n}{r} \leq k\cdot\binom{n}{k} \leq k\cdot\left(\frac{ne}{k}\right)^k \leq k\cdot\left(\frac{we}{2\ln s}\right)^k \leq 2^{(1.5)k\cdot\log(w/\ln s)} \leq s^{\frac{3n\log n}{w}},$$

where for the last inequality we used the definition of k and the bound $(w/\ln s) \leq w \leq n$.

Suppose without loss of generality that the top gate of C is a disjunction, i.e., $C = \bigvee_i \bigwedge_j g_{i,j}$. We can rewrite $C(x)$ as $\bigvee_{\text{leaves } \ell\in T}\left(\phi(x,\rho_\ell)\wedge C\restriction_{\rho_\ell}(x)\right)$, where, for a fixed restriction ρ_ℓ, the formula $\phi(x,\rho_\ell)$ indicates whether x is consistent with ρ_ℓ (i.e., whether x ends up at leaf ℓ of our decision tree T). It is easy to see that $\phi(x,\rho_\ell)$ can be written as a conjunction of at most n literals.

As written, the circuit above is a depth-$(d+2)$ size at most $(1+|T|+|T|\cdot s)$ circuit with fan-in at most w. By distributivity, we can rewrite each $\phi(x,\rho_\ell)\wedge C\restriction_{\rho_\ell}(x)$ as $\bigvee_i\left(\phi(x,\rho_\ell)\wedge\bigwedge_j\left(g_{i,j}\restriction_{\rho_\ell}(x)\right)\right)$. Plugging this into (3.1), we obtain a depth-d circuit C' with fan-in at most w, computing the same function as C, and the size of C' is at most $|T|\cdot s \leq s^{(4n\log n)/w}$, as required. ◀

▶ **Corollary 17** (Depth $(d+1)\to(d+1/2)$). *For any $d\geq 2$, n, and s_{yes}, s_{no} such that $\log s_{yes} \leq \frac{\log^2(s_{no}/4)}{n\log n}$, we have*

$$\mathsf{AC}^0_{d+1}\text{-}\mathtt{GapMCSP}_n[s_{yes},s_{no}] \leq^{\mathsf{poly}}_m \mathsf{AC}^0_{d+1/2}\text{-}\mathtt{GapMCSP}_n[2^{4\cdot\sqrt{n(\log n)(\log s_{yes})}},s_{no}]$$

with the identity functions as a reduction.

Proof. The "NO→NO" case is immediate: if $f\colon\{0,1\}^n\to\{0,1\}$ doesn't have size-s_{no} circuits with no restriction on the bottom fan-in, then f doesn't have size-s_{no} circuits with restricted bottom fan-in.

For the "YES→YES" case, we apply Lemma 16 to a depth-$(d+1)$ size-s_{yes} circuit for f, with $w = \sqrt{n(\log n)(\log s_{yes})}$. This results in a circuit for f of size at most $2^{4\cdot\sqrt{n(\log n)(\log s_{yes})}}$, with bottom fan-in at most $\sqrt{n(\log n)(\log s_{yes})}$. ◀

3.2 Depth $d + 1/2$ to $(d+1) + 1/2$

▶ **Theorem 18** (Hardness lifting). *Let f have $\mathsf{AC}^0_{d+1/2}$ circuit complexity s. Fix $s_0 > 0$. Then there is a function f' on $n' = n \cdot 16 \log s_0$ inputs with $\mathsf{AC}^0_{(d+1)+1/2}$ circuit complexity s' where $s' \le 2s2^{\sqrt{16 \log s \log s_0}}\sqrt{16 \log s \log s_0}$. Moreover, if $s_0 < \sqrt{s/3}$, then $s_0 \le s'$.*

Proof. The construction is as follows: given the truth table of $f\colon \{0,1\}^n \to \{0,1\}$, output the truth table of $f' = f \circ \oplus_\ell$ for $\ell = 16 \log s_0$. This takes time $2^{n\ell} = N^{16 \log s_0} \le N^{O(\log N)}$, quasi-polynomial in N since $s_0 \le N$. We argue the correctness next.

Bounding s' from below. Note that the parameter ℓ must be sufficiently larger than $\log s_0$ so that we can apply the Blockwise Switching Lemma to a depth-$(d+2)$ size-s_0 circuit with bottom fan-in $\log s_0$ that presumably computes $f \circ \oplus_\ell$ to obtain a depth-$(d+1)$ size-s circuit with bottom fan-in $\log s$ that computes f. We prove that if f' has a $\mathsf{AC}^0_{(d+1)+1/2}$ of size s_0, then f has a $\mathsf{AC}^0_{d+1/2}$ circuit of size $s \le 3(s_0)^2$.

Suppose $f \circ \oplus_\ell$ has a depth-$(d+2)$ circuit C' of size s_0 and bottom fan-in at most $\log s_0$. We shall hit C' with a blockwise random restriction ρ, where the blocks are the inputs to each $\oplus_\ell$. Since exactly one bit is left unset in each block, $C' \restriction_\rho$ computes f with some of the input bits potentially negated. For $C' \restriction_\rho$ to simplify to a depth-$(d+1)$ circuit with bottom fan-in at most $k \le \log(3s_0^2) \le \log s$, we need to argue that there exists a blockwise restriction ρ which makes every depth-2 bottom circuit of C' into a decision tree of depth at most k. By the Blockwise Switching Lemma (Lemma 10), this is implied if $s_0 \left(\frac{8 \log s_0}{\ell}\right)^k < 1$, which is equivalent to $2^{\log s_0 - k} < 1$, for our choice of $\ell = 16 \log s_0$. Thus, setting $k = \log s_0 + 1$ satisfies this inequality. Moreover, each bottom CNF or DNF of C' is turned into a DNF or CNF with 2^k clauses. So the size of $C' \restriction_\rho$ is at most $s_0 + s_0 \cdot 2^k \le 3(s_0)^2 \le s$, as required.

Bounding s' from above. Next we need to show that if f has a small depth-$(d+1/2)$ circuit, then $f \circ \oplus_\ell$ has a small depth-$(d+1+1/2)$ circuit. Note that computing the ℓ-bit parities by naive depth-2 circuits of size 2^ℓ is prohibitively expensive, as this would make the size of the new circuit for $f \circ \oplus_\ell$ at least $(s_0)^{16} > s'$, for our choice of $\ell = 16 \log s_0$ (which was dictated by the "NO→NO" case analysis above). Instead we will compute each $\oplus_\ell$ by a depth-3 circuit, as a parity of parities, adapting the standard construction of optimal size-$(\ell 2^{\sqrt{\ell}})$ depth-3 circuits. To get a final circuit for $f \circ \oplus_\ell$ to be of depth $d+1+1/2$, we will need to carefully balance the parameters of our partition of ℓ bits into ℓ_1 blocks of size ℓ_2 each, for ℓ_1 and ℓ_2 such that $\ell = \ell_1 \cdot \ell_2$.

Suppose f has a depth-$(d+1)$ circuit C of size s and bottom fan-in at most $\log s$, with all negations at the leaves; this at most doubles the size. Without loss of generality, assume that the bottom layer of gates consists of disjunctions with fan-in $\log s$. To obtain a circuit for $f \circ \oplus_\ell$, we will compose $\oplus_\ell$ with each of the bottom CNFs of C. Consider a particular CNF $h_i = \bigwedge_{j=1}^{k} g_{i,j}$ at the bottom of C, where $k \le s$ and each $g_{i,j}$ is a disjunction of at most $\log s$ literals.

For ℓ_1 to be chosen later, let $\ell_2 = \ell/\ell_1$. Using the trivial 2^{ℓ_1}-size CNF for computing $\oplus_{\ell_1}$, we can compute each $g_{i,j} \circ \oplus_{\ell_1}$ by an OR-AND-OR circuit, where the top OR gate has fan-in $\log s$ and the AND gates each have fan-in 2^{ℓ_1}. By distributivity, we can rewrite $g_{i,j} \circ \oplus_{\ell_1}$ as a CNF with $2^{\ell_1 \log s}$ clauses, each of width at most $\ell_1 \log s$.

Since C is a layered circuit, we can merge this CNF into h_i to obtain a depth-2 circuit computing $h_i \circ \oplus_{\ell_1}$. Finally, composing this with the DNF for $\oplus_{\ell_2}$, we get a depth-3 circuit with bottom fan-in ℓ_2 computing $h_i \circ \oplus_\ell$. Replacing each h_i in C with circuits constructed in this way, we obtain a depth-$(d+2)$ circuit for $f \circ \oplus_\ell$ with bottom fan-in ℓ_2. The subcircuit for computing each $g_{i,j} \circ \oplus_\ell$ is of size at most $\sigma = 1 + 2^{\ell_1 \log s} + 2^{\ell_2} \cdot \ell_1 \log s$. So the total size of the circuit for $f \circ \oplus_\ell$ is at most $s + s \cdot \sigma = s(\sigma + 1)$. If we set $\ell_2 = \sqrt{\ell \log s}$ and $\ell_1 = \sqrt{\frac{\ell}{\log s}}$, then the total size is at most

$$s\left(2 + 2^{\sqrt{\ell \log s}} + 2^{\sqrt{\ell \log s}} \cdot \sqrt{\ell \log s}\right) \leq (2s) \cdot 2^{\sqrt{\ell \log s}} \cdot \sqrt{\ell \log s} \leq s'.$$

Since the bottom fan-in is at most $\sqrt{\ell \log s} \leq \log s'$, this concludes the proof. ◀

▶ **Theorem 19** (Depth $(d + 1/2) \to ((d+1) + 1/2)$ GapMCSP)**.** *For any $d \geq 1$, n, s_{yes}, s'_{yes}, s'_{no}, and s_{no} such that $s_{yes} < s_{no}$, $s'_{yes} < s'_{no}$, $s_{no} \geq 3(s'_{no})^2$ and*

$$s'_{yes} \geq 2(s_{yes}) 2^{\sqrt{16(\log s_{yes})(\log s'_{no})}} \sqrt{16(\log s_{yes})(\log s'_{no})},$$

we have $\mathsf{AC}^0_{d+1/2}\text{-}\mathtt{GapMCSP}_n[s_{yes}, s_{no}] \leq^{\mathsf{qpoly}}_m \mathsf{AC}^0_{(d+1)+1/2}\text{-}\mathtt{GapMCSP}_{n'}[s'_{yes}, s'_{no}]$, *where* $n' = 16n \log s'_{no} \leq O(n^2)$.

Proof. We use the construction in Theorem 18 as the reduction function, with $s_0 = s'_{no}$. For the YES $\to$ YES side, if $AC^0_{d+1/2}(f) \leq s_{yes}$, then

$$\mathsf{AC}^0_{d+1+1/2}(f') \leq 2(s_{yes}) 2^{\sqrt{16(\log s_{yes})(\log s'_{no})}} \sqrt{16(\log s_{yes})(\log s'_{no})}$$

as desired. For the NO $\to$ NO side, if $AC^0_{d+1/2}(f) > s_{no}$, then $\mathsf{AC}^0_{d+1+1/2}(f') \geq s_0 = s'_{no}$. ◀

▶ Remark 20. If we apply this to succinct MCSP, we actually get a polytime reduction instead; constructing the naïve $f \circ \oplus_\ell$ circuit given a circuit for f takes polytime, it just makes the truth table too large.

3.3 Depth $d + 1/2$ to $d + 1$

▶ **Theorem 21** (Depth $(d + 1/2) \to (d+1)$)**.** *For any $d \geq 1$, $n, s_{yes}, s_{no}, s'_{yes}, s'_{no}$, such that $s_{yes} < s_{no}$, $s'_{yes} < s'_{no}$, $s_{no} \geq (s'_{no})^5$ and $s'_{yes} \geq 2(s_{yes})^3$, we have*

$$\mathsf{AC}^0_{d+1/2}\text{-}\mathtt{GapMCSP}_n[s_{yes}, s_{no}] \leq^{\mathsf{poly}}_m \mathsf{AC}^0_{d+1}\text{-}\mathtt{GapMCSP}_{2n}[s'_{yes}, s'_{no}]$$

Proof. The reduction is as follows: given the truth table of $f\colon \{0,1\}^n \to \{0,1\}$, output the truth table of $g = f \circ \oplus_2$. The size of the input for g is $2n$. The runtime of the reduction is $\mathsf{poly}(N)$. Next we argue the correctness of this reduction.

NO $\to$ NO. Suppose $f \circ \oplus_2\colon \{0,1\}^{2n} \to \{0,1\}$ is computable by a size-s'_{no} circuit C' of depth $d+1$. Without loss of generality, we may assume that the bottom gates of C' are ANDs. We will hit C' with a random blockwise restriction ρ. Consider a particular bottom AND-gate of fan-in t, for some $1 \leq t \leq n$. Since each block in a blockwise restriction is of size two, there must be at least $t/2$ variables from distinct blocks that feed into this AND gate. Each one of these variables will be chosen as a non-star variable by ρ with probability $1/2$, and then independently set to 0 with probability $1/2$. This would simplify the AND gate to the constant 0, with probability $1/4$. This happens independently for each of these

$t/2$ variables. Thus the probability that the AND gate of fan-in at least t survives a random restriction is at most $(3/4)^{t/2}$. By the union bound, the probability that any such AND gate survives is at most $s'_{no} \cdot (3/4)^{t/2}$, which is less than 1 for $t = 5(\log s'_{no})$. Thus there exists a blockwise restriction ρ which simplifies C' to a depth-$(d+1)$ circuit computing f, with size at most $s'_{no} \le s_{no}$ and bottom fan-in at most $5(\log s'_{no}) \le \log s_{no}$.

YES $\to$ YES. Suppose $f\colon \{0,1\}^n \to \{0,1\}$ is computable by a size-s_{yes} circuit C of depth $d+1$, with bottom fan-in at most $\log s_{yes}$. WLOG, assume the bottom gates of C are ANDs. Note that we can express the XOR and the negated XOR of two variables as the following 2-CNFs:

$$y \oplus z = (\bar{y} \vee \bar{z}) \wedge (y \vee z) \qquad \text{and} \qquad \neg(y \oplus z) = (\bar{y} \vee z) \wedge (y \vee \bar{z}).$$

Replacing the input literals of C by these circuits for (possibly negated) $\oplus_2$, and merging the bottom AND gate of C with the top AND gate of these parity circuits, we get a depth-$(d+2)$ circuit C' for $f \circ \oplus_2$, with 2-CNFs on $t = (2 \log s_{yes})$ clauses as the bottom depth-2 sub-circuits. By distributivity, we can rewrite each 2-CNF on t clauses as a t-DNF on 2^t terms. Then merge the OR gates of these DNFs with the OR gates at the preceding level in C', obtaining an equivalent depth-$(d+1)$ circuit C'' for $f \circ \oplus_2$, of size at most $s_{yes} + s_{yes} \cdot (s_{yes})^2 \le 2(s_{yes})^3 \le s'_{yes}$ (and bottom fan-in at most $(2 \log s_{yes}) \le \log s'_{yes}$). ◀

3.4 Combining the steps: Depth $d+1$ to $d+c$ for any constant $c > 1$

The reduction in Theorem 19 can be repeated multiple times, resulting in the overall reduction lifting hardness to constantly many levels. The following theorem shows how the parameters evolve over all steps of the reduction.

▶ **Theorem 22** (Depth $(d+1) \to (d+c)$)**.** *For any $d \ge 2$, $c > 1$, $n \ge n_0(\alpha, \delta, c)$, and $0 < \alpha < \delta < 1$ where $1 + \alpha < 2\delta$, we have*

$$\mathsf{AC}^0_{d+1}\text{-}\mathtt{GapMCSP}_n[2^{n^\alpha}, 2^{n^\delta}] \le_m^{\mathsf{qpoly}} \mathsf{AC}^0_{(d+c)}\text{-}\mathtt{GapMCSP}_{n'}[2^{(n')^\beta}, 2^{(n')^\gamma}],$$

where $n' = n^{(c-1)\delta+1}$, $\gamma \approx \frac{1}{c-1}$, $\beta \approx \frac{1}{c-1} - \frac{1}{(c-1)2^{c-1}} \cdot (1 - \frac{1+\alpha}{2\delta})$.

Proof. As outlined at the beginning of the section, we will create this reduction via composing the reductions in Corollary 17 and Theorems 19 and 21. Let $a = \frac{1+\alpha+\frac{\log\log n+4}{\log n}}{2}$.

Step 1. $\mathsf{AC}^0_{d+1}\text{-}\mathtt{GapMCSP}_n[2^{n^\alpha}, 2^{n^\delta}] \le_m^{\mathsf{poly}} \mathsf{AC}^0_{d+1/2}\text{-}\mathtt{GapMCSP}_n[2^{n^a}, 2^{n^\delta}]$
This follows immediately from Corollary 17 with $s_{yes} = 2^{n^\alpha}$ and $s_{no} = 2^{n^\delta}$.

Step 2.
$\mathsf{AC}^0_{d+1/2}\text{-}\mathtt{GapMCSP}_n[2^{n^a}, 2^{n^\delta}] \le_m^{\mathsf{qpoly}} \mathsf{AC}^0_{(d+c-1)+1/2}\text{-}\mathtt{GapMCSP}_{n^{(c-1)\delta+1}/2}[\exp_2\left(5n^{\frac{a+(2^{c-1}-1)\delta}{2^{c-1}}}\right),$
$\exp_2(\frac{n^\delta}{2^{c-1}} - \frac{2^{c-1}-1}{2^{c-1}} \log 3)]$
We will show each of n, s_{yes}, and s_{no} map to the corresponding values after $c-1$ applications of the reduction in Theorem 19. Define $n^{(i)}$, $s^{(i)}_{yes}$, and $s^{(i)}_{no}$ to be each value after applying i iterations of the reduction, with $n^{(0)}$, $s^{(0)}_{yes}$, and $s^{(0)}_{no}$ set to the initial values.

We will first show that $s^{(i)}_{no} = 2^{\frac{n^\delta}{2^i} - \frac{2^i-1}{2^i}\log 3}$; this is true for $i = 0$, and for larger i we have $s^{(i+1)}_{no} = \sqrt{\frac{s^{(i)}_{no}}{3}} = 2^{\frac{n^\delta}{2^{i+1}} - \frac{2^i-1}{2^{i+1}}\log 3 - \frac{\log 3}{2}} = 2^{\frac{n^\delta}{2^{i+1}} - \frac{2^{i+1}-1}{2^{i+1}}\log 3}$.

Next, we show that $n^{(i)} = 16^i n \prod_{j=1}^{i} [\frac{n^\delta}{2^j} - \frac{2^j-1}{2^j} \log 3]$; via padding, we can increase the number of variables to $n^{(c-1)\delta+1}/2$ at the end. Again, this is true for $i = 0$. For larger i,

$$n^{(i+1)} = 16 n^{(i)} \log s_{no}^{(i+1)} = 16^{i+1} n \prod_{j=1}^{i+1} \left[\frac{n^\delta}{2^j} - \frac{2^j - 1}{2^j} \log 3 \right].$$

Finally, for $s_{yes}^{(i)}$, we show that after i iterations of the Theorem 19 reduction, $s_{yes} = 2^{n^a}$, $s_{no} = 2^{n^\delta}$ would be mapped to at most $s'_{yes} = \exp_2(5n^{\frac{a+(2^i-1)\delta}{2^i}})$. For $i > 0$, assuming $s_{yes}^{(i)} \leq \exp_2(5n^{\frac{a+(2^i-1)\delta}{2^i}})$, we have $s_{yes}^{(i+1)}$ is at most

$$\exp_2\left(1 + 5n^{\frac{a+(2^i-1)\delta}{2^i}} + \sqrt{\frac{80}{2^i} n^{\frac{a+(2^i-1)\delta}{2^i}} (n^\delta - \Theta(2^i))} + O(\log n)\right) \leq \exp_2\left(5n^{\frac{a+(2^{i+1}-1)\delta}{2^{i+1}}}\right),$$

fixing n_0 sufficiently large. For $i = 0$, note that $n^a < 5n^{\frac{a+(2^0-1)\delta}{2^0}}$.

Step 3.

$\mathsf{AC}^0_{(d+c-1)+1/2}$-$\mathtt{GapMCSP}_{n^{(c-1)\delta+1}/2}[\exp_2\left(5n^{\frac{a+(2^{c-1}-1)\delta}{2^{c-1}}}\right), \exp_2(\frac{n^\delta}{2^{c-1}} - \frac{2^{c-1}-1}{2^{c-1}} \log 3)] \leq_m^{\mathsf{poly}}$
AC^0_{d+c}-$\mathtt{GapMCSP}_{n^{(c-1)\delta+1}}[2^{n^\beta}, 2^{n^\gamma}]$
This follows immediately from Theorem 21, setting

$$s_{yes} = \exp_2\left(5n^{\frac{a+(2^{c-1}-1)\delta}{2^{c-1}}}\right) \text{ and } s_{no} = \exp_2\left(\frac{n^\delta}{2^{c-1}} - \frac{2^{c-1}-1}{2^{c-1}} \log 3\right). \quad \blacktriangleleft$$

4 Constant-Depth Tolerant GapMCSP Reductions

We will show an analogous "hardness lifting" reduction from the GapMCSP problem for average-case circuits of depth d to depth $d+1$.

In this average case setting, instead of applying the machinery of Lemma 16, we can instead make use of the observation that bottom gates of large fan-in are almost always equal to their bias; see Theorem 23. Thus we get smaller gaps on the output side of the reduction, at a small cost to the tolerance parameter.

4.1 Tolerant depth $d+1$ to $d+1/2$ and reverse

▶ **Theorem 23** (Tolerant depth $(d+1) \to (d+1/2)$). *For any $0 \leq \epsilon_1, \epsilon_2 < 1/2$, $d \geq 1$, $n \geq 1$, and $s_{yes} < s_{no}$, we have*

$$\widehat{\mathsf{AC}}^0_{d+1}[\epsilon_1, \epsilon_2]\text{-}\mathtt{GapMCSP}_n[s_{yes}, s_{no}] \leq_m^{\mathsf{poly}} \widehat{\mathsf{AC}}^0_{d+1/2}[\epsilon_1 + 1/n, \epsilon_2]\text{-}\mathtt{GapMCSP}_n[(s_{yes})^2, s_{no}]$$

with the identity functions as a reduction.

Proof. The "NO"→"NO" case is obvious. For the "YES"→"YES" case, suppose C is a depth $d+1$ circuit of size s_{yes} that disagrees with f on at-most an ϵ_1-fraction of inputs. For each bottom gate of C with fan-in larger than $2 \log |C|$, replace the gate with a 1 if it is an OR, or a 0 if it is an AND. Call this new circuit with the replaced gates C'. For a uniformly-random sampled input, any of the replaced gates would disagree with this bit with probability at most $|C|^{-2}$, and so the probability C' disagrees with C on a uniformly-random input is at most $1/|C|$, via a union bound. Since $|C| \geq n$, this is at most $1/n$, and so C' disagrees with f on at most an $(\epsilon_1 + 1/n)$-fraction of inputs. Note that $|C'| \leq |C| \leq (s_{yes})^2$ and the bottom fan-in of C' is at most $2 \log s_{yes} \leq \log(s_{yes})^2$, as required. $\blacktriangleleft$

▶ **Theorem 24** (Tolerant depth $(d+1/2) \to (d+1)$)**.** *For any $0 \le \epsilon_1, \epsilon_2 < 1/2$, $d \ge 1$, $n \ge 1$, $s_{yes}, s_{no}, s'_{yes}, s'_{no}$ such that $s_{yes} < s_{no}$, we have, via the the identity functions as a reduction,*

$$\widehat{\mathsf{AC}}^0_{d+1/2}[\epsilon_1, \epsilon_2 + 1/n]\text{-}\mathtt{GapMCSP}_n[s_{yes}, s_{no}] \le^{\mathsf{poly}}_m \widehat{\mathsf{AC}}^0_{d+1}[\epsilon_1, \epsilon_2]\text{-}\mathtt{GapMCSP}_n[s_{yes}, \sqrt{s_{no}}].$$

Proof. The "YES"→"YES" case is obvious. For the "NO"→"NO" case, let C' be depth-$(d+1)$ circuit of size at most $s'_{no} = \sqrt{s_{no}}$ that ϵ_2-approximates f. As in the proof of Theorem 23 above, we replace by constants all bottom gates of C' that have fan-in larger than $2\log|C'|$, getting a new circuit C that computes f on all but at most $\epsilon_2 + (1/n)$ fraction of inputs. The size of C is at most $s'_{no} \le s_{no}$, and the bottom fan-in is at most $2\log s_{no}^{1/2} = \log s_{no}$, as required. ◀

4.2 Tolerant depth $d + 1/2$ to $(d+1) + 1/2$

▶ **Theorem 25** (Tolerant depth $(d+1/2) \to ((d+1)+1/2)$)**.** *For any $d \ge 1$, $n \ge 1$, $0 \le \epsilon_1, \epsilon_2 < 1/2$, s_{yes}, s'_{yes}, s'_{no}, and s_{no} such that $s_{yes} < s_{no}$, $s'_{yes} < s'_{no}$, $s_{no} \ge 3(s'_{no})^2(\epsilon_2 n + 1)$ and $s'_{yes} \ge 2(s_{yes})2^{\sqrt{16(\log s_{yes})(\log s'_{no})}}\sqrt{16(\log s_{yes})(\log s'_{no})}$, we have*

$$\widehat{\mathsf{AC}}^0_{d+1/2}[\epsilon_1, \epsilon_2 + 1/n]\text{-}\mathtt{GapMCSP}_n[s_{yes}, s_{no}] \le^{\mathsf{qpoly}}_m \widehat{\mathsf{AC}}^0_{(d+1)+1/2}[\epsilon_1, \epsilon_2]\text{-}\mathtt{GapMCSP}_{n'}[s'_{yes}, s'_{no}],$$

where $n' = 16n\log s'_{no} \le O(n^2)$.

Proof. We shall use the same reduction as in Theorem 19, outputting $f \circ \oplus_\ell$ on input f, where $\ell = 16\log s'_{no}$.

NO → NO. Let C' be a depth-$(d+2)$ circuit of size s'_{no} and bottom fan-in at most $\log s'_{no}$ that ϵ_2-approximates $f \circ \oplus_\ell$. We shall hit C' with a blockwise random restriction, as before. Here, we simultaneously require that $C' \restriction \rho$ simplifies to a depth-$(d+1)$ circuit with bounded bottom fan-in, and that its truth table is $(\epsilon_2 + 1/n)$-close to (some fixed shift of) f.

For any $x \in \{0,1\}^n$ and a blockwise restriction ρ, we denote by $\langle x, \rho\rangle$ the $(n \cdot \ell)$-tuple of bits obtained by placing x in the star positions of ρ. Clearly, picking x and ρ uniformly at random results in $\langle x, \rho\rangle$ being the uniform distribution on $\{0,1\}^{n\cdot\ell}$. By our assumption on C', we have $\mathsf{Exp}_{x,\rho}[C'(\langle x,\rho\rangle) \ne (f \circ \oplus_\ell)(\langle x,\rho\rangle)] \le \epsilon_2$. By Markov's Inequality,

$$\Pr_\rho\left[\mathsf{Exp}_x[C'(\langle x,\rho\rangle) \ne (f\circ\oplus_\ell)(\langle x,\rho\rangle)] > \epsilon_2 + \frac{1}{n}\right] < \frac{\epsilon_2}{\epsilon_2 + (1/n)}.$$

Hence, with probability at least $(\epsilon_2 \cdot n + 1)^{-1}$, for a randomly chosen blockwise restriction ρ

$$\begin{aligned}\mathsf{Exp}_x[C'(\langle x,\rho\rangle) \ne (f\circ\oplus_\ell)(\langle x,\rho\rangle)] &= \mathsf{Exp}_x[C' \restriction_\rho (x) \ne (f\circ\oplus_\ell)\restriction_\rho (x)] \\ &= \mathsf{Exp}_x[C' \restriction_\rho (x) \ne f(x \oplus b^\rho)] \le \epsilon_2 + \frac{1}{n},\end{aligned}$$

for $b^\rho = b_1 \dots b_n \in \{0,1\}^n$ such that b_i is the parity of assigned values in the ith block of ρ.

So, if $C' \restriction_\rho$ fails to simplify with probability less than $(\epsilon_2 \cdot n + 1)^{-1}$, then we are guaranteed there is some ρ such that $C' \restriction_\rho (x)$ agrees with $f(x \oplus b^\rho)$, a shift of f, on all but at most $(\epsilon_2 + (1/n))$-fraction of inputs $x \in \{0,1\}^n$, and is a depth-$(d+1)$ circuit with bounded bottom fan-in.

By the Blockwise Switching Lemma (Lemma 10), the probability that $C' \restriction \rho$ fails to simplify to depth $(d+1)$ circuit with bottom fan-in at most k is at most $s'_{no}\left(\frac{8\log s'_{no}}{\ell}\right)^k = 2^{\log s'_{no} - k}$, which is less than $(\epsilon_2 \cdot n + 1)^{-1}$ if we choose $k = \log(2s'_{no}(1+\epsilon_2 n))$.

Thus, there must exist a blockwise restriction ρ such that $C' \restriction_\rho$ is simplified and agrees with $f(x \oplus b^\rho)$ on all but at most $(\epsilon_2 + (1/n))$ fraction of inputs. We have that $C' \restriction_\rho$ is of size at most $s'_{no}(1+2^k) \leq s'_{no}(1+2s'_{no}(1+\epsilon_2 n)) \leq 3(s'_{no})^2(1+\epsilon_2 n) \leq s_{no}$. Also, the bottom fan-in is at most $k \leq \log s_{no}$ for our choice of k. Then the circuit $C(x) = C' \restriction_\rho (x \oplus b^\rho)$ agrees with $f(x)$ on all but at most $(\epsilon_2 + (1/n))$ fraction of inputs, and C has depth $(d+1)$, size at most s_{no}, and bottom fan-in at most $\log s_{no}$, as required.

YES $\to$ YES. Suppose f is ϵ_1-approximated by a depth-$(d+1)$ circuit C with size s_{yes} and bottom fan-in $\log s_{yes}$. Let g be the Boolean function computed by C. Using the same techniques as in the "YES$\to$YES" case analysis in the proof of Theorem 19, we construct a depth-$(d+1)$ circuit C' computing $g \circ \oplus_\ell$, with size at most s'_{yes} and bottom fan-in at most $\log s'_{yes}$.

We will argue that C' computes $f \circ \oplus_\ell$ on all but at most ϵ_1 fraction of inputs. Indeed, since the parity of a uniformly random string of bits is a uniformly random bit, we get that

$$\Pr_{z \in \{0,1\}^{n\ell}}[(f \circ \oplus_\ell)(z) = (g \circ \oplus_\ell)(z)] = \Pr_{x \in \{0,1\}^n}[f(x) = g(x)],$$

which is at most ϵ_1 by our assumption. This concludes the proof. ◀

4.3 Combining the steps: Tolerant depth $d+1$ to $d+2$

Using the above reductions, we can obtain a reduction from tolerant depth $d+1$ gap-MCSP to tolerant depth $d+2$ gap-MCSP. Extending this to depth $d+c$ can be done via repeatedly composing this reduction with itself.

▶ **Corollary 26.** *For any $d \geq 1, 0 \leq \epsilon_1, \epsilon_2 < 1/2, s_{yes}, s_{no}, s'_{yes}, s'_{no}$ where $s_{no} \geq (2\epsilon n+1){s'_{no}}^4$ and $s'_{yes} \geq 2(s_{yes})^2 \cdot 2^{\sqrt{16\log({s_{yes}}^2)\log({s'_{no}}^2)}}\sqrt{16\log({s_{yes}}^2)\log({s'_{no}}^2)}$, we have*

$$\widehat{\mathsf{AC}}^0_{d+1}[\epsilon_1, \epsilon_2 + \frac{2}{n}]\text{-}\mathtt{GapMCSP}_n[s_{yes}, s_{no}] \leq_m^{\mathsf{qpoly}} \widehat{\mathsf{AC}}^0_{d+2}[\epsilon_1 + \frac{1}{n}, \epsilon_2]\text{-}\mathtt{GapMCSP}_{32n\log s'_{no}}[s'_{yes}, s'_{no}].$$

Proof. We obtain the desired reduction by composing the reductions from Theorems 23, 25, and 24. Using $\langle \epsilon_1, \epsilon_2, s_{yes}, s_{no}\rangle_{d,n}$ as a shorthand for $\widehat{\mathsf{AC}}^0_d[\epsilon_1, \epsilon_2]\text{-}\mathtt{GapMCSP}_n[s_{yes}, s_{no}]$, the reductions operate as follows:

$$\begin{aligned}
\langle \epsilon_1, \epsilon_2 + \frac{2}{n}, s_{yes}, s_{no}\rangle_{d+1,n} &\leq_m^{\mathsf{poly}} \langle \epsilon_1 + \frac{1}{n}, \epsilon_2 + \frac{2}{n}, (s_{yes})^2, s_{no}\rangle_{d+1/2,n} && \text{Theorem 23}\\
&\leq_m^{\mathsf{qpoly}} \langle \epsilon_1 + \frac{1}{n}, \epsilon_2 + \frac{1}{n}, s'_{yes}, (s'_{no})^2\rangle_{d+1+1/2,32n\log s'_{no}} && \text{Theorem 25}\\
&\leq_m^{\mathsf{poly}} \langle \epsilon_1 + \frac{1}{n}, \epsilon_2, s'_{yes}, s'_{no}\rangle_{d+2,32n\log s'_{no}} && \text{Theorem 24}
\end{aligned}$$ ◀

5 Barriers to More Efficient Natural Reductions

Our reductions are deterministic, many-one, and "simple" in the original size parameter. However, they require quasi-polynomial time. Here, we give evidence that improving such "nice" reductions to run in polynomial time for the *exact* MCSP is difficult: such reductions would immediately give breakthrough circuit lower bounds or non-trivial MCSP algorithms, and either outcome seems like dramatic progress.[2] To begin, observe that every reduction we present is qpoly-Natural in the following sense.

[2] Similar arguments apply to the gap-versions of the problem that we study above, but we argue about the exact version here to facilitate exposition.

▶ **Definition 27** (Natural Reductions between Parametric Problems). *Let A and B be* parametric problems, *that is, inputs are of the form: $\{\langle x, s\rangle : x \in \{0,1\}^n,\ s \in \mathbb{N}\}$. We call a* parametric reduction $R = \langle R_I, R_P\rangle$ *where R_I outputs instances and R_P outputs parameters, $t(\cdot)$-*natural *if it is:*

- ***Parametric Many-one:*** $\langle x, s\rangle \in A \iff \langle R_I(x,s),\ R_P(x,s)\rangle \in B$
- ***Parameter-Value Uniform:*** $R_P(x,s)$ *depends* ***only*** *on the size of the input and value of the parameter; we will treat R_P as a function from $\mathbb{N}\times\mathbb{N}$ in this case.*
- ***$t(\cdot)$-Efficient:*** *The combined runtime of R_I and R_P is bounded by $t(|x|,\ s)$.*

A natural reduction R from Λ-MCSP to Γ-MCSP is many-one, so a Λ-MCSP algorithm follows by brute-force search through Γ-circuits, and Λ-to-Γ lifting follows by mapping a Λ-hard function h through R. This gives the next two lemmas. Kabanets and Cai used the same reasoning to prove that NP-hardness of MCSP under poly-time natural reductions would imply breakthrough circuit lower bounds (Theorem 15 of [20]). Removing NP-hardness from the picture, we instead obtain the following:

▶ **Lemma 28** (Black-Box MCSP Algorithms from Natural MCSP-Redux). *If there is a* poly*-Natural Reduction from Λ-MCSP to Γ-MCSP, then there is a fixed constant $k \in \mathbb{N}$ such that $\Lambda\text{-MCSP}_n \in \mathsf{TIME}[\mathsf{poly}(nk) \times \Gamma\text{-count}(R_P(2^n, s))]$*

Proof. Fix a reasonable encoding of Γ-circuits that admits efficient evaluation. Then write Γ-count(s) for the total number of circuits so encoded that witness Γ-measure at most s. On input (f, s) to $\Lambda\text{-MCSP}_n$ we first run (f, s) through the natural reduction R to obtain (f', s'). Just as above, because R is poly-time, there is a fixed k such that $t(n) = 2^{kn}$. This means $|f'| \le 2^{kn}$, so we obtain an instance of Γ-MCSP with new size parameter $s' = R_P(2^n, s)$ on at most kn input variables.

Then, because R is parametric many-one, a (yes, no)-instance of $\Lambda\text{-MCSP}_n$ becomes a (yes, no)-instance of $\Gamma\text{-MCSP}_{kn}$ (respectively). So, we can solve the resulting instance of Γ-MCSP by brute-force search over the set of all s'-measure-witnessing Γ-circuits, and answer accordingly. We must evaluate a s'-size Γ-circuit on $\le kn$ bits at most Γ-count(s') times. This takes $\mathsf{poly}(nk) \cdot \Gamma$-count$(s')$ time in total. ◀

Lifting begins with pre-existing lower bounds for Λ, which we formalize below. Many concrete circuit lower bounds are *far more* explicit, but this weak notion will suffice for lifting via natural and efficient inter-MCSP reductions.

▶ **Definition 29** (Explicit Complexity Lower Bounds). *Let $H = \{h_n\}_{n\in\mathbb{N}}$ be a sequence of Boolean functions in* E*, and let $s_\Lambda : \mathbb{N} \to \mathbb{N}$ be a function in* FP*. We call the pair $\langle H, s_\Lambda\rangle$ an* explicit Λ-complexity lower bound *if $\forall n\ \Lambda(h_n) > s_\Lambda(n)$.*

▶ **Lemma 30** (Black-Box Lifting from Natural MCSP-Redux). *Let $\langle H, s\rangle$ be a Λ-complexity lower bound. If there is a* poly*-Natural Reduction R from Λ-MCSP to Γ-MCSP, then there exists a constant k and sequence of m-input Boolean functions H' such that $\langle H', R_P(2^{m/k}, s(m/k))\rangle$ is an explicit Γ-complexity lower bound.*

Proof. Fix an explicit Λ-complexity lower bound $\langle H, s\rangle$ and poly-natural reduction $R = \langle R_I, R_P\rangle$ from Λ-MCSP to Γ-MCSP. Now run the reduction: let H' be the sequence $h'_n = R_I(h_n, s(n))$ and let $s'(n) = R_P(h_n, s(n))$. We know $(h_n, s(n)) \notin \Lambda$-MCSP by the hardness assumption about H. Then, because R is parametric many-one, $(h'_n, s'(n)) \notin \Gamma$-MCSP and thus $\Gamma(h'_n) > s'(n)$. To make this explicit, we bound the runtime of answering queries according to h' on inputs x of m bits. This amounts to re-indexing the sequence H' to ensure that a Γ-hard function is defined everywhere and computable in E.

First, because R is poly-time, there is a fixed k such that $t(n) = 2^{kn}$. This means $|h'_n| \leq 2^{kn}$, so we send each input length n through the reduction to a new input length of at most kn. We evaluate h at m/k input bits and pad to fill in the gaps. Propagating this padded sequence of functions through the parameter-map R_P, we obtained the claimed Γ-complexity lower bound. ◀

5.1 Efficient Natural Reductions Between AC^0_d-, AC^0_{d+1}-MCSP: Win/Win

Notice how both applications of poly-Natural reductions depend quantitatively on R_P, the size parameter of the reduction. For lifting, we want $R_P(\cdot)$ *large enough* to improve the best known Γ-complexity lower bound by starting with a stronger lower bound for Λ. For solving Λ-MCSP by brute-force on Γ-MCSP, we want $R_P(\cdot)$ *small enough* such that searching all relevant Γ-circuits is faster than trivial brute-force over all relevant Λ-circuits. This observation suggests a case analysis of the function R_P, to obtain either a non-trivial MCSP algorithm or improved circuit lower bounds. For poly-Natural reductions from AC^0_d-MCSP to AC^0_{d+1}-MCSP, such a win/win argument succeeds. Informally, we have the following:

▶ **Theorem 31** (poly-Natural MCSP Reduction Win/Win)**.** *Suppose there is a* poly*-Natural reduction from* AC^0_d-MCSP *to* $\mathsf{AC}^0_{d'}$-MCSP*, for* $d' > d$*. Then, either:*

- *There is a surprisingly fast algorithm for* AC^0_d-MCSP*, or*
- *There are breakthrough explicit circuit lower bounds against* $\mathsf{AC}^0_{d'}[2^{\Omega(n^{1/d})}]$ *for* $d < d'$ *!*

We spend the remainder of this section formalizing and proving variations on the above.

5.2 Quantitative Consequences of a Hardness Hypothesis for MCSP

We first formulate an appropriate hypothesis about the hardness of MCSP.

▶ **Definition 32** (Weak Exponential Time Hypothesis (WETH) for Λ-MCSP)**.** *There exists an* $\epsilon > 0$ *such that for all "nice" size functions* $s(n)$*,* $\Lambda\text{-MCSP}_n[s(n)] \notin \mathsf{TIME}[2^{s(n)^\epsilon}]$.

For the general MCSP (when Λ is the class of unrestricted Boolean circuits), it can be shown that the WETH for MCSP is implied by the cryptographic conjecture that exponentially-strong one-way functions exist (using the ideas of [28, 20, 2]). One can also show that if WETH for general MCSP is false, then $\mathsf{NEXP} \not\subset \mathsf{P/poly}$ (using the ideas of [19]). For every $d \geq 2$, the WETH for AC^0_d-MCSP is also reasonable to assume, although we don't seem to have any strong evidence to support it yet (see [3] for some cryptographic hardness of AC^0_d-MCSP for large d).

Under this hypothesis, we establish barriers to giving poly-Natural reductions from AC^0_d-MCSP to AC^0_{d+c}-MCSP. We begin by recalling the best-known AC^0_d circuit lower bounds.

▶ **Theorem 33** (Håstad [13])**.** *Any depth* $(d+1)$ *alternating circuit computing* $\oplus_n$ *requires* $2^{\Omega(n^{1/d})}$ *gates. Furthermore, this bound is clearly* explicit *as in Definition 29.*

▶ **Theorem 34.** *Suppose there is a* poly*-Natural reduction from* AC^0_d-MCSP *to* $\mathsf{AC}^0_{d'}$-MCSP*, for* $d' > d$*. Then, either:*

- *The WETH for* AC^0_d-MCSP *is* ***false****, or*
- *There is an explicit circuit lower bound with* $s(n) = 2^{\Omega(n^{1/(d-1)})}$ *against* $\mathsf{AC}^0_{d'}$.

Proof. Assume such a poly-Natural reduction $R = \langle R_I, R_P \rangle$ exists, with run-time 2^{kn}. We reason by cases on bounds for R_P.

Suppose R_P is small. That is, $\forall\epsilon.R_P(2^n, s(n)) < s(n)^\epsilon$. Substituting into the black-box MCSP algorithm above, we have that $\forall\epsilon.\mathsf{AC}^0_d\text{-}\mathtt{MCSP}_n \in \mathsf{TIME}[\mathsf{poly}(nk) \times \mathsf{AC}^0_{d'}\text{-}\,\mathrm{count}(s(n)^\epsilon)] \in \mathsf{TIME}[2^{s(n)^{2\epsilon}}]$, where the first inclusion is by Lemma 28, and second by counting $\mathsf{AC}^0_{d'}$ circuits. This contradicts the MCSP-WETH for AC^0_d.

Suppose R_P is large. That is, $\exists\epsilon.R_P(2^n, s(n)) > s(n)^\epsilon$. Lifting $\oplus$ through R we have that there is an explicit sequence of Boolean functions H on m-bit inputs such that we have the following explicit $\mathsf{AC}^0_{d'}$-complexity bounds: $R_P(2^{m/k}, s(m/k)) > s(m/k)^\epsilon > 2^{\Omega(m^{1/(d-1)})}$. Here, the lower bound is by Lemma 30, the first inequality by size assumption about R_P, and the last by application of Håstad's bound. ◀

When $d' > d$, the lower-bound case above would be a breakthrough in circuit complexity.

▶ **Corollary 35** (Breakthrough Circuit Lower Bounds for Alternating Constant-Depth)**.** *Suppose the WETH for* $\mathsf{AC}^0_d\text{-}\mathtt{MCSP}$ *holds, for every* $d \geq 2$*. Then, if* $\forall d > d_0$ *we have a* poly*-Natural reduction* R_d *from* $\mathsf{AC}^0_d\text{-}\mathtt{MCSP}$ *to* $\mathsf{AC}^0_{(d+1)}\text{-}\mathtt{MCSP}$*, then there is a fixed constant* α *such that, for each depth* $d > d_0$*, there is a Boolean function* $f^d \in E$ *such that any depth-*d *alternating circuit computing* f^d_n *requires* $2^{\Omega_d(n^\alpha)}$ *gates.*

Proof. Fix any constant $d > d_0$. We first compose R_d with itself sufficiently many times to obtain a many-one reduction R'_d all the way from $\mathsf{AC}^0_{d_0}\text{-}\mathtt{MCSP}$ to $\mathsf{AC}^0_d\text{-}\mathtt{MCSP}$. Observe that R'_d remains poly-Natural, because all the polynomial resource bounds are closed under a constant number of compositions – though the leading constant exponent of runtime for R'_d certainly increases proportional to the gap between d and d_0; this is precisely what is hidden by Ω_d in the bound. To conclude, we apply black box lifting (Lemma 30) to the composed poly-Natural reduction R'_d, with Håstad's lower bound for $\oplus$ at depth d_0, getting $\alpha = 1/d_0$ in the theorem. ◀

Combining with a simulation of shallow formulas by constant-depth circuits, we get

▶ **Lemma 36** (Folklore)**.** *Any sequence* f_n *of Boolean functions on* n *inputs computable by formulas of depth* $c\log(n)$ *is computable by depth-*d *alternating* circuits *of size* $2^d \times 2^{n^{(c/d)}}$*.*

▶ **Theorem 37** (Breakthrough Circuit Lower Bounds for Formulas)**.** *Suppose the WETH for* $\mathsf{AC}^0_d\text{-}\mathtt{MCSP}$ *holds, for every* $d \geq 2$*. Then, if* $\forall d > d_0$ *we have a* poly*-Natural reduction* R_d *from* $\mathsf{AC}^0_d\text{-}\mathtt{MCSP}$ *to* $\mathsf{AC}^0_{(d+1)}\text{-}\mathtt{MCSP}$*, for every fixed* k *there exists* f^k *a sequence of Boolean functions in* E*, such that* f^k *does not have size-*n^k *formulas.*

Proof. Fix constant k, and let $c \in \mathbb{N}$ be the leading constant that results from re-balancing an arbitrary n^k-size formula to log-depth. Any function computed by such a formula will have – for every d – AC^0_d circuits of size $\approx 2^{n^{c/d}}$ by Lemma 36. Therefore, if we choose d such that $1/d_0 > c/d$, the size bound that results from lifting $\oplus$ through iterated composition of R_d exceeds the constant-depth simulation-size of any n^k-size formula. The rest of this argument is identical to the proof of Corollary 35 above. ◀

6 Open Questions

One obvious question is whether one can show the NP-completeness of AC^0_d-MCSP for any constant depth $d \geq 3$ by proving that depth 3 circuit complexity is NP-hard to approximate to within the factors needed by our lifting theorems? Note that while we have hardness of approximation result for DNFs [23, 3], the (almost tight) approximation gap there is not strong enough for our lifting theorems to apply. For depth-3 circuits, on the other hand, there are no known hardness of approximation results.

Another natural question is to tighten the gap (the approximation factor) of our lifting theorems. Finally, can one provide more evidence supporting the Weak Exponential Time Hypothesis for AC^0_d-MCSP?

References

1 Eric Allender. The new complexity landscape around circuit minimization. In Alberto Leporati, Carlos Martín-Vide, Dana Shapira, and Claudio Zandron, editors, *Language and Automata Theory and Applications*, pages 3–16, Cham, 2020. Springer International Publishing.

2 Eric Allender, Harry Buhrman, Michal Koucký, Dieter van Melkebeek, and Detlef Ronneburger. Power from random strings. *SIAM J. Comput.*, 35(6):1467–1493, 2006. doi:10.1137/050628994.

3 Eric Allender, Lisa Hellerstein, Paul McCabe, Toniann Pitassi, and Michael E. Saks. Minimizing Disjunctive Normal Form Formulas and AC^0 Circuits Given a Truth Table. *SIAM J. Comput.*, 38(1):63–84, 2008. doi:10.1137/060664537.

4 Eric Allender and Shuichi Hirahara. New insights on the (non-)hardness of circuit minimization and related problems. *ACM Transactions on Computation Theory (ToCT)*, 11(4):1–27, 2019.

5 Eric Allender, Dhiraj Holden, and Valentine Kabanets. The minimum oracle circuit size problem. In *32nd International Symposium on Theoretical Aspects of Computer Science, STACS 2015, March 4-7, 2015, Garching, Germany*, pages 21–33, 2015. doi:10.4230/LIPIcs.STACS.2015.21.

6 Alexander E. Andreev. On a method for obtaining more than quadratic effective lower bounds for the complexity of π-schemes. *Moscow Univ. Math. Bull.*, 42:63—-66, 1987.

7 Paul Beame. A switching lemma primer, 1994.

8 Eli Ben-Sasson and Avi Wigderson. Short proofs are narrow—resolution made simple. *Journal of the ACM*, 48(2):149–169, 2001.

9 Vàclav Chvátal. A greedy heuristic for the set-covering problem. *Math. Oper. Res.*, 4(3):233–235, 1979. doi:10.1287/moor.4.3.233.

10 Susanna F. de Rezende, Or Meir, Jakob Nordström, Toniann Pitassi, Robert Robere, and Marc Vinyals. Lifting with simple gadgets and applications to circuit and proof complexity. In *61st IEEE Annual Symposium on Foundations of Computer Science, FOCS 2020, Durham, NC, USA, November 16-19, 2020*, pages 24–30. IEEE, 2020.

11 Vitaly Feldman. Hardness of approximate two-level logic minimization and PAC learning with membership queries. In *Proceedings of the Thirty-Eighth Annual ACM Symposium on Theory of Computing*, STOC '06, page 363–372, New York, NY, USA, 2006. Association for Computing Machinery. doi:10.1145/1132516.1132569.

12 Mika Göös, Toniann Pitassi, and Thomas Watson. Deterministic communication vs. partition number. *SIAM Journal on Computing*, 47(6):2435–2450, 2018.

13 Johan Håstad. Almost optimal lower bounds for small depth circuits. In Juris Hartmanis, editor, *Proceedings of the 18th Annual ACM Symposium on Theory of Computing, May 28-30, 1986, Berkeley, California, USA*, pages 6–20. ACM, 1986. doi:10.1145/12130.12132.

14 Johan Håstad, Benjamin Rossman, Rocco A. Servedio, and Li-Yang Tan. An average-case depth hierarchy theorem for boolean circuits. *J. ACM*, 64(5):35:1–35:27, 2017. doi:10.1145/3095799.

15 Shuichi Hirahara, Igor C. Oliveira, and Rahul Santhanam. NP-hardness of minimum circuit size problem for OR-AND-MOD circuits. In *33rd Computational Complexity Conference (CCC 2018)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2018.

16 Shuichi Hirahara and Osamu Watanabe. Limits of minimum circuit size problem as oracle. In *31st Conference on Computational Complexity, CCC*, pages 18:1–18:20, 2016. doi:10.4230/LIPIcs.CCC.2016.18.

17 John M. Hitchcock and A. Pavan. On the NP-completeness of the minimum circuit size problem. In *35th IARCS Annual Conference on Foundation of Software Technology and Theoretical Computer Science, FSTTCS*, pages 236–245, 2015. doi:10.4230/LIPIcs.FSTTCS.2015.236.

18 Rahul Ilango. Constant depth formula and partial function versions of MCSP are hard. In *2020 IEEE 61st Annual Symposium on Foundations of Computer Science (FOCS)*, pages 424–433. IEEE, 2020.

19 Russell Impagliazzo, Valentine Kabanets, and Avi Wigderson. In search of an easy witness: Exponential time vs. probabilistic polynomial time. *J. of Computer and System Sciences*, 65(4):672–694, 2002.

20 Valentine Kabanets and Jin-Yi Cai. Circuit minimization problem. In *Proceedings of the 32nd Annual ACM Symposium on Theory of Computing (STOC)*, pages 73–79, 2000.

21 Richard M. Karp, F. E. McFarlin, J. P. Roth, and J. R. Wilts. A computer program for the synthesis of combinational switching circuits. In *2nd Annual Symposium on Switching Circuit Theory and Logical Design (SWCT 1961)*, pages 182–194, 1961. doi:10.1109/FOCS.1961.1.

22 Subhash Khot and Rishi Saket. Hardness of minimizing and learning DNF expressions. In *2008 IEEE 49th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 231–240, Los Alamitos, CA, USA, October 2008. IEEE Computer Society. doi:10.1109/FOCS.2008.37.

23 William J. Masek. Some NP-complete set covering problems, 1979.

24 Cody D. Murray and Ryan R. Williams. On the (non) NP-hardness of computing circuit complexity. In *30th Conference on Computational Complexity, CCC*, pages 365–380, 2015. doi:10.4230/LIPIcs.CCC.2015.365.

25 Toniann Pitassi and Robert Robere. Lifting nullstellensatz to monotone span programs over any field. In *Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing*, pages 1207–1219, 2018.

26 Ran Raz and Pierre McKenzie. Separation of the monotone NC hierarchy. In *Proceedings 38th Annual Symposium on Foundations of Computer Science*, pages 234–243. IEEE, 1997.

27 Alexander A. Razborov. Bounded arithmetic and lower bounds in boolean complexity. In Peter Clote and Jeffrey B. Remmel, editors, *Feasible Mathematics II*, pages 344–386, Boston, MA, 1995. Birkhäuser Boston.

28 Alexander A. Razborov and Steven Rudich. Natural proofs. *J. of Computer and System Sciences*, 55(1):24–35, 1997.

29 Michael Saks and Rahul Santhanam. Circuit lower bounds from NP-hardness of MCSP under Turing reductions. In *35th Computational Complexity Conference (CCC 2020)*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2020.

30 Boris A. Trakhtenbrot. A Survey of Russian Approaches to Perebor (Brute-Force Searches) Algorithms. *IEEE Annals of the History of Computing*, 6(4):384–400, 1984. doi:10.1109/MAHC.1984.10036.

Sparsification of Directed Graphs via Cut Balance

Ruoxu Cen
Duke University, Durham, NC, USA

Yu Cheng
University of Illinois at Chicago, IL, USA

Debmalya Panigrahi
Duke University, Durham, NC, USA

Kevin Sun
Duke University, Durham, NC, USA

Abstract

In this paper, we consider the problem of designing cut sparsifiers and sketches for directed graphs. To bypass known lower bounds, we allow the sparsifier/sketch to depend on the *balance* of the input graph, which smoothly interpolates between undirected and directed graphs. We give nearly matching upper and lower bounds for both *for-all* (cf. Benczúr and Karger, STOC 1996) and *for-each* (Andoni et al., ITCS 2016) cut sparsifiers/sketches as a function of cut balance, defined the maximum ratio of the cut value in the two directions of a directed graph (Ene et al., STOC 2016). We also show an interesting application of digraph sparsification via cut balance by using it to give a very short proof of a celebrated maximum flow result of Karger and Levine (STOC 2002).

2012 ACM Subject Classification Theory of computation → Sparsification and spanners

Keywords and phrases Graph sparsification, directed graphs, cut sketches, space complexity

Digital Object Identifier 10.4230/LIPIcs.ICALP.2021.45

Category Track A: Algorithms, Complexity and Games

Related Version *Full Version*: `https://arxiv.org/abs/2006.01975`

Funding *Debmalya Panigrahi and Kevin Sun*: This work was supported in part by NSF grants CCF-1535972, CCF-1955703, and an NSF CAREER Award CCF-1750140.

Acknowledgements Part of the work was done while Yu Cheng was visiting the Institute for Advanced Study.

1 Introduction

Graph sparsification, originally introduced by Benczúr and Karger as a means of obtaining faster maximum flow algorithms [8], has become a fundamental tool in graph algorithms. The goal of graph sparsification is to replace an arbitrary graph with a sparse graph (called the graph sparsifier) on the same set of n vertices but with only $O(n \cdot \text{poly}(\log n, 1/\epsilon))$ edges, while approximately preserving the value of every cut up to a factor of $1 \pm \epsilon$ for any given $\epsilon > 0$. Since their work, several graph sparsification techniques have been discovered (e.g., [17]), the idea has been extended to other models of computation such as data streaming (e.g., [1]) and sketching (e.g., [6]), stronger notions such as spectral sparsification that preserves all quadratic forms have been proposed (e.g., [43]), and far-reaching generalizations such as the Kadison-Singer conjecture have been established [37]. On the applications front, graph sparsification has been heavily used to obtain a tradeoff between algorithmic accuracy and efficiency for a variety of "cut-based" problems such as maximum flows, minimum cuts, balanced separators, etc.

In spite of its widespread use, one restriction is that most sparsification techniques only apply to undirected graphs. There is a fundamental reason for this restriction – there are directed graphs that *cannot* be sparsified (see Fig. 1 for an example). Indeed, the lower

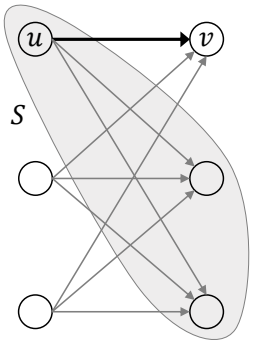


Figure 1 The graph is a complete bipartite graph where all edges go from the left vertices L to the right vertices R. For any $u \in L$ and $v \in R$, the only edge that leaves the set $S = \{u\} \cup (R \setminus \{v\})$ is the edge (u, v). Consequently, if we want to preserve the value of all directed cuts in this graph, we have to (approximately) store the weight of every edge.

bound holds even for *cut sketches*, where one does not insist on a graph being output as the sparsifier, but simply a succinct data structure from which the cut values of the original graph can be (approximately) retrieved.

A qualitative distinction between directed and undirected graphs is in terms of the *balance* of cuts, i.e., the ratio between incoming and outgoing edges in any given cut. An equivalent view of an undirected graph is by bi-directing its edges, which results in a graph with perfect balance, i.e., every cut has exactly the same number of incoming and outgoing edges.[1] Cut balance, therefore, smoothly interpolates between undirected and directed graphs, which leads to the question: *can we design cut sparsifiers/sketches for directed graphs that depend on cut balance?* We answer this question in the affirmative in this paper, and show that this view of sparsification leads to interesting consequences.

We note that the use of cut balance to bridge between undirected and directed graphs predates our work. Ene et al. [15] introduced the notion of parameterizing digraphs by their cut balance (or simply balance) and defined it as the largest ratio between the value of a cut in its two directions. Using this view, they extended two classic operations on undirected graphs – oblivious routing and fast approximate maximum flows – to directed graphs with a dependence on the balance. In this paper, we show that this phenomenon is exhibited by cut sparsification as well.

1.1 Our Results

We consider the two canonical forms of cut sparsification considered in the literature. The first is the classic version introduced by Benczúr and Karger [8], where all cuts must simultaneously be approximately preserved *whp*;[2] we call this *for-all* sparsification. The second, more relaxed, notion is due to Andoni et al. [6], where *any* cut must be approximately preserved *whp* instead of all cuts simultaneously; we call this *for-each* sparsification. For both these notions of sparsification, previous results on undirected graphs can be extended to β-balanced graphs by boosting sampling probabilities in undirected sparsification algorithms by a factor of β, thereby losing an additional factor of β in the size of the sparsifier/sketch (see also Ikeda and Tanigawa [20]). Is it possible to do better than losing a factor of β?

[1] Note that all Eulerian digraphs, whether or not derived from undirected graphs, exhibit perfect cut balance.

[2] with high probability

Our first result sharpens this naïve bound in for-each sparsification, by constructing cut sketches that improve the dependence on β to $\sqrt{\beta}$. We also show that this dependence is tight by constructing a matching lower bound. This pair of results resolves the precise dependence of for-each sparsification in directed graphs on the balance of the graph.

▶ **Theorem 1** (Upper Bound). *For any β-balanced graph with n vertices, m edges, and polynomially-bounded edge weights, there is an $\widetilde{O}(m + \sqrt{\beta}n/\epsilon)$-time algorithm[3] that constructs a $(1 \pm \epsilon)$ for-each cut sketch of size $\widetilde{O}(\sqrt{\beta}n/\epsilon)$ bits.*

(Lower Bound) Fix any $\beta \geq 1$, $0 < \epsilon < 1$, and n such that $(\beta/\epsilon)^{1/2} \leq n/2$. Any $(1 \pm \epsilon)$ for-each cut sketching algorithm for β-balanced graphs with n vertices must output at least $\Omega(\sqrt{\beta}n/\sqrt{\epsilon})$ bits in the worst case.

In for-all sparsification, we are not as lucky; we show that the linear dependence on β is tight in this case. (In fact, Ikeda and Tanigawa [20] had conjectured that better for-all sparsifiers can be constructed by sampling edges according to directed connectivity parameters; our lower bound construction refutes this conjecture and shows that such more aggressive sampling may not produce a sparsifier at all.)

▶ **Theorem 2.** *Fix any $\beta \geq 1$, $0 < \epsilon < 1$, and n such that $\beta/\epsilon \leq n/2$. Any $(1 \pm \epsilon)$ for-all cut sketching algorithm for n-node β-balanced graphs must output at least $\Omega(n\beta/\epsilon)$ bits.*

But, we note that the upper bound only applies to digraphs where *all* cuts are β-balanced. In general, the balance parameter for different cuts in a digraph may be highly non-uniform: some cuts could be very balanced and some others very unbalanced. For such graphs, we show a more refined result: for *any* value $\beta \geq 1$, we construct a sparsfier that approximately preserves all β-balanced cuts losing only an additional factor β in the size of the sparsifier. Note that this result holds for any value of β irrespective of the balance parameter of the graph; if β is the balance parameter, then it recovers the tight bound for β-balanced graphs.

▶ **Theorem 3.** *For any directed graph with n vertices, m edges, and non-negative edge weights, and any $\beta \geq 1$, there is an $\widetilde{O}(m)$-time algorithm that returns a (weighted) subgraph with $\widetilde{O}(\beta n/\epsilon^2)$ edges and preserves the values of all β-balanced cuts up to a factor of $1 \pm \epsilon$.*

We remark that digraph sparsification using cut balance has interesting consequences. In particular, note that for residual graphs produced by s-t maximum flow algorithms in undirected graphs, we can precisely bound the balance parameter on all cuts separating the source and the sink. Using this observation, we give a very short proof of the celebrated maximum flow result of Karger and Levine [27] via the digraph sparsification results.

1.2 Our Techniques

First, we outline the main ideas in our for-each cut sketch. In previous results on for-each cut sketches of undirected graphs [6, 21], the main idea was to (recursively) partition the graph into "sparse" and "dense" parts, and then maintain the sparse parts exactly along with a sample of the dense parts. A directed subgraph, however, can simultaneously be too dense to preserve exactly but also not amenable to sampling (e.g., a complete bipartite digraph). Of course, the balance parameter helps bridge this gap, but the cut balance of a subgraph that the algorithm encounters during recursion can be much worse than that of the original

[3] This runtime bound assumes that the value of β is known. If not, then β can be computed using an algorithm of Ene et al. [15] in $\widetilde{O}(\beta^2 m)$ time.

graph. Indeed, individual subgraphs might not even be strongly connected (i.e., have balance ∞), even if the original graph were Eulerian (i.e., has balance 1). This makes the (recursive) local sketching techniques in previous works unusable for directed graphs.

Our main technical contribution is a new *global* cut sketch construction. We design a cut sketch whose variance can be large on individual dense regions of the input digraph that are well-connected in an undirected sense, but we crucially show that the *cumulative variance of our estimator across all these well-connected regions of the digraph is small*. This helps eliminate the need for local cut sketches in each dense subgraph, and simplifies the recovery algorithm to the natural estimator that appropriately scales the number of sampled edges in the queried cut. Moreover, to obtain the right dependence on β, we need to carefully analyze the variance of our estimator. Our new variance analysis works for undirected graphs as well, which tightens the analysis of [6] and consequently leads to undirected cut sketching algorithms that do not require downsampling or low-accuracy for-all sparsifiers.

Next, we turn to for-all sparsification. Our first result is the lower bound on for-all sparsifiers and cut sketches. For any β and n, we construct a family $\mathcal{G}$ of β-balanced graphs on n vertices that satisfies two conflicting properties: $\mathcal{G}$ is a large family, yet for each graph $G \in \mathcal{G}$, the number of graphs in $\mathcal{G}$ that approximate all cuts of G is small. For any graph G and cut S, let $\delta_G(S)$ denote the value of S and $E_G(S)$ denote the edges crossing S. Notice that there are many possible graphs H such that $\delta_H(S) \approx \delta_G(S)$, because $E_H(S)$ and $E_G(S)$ could differ in numerous ways. Thus, to ensure that the number of cut approximators is small, we carefully design $\mathcal{G}$ such that for any $G, H \in \mathcal{G}$ and cut S, if $\delta_H(S) \approx \delta_G(S)$, then $E_H(S) \approx E_G(S)$. We show that this can be done by considering a large family of bipartite graphs that all contain a fixed (directed) matching. Consequently, any sketching algorithm must produce a large number of different cut sketches for the graphs in $\mathcal{G}$, which translates to a lower bound on the size of the cut sketches using standard information theory.

Finally, we refine for-all sparsification in digraphs by showing that we can preserve all balanced cuts, irrespective of the balance parameter of the entire graph G. More specifically, at a cost of an additional factor of β in the size of the sparsifer, we can preserve all β-balanced cuts, and provide an approximation for α-balanced cuts with $\alpha > \beta$ that degrades gracefully as α gets larger. For this purpose, we adopt a (recursive) graph decomposition due to Benczúr and Karger [8] that expresses a graph as a weighted sum of subgraphs, each of which corresponds to a particular edge sampling rate. Now we can boost the (undirected) sampling rate by a factor of β. If the balance of every subgraph in the decomposition is also β, then the undirected analysis carries over to the directed case. However, in general, each subgraph can be very unbalanced, so we cannot bound the estimation error in each individual subgraph. Our main technical contribution is to show that even though we do not preserve the cut values in individual subgraphs, we do so globally across all the subgraphs.

1.3 Related Work

Graph Sparsification. Graph sparsification was introduced by Benczúr and Karger [8] ("for-all" cut sparsification), and has led to research in a number of directions: Fung et al. [17] and Kapralov and Panigrahy [26] gave new algorithms for preserving cuts in a sparsifier; Spielman and Teng [43] generalized to spectral sparsfiers that preserved all quadratic forms, which led to further research both in reducing the size of the sparsifier [42, 7] and developing faster algorithms (e.g., [34, 5, 35, 11, 33, 30, 32, 31]); faster algorithms for fundamental graph problems such as maximum flow utilized sparsification results (e.g., [8, 40]); Ahn and Guha [1] introduced sparsification in the streaming model, which has led to a large body of work for both cut (e.g., [2, 3, 18]) and spectral sparsifiers (e.g., [25, 24, 23, 4]) in graph

streams; both cut [29, 38] and spectral [41] sparsification have been studied in hypergraphs. For lower bounds, Andoni et al. [6] showed that any data structure that $(1\pm\epsilon)$-approximately stores the sizes of all cuts in an undirected graph must use $\Omega(n/\epsilon^2)$ bits. Carlson et al. [10] improved this lower bound to $\Omega(n\log n/\epsilon^2)$ bits, matching existing upper bounds.

Andoni et al. [6] first proposed the notion of "for-each" cut (and spectral) sketches, where the sparsifier preserves the value of *any* cut rather than all cuts simultaneously. They showed that for any undirected graph with n vertices, a $(1\pm\epsilon)$ for-each cut sketch of size $\widetilde{O}(n/\epsilon)$ exists and can be computed in polynomial time. Subsequently, Jambulapati and Sidford [21] gave the first nearly-linear time algorithm for constructing $(1\pm\epsilon)$ for-each graph sketches of size $\widetilde{O}(n/\epsilon)$. Their sketch not only approximates cut values, but also approximately preserves the quadratic form of any undirected Laplacian matrix (and its pseudoinverse). Chu et al. [11] showed how to construct a *graph* containing $\widetilde{O}(n^{1+o(1)}/\epsilon)$ edges that satisfies the "for-each" requirement for spectral queries.

Directed Graphs. Cohen et al. [14, 13] proposed a directed notion of spectral sparsifiers and used it to obtain nearly-linear time algorithms for solving directed Laplacian linear systems and computing various properties of directed random walks. However, their directed spectral sparsifiers only work for Eulerian graphs, i.e., for $\beta = 1$. Zhang et al. [44] proposed a notion of spectral sparsification that works for all directed graphs, but their definition does not preserve cut values. More generally, there have been attempts at bridging the divide between directed and undirected graphs for other problems. For instance, Lin [36] defined the imbalance of a graph as the sum of the absolute difference of in- and out-capacities at all vertices, and used it to generalize the max-flow algorithm of Karger and Levine [27] from undirected graphs to digraphs. Digraphs have also been parameterized by directed extensions of treewidth [22], and similar notions of DAG-width [9, 39] and Kelly-width [19], which led to FPT algorithms based on these parameters, much like for undirected bounded treewidth graphs. In spectral graph theory, directed analogs of Cheeger's inequality have been defined [12], particularly in the context of analyzing the spectrum of digraphs. Closest to our work is that of Ene et al. [15] who proposed cut balance of digraphs that we use in this paper, although in the context of oblivious routing and max-flow algorithms.

2 Preliminaries

Basic Notations. Let $G = (V, E, w)$ be a weighted *directed* graph with $n = |V|$ vertices and $m = |E|$ edges. Every edge $e \in E$ has a given non-negative weight $w_e \geq 0$. When working with unweighted graphs, (i.e., $w_e = 1$ for all $e \in E$), we will omit the edge weights w_e.

For two sets of vertices $S \subseteq V$ and $T \subseteq V$, we use $E(S,T) = \{(u,v) \in E : u \in S, v \in T\}$ to denote the set of edges in E that go from S to T. We use $w(S,T) = \sum_{e\in E(S,T)} w_e$ to denote the total weight of the edges from S to T. For a vertex $u \in V$ and a set of vertices $S \subseteq V$, we write $E(u,S)$ for $E(\{u\},S)$, and we define $E(S,u)$, $w(u,S)$, and $w(S,u)$ similarly.

We often write $\overline{S}$ as a shorthand for $V \setminus S$. Given a component V_i and a subset of its vertices $S_i \subseteq V_i$, we can similarly define $\overline{S_i} = V_i \setminus S_i$. For example, using this notation, we write $w(S,\overline{S})$ for $w(S, V\setminus S)$ and similarly $w(S_i,\overline{S_i}) = w(S_i, V_i \setminus S_i)$.

The *conductance* of an undirected graph $G = (V, E, w)$ is defined as

$$\phi(G) = \min_{\varnothing\neq S\subset V} \frac{w(S,\overline{S})}{\min\big(w(S,V), w(\overline{S},V)\big)}. \tag{1}$$

▶ **Definition 4** (β-Balanced). *A strongly connected digraph $G = (V, E, w)$ is* β-balanced *if, for all $\varnothing \subseteq S \subseteq V$, it holds that $w(S,\overline{S}) \leq \beta \cdot w(\overline{S}, S)$.*

Directed Cut Sparsifiers and Cut Sketches. We consider two notions of sparsification. The first is the classic "for-all" sparsifier that approximately preserves the values of all cuts.

▶ **Definition 5** (For-All Cut Sparsifier). *Let $G = (V, E_G, w_G)$ and $H = (V, E_H, w_H)$ be two weighted directed graphs. Fix $0 < \epsilon < 1$. We say H is a $(1 \pm \epsilon)$ for-all cut sparsifier of G iff the following holds for all $S \subseteq V$:*

$$(1 - \epsilon) \cdot w_G(S, V \setminus S) \leq w_H(S, V \setminus S) \leq (1 + \epsilon) \cdot w_G(S, V \setminus S).$$

Instead of a graph that preserves cut values, if we allow any data structure from which the cut values can be (approximately) recovered, we call it a cut sketch.

▶ **Definition 6** (For-All Cut Sketch). *Let $G = (V, E, w)$ be a weighted directed graph. Fix $0 < \epsilon < 1$. A (deterministic) function g outputs a $(1 \pm \epsilon)$ for-all cut sketch of G if there exists a recovering function f such that, for all $S \subseteq V$:*

$$(1 - \epsilon) \cdot w(S, V \setminus S) \leq f(S, \mathrm{sk}(G)) \leq (1 + \epsilon) \cdot w(S, V \setminus S).$$

Next we consider a weaker notion of graph sparsification, where instead of approximating the value of all cuts, we only require the value of any individual cut to be approximately preserved with (high) constant probability.

▶ **Definition 7** (For-Each Cut Sketch). *Let $G = (V, E, w)$ be a weighted directed graph. Fix $0 < \epsilon < 1$. A function g outputs a for-each $(1 \pm \epsilon)$-cut sketch of G if there exists a recovering function f such that, for each $S \subseteq V$, with probability at least $2/3$,*

$$(1 - \epsilon) \cdot w(S, V \setminus S) \leq f(S, g(G)) \leq (1 + \epsilon) \cdot w(S, V \setminus S).$$

3 For-All Sparsification: $\widetilde{O}(n \cdot \beta/\epsilon^2)$ Upper Bound

In this section, we extend the seminal work of Bencúr and Karger [8] to directed graphs using cut balance. For undirected graphs, they showed that sampling every edge inversely proportional to a quantity known as its strength (see Definition 11) preserves all cuts with high probability. We show that, by boosting this sampling probability by a factor of β, this procedure can preserve the value of β-balanced cuts in any directed graph.

We then show that this sampling theorem can be applied in a black-box manner to recover the analysis of a celebrated maximum flow algorithm for undirected graphs given by Karger and Levine [27]. At each step of the algorithm, they sample edges from the residual network (which is directed) of an undirected graph. Using a customized version of the sparsification result from Benczúr and Karger [8], they show that with high probability, the sample contains an augmenting path. In contrast, our sampling procedure can be applied directly to the residual network, which simplifies the analysis of the algorithm.

The following theorem is our main result of this section:

▶ **Theorem 8.** *Let $G = (V, E)$ be a directed graph where each edge e has weight $u_e \geq 0$, and let ϵ, β be parameters. There is an $\widetilde{O}(m)$-time algorithm that returns a weighted subgraph H that satisfies the following with high probability: for every α-balanced cut U,*

$$\left(1 - \epsilon\sqrt{\frac{\alpha+1}{\beta+1}}\right) \cdot \delta_G(U) \leq \delta_H(U) \leq \left(1 + \epsilon\sqrt{\frac{\alpha+1}{\beta+1}}\right) \cdot \delta_G(U).$$

where $\delta_G(U)$ and $\delta_H(U)$ denote the cut value of U in G and H, respectively. Furthermore, H contains $O(\beta n \log n/\epsilon^2)$ edges in expectation.

Note that for the special case where the graph G is β-balanced, Theorem 3 is implied by Theorem 8: all cut values are preserved. This is the main result of Ikeda and Tanigawa [20].

▶ **Corollary 9** (Ikeda and Tanigawa [20]). *Consider the same setting as Theorem 8. If G is β-balanced, then with high probability, H approximates every cut of G up to a $(1 \pm \epsilon)$ factor.*

Before proving Theorem 8, we give an application of digraph sparsification to the maximum flow problem. In particular, we prove the correctness of the $\widetilde{O}(m + nv)$-time maximum flow algorithm given by Karger and Levine [27], where v is the value of the maximum flow. This algorithm is an adaptation of the classic augmenting paths algorithm of Ford and Fulkerson [16], but with the following crucial observation. Let f denote the current flow value in any iteration, and $\gamma = (v - f)/v$ denote the fraction of remaining flow in the residual network. Karger and Levine [27] show that, by boosting the undirected sampling procedure of Benczúr and Karger [8] by a factor of $1/\gamma$ and applying it to the residual network, the resulting sample contains an augmenting path with high probability. This saves on running time since the search for an augmenting path can then be performed on the sampled graph instead of the entire residual network. (See the full paper for more details.)

In contrast, we show that we can directly apply digraph sparsification to the residual network, with $\beta = 2/\gamma$ to obtain a short proof of the Karger-Levine theorem:

▶ **Theorem 10** (Karger and Levine [27]). *Suppose we apply the algorithm in Theorem 8 to the residual network in a maximum flow computation, with $\epsilon = 0.1$ and $\beta = 2/\gamma$, where $\gamma = (v - f)/v$ is the fraction of flow remaining in the residual network. Then with high probability, there is an augmenting path in the sample.*

Proof. We claim that every s-t cut S in the residual graph is β-balanced, where $\beta = 2/\gamma$. Suppose S initially contains capacity $c \geq v$, and currently, x units of flow are entering S. Since the flow value is $(1 - \gamma)v$, the amount of flow leaving S is $x + (1 - \gamma)v$. At the same time, the x units of flow entering S create a residual capacity of x leaving S. Thus, the total residual capacity leaving S is $c - x - (1 - \gamma)v + x = c - (1 - \gamma)v$. We can similarly show that the residual capacity entering S is $c + x + (1 - \gamma)v - x = c + (1 - \gamma)v$. Thus, in the residual graph, the balance of S is at most $\frac{c+(1-\gamma)v}{c-(1-\gamma)v} \leq \frac{2-\gamma}{\gamma} \leq \frac{2}{\gamma}$.

Now by setting $\epsilon = 0.1$ and $\beta = 2/\gamma$, Theorem 8 implies that the sparsifier H preserves all $(2/\gamma)$-balanced cuts up to a (1 ± 0.1) factor with high probability. Since every s-t cut is $(2/\gamma)$-balanced, this implies that there exists an augmenting path in H, as desired. ◀

In the rest of this section, we prove Theorem 8. Before we give our algorithm, we state the definitions and results that we need from previous work.

▶ **Definition 11** (Strength and strong components). *The* strength *of an edge e, denoted by k_e, is the largest k such that there exists a k-edge-connected vertex-induced subgraph of G containing e. A* k-strong component *is the subgraph induced by edges with strength at least k.*

▶ **Lemma 12** (Benczúr and Karger [8]). *The strong components of an undirected graph form a laminar family, and a graph on n vertices has at most $n - 1$ nontrivial strong components.*

▶ **Lemma 13** (Benczúr and Karger [8]). *In any graph with edge weights u_e and strengths k_e, we have $\sum_e u_e/k_e \leq n - 1$. Furthermore, there exists an $O(m \log^3 n)$-time algorithm that returns, for every edge e, an estimate $\widetilde{k}_e$ of k_e satisfying $\widetilde{k}_e \leq k_e$ and $\sum_e u_e/\widetilde{k}_e = O(n)$.*

We now describe our algorithm (Algorithm 1). The input is a directed graph where each edge e has weight u_e. We first compute approximate edge strengths $\widetilde{k}_e$ as given in Lemma 13. Then we sample each edge e proportional to $(\beta+1)u_e/\widetilde{k}_e$, where $\beta \geq 1$ is a chosen parameter. We choose the weight of the sampled edges so that we get an unbiased estimator.

Algorithm 1 For-all sparsification for directed graphs.

Input : An n-vertex directed graph $G = (V, E, u)$ with edge weights u_e, $0 < \epsilon < 1$, $\beta \geq 1$, and a constant $d > 2$.
Output : A subgraph H that satisfies Theorem 8.

```
Use Lemma 13 to compute an estimated edge strength k̃_e ≤ k_e for every edge e ∈ E.
Let ρ = 3d(β + 1) log n/ε².
for each edge e ∈ E do
    Sample e with probability p_e = ρ · u_e/k̃_e.
    if e is sampled then add e to H with weight w_e = k̃_e/ρ.
return H.
```

Now we analyze the output H of Algorithm 1. Without loss of generality, we assume that the algorithm uses the actual edge strengths k_e rather than the estimates $\widetilde{k}_e$. This is because $\widetilde{k}_e \leq k_e$ and it does not hurt to oversample in importance sampling.

For each strong component G_i of G (see Definition 11), let H_i denote the corresponding component in H. Because the way we choose sampling probabilities and edge weights in H, we have $\mathbb{E}[H_i] = G_i$. Let $\alpha_i = (k_i - k_{p(i)})/\rho$ where $p(i)$ is G_i's parent in the laminar family formed by strong components (see Lemma 12). As shown by Benczúr and Karger [8], this results in a decomposition of G into its strong components, that is, $G = \sum_i \alpha_i G_i$.

For a component G_i and a cut U, let $\delta_{G_i}(U)$ be the total capacity of edges leaving U in G_i, and let $\delta^{\text{un}}_{G_i}(U)$ be the corresponding value for the undirected version of G_i. The following lemma shows that for every strong component G_i, with high probability, $\delta_{G_i}(U)$ is preserved in H up to a relative error for every cut U.

▶ **Lemma 14.** *Let H be the output of Algorithm 1. For each strong component G_i (defined in Definition 11), the following holds with probability at least $1 - O(n^{-d+2})$: for any cut U, we have $\left|\delta_{H_i}(U) - \delta_{G_i}(U)\right| \leq \zeta(U) \cdot \delta_{G_i}(U)$ where $\zeta(U) = \epsilon\sqrt{\delta^{\text{un}}_{G_i}(U)/(\delta_{G_i}(U)(\beta+1))}$.*

Proof. For any cut U, let $\delta_p(U)$ denote the sum of u_e/k_e over the (undirected) edges crossing U_j in G_i. Order the r cuts intersecting G_i such that $1 = \delta_p(U_1) \leq \cdots \leq \delta_p(U_r)$, and let

$$q_j = \Pr\left(\left|\delta_{H_i}(U_j) - \delta_{G_i}(U_j)\right| > \zeta(U) \cdot \mathbb{E}\left[\delta_{H_i}(U_j)\right]\right).$$

By a Chernoff bound, we have

$$q_j \leq 2\exp\left(-\frac{(\zeta(U))^2 \cdot \delta_{G_i}(U_j)}{3}\right) = 2\exp\left(-\frac{\epsilon^2 \cdot \delta^{\text{un}}_{G_i}(U_j)}{3(\beta+1)}\right), \tag{2}$$

where the equality follows substituting the definition of $\zeta(U)$. Let $E(i,j)$ denote the set of edges crossing U_j in G_i. Since each edge e in G_i has weight p_e, we have

$$\delta^{\text{un}}_{G_i}(U_j) = \sum_{e \in E(i,j)} p_e = \sum_{e \in E(i,j)} \frac{3d(\beta+1)u_e}{\epsilon^2 k_e} \cdot \log n,$$

Substituting this into Eq. (2) shows

$$q_j \le 2\exp\left(-d\sum_{e\in E(i,j)}\frac{u_e}{k_e}\cdot\log n\right) = 2n^{-d\cdot\delta_p(U_j)}.$$

Since $\delta_p(U_j)\ge 1$, we have $q_j \le 2n^{-d}$, so

$$\sum_{j\le n^2} q_j \le n^2\cdot 2n^{-d} = 2n^{-d+2} = O(n^{-d+2}). \tag{3}$$

For $j\ge n^2$, we express j as $j = n^{2\lambda}$. The number of λ-minimum cuts is at most j (see, e.g., Karger and Stein [28]) and $\delta_p(U_1)=1$, so $\delta_p(U_j)\ge\lambda$, so $\delta_p(U_j)\ge\frac{\log j}{2\log n}$. This implies, for $j\ge n^2$, $q_j\le 2n^{-(d\log j)/(2\log n)} = 2j^{-d/2}$. Combining this with Eq. (3), we can conclude

$$\sum_{j\ge 1} q_j \le O(n^{-d+2}) + 2\int_{n^2}^{\infty} j^{-d/2}dj = O(n^{-d+2}). \qquad \blacktriangleleft$$

Now we are ready to prove Theorem 8.

Proof of Theorem 8. By Lemma 13, the expected number of edges in H is $\sum_e p_e = O(\beta n\log n/\epsilon^2)$, as claimed. Now consider an α-balanced cut U. We have

$$\left|\delta_H(U)-\delta_G(U)\right| = \left|\sum_i \alpha_i\delta_{H_i}(U)-\alpha_i\delta_{G_i}(U)\right| \le \sum_i \alpha_i\left|\delta_{H_i}(U)-\delta_{G_i}(U)\right|.$$

Taking a union bound over all strong components, we know that with high probability, Lemma 14 holds for every strong component. Thus, the quantity above is at most

$$\begin{aligned}\sum_i \alpha_i\zeta(U)\cdot\delta_{G_i}(U) &= \sum_i \alpha_i\epsilon\sqrt{\delta^{\mathrm{un}}_{G_i}(U)\delta_{G_i}(U)/(\beta+1)}\\ &\le \frac{\epsilon}{\sqrt{\beta+1}}\sqrt{\sum_i \alpha_i\delta^{\mathrm{un}}_{G_i}(U)\sum_i\alpha_i\delta_{G_i}(U)} && \text{(Cauchy-Schwarz)}\\ &= \frac{\epsilon}{\sqrt{\beta+1}}\sqrt{\delta^{\mathrm{un}}_G(U)\delta_G(U)}. && (\textstyle\sum_i\alpha_i\delta_{G_i}(U)=\delta_G(U))\end{aligned}$$

We conclude the proof by noting that $\delta^{\mathrm{un}}_G(U)\le(\alpha+1)\cdot\delta_G(U)$, since U is α-balanced. $\blacktriangleleft$

4 For-All Sparsification: $\Omega(n\cdot\beta/\epsilon)$ Lower Bound

Our goal in this section to prove a lower bound whose dependence on β matches the linear upper bound given by Ikeda and Tanigawa [20] on the size of for-all cut sketches:

▶ **Theorem 15.** *Fix $\beta\ge 1$ and $0<\epsilon<1$ where $\beta/\epsilon\le n/2$. Any $(1\pm\epsilon)$ for-all cut sketching algorithm for n-node β-balanced graphs must output $\Omega(n\cdot\beta/\epsilon)$ bits in the worst case.*

We prove a special case of our lower bound for $\beta=\Theta(n)$ and $\epsilon=\Theta(1)$ (Lemma 16). The proof for this special case contains the main ideas of our lower-bound construction for general values of β and ϵ. We defer the proof of Theorem 15 to the full paper.

▶ **Lemma 16.** *Let $\beta=8n$ and let ϵ be a sufficiently small universal constant. Any $(1\pm\epsilon)$ for-all cut sketching algorithm for n-node β-balanced graphs must output $\Omega(\beta n)$ bits in the worst case.*

We give an overview of how we prove Lemma 16. We can w.l.o.g focus on deterministic cut sketching algorithms, because running time is not a concern in Lemma 16, any randomized sketching algorithm can be derandomized by enumerating all possible coin flips.

We will choose a set of graphs $\mathcal{G}$ such that the following conditions hold:

- Every graph in $\mathcal{G}$ is β-balanced.
- The size of $\mathcal{G}$ is large (Lemma 17).
- There exists a ℓ with $|\mathcal{G}|/\ell = 2^{\Omega(\beta n)}$ such that, for every graph $G \in \mathcal{G}$, there are at most ℓ graphs in $\mathcal{G}$ that can share a $(1 \pm \epsilon)$-cut sketch with G (Lemma 18).

This way, each cut sketch works for at most ℓ graphs in $\mathcal{G}$, so any algorithm must produce at least $|\mathcal{G}|/\ell = 2^{\Omega(\beta n)}$ different cut sketches for all graphs in $\mathcal{G}$, which implies that the algorithm must output at least $\log_2(|\mathcal{G}|/\ell) = \Omega(\beta n)$ bits.

Formally, consider the set of graphs $\mathcal{G}_{2n}$ with $2n$ vertices defined as follows: every graph $G \in \mathcal{G}_{2n}$ is an unweighted bipartite graph with bipartitions L, R satisfying $|L| = |R| = n$. Fix a perfect matching from L to R. The set $\mathcal{G}_{2n}$ is defined to contain all graphs G such that the edges from L to R is exactly this perfect matching (and the set of edges from R to L are arbitrary). Let $\mathcal{G}_{2n,\beta} \subseteq \mathcal{G}_{2n}$ be the subset of graphs in $\mathcal{G}_{2n}$ that are β-balanced.

As described above, Lemma 17 gives a lower bound on the size of $\mathcal{G}_{2n,\beta}$.

▶ **Lemma 17.** *Let n_0 be a sufficiently large universal constant. If $n \geq n_0$ and $\beta = 8n$, then $|\mathcal{G}_{2n,\beta}| \geq 2^{n^2/2}$.*

The next lemma upper bounds the maximum number of graphs in $\mathcal{G}$ that can share an $(1 \pm \epsilon)$-cut sketch. Notice that if G and H have the same $(1 \pm \epsilon)$-cut sketch, then H must be a $(1 \pm 3\epsilon)$-cut sparsifier of G.

▶ **Lemma 18.** *Let $\epsilon > 0$ be a sufficiently small universal constant. For every $G \in \mathcal{G}_{2n}$, the number of graphs in $\mathcal{G}_{2n}$ that are $(1 \pm 3\epsilon)$-cut sparsifiers of G is at most $2^{n^2/4}$.*

We now prove Lemma 16 and defer the proofs of Lemmas 17 and 18 to the full paper.

Proof of Lemma 16. We work with graphs with $2n$ vertices (rather than n vertices) to make the presentation easier. This is equivalent because we aim to prove a lower bound of $\Omega(\beta n)$.

Fix any $(1 \pm \epsilon)$ for-all cut sketching algorithm. Consider running this algorithm on all graphs in $\mathcal{G}_{2n,\beta}$. Every graph in $\mathcal{G}_{2n,\beta}$ is β-balanced, so the algorithm must map every $G \in \mathcal{G}_{2n,\beta}$ to a bit string (i.e., cut sketch), and graphs that are not $(1 \pm 3\epsilon)$-cut sparsifiers of each other must be mapped to different strings. By Lemma 17, there are at least $2^{n^2/2}$ graphs in $\mathcal{G}_{2n,\beta}$, and by Lemma 18, at most $2^{n^2/4}$ graphs can be mapped to the same bit string. Therefore, the algorithm must output at least $\frac{2^{n^2/2}}{2^{n^2/4}} = 2^{n^2/4}$ distinct bit strings. This implies that the algorithm must output at least $\frac{n^2}{4} = \frac{1}{32}\beta n$ bits in the worst case. ◀

5 For-Each Cut Sketch: $\widetilde{O}(n \cdot \sqrt{\beta}/\epsilon)$ Upper Bound

In this section, we give an upper bound on the size of cut sketches in the for-each setting.

▶ **Theorem 19.** *Let G be an n-vertex β-balanced graph with edge weights in $[1, \mathrm{poly}(n)]$. There exists a $(1 \pm \epsilon)$ for-each cut sketch of size $O(n\beta^{1/2}\log^3 n/\epsilon)$ bits that approximates the value $w(S, V \setminus S)$ of every directed cut $S \subseteq V$ with high probability.*

We will prove a lower bound of $\Omega(n \cdot (\beta/\epsilon)^{1/2})$ bits in Section 6.

Overview of Our Approach. Our approach is inspired by the cut sketching algorithm for undirected graphs by Andoni et al. [6]. We first partition the edges into $\ell = O(\log n)$ disjoint sets $(E_i)_{i=1}^{\ell}$ based on their weights. Edges in $G_i = (V, E_i, w)$ have roughly the same weight and we can essentially treat G_i as an unweighted graph. For each graph G_i, we ignore edge directions, and iteratively remove and store edges belonging to sparse cuts.

Note that G_i may not be balanced. Even when G_i is balanced, the dense components of G_i may not be balanced. Despite this, we show that we can estimate the dense components' contribution to the cut value via random sampling. This is because we can bound the variance within each component, and in the end, upper bound their sum (i.e., the overall variance) using the β-balance condition.

One of our main technical contributions is to derive a tighter upper bound on the variance of random sampling. If we trace our analysis back to the undirected case, we remove some redundant terms in the analysis of [6]. This tighter variance bound is critical, because we cannot obtain the right space dependence on β without it (see the full paper). In addition, our new analysis can be traced back to the undirected case, which will simplify the algorithm of [6]. We can obtain a for-each sketching algorithms for undirected graphs without downsampling or low-accuracy for-all sparsifiers, and the output is a graph.

5.1 Sketching and Recovery Algorithms

Let $G = (V, E, w)$ be an n-node β-balanced directed graph with edge weights $w_e \in [1, \text{poly}(n)]$. It is worth noting that, when constructing the cut sketch, we do not know the cut query $S \subseteq V$. The cut query S is only given as input to the recovery algorithm.

The Sketching Algorithm. We describe our overall cut sketching algorithm (Algorithm 2). We partition the edges into $O(\log n)$ weight classes. For each weight class, we iteratively store and remove all edges that belong to some λ-sparse cut (defined in Equation 4). When there are no λ-sparse cuts remain, we sample α incoming and outgoing edges at each vertex among the remaining edges. The values of λ and α will be specified later in our analysis.

For a directed graph $G = (V, E, w)$, we say a cut $(S, \overline{S})$ is λ-*sparse* if the following holds:

$$|E(S, \overline{S})| + |E(\overline{S}, S)| \leq \lambda \cdot \min\big(|S|, |\overline{S}|\big). \tag{4}$$

The Recovery Algorithm. Algorithm 3 is our recovery algorithm that queries the cut sketch $\text{sk}(G)$ (i.e., the output of Algorithm 2). We first establish some notation. Recall that Algorithm 2 decomposes G into $(G_i)_{i=1}^{\ell}$ according to the edge weights. Let $V_i = (V_{ij})_j$ denote the set of dense components in G_i after we iteratively remove the sparse cuts in G_i.

Algorithm 3 approximates $w(S, \overline{S})$ by adding the total contribution of the sparse-cut edges and the dense-component edges. Let J_S denote the total weight of sparse-cut edges that go from S to $\overline{S}$ in all of the graphs G_i (which we store deterministically). Let I_S be the estimator for the total weight of dense-component edges leaving S in all G_i as defined in Algorithm 3. Algorithm 3 returns $I_S + J_S$ as the final answer.

Correctness and Size Guarantees. We state the correctness of our recovery algorithm (Algorithm 3) in Lemma 20 and the output size of our sketching algorithm (Algorithm 2) in Lemma 21. Theorem 19 follows immediately from Lemmas 20 and 21; we prove the latter before proving the former.

▶ **Lemma 20** (Correctness of Algorithm 3). *Let* $\text{sk}(G)$ *be the output of Algorithm 2. Fix a cut query* $S \subseteq V$*. With probability at least* $2/3$*, the value* $(I_S + J_S)$ *returned by Algorithm 3 on input* $(S, \text{sk}(G))$ *satisfies* $\Big|(I_S + J_S) - w(S, \overline{S})\Big| \leq O(\epsilon) \cdot w(S, \overline{S})$.

Algorithm 2 Compute a $(1 \pm O(\epsilon))$ for-each cut sketch.

Input : An n-vertex β-balanced graph $G = (V, E, w)$ with edge weights $w_e \in [1, \text{poly}(n)]$, and $0 < \epsilon < 1$.
Output : A $(1 \pm O(\epsilon))$ for-each cut sketch $\text{sk}(G)$ of size $\widetilde{O}(n\beta^{1/2}/\epsilon)$.

Set $\alpha = \lambda = \beta^{1/2}/\epsilon$.
Partition the edges into $\ell = O(\log n)$ weight classes $E_1, \ldots, E_\ell$ where $E_i = \{e : w_e \in [2^{i-1}, 2^i)\}$.
Each weight class E_i defines a (possibly unbalanced) graph $G_i = (V, E_i, w)$.
for $i = 1$ **to** ℓ **do**
  while *there exists a λ-sparse cut (defined in Equation (4)) in G_i* **do**
    Remove all edges (in both directions) in this cut and store them in $\text{sk}(G_i)$.
  In $\text{sk}(G_i)$, store the (dense) components $\{V_{ij}\}_j$ of G_i.
  For every V_{ij} and every $u \in V_{ij}$, store the number of (remaining) incoming and outgoing edges at u in G_i, i.e., $d^{\text{in}}_{ij}(u) = |E_i(V_{ij}, u)|$ and $d^{\text{out}}_{ij}(u) = |E_i(u, V_{ij})|$.
  At each vertex $u \in V$, sample with replacement α edges from the (remaining) outgoing edges (u, v) and store them in $\text{sk}(G_i)$. Do the same for incoming edges.
return $\text{sk}(G) = \bigcup_i \text{sk}(G_i)$.

Algorithm 3 Query the cut value $w(S, \overline{S})$ from $\text{sk}(G)$.

Input : A cut query $S \subseteq V$ and a cut sketch $\text{sk}(G)$ (output of Algorithm 2).

for *each* $\text{sk}(G_i)$ *in* $\text{sk}(G)$ **do**
  for *each dense component V_{ij} in G_i* **do**
    Let S_{ij} denote the smaller set of $(V_{ij} \cap S)$ and $(V_{ij} \cap \overline{S})$.
    if $S_{ij} = V_{ij} \cap S$ **then**
      Estimate the total weight of edges leaving S_{ij}: For every $u \in S_{ij}$, set

$$I_{S_{ij}}(u) = \frac{d^{\text{out}}_{ij}(u)}{\alpha} \sum_{q=1}^{\alpha} \chi_{ij}(u, q) w_{ij}(u, q) \tag{5}$$

      where $d^{\text{out}}_{ij}(u)$ is the out-degree of u in V_{ij}, $\chi_{ij}(u, q) = 1$ if the q-th sampled outgoing edge at u crosses S and $\chi_{ij}(u, q) = 0$ otherwise, and $w_{ij}(u, q)$ is the weight of the q-th sampled edge.
    else
      Estimate the total weight of edges entering $S_{ij} = V_{ij} \cap \overline{S}$ instead:
      For every $u \in S_{ij}$, set $I_{S_{ij}}(u)$ as in (5), using $d^{\text{in}}_{ij}(u)$ instead of $d^{\text{out}}_{ij}(u)$, and $\chi_{ij}(u, q)$ indicates if the q-th sampled incoming edge at u crosses S.
    The estimated contribution from V_{ij} is $I_{V_{ij}} = \sum_{u \in S_{ij}} I_{S_{ij}}(u)$.
  The estimated contribution from G_i is $I_{G_i} = \sum_{V_{ij} \in G_i} I_{V_{ij}}$.
Compute $I_S = \sum_i I_{G_i}$, the estimate of the cut value from all dense-component edges.
Compute J_S, the total weight of λ-sparse cut edges that leaves S in all G_i's.
return $I_S + J_S$.

▶ **Lemma 21** (Output Size of Algorithm 2). *The output* $\mathrm{sk}(G)$ *of Algorithm 2 has size* $\widetilde{O}(n(\lambda+\alpha)) = \widetilde{O}(n\beta^{1/2}/\epsilon)$.

Proof. Without loss of generality, we assume $\epsilon = \Omega(1/n)$, otherwise we can store all edges exactly using $\widetilde{O}(n/\epsilon)$ bits. Algorithm 2 produces $\ell = O(\log n)$ weight classes; each weight class defines a graph G_i. In every G_i:

- First we iteratively store and remove edges in λ-sparse cuts. We can upper bound the total number of edge removed using the following charging argument: When a λ-sparse cut is removed, we charge the cut size evenly to the vertices on the smaller side of the cut. Since the cut is λ-sparse, every vertex on the smaller side gets charged at most λ edges. Each vertex can be charged at most $O(\log n)$ times because it can be in the smaller side $O(\log n)$ times. Therefore, $\mathrm{sk}(G)$ stores at most $O(\lambda n \log n)$ sparse edges, which takes $O(\lambda n \log^2 n)$ bits.
- On the remaining graph, the connected components are disjoint, so we can also store the partition of vertices into these dense components in $O(n \log n)$ bits.
- We can store the (remaining) in- and out-degree of every vertex in $O(n \log n)$ bits.
- We sample $O(\alpha)$ edges at each vertex in V, which requires $O(\alpha n \log n)$ bits.

Thus, for every G_i we store $O(\lambda n \log^2 n + n \log n + \alpha n \log n) = O(n(\lambda+\alpha)\log^2 n)$ bits. Since $\alpha = \lambda = \beta^{1/2}/\epsilon$, the size of $\mathrm{sk}(G_i)$ is $O(n\beta^{1/2}\log^2 n/\epsilon)$. The size of $\mathrm{sk}(G) = \bigcup_i \mathrm{sk}(G_i)$ is

$$O(\log n) \cdot O(n\beta^{1/2}\log^2 n/\epsilon) = O(n\beta^{1/2}\log^3 n/\epsilon). \qquad \blacktriangleleft$$

Now we prove Lemma 20 (the correctness of Algorithm 3). Note that it follows immediately from the following lemma and Chebyshev's inequality.

▶ **Lemma 22.** *The estimator returned in Algorithm 3 is unbiased, i.e.,* $\mathbb{E}[I_S] + J_S = w(S,\overline{S})$. *Moreover, the variance of* I_S *is* $\mathrm{Var}[I_S] \le O(\beta/\alpha\lambda)w(S,\overline{S})^2$.

Proof of Lemma 20. Algorithm 2 sets $\alpha = \lambda = \beta^{1/2}\epsilon^{-1}$, so Lemma 22 implies $\mathrm{Var}[I_S] \le O(\epsilon^2)\cdot w(S,\overline{S})^2$. By Chebyshev's inequality, with probability at least $2/3$,

$$\left|(I_S + J_S) - w(S,\overline{S})\right| \le O(\epsilon)\cdot w(S,\overline{S}). \qquad \blacktriangleleft$$

To prove Theorem 19, all that remains is to prove Lemma 22.

Proof of Lemma 22. Recall that our estimator is

$$I_S = \sum_{G_i}\sum_{V_{ij}}\sum_{u\in S_{ij}} I_{S_{ij}}(u),$$

where i sums over the graphs G_i defined according to the edge weights, j sums over the dense components in each G_i after all sparse cuts are removed, and u sums over the vertices of S_{ij}. Without loss of generality, we can assume $|V_{ij}\cap S| \le |V_{ij}\cap\overline{S}|$ and hence $S_{ij} = V_{ij}\cap S$. Otherwise, Algorithm 3 works with $V_{ij}\cap\overline{S}$ and queries for the incoming edges instead. Under this assumption, we always work with outgoing edges:

$$I_{S_{ij}}(u) = \frac{d_{ij}^{\mathrm{out}}(u)}{\alpha}\sum_{q=1}^{\alpha}\chi_{ij}(u,q)w_{ij}(u,q).$$

Every edge $e \in E(S,\overline{S})$ belongs to exactly one G_i, and in that G_i it is either a sparse-cut edge, or a dense-component edge in exactly one V_{ij}. Consequently, to prove $I_S + J_S$ is unbiased, it suffices to prove that $I_{S_{ij}}(u)$ is unbiased. In the dense component V_{ij} of G_i, the

total contribution of edges leaving u to $w(S,\overline{S})$ is $\sum_{r=1}^{d_{ij}^{\text{out}}(u)} \chi(u,r)\cdot w(u,r)$, where r indexes the edges leaving u, $w(u,r)$ is the weight of the r-th edge leaving u, and $\chi(u,r)$ indicates if this edge goes from S to $\overline{S}$. Let $e(u,r)$ denote the r-th edge leaving u and $e_{ij}(u,q)$ denote the q-th sampled edge leaving u within V_{ij}. Summing over the α sampled edges, we have

$$\begin{aligned}\mathbb{E}\left[I_{S_{ij}}(u)\right] &= \frac{d_{ij}^{\text{out}}(u)}{\alpha}\cdot\sum_{q=1}^{\alpha}\mathbb{E}\left[\chi_{ij}(u,q)\cdot w_{ij}(u,q)\right]\\ &= \frac{d_{ij}^{\text{out}}(u)}{\alpha}\cdot\sum_{q=1}^{\alpha}\sum_{r=1}^{d_{ij}^{\text{out}}(u)}\Pr[e_{ij}(u,q)=e(u,r)]\cdot\chi(u,r)\cdot w(u,r)\\ &= \sum_{r=1}^{d_{ij}^{\text{out}}(u)}\chi(u,r)\cdot w(u,r),\end{aligned}$$

where the last equality holds because each sample has the same variance, and the q-th sample is drawn uniformly among all outgoing edges at u, i.e., $\Pr[e_{ij}(u,q)=e(u,r)]=\frac{1}{d_{ij}^{\text{out}}(u)}$. The expectation of $I_{S_{ij}}(u)$ is exactly the contribution of edges leaving u to $w(S,\overline{S})$, so $I_{S_{ij}}(u)$ is unbiased.

For the rest of the proof, we upper bound the variance of I_S. We assume without loss of generality that $J_S=0$, i.e., no sparse edges were ever stored and removed by Algorithm 2. This is because we are trying to prove the statement

$$\text{Var}\left[I_S\right]\le O\left(\frac{\beta}{\alpha\lambda}\right)w(S,\overline{S})^2 = O\left(\frac{\beta}{\alpha\lambda}\right)\cdot\left(\mathbb{E}\left[I_S\right]+J_S\right)^2,$$

so setting $J_S=0$ only makes the right-hand side smaller and hence, the proof more difficult.

We introduce some notation: recall that $(V_{ij})_j$ is the set of dense components of G_i and $S_{ij}=V_{ij}\cap S$. We use $X_{ij}=|E_i(S_{ij},\overline{S_{ij}})|$ to denote the *number* of edges from S_{ij} to $V_{ij}\cap\overline{S}$ in G_i, and $\overline{X_{ij}}=|E_i(\overline{S_{ij}},S_{ij})|$ the number of edges in the reverse direction. Let $X_i=\sum_j X_{ij}$ and $\overline{X_i}=\sum_j\overline{X_{ij}}$, so that X_i is the total number of dense-component edges that go from S to $\overline{S}$ in G_i.

Since there are no λ-sparse cuts (defined in (4)) at the end of Algorithm 2, we have $X_{ij}+\overline{X_{ij}}>\lambda\min(|S_{ij}|,|\overline{S_{ij}}|)$. Since we assume $|S_{ij}|\le|\overline{S_{ij}}|$, this condition implies the following for every dense component V_{ij}, $\lambda\left|S_{ij}\right|\le X_{ij}+\overline{X_{ij}}$.

Fix any $u\in S_{ij}$. We first upper bound $\text{Var}\left[I_{S_{ij}}(u)\right]$ and then work our way up the definition of I_S. Since $\chi_{ij}(u,q)$ is a Bernoulli random variable with mean $\left|E_i(u,\overline{S_{ij}})\right|/d_{ij}^{\text{out}}(u)$, its variance is

$$\text{Var}\left[\chi_{ij}(u,q)\right]=\frac{\left|E_i(u,\overline{S_{ij}})\right|}{d_{ij}^{\text{out}}(u)}\cdot\frac{\left|E_i(u,S_{ij})\right|}{d_{ij}^{\text{out}}(u)}.$$

Now from the definition of $I_{S_{ij}}(u)$, we have

$$\begin{aligned}
\operatorname{Var}\left[I_{S_{ij}}(u)\right] &= \frac{d_{ij}^{\text{out}}(u)^2}{\alpha^2}\sum_{q=1}^{\alpha}\operatorname{Var}\left[\chi_{ij}(u,q)\right]w_{ij}(u,q)^2 \\
&\le \frac{d_{ij}^{\text{out}}(u)^2}{\alpha^2}\cdot\alpha\cdot\frac{\left|E_i(u,\overline{S_{ij}})\right|}{d_{ij}^{\text{out}}(u)}\cdot\frac{\left|E_i(u,S_{ij})\right|}{d_{ij}^{\text{out}}(u)}\cdot 2^{2i} && (w_e \le 2^i \text{ for all } e\in E_i) \\
&= \frac{2^{2i}}{\alpha}\left|E_i(u,\overline{S_{ij}})\right|\cdot\left|E_i(u,S_{ij})\right| \\
&\le \frac{2^{2i}}{\alpha}\left|E_i(u,\overline{S_{ij}})\right|\cdot\left|S_{ij}\right|. && (\left|E_i(u,S_{ij})\right| \le |S_{ij}|)
\end{aligned}$$

In Algorithm 3, we set $I_{V_{ij}} = \sum_{u\in S_{ij}} I_{S_{ij}}(u)$, so

$$\begin{aligned}
\operatorname{Var}\left[I_{V_{ij}}\right] = \sum_{u\in S_{ij}}\operatorname{Var}\left[I_{S_{ij}}(u)\right] &\le \sum_{u\in S_{ij}}\frac{2^{2i}}{\alpha}\left|S_{ij}\right|\cdot\left|E_i(u,\overline{S_{ij}})\right| \\
&= \frac{2^{2i}}{\alpha}\cdot\left|S_{ij}\right|\cdot X_{ij} && (X_{ij} = \left|E_i(S_{ij},\overline{S_{ij}})\right|) \\
&\le \frac{2^{2i}}{\alpha\lambda}\left(X_{ij}+\overline{X_{ij}}\right)X_{ij}. && ((S_{ij},\overline{S_{ij}}) \text{ is not } \lambda\text{-sparse})
\end{aligned}$$

Summing across every dense component V_{ij} in G_i, we get

$$\begin{aligned}
\operatorname{Var}\left[I_{G_i}\right] = \sum_j \operatorname{Var}\left[I_{V_{ij}}\right] &\le \frac{2^{2i}}{\alpha\lambda}\sum_j\left(X_{ij}+\overline{X_{ij}}\right)X_{ij} \\
&\le \frac{2^{2i}}{\alpha\lambda}\left(X_i+\overline{X}_i\right)X_i. && (X_i = \textstyle\sum_j X_{ij} \text{ and } \overline{X_i} = \sum_j \overline{X_{ij}})
\end{aligned}$$

Finally, we sum across the weight classes indexed by i to obtain

$$\begin{aligned}
\operatorname{Var}\left[I_S\right] &= \sum_i \operatorname{Var}\left[I_{G_i}\right] \\
&\le \frac{1}{\alpha\lambda}\sum_i\left(2^i\right)^2\left(X_i+\overline{X}_i\right)X_i \\
&= \frac{1}{\alpha\lambda}\left[\sum_i\left(2^iX_i\right)^2+\sum_i 2^i\overline{X}_i\cdot\sum_i 2^iX_i\right] \\
&\le \frac{4}{\alpha\lambda}\left[w(S,\overline{S})^2+w(\overline{S},S)\cdot w(S,\overline{S})\right] && (w_e \ge 2^{i-1} \text{ for all } e\in E_i) \\
&\le \frac{4}{\alpha\lambda}\left[w(S,\overline{S})^2+\beta w(S,\overline{S})^2\right] && (w(\overline{S},S)\le\beta w(S,\overline{S})) \\
&= O\left(\frac{\beta}{\alpha\lambda}\right)w(S,\overline{S})^2. && \blacktriangleleft
\end{aligned}$$

5.2 For-Each Cut Sketch: Faster Algorithms

We can speed up the algorithms in the previous section and prove the following theorem.

▶ **Theorem 23.** *Consider the same setting as in Theorem 19. That is, a $(1\pm\epsilon)$ for-each cut sketch of size $O(\beta^{1/2}n\log^5 n/\epsilon)$ bits exists for any n-vertex β-balanced graph G. Now in addition, we can compute such a cut sketch in time $\widetilde{O}(m+\beta^{1/2}n/\epsilon)$.*

For undirected graphs, Jambulapati and Sidford [21] showed how to construct for-each cut sketches in nearly-linear time. Instead of trying to repeatedly find sparse cuts, they showed how to sketch expander graphs (graphs with high conductance) and then decompose the input graph using expander partitioning algorithms.

Intuitively, we should be able to speed up our algorithm using a similar approach, because when we partition the graph by removing sparse cuts, we do not look at the direction of the edges. However, the analysis in [21] does not apply to our setting because they focus on sketching quadratic forms. The quadratic form of a directed Laplacian ignores edge directions and hence does not preserve the directed cut values. On a more technical level, their analysis relies heavily on the notion of conductance, which is not canonically defined for directed graphs. In our setting, we cannot bound the variance of our estimator even if we have a directed graph whose undirected version is an expander (see the full paper for more details).

Because our main focus in this paper is to derive cut sketches with *optimal size* (especially with a tight dependence on β) rather than to obtain best runtime of computing such cut sketches, we defer the proof of Theorem 23 to the full paper.

6 For-Each Cut Sketch: $\Omega(n \cdot \sqrt{\beta/\epsilon})$ Lower Bound

In this section, we prove that the size of for-each cut sketches must scale with $\sqrt{\beta}$.

▶ **Theorem 24.** *Fix $\beta \geq 1$ and $0 < \epsilon < 1$ with $\left(\beta/\epsilon\right)^{1/2} \leq \frac{n}{2}$. A $(1 \pm \epsilon)$ for-each cut sketching algorithm for n-node β-balanced graphs must output $\Omega(n \cdot (\beta/\epsilon)^{1/2})$ bits in the worst case.*

To prove this, we will need the following folklore result from communication complexity:

▶ **Lemma 25.** *Given a bit string $s \in \{0,1\}^N$, if there is a data structure D that allows one to recover each bit of s with marginal probability at least $2/3$, then D must use $\Omega(N)$ bits.*

We first prove a special case of our lower bound for specific values of $\beta = \Theta(n^2)$ and $\epsilon = \Theta(1)$ (Lemma 26). The proof for this special case is easier to explain and it contains the key ingredients of our construction for the general lower bound.

▶ **Lemma 26.** *For $\beta = n^2$ and $\epsilon = \frac{1}{10}$, any $(1 \pm \epsilon)$ for-each cut sketching algorithm for n-node β-balanced graphs must output $\Omega(n \cdot \beta^{1/2})$ bits in the worst case.*

Proof. At a high level, we will encode a bit string s of length $\Omega(n^2)$ into an n-node β-balanced graph, such that given a $(1 \pm \epsilon)$ for-each cut sketch, we can recover each bit of s with high constant probability. Then by Lemma 25, the cut sketch must have use $\Omega(|s|) = \Omega(n^2) = \Omega(n\beta^{1/2})$ bits.

Given a bit string s of length $\frac{n^2}{4}$, we construct a graph as follows. We start with an $\frac{n}{2} \times \frac{n}{2}$ complete bipartite digraph where edges go from left to right. We set the weight of the i-th bipartite edge to $s_i + 1$ (so either 1 or 2). We add a unit-weight cycle that leaves each side exactly once. See Fig. 2 for an example of our construction.

We first show that the graph is β-balanced for $\beta = n^2$. The graph is strongly connected because it contains a cycle. Note that all edge weights are in $[1, 2]$ and there are in total $\frac{n^2}{4} + n \leq \frac{n^2}{2}$ edges in the graph. Therefore, for every non-empty set $S \subset V$, the total weight of edges leaving (or entering) S is at least 1 and at most n^2, so the graph is (n^2)-balanced.

It remains to show that we can recover each bit of s from a $(1 \pm \epsilon)$ cut sketch. Let L denote the left vertices and R the right vertices. Fix any coordinate of s and suppose it corresponds to the edge (u, v) for some $u \in L$ and $v \in R$. To recover this bit of s, we need to decide whether $w(u, v)$ is 1 or 2. Consider the cut value leaving $S = \{u\} \cup R \setminus \{v\}$. The cycle contributes a fixed amount to this cut (independent of the weights of the bipartite

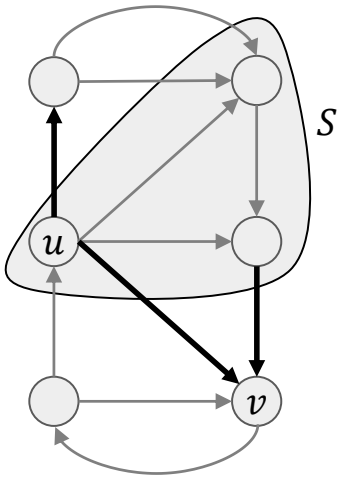


Figure 2 In this example, the cut value $w(S, \overline{S})$ is 2 or 3, depending on the value of $w(u, v)$. Thus, a (1 ± 0.1)-approximation to $w(S, \overline{S})$ allows us to decode the bit in s corresponding to edge (u, v). (For readability we omit other bipartite edges from L to R.)

edges), which is at most 3. More importantly, (u, v) is the only bipartite edge leaving S. Since $\epsilon = \frac{1}{10}$ and the sketch returns this cut value within a factor of $(1 \pm \epsilon)$ with probability at least $2/3$, we can recover $w(u, v)$ with probability at least $2/3$. ◀

Our lower bound construction for general values of β and ϵ builds on the one in the proof of Lemma 26. At a high level, instead of using a bipartite graph with two clusters, we will use multiple clusters where the size of each cluster depends on β and ϵ.

Proof of Theorem 24. Let $k = \sqrt{\beta/\epsilon}$. We will encode a bit string s of length $\Omega(nk)$ into an n-node graph G such that (1) G is (3β)-balanced, and (2) we can recover each bit of s with high constant probability given a $(1 \pm c \cdot \epsilon)$ for-each cut sketch of G where $c = 10^{-2}$. By Lemma 25, the cut sketch must have at least $\Omega(nk) = \Omega(n \cdot (\beta/\epsilon)^{1/2})$ bits.

Without loss of generality, we assume k is an integer and n is a multiple of k. We partition n vertices into $t = n/k$ clusters of size k, which we denote by $V_1, \ldots, V_t$. Since we assume $n \geq 2k$, there are at least two clusters.

Let s be a bit string of length $k^2(t-1) = \Omega(nk)$. We partition s into $(t-1)$ blocks where each block has length k^2. We encode the i-th block of s in a $k \times k$ complete bipartite digraph where edges go from V_i to V_{i+1}. As in the proof of Lemma 26, a bipartite edge (u, v) for $u \in V_i$ and $v \in V_{i+1}$ has weight $s_{i,(u,v)} + 1$ (so either 1 or 2). For every $1 \leq i \leq t-1$, we add a cycle between V_i and V_{i+1} that leaves V_i and V_{i+1} exactly once. Now, in contrast to the previous construction, these cycle edges have weight $1/\epsilon$.

We first show that G is (3β)-balanced. Fix any non-empty set $S \subset V$. Let G_i denote the subgraph between V_i and V_{i+1} which contains k^2 bipartite edges and one cycle. Let $w_i(S, \overline{S})$ denote the total weight of edges leaving S in G_i. We will show that $w_i(S, \overline{S})$ and $w_i(S, \overline{S})$ are within a factor of 3β of each other. Because G is strongly connected and $w(S, \overline{S}) = \sum_{i=1}^{t-1} w_i(S, \overline{S})$, we can conclude that G is (3β)-balanced.

Without loss of generality, we assume both $w_i(S, \overline{S})$ and $w_i(\overline{S}, S)$ are positive. The cut value $w_i(S, \overline{S})$ remains the same if we restrict G_i on vertices $(V_i \cup V_{i+1})$ and consider the cut query $S \cap (V_i \cup V_{i+1})$. The cycle contributes equally in both directions, so without loss of generality, we can assume the cycle has minimum contribution, which is $\frac{1}{\epsilon}$. (If the cycle

contributes more, the cut is more balanced.) The total weight of the bipartite edges is at most $2k^2 = \frac{2\beta}{\epsilon}$. Therefore, the ratio between the cut values in both directions is at most $\frac{(2\beta/\epsilon)+(1/\epsilon)}{1/\epsilon} = 2\beta + 1 \leq 3\beta$.

It remains to show that we can recover every bit of s from a cut sketch. Fix any bit of s. Suppose this bit $s_{i,(u,v)}$ corresponds to the edge (u, v) for some $u \in V_i$ and $v \in V_{i+1}$, we query the cut value leaving $S_{(u,v)} = \{u\} \cup \left(V_{i+1} \setminus \{v\}\right) \bigcup_{j=i+2}^{t-1} V_j$. The only bipartite edge leaving $S_{(u,v)}$ is the edge (u, v), which has weight either 1 or 2. There are at most 5 cycle edges leaving $S_{(u,v)}$ (at most 1 from G_{i-1}, 3 from G_i, and 1 from G_{i+1}), whose total weight is fixed and at most $\frac{5}{\epsilon}$. Therefore, if we can compute an $(1 \pm c \cdot \epsilon)$ approximation to the cut value for $c = 10^{-2}$, we can recover the corresponding bit of s. ◀

7 Conclusion

In this paper, we considered the question of sparsifying directed graphs. We focused on graphs that are β-balanced, where the ratio between the cut value in two directions is at most β. We gave upper and lower bounds on the size of the cut sketch with almost tight dependence on β, under both the standard "for-all" notion (i.e., simultaneously preserving the value of all cuts) and the "for-each" notion (introduced by Andoni *et. al* [6]) of cut sparsification. More specifically, we showed that under the "for-all" notion, the linear dependence on β obtained by Ikeda and Tanigawa [20] is tight. For the "for-each" notion, we gave a data structure that preserves cut values whose size scales as $\sqrt{\beta}$, thereby beating the "for-all" lower bound. We also showed that this dependence on $\sqrt{\beta}$ is tight. Our lower bounds hold not only for sparsifiers (i.e., graph encodings), but also for arbitrary data structures.

An interesting direction for future work is to consider the *spectral* sparsification of directed graphs. Cohen et al. [14, 13] (see also Chu et al. [11]) introduced a novel definition of directed sparsification and leveraged it to solve directed Laplacian linear systems. However, their work is not immediately relevant to ours because their directed spectral sparsifiers do not necessarily preserve directed cut values. This motivates the following natural question: *is there a notion of spectral sparsification that generalizes cut sparsification in directed graphs?* (Note that this is indeed the case for undirected graphs, where spectral sparsifiers also preserve cut values.) A natural candidate would be a sparse graph that preserves $\sum_{(u,v)\in E} \left((x_u - x_v)^+\right)^2$ for all real vectors x, where $y^+ = \max(0, y)$. Note that if $x \in \{0, 1\}^{|V|}$, then this sum represents directed cut values, which is analogous to the correspondence between cut and spectral sparsification in undirected graphs. It would be interesting to explore if preserving this sum in directed graphs has interesting applications beyond preserving cuts, and if so, whether there exist sparse graphs that preserve this sum approximately for balanced directed graphs.

References

1 Kook Jin Ahn and Sudipto Guha. Graph sparsification in the semi-streaming model. In *International Colloquium on Automata, Languages, and Programming*, pages 328–338. Springer, 2009.

2 Kook Jin Ahn, Sudipto Guha, and Andrew McGregor. Analyzing graph structure via linear measurements. In *Proceedings of the Twenty-Third ACM-SIAM Symposium on Discrete Algorithms*, pages 459–467, 2012.

3 Kook Jin Ahn, Sudipto Guha, and Andrew McGregor. Graph sketches: sparsification, spanners, and subgraphs. In *Proceedings of the 31st ACM Symposium on Principles of Database Systems*, pages 5–14, 2012.

4 Kook Jin Ahn, Sudipto Guha, and Andrew McGregor. Spectral sparsification in dynamic graph streams. In *Proceedings of the 16th International Workshop on Approximation Algorithms for Combinatorial Optimization Problems and the 17th International Workshop on Randomization and Computation*, pages 1–10, 2013.

5 Zeyuan Allen Zhu, Zhenyu Liao, and Lorenzo Orecchia. Spectral sparsification and regret minimization beyond matrix multiplicative updates. In *Proceedings of the 47th ACM Symposium on Theory of Computing*, pages 237–245, 2015.

6 Alexandr Andoni, Jiecao Chen, Robert Krauthgamer, Bo Qin, David P Woodruff, and Qin Zhang. On sketching quadratic forms. In *Proceedings of the 2016 ACM Conference on Innovations in Theoretical Computer Science*, pages 311–319. ACM, 2016.

7 Joshua Batson, Daniel A Spielman, and Nikhil Srivastava. Twice-Ramanujan sparsifiers. *SIAM Journal on Computing*, 41(6):1704–1721, 2012.

8 András A Benczúr and David R Karger. Randomized approximation schemes for cuts and flows in capacitated graphs. *SIAM Journal on Computing*, 44(2):290–319, 2015.

9 Dietmar Berwanger, Anuj Dawar, Paul Hunter, Stephan Kreutzer, and Jan Obdržálek. The DAG-width of directed graphs. *Journal of Combinatorial Theory, Series B*, 102(4):900–923, 2012.

10 Charles Carlson, Alexandra Kolla, Nikhil Srivastava, and Luca Trevisan. Optimal lower bounds for sketching graph cuts. In *Proceedings of the 30th ACM-SIAM Symposium on Discrete Algorithms*, pages 2565–2569, 2019.

11 Timothy Chu, Yu Gao, Richard Peng, Sushant Sachdeva, Saurabh Sawlani, and Junxing Wang. Graph sparsification, spectral sketches, and faster resistance computation, via short cycle decompositions. In *Proceedings of the 59th IEEE Annual Symposium on Foundations of Computer Science*, pages 361–372, 2018.

12 Fan Chung. Laplacians and the cheeger inequality for directed graphs. *Annals of Combinatorics*, 9(1):1–19, 2005.

13 Michael B. Cohen, Jonathan Kelner, Rasmus Kyng, John Peebles, Richard Peng, Anup B. Rao, and Aaron Sidford. Solving directed Laplacian systems in nearly-linear time through sparse LU factorizations. In *Proceedings of the 59th IEEE Symposium on Foundations of Computer Science*, pages 898–909, 2018.

14 Michael B. Cohen, Jonathan Kelner, John Peebles, Richard Peng, Anup B. Rao, Aaron Sidford, and Adrian Vladu. Almost-linear-time algorithms for Markov chains and new spectral primitives for directed graphs. In *Proceedings of the 49th ACM Symposium on Theory of Computing*, pages 410–419, 2017.

15 Alina Ene, Gary Miller, Jakub Pachocki, and Aaron Sidford. Routing under balance. In *Proceedings of the forty-eighth annual ACM symposium on Theory of Computing*, pages 598–611. ACM, 2016.

16 Lester Randolph Ford and Delbert R Fulkerson. Maximal flow through a network. *Canadian journal of Mathematics*, 8:399–404, 1956.

17 Wai Shing Fung, Ramesh Hariharan, Nicholas JA Harvey, and Debmalya Panigrahi. A general framework for graph sparsification. In *Proceedings of the forty-third annual ACM symposium on Theory of computing*, pages 71–80. ACM, 2011.

18 Ashish Goel, Michael Kapralov, and Sanjeev Khanna. Graph sparsification via refinement sampling. *CoRR*, abs/1004.4915, 2010. `arXiv:1004.4915`.

19 Paul Hunter and Stephan Kreutzer. Digraph measures: Kelly decompositions, games, and orderings. *Theoretical Computer Science*, 399(3):206–219, 2008.

20 Motoki Ikeda and Shin-ichi Tanigawa. Cut sparsifiers for balanced digraphs. In *International Workshop on Approximation and Online Algorithms*, pages 277–294. Springer, 2018.

21 Arun Jambulapati and Aaron Sidford. Efficient $\widetilde{O}(n/\epsilon)$ spectral sketches for the Laplacian and its pseudoinverse. In *Proceedings of the 29th ACM-SIAM Symposium on Discrete Algorithms*, pages 2487–2503, 2018.

22 Thor Johnson, Neil Robertson, Paul D Seymour, and Robin Thomas. Directed tree-width. *Journal of Combinatorial Theory, Series B*, 82(1):138–154, 2001.

23 Michael Kapralov, Yin Tat Lee, Cameron Musco, Christopher Musco, and Aaron Sidford. Single pass spectral sparsification in dynamic streams. *SIAM J. Comput.*, 46(1):456–477, 2017.

24 Michael Kapralov, Aida Mousavifar, Cameron Musco, Christopher Musco, and Navid Nouri. Faster spectral sparsification in dynamic streams. *CoRR*, abs/1903.12165, 2019. `arXiv:1903.12165`.

25 Michael Kapralov, Navid Nouri, Aaron Sidford, and Jakab Tardos. Dynamic streaming spectral sparsification in nearly linear time and space. *CoRR*, abs/1903.12150, 2019. `arXiv:1903.12150`.

26 Michael Kapralov and Rina Panigrahy. Spectral sparsification via random spanners. In *Innovations in Theoretical Computer Science 2012, Cambridge, MA, USA, January 8-10, 2012*, pages 393–398, 2012.

27 David R Karger and Matthew S Levine. Random sampling in residual graphs. In *Proceedings of the thiry-fourth annual ACM symposium on Theory of computing*, pages 63–66. ACM, 2002.

28 David R Karger and Clifford Stein. A new approach to the minimum cut problem. *Journal of the ACM (JACM)*, 43(4):601–640, 1996.

29 Dmitry Kogan and Robert Krauthgamer. Sketching cuts in graphs and hypergraphs. In *Proceedings of the 2015 Conference on Innovations in Theoretical Computer Science*, pages 367–376, 2015.

30 Ioannis Koutis, Alex Levin, and Richard Peng. Improved spectral sparsification and numerical algorithms for SDD matrices. In *29th International Symposium on Theoretical Aspects of Computer Science*, pages 266–277, 2012.

31 Ioannis Koutis, Gary L. Miller, and Richard Peng. Approaching optimality for solving SDD linear systems. In *Proceedings of the 51th Annual IEEE Symposium on Foundations of Computer Science*, pages 235–244, 2010.

32 Ioannis Koutis, Gary L. Miller, and Richard Peng. A nearly-$m \log n$ time solver for SDD linear systems. In *Proceedings of the 52nd IEEE Symposium on Foundations of Computer Science*, pages 590–598, 2011.

33 Rasmus Kyng, Yin Tat Lee, Richard Peng, Sushant Sachdeva, and Daniel A. Spielman. Sparsified Cholesky and multigrid solvers for connection laplacians. In *Proceedings of the 48th ACM Symposium on Theory of Computing*, pages 842–850, 2016.

34 Yin Tat Lee and He Sun. Constructing linear-sized spectral sparsification in almost-linear time. In *Proceedings of the 56th IEEE Symposium on Foundations of Computer Science*, pages 250–269, 2015.

35 Yin Tat Lee and He Sun. An SDP-based algorithm for linear-sized spectral sparsification. In *Proceedings of the 49th ACM Symposium on Theory of Computing*, pages 678–687, 2017.

36 Henry Lin. Reducing directed max flow to undirected max flow. *Unpublished Manuscript*, 2009.

37 Adam W Marcus, Daniel A Spielman, and Nikhil Srivastava. Interlacing families II: Mixed characteristic polynomials and the kadison—singer problem. *Annals of Mathematics*, pages 327–350, 2015.

38 Ilan Newman and Yuri Rabinovich. On multiplicative Lambda-approximations and some geometric applications. *SIAM J. Comput.*, 42(3):855–883, 2013.

39 Jan Obdržálek. DAG-width: connectivity measure for directed graphs. In *Proceedings of the seventeenth annual ACM-SIAM symposium on Discrete algorithm*, pages 814–821, 2006.

40 Jonah Sherman. Nearly maximum flows in nearly linear time. In *Proceedings of the 54th Annual IEEE Symposium on Foundations of Computer Science*, pages 263–269, 2013.

41 Tasuku Soma and Yuichi Yoshida. Spectral sparsification of hypergraphs. In *Proceedings of the 13th ACM-SIAM Symposium on Discrete Algorithms*, pages 2570–2581, 2019.

42 Daniel A Spielman and Nikhil Srivastava. Graph sparsification by effective resistances. *SIAM Journal on Computing*, 40(6):1913–1926, 2011.

43 Daniel A. Spielman and Shang-Hua Teng. Spectral sparsification of graphs. *SIAM Journal on Computing*, 40(4):981–1025, 2011.

44 Ying Zhang, Zhiqiang Zhao, and Zhuo Feng. Towards scalable spectral sparsification of directed graphs. In *Proceedings of the 15th IEEE International Conference on Embedded Software and Systems*, pages 1–2, 2019.

Fault Tolerant Max-Cut

Keren Censor-Hillel
Department of Computer Science, Technion, Haifa, Israel

Noa Marelly
Department of Computer Science, Technion, Haifa, Israel

Roy Schwartz
Department of Computer Science, Technion, Haifa, Israel

Tigran Tonoyan
Department of Computer Science, Technion, Haifa, Israel

Abstract

In this work, we initiate the study of fault tolerant Max-Cut, where given an edge-weighted undirected graph $G = (V, E)$, the goal is to find a cut $S \subseteq V$ that maximizes the total weight of edges that cross S even after an adversary removes k vertices from G. We consider two types of adversaries: an adaptive adversary that sees the outcome of the random coin tosses used by the algorithm, and an oblivious adversary that does not. For any constant number of failures k we present an approximation of $(0.878-\epsilon)$ against an adaptive adversary and of $\alpha_{GW} \approx 0.8786$ against an oblivious adversary (here α_{GW} is the approximation achieved by the random hyperplane algorithm of [Goemans-Williamson J. ACM '95]). Additionally, we present a hardness of approximation of α_{GW} against both types of adversaries, rendering our results (virtually) tight.

The non-linear nature of the fault tolerant objective makes the design and analysis of algorithms harder when compared to the classic Max-Cut. Hence, we employ approaches ranging from multi-objective optimization to LP duality and the ellipsoid algorithm to obtain our results.

2012 ACM Subject Classification Mathematics of computing → Approximation algorithms; Mathematics of computing → Graph algorithms

Keywords and phrases fault-tolerance, max-cut, approximation

Digital Object Identifier 10.4230/LIPIcs.ICALP.2021.46

Category Track A: Algorithms, Complexity and Games

Related Version *Full Version*: `https://arxiv.org/abs/2105.01138` [19]

Funding This project has received funding from the European Research Council (ERC), under the European Unions Horizon 2020 research and innovation programme under grant agreement No 755839, and from Israel Science Foundation (ISF), under grant No 1336/16.

1 Introduction

In this work, we initiate the study of *fault tolerant Max-Cut*. In the classic Max-Cut problem, we are given an undirected graph $G = (V, E)$ equipped with non-negative edge weights $w : E \to \mathbb{R}_+$. The goal is to find a cut $S \subseteq V$ that maximizes the total weight of edges that cross S. Max-Cut is one of Karp's 21 NP-complete problems [37] and has been for close to three decades a case study for the introduction of new approaches both in the theory of algorithms and the complexity theory. Perhaps the two most prominent examples of the above are: (1) the random hyperplane rounding method of Goemans and Williamson for semi-definite programs [29], which yields an approximation of $\alpha_{GW} \approx 0.8786$ for Max-Cut; and (2) the Unique Games Conjecture of Khot [38]. The former has opened an entirely new area in the field of approximation algorithms with applications to a wide range of problems, *e.g.*, Max-DiCut [26,42,44], Max-Bisection [5,53], Max-Agreement [20,56], Max-2SAT [26,42],

Max-SAT [4, 7], and Cut Norm [2], to name a few. The latter has been a dominant method for proving hardness of approximation results in the last two decades, *e.g.*, the celebrated tight hardness for Max-Cut [39, 45], and Vertex Cover [40].

Motivated by large scale real life systems, fault tolerant algorithms seek to find a solution to a given optimization problem that is resilient to failures of some parts of the input. The above can be intuitively formulated as a two step process: (1) the algorithm finds a solution to the problem at hand; and (2) an adversary removes parts of the input. The goal of the algorithm is that no matter which part of the input the adversary removes, the remaining solution after removal still retains some desired properties despite the removal. Typically, the focus of fault tolerance has been network design problems, *e.g.*, BFS [33, 48, 50–52] and spanners [15–17, 25, 41, 47, 55]. Additional related algorithmic problems for which fault tolerant algorithms were studied include, *e.g.*, single source reachability [9, 10], connected dominating set [18, 59], and facility location [23, 32, 36, 57].

In this work, we initiate the study of fault tolerant Max-Cut, where the adversary can remove vertices from the graph (all edges touching the removed vertices are also deleted). Intuitively, fault tolerant Max-Cut can be seen as a two players game, in which one player (the algorithm) chooses a cut and the other player (the adversary) removes up to a prespecified number k of vertices. The algorithm desires to maximize the total weight of edges crossing the cut, while the adversary aims to minimize the total weight of edges crossing the cut.

We study two types of adversaries. The first is an *adaptive adversary* that chooses which k vertices to fail *after* seeing the cut the algorithm produces. Specifically, the adaptive adversary knows the input, how the algorithm operates, and if the algorithm is randomized, the adaptive adversary also knows the outcome of all random coin tosses used by the algorithm. The second type of adversary is an *oblivious adversary*. Similarly to the adaptive adversary, the oblivious adversary knows the input and how the algorithm operates. However, in contrast to the adaptive adversary, the oblivious adversary does not know the outcome of the random coin tosses used by the algorithm, in case the latter is randomized (equivalently, the oblivious adversary only knows the distribution over cuts the algorithm produces). Thus, the oblivious adversary is required to choose which k vertices to fail without the knowledge of which cut was sampled. To the best of our knowledge only adaptive adversaries were studied in the fault tolerance literature.

The Challenges. The fault tolerant Max-Cut problem differs considerably from classic Max-Cut for several reasons. First, the structure of the solutions may be different. Specifically, there are instances for which an optimal solution to fault tolerant Max-Cut is not an optimal solution to classic Max-Cut, and vice versa (refer to [19] for details). Furthermore, it might be the case that the ratio between the values of the optimal solutions is large or even unbounded.

Second, the application of known techniques (which can be successfully applied to Max-Cut) to fault tolerant Max-Cut imposes some obstacles that arise from the non-linear nature of the fault tolerant objective. For example, the random hyperplane rounding method of Goemans and Williamson cannot be analyzed in a straightforward manner as one is required to lower bound the expectation of the minimum value (over all possible actions of the adversary) of the cut the random hyperplane defines, as opposed to just the expected value of the cut the random hyperplane defines. Moreover, even analyzing the simplest known algorithm for Max-Cut, *i.e.*, choosing a uniform random cut, requires great care (refer to [19] for further details). Hence, the design and analysis of algorithms for fault tolerant Max-Cut requires some new insights into the problem.

1.1 Our Contributions

Adaptive Adversary. When focusing on an adaptive adversary, our main result is an (almost) tight approximation of $0.878-\epsilon$, for any constant number k of failures and unweighted graphs. This is summarized in the following theorem (it is important to note that the constant in the theorem is slightly smaller than the Goemans-Williamson approximation factor α_{GW}).

▶ **Theorem 1.1.** *For every constant $k > 0$ and $\epsilon > 0$, there is a polynomial time $(0.878-\epsilon)$-approximation algorithm for fault tolerant Max-Cut on unweighted graphs against an adaptive adversary and k faults.*

Our algorithm is based on viewing fault tolerant Max-Cut against an adaptive adversary as a multi-objective optimization problem, where for every possible subset of k vertices the adversary can fail, one can define a different objective. The goal is to maximize the worst, *i.e.*, minimum, objective. This approach does not suffice, since all known results for the multi-objective variant of Max-Cut (formally known as Simultaneous Max-Cut [12, 13]) can handle only a constant number of objectives. In our case, even when a single failure is allowed, the number of objectives equals n. Hence, to overcome this difficulty, we incorporate local search into the above multi-objective approach to obtain the claimed result in Theorem 1.1.

Oblivious Adversary. When focusing on an oblivious adversary, our main result is a tight approximation of α_{GW} for any constant number k of failures. However, in contrast to the adaptive adversary setting, this result holds for general weighted graphs and achieves the α_{GW}-approximation guarantee exactly. This is summarized in the following theorem.

▶ **Theorem 1.2.** *For every constant $k > 0$, there is a polynomial time α_{GW}-approximation algorithm for fault tolerant Max-Cut on general weighted graphs against an oblivious adversary and k faults.*

The approach we adopt for approximating fault tolerant Max-Cut against an oblivious adversary significantly differs from the approach taken against an adaptive adversary. Surprisingly, our algorithm is based on an approximation-preserving reduction from fault tolerant Max-Cut to the classic Max-Cut problem. This reduction uses LP duality alongside the ellipsoid algorithm and is achieved by presenting a suitable approximate dual separation oracle for a configuration LP that encodes the distribution over cuts that the algorithm produces.

Hardness of Approximation. We prove that fault tolerant Max-Cut in unweighted graphs, against both adaptive and oblivious adversaries, cannot be approximated better than α_{GW} without breaking well-known hardness assumptions. It is important to note that this settles the approximability of the oblivious adversary setting (see Theorem 1.2 above), and almost settles the approximability of the adaptive adversary setting (see Theorem 1.1 above) as the constant in Theorem 1.1 is slightly smaller than α_{GW}.

▶ **Theorem 1.3.** *Assuming the Unique Games Conjecture and $NP \nsubseteq BPP$, there is no polynomial time $(\alpha_{GW}+\epsilon)$-approximation algorithm for fault tolerant Max-Cut in unweighted graphs, for any constant $\epsilon > 0$. This holds for both adaptive and oblivious adversaries.*

Simple Purely Combinatorial Algorithms. While Theorem 1.1 provides an (almost) tight result against an adaptive adversary, and Theorem 1.2 provides a tight result against an oblivious adversary, the techniques we employ yield algorithms which are polynomial but

not simple. For example, the work of [12] for approximating Simultaneous Max-Cut, an important ingredient in the design of our algorithm against an adaptive adversary, is based on SDP hierarchies and the running time is exponential in the number of objectives. In contrast, the classic Max-Cut problem admits some very simple and fast heuristics, *e.g.*, choosing a random uniform cut. Thus, we also aim to study simple and purely combinatorial algorithms for fault tolerant Max-Cut.

We prove that fault tolerant Max-Cut does yield a simple purely combinatorial local search algorithm with a provable approximation guarantee against an adaptive adversary. Unfortunately, the classic local search for Max-Cut, that in each step moves a single vertex from one side of the cut to the other side, fails in the fault tolerant setting. Nonetheless, we prove that a local search that allows for a slightly richer family of local improvement steps suffices. This is summarized in the following theorem (refer to Section 3.2 for additional details).

▶ **Theorem 1.4.** *There is a purely combinatorial polynomial time* $1/2$*-approximation algorithm for fault tolerant Max-Cut on unweighted input graphs against an adaptive adversary and a single fault.*

We further study how a *uniform random cut* performs against both types of adversaries (deferred to the full version [19]), and prove that this performance depends on the type of the adversary. Specifically, for an oblivious adversary an approximation of $1/2$ is achieved, by a uniform random cut. However, this is not the case when considering an adaptive adversary, since we prove that a uniform random cut cannot achieve an approximation better than $1/4$.

1.2 Related Work

The weighted version of Max-Cut is one of Karp's NP-complete problems [37], and the unweighted version is also known to be NP-complete [27]. In general graphs, one cannot obtain an approximation factor better than $16/17$ for the undirected version, or better than $12/13$ for the directed version, unless $P = NP$ [34, 58]. The best known approximation for Max-Cut is the celebrated random hyperplane algorithm of Goemans and Williamson that obtains an approximation factor of roughly 0.8786 by rounding the natural semi-definite programming relaxation [29]. This is the best approximation that one can achieve, assuming the Unique Games Conjecture of Khot [39] and $P \neq NP$.

The problem of fault tolerant Max-Cut against an adaptive adversary that we introduce in this paper can be viewed as a special case of Simultaneous Max-Cut, in which the input is a collection of τ weighted graphs on the same vertex set and the goal is to partition the vertices into two parts, such that the size of the cut is large in every given graph. In a straightforward manner, our problem would imply $\tau = \binom{n}{k}$, which is unacceptable since the known approximations for Simultaneous Max-Cut are for a constant number of instances only [3, 12, 13]. Nonetheless, we do use the algorithm from [12] to obtain an algorithm that achieves an approximation of 0.878 for fault tolerant Max-Cut against an adaptive adversary. The state-of-the-art for Simultaneous Max-Cut is a polynomial 0.878-approximation for any constant number of input graphs [12], which is nearly optimal since assuming the Unique Games conjecture, Simultaneous Max-Cut cannot be approximated better than $(\alpha_{GW} - \delta)$ (where $\delta \geq 10^{-5}$) [11].

One more notion of resilience is that of robust submodular maximization, see, *e.g.*, [6, 46]. Given a submodular function f and, *e.g.*, a cardinality constraint k, a set A is robust against τ failures if $A = \arg\max_{A \subseteq V, |A| \leq k} \min_{Z \subseteq A, |Z| \leq \tau} f(A - Z)$, i.e., a subset of size at most k that achieves the maximal value after at most τ elements are removed from the solution.

Note that this notion of robustness differs from fault tolerance. The reason is that the failed elements are removed from the solution, as opposed to removed from the instance. Specifically, when considering the cut function of an undirected graph (which is submodular) the removal of a vertex from S (as in robust) differs from removing the same vertex from the graph (as in fault tolerant).

Due to the importance of coping with failures, the fault tolerance of many additional fundamental problems has been extensively studied. Prime examples are replacement paths [1, 21, 22, 30, 54], BFS trees [33, 48, 50–52], spanners [15–17, 25, 41, 47, 55], connected dominating sets [18, 59], and more [8–10, 14, 23, 32, 36, 57]

Fault tolerance was also studied in the distributed setting, such as for BFS trees [28], MST [28], and spanners [25, 49].

Paper Organization. Section 2 contains all required formal definitions and preliminary lemmas used throughout the paper. Section 3 deals with the adaptive adversary, whereas Section 4 deals with the oblivious adversary. In Section 5 we show a hardness of approximation result. Missing proofs and the analysis of a random cut appear in the full version [19].

2 Preliminaries

Graph Notations. We consider only edge-weighted graphs $G = (V, E, w)$ with positive integer weights w_e assigned to the edges $e \in E$. By *unweighted* graphs we mean graphs with $w_e = 1$, for all $e \in E$. A *cut* S in a graph $G = (V, E, w)$ is a subset of vertices $S \subseteq V$. We let $\delta(S, G) = \{e \in E : |e \cap S| = 1\}$ denote the set of all *crossing edges* of S in the graph G. The *size* or *weight* of a cut S, denoted by $C_{S,G}$, is the total weight of the crossing edges: $C_{S,G} = \sum_{e \in \delta(S,G)} w_e$. When G is clear from the context, we use C_S and $\delta(S)$.

For a set $F \subseteq V$ of vertices, the *degree* $d(F)$ of F is the total weight of edges adjacent to F: $d(F) = \sum_{e \in E : e \cap F \neq \emptyset} w_e$. For a subset $F \subseteq V$ and cut $S \subseteq V$, the *crossing degree* $d_S(F)$ of F is the total weight of edges adjacent to F that cross S: $d_S(F) = \sum_{e \in \delta(S) : e \cap F \neq \emptyset} w_e$. We use $d(v)$ and $d_S(v)$, if $F = \{v\}$. We also let $n = |V|$, $m = |E|$, and $\Delta = \max_{v \in V} d(v)$. Finally, we let 2^V and $\binom{V}{k}$ denote the collection of all and all size-k subsets of V, respectively.

The Adaptive Adversary. We define the *k-FT value of a cut against an adaptive adversary* to be the minimal size of the cut, subsequent to a failure of any k vertices. Formally, for a cut S in a graph $G = (V, E, w)$ and a constant $k > 0$, the k-FT value of S is defined as $\varphi(S, k, G) = \min_{F \in \binom{V}{k}} C_{S-F, G-F}$.

▶ **Definition 2.1** (*k-AFTcut*). *Given an edge-weighted graph $G = (V, E, w)$ and a number $k \in \mathbb{N}$, a cut S is a* k-adaptive fault tolerant cut*, or k-AFTcut for short, if $\varphi(S, k, G) = \max_{S' \subseteq V} \{\varphi(S', k, G)\}$.*

We usually omit G and/or k from $\varphi(S, k, G)$ when G is clear from the context and $k = 1$. The *Max-Cut* problem, i.e., that of finding a cut with the largest size, corresponds to the special case $k = 0$, but will always be denoted by Max-Cut.

The Oblivious Adversary. We represent a randomized algorithm that finds a cut in a graph $G = (V, E, w)$ by a probability distribution $\mathcal{D}$ over all possible cuts 2^V. For a distribution $\mathcal{D}$ over cuts, we define the *k-FT value of $\mathcal{D}$* to be the minimal expected size of the cut, subsequent to the failure of any k vertices. Formally, for a graph $G = (V, E, w)$, a distribution $\mathcal{D}$ over cuts and a constant $k > 0$, we define the k-FT value of $\mathcal{D}$, denoted by $\mu(\mathcal{D}, k, G)$, as $\mu(\mathcal{D}, k, G) = \min_{F \in \binom{V}{k}} \mathop{\mathbb{E}}_{S \sim \mathcal{D}} [C_{S-F, G-F}]$.

▶ **Definition 2.2** (*k-OFTcut*). *Given an edge-weighted graph $G = (V, E, w)$ and a number $k \in \mathbb{N}$, a distribution $\mathcal{D}$ over all cuts 2^V is a k-oblivious fault tolerant cut, or k-OFTcut for short, if $\mu(\mathcal{D}, k, G) = \max_{\mathcal{D}'} \{\mu(\mathcal{D}', k, G)\}$.*

Note that here we assume the adversary chooses the set F of faults deterministically; it easily follows from the linearity of expectation that the adversary always has a deterministic best choice – a subset that has the largest expected crossing degree.

Greedy steps and stable cuts. We assume here that we are given an *unweighted* graph $G = (V, E)$. A key observation in our algorithms against an adaptive adversary is that any solution can be transformed into another one where each vertex contributes *many* of its edges to the cut. If a vertex contributes too little, we can just move it to the opposite side of the cut: while this could increase the crossing degree of some vertices (negative contribution to the FT value), it increases the cut size by more, giving a positive net contribution to the FT value. We prove this formally in Lemma 2.4, after some formal definitions.

For every $v \in V$ and $S \subseteq V$, let $S \oplus v$ denote the cut obtained from S by switching v to its opposite side, that is, $S \oplus v = S - v$, if $v \in S$, and $S \oplus v = S \cup \{v\}$, otherwise. Given a subset $S \subseteq V$, a constant $k \in \mathbb{N}$, and a vertex $v \in V$, we say that replacing S with $S \oplus v$, i.e., moving v to its opposite side w.r.t. S, is a *k-greedy step* if $d_S(v) \leq (d(v) - k)/2$. A cut S is *k-stable* if it has no k-greedy step, that is, for every $v \in V$, it holds that $d_S(v) > (d(v) - k)/2$. For $k = 1$, we use *stable* instead of 1-stable.

▶ **Observation 2.3.** *For every cut S and a vertex v, it holds that $C_{S\oplus v} - C_S = d_{S\oplus v}(v) - d_S(v)$.*

▶ **Lemma 2.4.** *Let $v \in V$ be a vertex, $S \subseteq V$ be a cut, and $k > 0$ be an integer, such that $d_S(v) \leq (d(v) - k)/2$; then $C_{S\oplus v} \geq C_S + k$, and $\varphi(S \oplus v, k) \geq \varphi(S, k)$.*

Proof. Assume, without loss of generality, that $v \in S$ (otherwise, we swap S and $V - S$). Observation 2.3 implies that $C_{S\oplus v} \geq C_S + k$, since $d_S(v) + k \leq d(v) - d_S(v) = d_{S\oplus v}(v)$.

For the second claim, we show that for every $F \in \binom{V}{k}$, $C_{S-F,G-F} \leq C_{S\oplus v-F,G-F}$. Assume that $v \notin F$, as otherwise $S - F = S \oplus v - F$, and the claim holds trivially. Recall that $C_{S\oplus v} \geq C_S + k$. In addition, $d_{S\oplus v}(F) \leq d_S(F) + k$, since for every $u \in F$, at most one crossing edge is added to the cut (the edge $\{u, v\}$). Putting those together, we have that: $C_{S-F,G-F} = C_S - d_S(F) \leq C_{S\oplus v} - d_{S\oplus v}(F) = C_{S\oplus v-F,G-F}$. Since this holds for every F, we have that $\varphi(S \oplus v, k) \geq \varphi(S, k)$. ◀

By repeatedly applying a k-greedy step to a cut, we keep increasing the cut value, while not decreasing the k-FT value; thus, after at most m greedy steps, we have a k-stable cut with a k-FT value at least as good as the original one. We let StabilizeCut(G,S,k) denote this procedure, which takes as input a graph G, a cut S in G, and a number k, then starting with S, repeatedly applies a (arbitrary) k-greedy step, while there is one, and returns the obtained k-stable cut. The following corollary follows from the reasoning above (the second claim follows by applying StabilizeCut to an optimal *k-AFTcut*).

▶ **Corollary 2.5.** *Let S be a cut in graph $G = (V, E)$, and let k be a positive integer. Let $S' = \textsc{StabilizeCut}(G, S, k)$. It holds that S' is k-stable, $C_{S'} \geq C_S$ and $\varphi(S', k) \geq \varphi(S, k)$. In particular, every unweighted graph $G = (V, E)$ has a k-stable optimal k-AFTcut.*

3 Fault Tolerance Against an Adaptive Adversary

3.1 A 0.878-Approximation for Multiple Faults

In this section, we give a $(0.878-\epsilon)$-approximation algorithm for k-*AFTcut* on unweighted graphs, for constants $k, \epsilon > 0$. A core tool that we use in our algorithm is an algorithm for the *Simultaneous Max-Cut* problem, where given several graphs defined over the same vertex set, the goal is to find a cut that is large for all graphs simultaneously. A 0.878-approximation algorithm for this problem with a *constant* number of graphs has been given in [12]. The algorithm is based on semidefinite programming techniques.

The main idea behind our algorithm is to separate a constant number of "heavy" (high-degree) vertices for which the following holds; given a cut which is large subsequent to any failure of k *heavy* vertices, the cut is large even if light (non-heavy) vertices fail as well. For such a heavy set, a good approximation for Simultaneous Max-Cut on the instances obtained by removing each possibility of k heavy vertices from G, should be a good approximation for k-*AFTcut* on G. We give a greedy algorithm that selects the set of heavy vertices. We then consider two cases. We show that if the heavy vertices do not cover most of the edges in the graph (the "non-shallow" case), then an approximate solution for Simultaneous Max-Cut with respect to the heavy set gives an approximate solution for k-*AFTcut*. Otherwise (the "shallow" case), we identify a set of "super-heavy" vertices, which is shown to fail in any near-optimal solution. Therefore, finding a near-optimal solution for the original graph reduces to finding a near-optimal solution on the graph remaining by removing the "super-heavy" vertices. We show that it can be solved via brutforce, or by finding a good solution to Max-Cut (e.g., obtained via [29]). We prove the following theorem.

▶ **Theorem 1.1.** *For every constant $k > 0$ and $\epsilon > 0$, there is a polynomial time $(0.878-\epsilon)$-approximation algorithm for fault tolerant Max-Cut on unweighted graphs against an adaptive adversary and k faults.*

Before proceeding to the algorithm, we introduce the Simultaneous Max-Cut framework.

▶ **Definition 3.1** (Simultaneous Max-Cut). *Let V be a vertex set. We are given k edge-weighted graphs, $G_i = (V, E_i)$, $i = 1, \ldots, k$, on the vertex set V, where the weights are normalized, so that $\sum_{e \in E_i} w_e = 1$, for each i. In the (Pareto)* Simultaneous Max-Cut *problem, given the graphs G_i together with thresholds $c_i \in [0, 1]$, the goal is to find a cut $S^* \subseteq V$ such that $C_{S^*, G_i} \geq c_i$, for every i. We say that an algorithm is an α-approximation algorithm for the problem if for every input G_i, c_i, $i = 1, \ldots, k$, where there exists a cut S^* such that $C_{S^*, G_i} \geq c_i$ for every i, the algorithm returns a cut $\widetilde{S}$ such that $C_{\widetilde{S}, G_i} \geq \alpha c_i$, for every i.*

▶ **Theorem 3.2.** *[12] For every constant $k \geq 1$ and parameter $n \geq 1$, there is a polynomial-in-n algorithm that computes an α_{SMC}-approximate solution to any Simultaneous Max-Cut instance with k weighted graphs on a vertex set of size n, in which all non-zero edge-weights are lower-bounded by $\exp(n^{-c})$, for constants k and c, and $\alpha_{SMC} = 0.878$.*

We apply the Simultaneous Max-Cut framework for unweighted graphs G_i. We let SimultaneousMC denote the algorithm that gets as input a constant number of unweighted graphs G_i, $i = 1, \ldots, k$, and returns a cut $\widetilde{S}$ with the following property: for every cut S^* and number c such that $C_{S^*, G_i} \geq c$, for all i, it holds that $C_{\widetilde{S}, G_i} \geq \alpha_{SMC} \cdot c$, for all i. This can be achieved by combining the algorithm given in Theorem 3.2 (by appropriately scaling the edge-weights and the thresholds) with a binary search on c.

Algorithm 1 $(\alpha_{SMC} - \epsilon)$-approximation for k-*AFTcut*.

Input: $G = (V, E)$, k, ϵ
Output: $(\alpha_{SMC} - \epsilon)$-approximation for k-*AFTcut*
$H \leftarrow$ HeavyVertices(G, k, ϵ)
$\widetilde{S} \leftarrow$ SimultaneousMC($\{G_{\neg F} : F \in \binom{H}{k}\}$)
if $(H, \widetilde{S})$ *is shallow* **then**
  return ShallowFTCut(G, H, $\widetilde{S}$, k, ϵ)
else
  return $\widetilde{S}$

In addition to the Simultaneous Max-Cut algorithm, we use the α_{GW}-approximation for Max-Cut due to Goemans and Williamson [29], for $\alpha_{GW} \approx 0.8786$. We use Goemans-Williamson (with input G) to denote this algorithm. Note that the actual value of the approximation factor α_{SMC} is slightly larger than 0.878 but is less than α_{GW}.

The Main Algorithm. The inputs to the algorithm (see the pseudocode in Algorithm 1) are an unweighted graph G, and parameters k (number of faults) and ϵ (precision). First, it computes the set H of heavy vertices via the subroutine HeavyVertices, then applies SimultaneousMC on a collection $\{G_{\neg F} : F \in \binom{H}{k}\}$ of subgraphs containing one subgraph for every failure of k heavy vertices. The following notation is used: for a subset $F \subseteq V$ of vertices, we let $G_{\neg F} = (V, E_{\neg F})$, where $E_{\neg F} = \{e \in E : e \cap F = \emptyset\}$. Note that in $G_{\neg F}$, we do not remove the vertices of F from the graph, as opposed to $G - F$, but only the edges adjacent to F.

The pair $(H, \widetilde{S})$ is *shallow* if all vertices in $V - H$ have degree at most $3k$, and there are k vertices in H whose removal reduces the weight of $\widetilde{S}$ below $3k^2/\epsilon$. To state this formally, let us introduce a notation that will be useful later too. For a cut $S \subseteq V$, we use $C_{S-k\times H}$ to denote the smallest size of the cut after the failure of any k vertices from H, i.e., $C_{S-k\times H} = \min\{C_{S-F,G-F} : F \in \binom{H}{k}\}$. Thus, $(H, \widetilde{S})$ is shallow if we have $\max_{v\in V-H} d(v) \leq 3k$ and $C_{\widetilde{S}-k\times H} < 3k^2/\epsilon$. If $(H, \widetilde{S})$ is not shallow, the algorithm simply returns $\widetilde{S}$. Otherwise, we recompute the cut via ShallowFTCut, using alternative methods.

The proof of Theorem 1.1 is split into two parts, addressing shallow and non-shallow cases separately. The running time is dominated by Simultaneous Max-Cut. Before giving further details, let us mention how the proof follows from the main lemmas addressing those cases.

Proof of Theorem 1.1. Let G be a graph and let S^* be an optimal k-*AFTcut* on G. Let $\widetilde{S}$ be the output of Algorithm 1 on G, k, ϵ. We show that $\varphi(\widetilde{S}, k) \geq (\alpha_{SMC} - \epsilon) \cdot \varphi(S^*, k)$. Lemma 3.4 provides this for the non-shallow case, while Lemma 3.5 provides it in the shallow case. The algorithm is indeed polynomial, since the sub-routines are such, and the input to SimultaneousMC consists of $\binom{|H|}{k} = O(k/\epsilon)^k = O(1)$ subsets, where $|H| = O(k^2/\epsilon)$ is proven in Lemma 3.3. ◀

The selection of heavy vertices (Algorithm 2) is done by a simple greedy procedure, where we sequentially select vertices in the heavy set H in a non-increasing order by degree. The selection stops either when the remaining vertices $(V - H)$ have a small degree (at most $3k$) or when H has sufficiently many incident edges (used in Lemma 3.4). By Corollary 2.5, any cut can be transformed into one with a similar k-FT value, where every vertex v has crossing

Algorithm 2 HeavyVertices.

```
1 Input: G = (V, E), k, ε
2 Output: H ⊆ V, the set of heavy vertices
3 Let v_1, ..., v_n be an ordering of vertices by non-increasing degree
4 σ ← 0, i ← 1, H ← {v_1, ..., v_k}
5 while d(v_{k+i}) > (ε · α_SMC/k) · σ and d(v_{k+i}) > 3k do
6     σ ← σ + (d(v_{k+i}) − 3k)/4
7     H ← H ∪ {v_{k+i}}
8     i ← i + 1
9 return H
```

degree at least $(d(v)-k)/2$, and at least $(d(v)-3k)/2$, after k failures. Thus, heavy vertices are guaranteed to contribute σ in the "stable version" of every cut. The degree constraint ensures that we do not select vertices that are unnecessary, according to this logic, which helps us keep the size of H bounded.

▶ **Lemma 3.3.** *Algorithm 2 terminates within $t = 4(3k^2+k)/(\epsilon \cdot \alpha_{SMC})$ iterations. In particular, $|H| \leq t + k$.*

Proof. If $d(v_{k+t}) \leq 3k$, then by the condition in Line 5, the algorithm terminates before the t-th iteration; therefore, assume $d(v_{k+t}) > 3k$. For every $i \leq t$, after the i-th iteration, it holds that $\sigma_i = \sum_{j=1}^{i} (d(v_{k+j}) - 3k)/4$; thus, after t iterations,

$$\sigma_t = \sum_{j=1}^{t} \frac{d(v_{k+j})-3k}{4} \geq t \cdot \frac{d(v_{k+t})-3k}{4} = \frac{3k^2+k}{\epsilon \cdot \alpha_{SMC}} \cdot (d(v_{k+t})-3k)$$
$$= \frac{k}{\epsilon \cdot \alpha_{SMC}} \cdot d(v_{k+t}) + \frac{3k^2}{\epsilon \cdot \alpha_{SMC}} \cdot d(v_{k+t}) - \frac{3k^2(3k+1)}{\epsilon \cdot \alpha_{SMC}} \geq \frac{k}{\epsilon \cdot \alpha_{SMC}} d(v_{k+t}) \ ,$$

where in the first inequality, we use the fact that the vertices are processed in a non-increasing order of degrees, and in the last inequality, we use the assumption that $d(v_{k+t}) \geq 3k+1$. It follows that $d(v_{k+t}) \leq (\epsilon \cdot \alpha_{SMC}/k) \cdot \sigma_t$, and using $d(v_{k+t+1}) \leq d(v_{k+t})$, we get that the algorithm terminates within the first t iterations, by the condition in Line 5. ◀

Non-Shallow Case. In this case, we have either $\max_{v \in V-H} d(v) > 3k$ or $C_{\widetilde{S}-k\times H} \geq 3k^2/\epsilon$. If the former holds, we see from Algorithm 2 that for all light vertices $v \notin H$, $d(v) \leq (\epsilon \cdot \alpha_{SMC}/k) \cdot \sigma$. As it was observed earlier, Corollary 2.5 implies that every cut can be turned into another one with no smaller FT value, such that H contributes σ edges to the cut, even after k failures. In such a cut, a failure of k light vertices would affect only an ϵ fraction of the cut. A similar reasoning applies in the other case, when $\max_{v \in V-H} d(v) \leq 3k$: here we have $C_{\widetilde{S}-k\times H} \geq 3k^2/\epsilon$, which leads to similar conclusions as above. Putting these together with the near-optimality of $\widetilde{S}$, we obtain the main lemma of the non-shallow case (see [19]).

▶ **Lemma 3.4.** *If $(H, \widetilde{S})$ is not shallow, then it holds that $\varphi(\widetilde{S}, k) \geq (\alpha_{SMC} - \epsilon)\varphi(S^*_{ft}, k)$, for an optimal k-AFTcut S^*_{ft}.*

Shallow Case. Recall that in this case, we have that $\max_{v \in V-H} d(v) \leq 3k$, and $C_{\widetilde{S}-k\times H} < 3k^2/\epsilon$. The subroutine ShallowFTCut (Algorithm 3) constructs another cut, $\hat{S}$, which we prove in Lemma 3.5 is a $(\alpha_{SMC} - \epsilon)$-approximation for k-$AFTcut$. A key role in this case is played by the following (possibly empty) set $\hat{H}$ of *super-heavy* vertices, where $\hat{H} = \left\{v \in V : (d(v) - 3k)/2 > C_{\widetilde{S}-k\times H}/\alpha_{SMC}\right\}$. We show that $\hat{H}$ is contained in *every worst-case failure set* of a cut where there is no k-greedy step of a vertex in $\hat{H}$. We let $G_R = G - \hat{H}$, and let m_R be the number of edges in G_R. Note that the degree of every $v \notin \hat{H}$ is bounded by some constant ℓ.

Algorithm 3 ShallowFTCut.

```
Input: G = (V, E), H, S̃, k, ε
Output: Cut Ŝ ⊆ V
if m_R < 2kℓ/(α_SMC ε) then
    for every S' ⊆ V_R (V_R is of constant size) do
        Compute φ(S', k − |Ĥ|)
    Ŝ ← arg max_{S'⊆V_R} φ(S', k − |Ĥ|)
else
    Ŝ ← Goemans-Williamson(G_R)
while ∃v ∈ Ĥ such that d_Ŝ(v) ≤ (d(v) − k)/2 do
    Ŝ ← Ŝ ⊕ v
return Ŝ
```

First, we compute a near-optimal $(k-|\hat{H}|)$-$AFTcut$, $\hat{S}$, on G_R, then add $\hat{H}$ and repeatedly apply k-greedy steps to the vertices in $\hat{H}$, while there are any. To compute a cut in G_R, we distinguish between two cases. If there are many edges, i.e., $m_R \geq ck\ell/\epsilon$, for a constant c, then we let $\hat{S}$ =Goemans-Williamson(G_R). This suffices, since the cut is of size at least $m_R/2 = \Omega(k\ell/\epsilon)$, and the degrees are bounded by ℓ , so failures do not affect the cut size significantly, and we get an $(\alpha_{GW} - \epsilon)$-approximation. If, on the other hand, there are few edges, i.e., $m_R < ck\ell/\epsilon$, then we can compute an optimal $(k - |\hat{H}|)$-$AFTcut$ in G_R via brute-force. Also using that $\hat{H}$ belongs to every worst-cast failure set of a cut not having a k-greedy step of a vertex in $\hat{H}$ (including the one we constructed, and the optimal ones that exist by Corollary 2.5), we get the following main lemma (proved in the full version [19]).

▶ **Lemma 3.5.** *If $(H, \widetilde{S})$ is shallow, then it holds that $\varphi(\hat{S}, k) \geq (\alpha_{SMC} - \epsilon) \cdot \varphi(S^*_{ft}, k)$, for an optimal k-$AFTcut$ S^*_{ft}.*

3.2 A Combinatorial 1/2-Approximation for a Single Fault

In the case of a single fault, we have the following result, that is, a simple and efficient 1/2-approximation for the case of a single fault. Moreover, we show that an FT value of $(m - \Delta)/2$ can be achieved, for $\Delta \geq 3$, while $m - \Delta$ is an (easy) upper bound.

▶ **Theorem 1.4.** *There is a purely combinatorial polynomial time 1/2-approximation algorithm for fault tolerant Max-Cut on unweighted input graphs against an adaptive adversary and a single fault.*

In the discussion below, we call a vertex v *critical* for a cut S if $C_{S-v,G-v} = \varphi(S)$. It is well-known (and easy to show) that every stable cut is a 1/2-approximate Max-Cut. This even holds for $AFTcut$, with $\Delta = 2$ (see [19]). However, in general, while we know

Algorithm 4 Combinatorial 1/2-approximation for *AFTcut*.

Input: $G = (V, E)$
if $\Delta \leq 2$ **then return** $\textsc{StabilizeCut}(G, \emptyset, 1)$
$\widetilde{S} \leftarrow \emptyset$
while $\varphi(\widetilde{S}) < (m - \Delta)/2$ **do**
  if $\exists v, \varphi(\widetilde{S} \oplus v) \geq (m - \Delta)/2$ **then**
    $\widetilde{S} \leftarrow \widetilde{S} \oplus v$ `// type-0 step`
  else if $\exists v, d_{\widetilde{S}}(v) < d(v)/2$ **then**
    $\widetilde{S} \leftarrow \widetilde{S} \oplus v$ `// type-1 step`
  else if $\exists v, w, \left(d_{\widetilde{S}}(v) = \frac{d(v)}{2}$ **and** $\varphi(\widetilde{S} \oplus v) \geq \varphi(\widetilde{S})$ **and** $d_{\widetilde{S} \oplus v}(w) < \frac{d(w)}{2}\right)$ **then**
    $\widetilde{S} \leftarrow \widetilde{S} \oplus v$ `// build-up for another type-1 step`
  else
    $v \leftarrow$ a vertex such that $d_{\widetilde{S}}(v) = d(v)/2$ and $\varphi(\widetilde{S} \oplus v) > \varphi(\widetilde{S})$
    $\widetilde{S} \leftarrow \widetilde{S} \oplus v$ `// type-2 step`
return $\widetilde{S}$

that greedy steps (moving a vertex v with $d(v) < d_S(v)/2$) never decrease the FT value (Lemma 2.4), a stable cut can be a poor approximation for *AFTcut*. Consider, for example, a graph that consists of t triangles with a single common vertex u. Note that $d(u) = \Delta = 2t$, $d(v) = 2$, for every $v \neq u$, and $m = 3t$. The cut $S' = \{u\}$ is a stable cut, with $\varphi(S') = 0$. In order to transform S' into a 1/2-approximation, we have to decrease the crossing degree of the critical vertex u without decreasing the size of the cut. This can be done by moving a neighbor v of u from the opposite side of the cut, since $d_{S'}(v) = d(v)/2$.

In general, moving such vertex v (which we call a *neutral* move below) does not change the size of the cut, and decreases the crossing degree of u. Nevertheless, it does not always imply that the FT value increases, as there can be an additional critical vertex u' in S that is not affected, or that moving v creates a new critical vertex u'' with the same crossing degree as u.

Our algorithm (see Algorithm 4) is based on some key structural properties of stable cuts that we prove. Essentially, we show that any given cut S with FT value less than $(m - \Delta)/2$ either admits a *greedy step*, or a *neutral move followed by a greedy step*, or a *neutral move that increases the FT value*. Our algorithm is then a repeated application of such steps until the cut has the desired FT value; thus, it can be seen as a local search over two-move combinations, for maximizing the *sum* of the cut size and FT value.

Our key technical observation is that in a balanced cut S with an FT value less than $(m - \Delta)/2$, the critical vertex is unique. Moreover, letting $x_S(v) = d_S(v) - d(v)/2$ denote the *excess* contribution of a vertex v to the cut, it holds for the critical vertex u that $x_S(u) > \sum_{v \neq u} x_S(v) + \Delta - d(u)$, as proved in the following lemma.

▶ **Lemma 3.6.** *Let S be a stable cut in a graph $G = (V, E)$ such that $\varphi(S) < (m - \Delta)/2$. Then S has a unique critical vertex vertex u, and u satisfies*

$$d_S(u) > \sum_{v \neq u} x_S(v) + \Delta - d(u)/2 \ . \tag{1}$$

Moreover, u has a neighbor w in its opposite side of the cut, which satisfies $x_S(w) = 0$.

Proof. First, we show that S has a unique critical vertex. Since for every vertex v, $d_S(v) = d(v)/2 + x_S(v)$ and $\sum_{v\in V} d(v)/2 = m$, we get that

$$C_S = \frac{1}{2}\sum_{v\in V} d_S(v) = \frac{1}{2}\sum_{v\in V}\left(\frac{d(v)}{2} + x_S(v)\right) = \frac{m}{2} + \sum_{v\in V}\frac{x_S(v)}{2} \ . \tag{2}$$

Let u be a critical vertex, and assume, without loss of generality, that $u \in S$ (otherwise we swap S and $V-S$). On one hand, we have $\varphi(S) = C_S - d_S(u) = m/2 + \sum_{v\in V} x_S(v)/2 - d_S(u)$, and on the other hand, we have $\varphi(S) < (m-\Delta)/2$, which together imply:

$$m/2 + \sum_{v\in V} x_S(v)/2 - d_S(u) < (m-\Delta)/2 \ .$$

After a rearrangement, the latter implies (1). Using $d_S(u) = d(u)/2 + x_S(u)$ in (1) and simplifying, we get $x_S(u) > \sum_{v\neq u} x_S(v) + \Delta - d(u) \geq \sum_{v\neq u} x_S(v)$. Since u is an arbitrary critical vertex, this implies that u is the only critical vertex of S.

Next, let us show that there is a neighbor $w \in V - S$ of u (recall that $u \in S$) with $x_S(w) = 0$. Assume to the contrary that for every $v \notin S$ such that $\{u, v\} \in E$, it holds that $x_S(v) \geq 1/2$ (recall that S is stable, and hence $x_S(v)$ is a non-negative integer multiple of $1/2$). Using (2), this implies:

$$\begin{aligned} C_S = \frac{m}{2} + \sum_{v\in V}\frac{x_S(v)}{2} &\geq \frac{m}{2} + \frac{1}{2}\left(x_S(u) + \sum_{v:\{u,v\}\in\delta(S)} x_S(v)\right) \\ &\geq \frac{m}{2} + \frac{1}{2}\left(x_S(u) + \frac{d_S(u)}{2}\right) \geq \frac{m}{2} + x_S(u) \ , \end{aligned}$$

where we use $d_S(u) = |\{v : \{u, v\} \in \delta(S)\}|$ in the second inequality, and $d_S(u) = d(u)/2 + x_S(u) \geq 2x_S(u)$, in the third one. Since u is the critical vertex of S, this gives that

$$\varphi(S) = C_S - d_S(u) \geq \frac{m}{2} + x_S(u) - d_S(u) = \frac{m}{2} - \frac{d(u)}{2} \geq (m-\Delta)/2 \ ,$$

in contradiction to $\varphi(S) < (m-\Delta)/2$. This completes the proof. ◀

Note that in a stable cut S, $x_S(v)$ is a non-negative multiple of $1/2$, for all v. In most typical cases (e.g., when $d(u) < \Delta$, or when there are not too few nodes v with $x_S(v) > 0$), the inequality from the lemma quickly gives us the properties we claimed. However, covering all cases turns out to be quite tedious. The complete analysis can be found in the full version [19].

4 Fault Tolerance Against an Oblivious Adversary

We give an algorithm that approximates the fault tolerant Max-Cut against the oblivious adversary with (constant) k faults within an α_{GW}-approximation factor. The main idea is to frame the problem as a linear program (LP) with an exponential number of variables, then reduce the number of variables using a solution of its dual (with an exponential number of constraints but a polynomial number of variables). The dual is approximately solved by the ellipsoid algorithm together with an approximate separation oracle that is given by a Max-Cut algorithm. A similar approach has been used, e.g. in [35], for an unrelated problem.

▶ **Theorem 1.2.** *For every constant $k > 0$, there is a polynomial time α_{GW}-approximation algorithm for fault tolerant Max-Cut on general weighted graphs against an oblivious adversary and k faults.*

For simplicity, we present the algorithm for a single fault. In the full version [19], we show how to extend it to any constant number k of faults. The $OFTcut$ problem can be formulated as the following LP, $(Primal_1)$, with an exponential number of variables.

$$\max \quad \sum_{S\subseteq V} P_S \cdot \sum_{e\in\delta(S)} w_e - Z \qquad (Primal_1)$$

$$\text{s.t.} \quad \sum_{S\subseteq V} P_S \cdot \sum_{v:\{u,v\}\in\delta(S)} w_{\{u,v\}} \le Z \qquad \forall u \in V \qquad (3)$$

$$\sum_{S\subseteq V} P_S \le 1 \qquad (4)$$

$$0 \le P_S \qquad \forall S \subseteq V \qquad (5)$$

The variable P_S represents the probability assigned to the cut $S \subseteq V$. The variable Z represents the expected weight that the adversary removes from the graph. Constraints (4-5) make P_S a probability distribution. In (3), for each vertex u, we bound by Z the expected weight that is removed from the cut when u fails. To see that the left hand side is indeed the expected removed weight, note that it equals $\sum_{S\subseteq V} P_S \cdot d_S(u)$.

Consider the dual problem of the LP above, $(Dual_1)$:

$$\min \quad Y \qquad (Dual_1)$$

$$\text{s.t.} \quad \sum_{\{u,v\}\in\delta(S)} w_{\{u,v\}} - \sum_{u\in V} X_u \sum_{v:\{u,v\}\in\delta(S)} w_{\{u,v\}} \le Y \qquad \forall S \subseteq V \qquad (6)$$

$$\sum_{u\in V} X_u \le 1 \qquad (7)$$

$$0 \le X_u \qquad \forall u \in V \qquad (8)$$

The dual LP captures the following problem: The adversary picks a distribution over the vertices, and the algorithm picks a cut (depending on the choice of the adversary). The goal of the adversary is to choose its distribution (without knowing the cut choice of the algorithm) so as to minimize the expected cut size after a random failure from its distribution.

The dual LP $(Dual_1)$ has an exponential number of constraints but only $|V|+1$ variables. Such LPs can be solved efficiently via the *ellipsoid method* [31], given an efficient *separation oracle*. The latter is an algorithm that given an assignment of values to the variables of the LP, reports a violated constraint if the assignment is infeasible, or otherwise reports that it is feasible. For the particular case of $(Dual_1)$, the ellipsoid algorithm can be viewed as a binary search over the values of Y, such that in each stage (fixed Y), a black-box procedure does a polynomial number of queries to a given separation oracle, and either reports the first solution $\{X_u\}_{u\in V}$ it finds such that $\{X_u\}_{u\in V}, Y$ is feasible according to the oracle, or reports that there is no such solution.

Let us see what a separation oracle looks like in our case. For given values $\{X_u\}_{u\in V}, Y$, let $G' = (V', E', w')$ be the graph with weights $w'_{\{u,v\}} = (1-X_u-X_v)w_{\{u,v\}}$. With this notation, constraint (6) becomes $C_{S,G'} \le Y$. In order to see if a given assignment of variables is feasible, it thus suffices to find a maximum weight cut S^* in G' and test if $C_{S^*,G'} \le Y$. Since Max-Cut

is hard to solve exactly, we use an *approximate separation oracle*. Given $\{X_u\}_{u\in V}, Y$, it immediately returns the constraint (7), if it is violated, and otherwise computes a cut S_{ALG} in G' using a derandomized variant of the Goemans-Williamson algorithm [43], which we denote by Derandomized-Goemans-Williamson. Given an assignment $\{X_u\}_{u\in V}, Y$ to the variables in the LP, the oracle either outputs `feasible` or a violated constraint. If the size of the cut is larger than Y, it returns the violated constraint (6) corresponding to S_{ALG}, otherwise it reports that the solution is feasible (see Algorithm 5).

Algorithm 5 Approximate separation oracle.

Input: $\{X_u\}_{u\in V}, Y, G$
if $\sum_{u\in V} X_u > 1$ **then**
  return violated constraint $\sum_{u\in V} X_u \leq 1$
$S_{ALG} \leftarrow$ Derandomized-Goemans-Williamson(G')
if $C_{S_{ALG},G'} > Y$ **then**
  return violated constraint for subset S_{ALG}
else
  return `feasible`

We show (Lemma 4.1) that if $\{X_u\}_{u\in V}, Y$ is feasible, then the oracle reports that it is feasible, and otherwise, it either reports a violated constraint, or incorrectly reports that it is feasible, in which case, however, $\{X_u\}_{u\in V}, Y/\alpha_{GW}$ is feasible.

▶ **Lemma 4.1.** *Given an assignment $\{X_u\}_{u\in V}, Y$ to the variables in $(Dual_1)$ as input to the separation oracle in Algorithm 5, it holds that:*

1. *if the assignment is feasible, then the oracle returns* `feasible`*,*
2. *if the assignment is infeasible, then either the oracle outputs a violated constraint, or reports* `feasible`*, in which case $\{X_u\}_{u\in V}, Y/\alpha_{GW}$ is feasible.*

Proof. Let $\{X_u\}_{u\in V}, Y$ be an assignment to the variables of $(Dual_1)$. If it is feasible, then it holds that $\sum_{u\in V} X_u \leq 1$, and in addition every $S \subseteq V$ satisfies $C_{S,G'} \leq Y$, therefore the oracle returns `feasible`.

If the assignment is infeasible, there are two cases. If $\sum_{u\in V} X_u > 1$, the oracle returns this violated constraint. Otherwise, there is a subset $S' \subseteq V$ such that $C_{S',G'} > Y$. Let S^* be an optimal solution for Max-Cut on G', and note that $C_{S^*,G'} > Y$. If $\alpha_{GW} \cdot C_{S^*,G'} > Y$, then we also have that $C_{S_{ALG},G'} > Y$ (since S_{ALG} is an α_{GW}-approximate Max-Cut), and the oracle returns the violated constraint for S_{ALG}. Otherwise, $C_{S^*,G'} \leq Y/\alpha_{GW}$. Since S^* is an optimal solution for Max-Cut on G', it follows that for every $S \subseteq V$, it holds that $C_{S,G'} \leq Y/\alpha_{GW}$, i.e., the solution $\{X_u\}_{u\in V}, Y/\alpha_{GW}$ is feasible. ◀

It is not hard to see that the application of the ellipsoid algorithm on $(Dual_1)$ takes a polynomial time (i.e., at most as much time as it would take with an exact separation oracle), since our approximate oracle is (possibly) incorrect only on *the last* call from the ellipsoid algorithm (for a given Y), when it incorrectly reports a solution as feasible.

The output of the ellipsoid algorithm/binary search is an assignment $\{X_u\}_{u\in V}, Y$ to the variables of $(Dual_1)$ such that $\{X_u\}_{u\in V}, Y$ is feasible according to the oracle, while $Y - \epsilon$ is infeasible with every assignment to the X variables, where ϵ is the precision of the binary search. As observed above, we have that $\{X_u\}_{u\in V}, Y/\alpha_{GW}$ is feasible, and it follows that if

Y^* is the optimal value of $(Dual_1)$, then $Y - \epsilon \leq Y^* \leq Y/\alpha_{GW}$. Since the ellipsoid algorithm queries the oracle a polynomial number of times, there is a set $\mathcal{H} \subseteq 2^V$ of a polynomial number of cuts S, for which constraint (6) is queried. Consider a modified variant of $(Dual_1)$, called $(Dual_2)$, where only constraints of cuts in $\mathcal{H}$ are present. Let Y_2^* be the optimal value of $(Dual_2)$. Note that $Y_2^* \leq Y^*$. Note also that the ellipsoid algorithm returns exactly the same solution $\{X_u\}_{u \in V}, Y$, when executed on $(Dual_1)$ and $(Dual_2)$ (since our algorithm is deterministic, and only constraints in $\mathcal{H}$ are queried); hence, we have $Y - \epsilon \leq Y_2^*$. Finally, let us consider the primal LP corresponding to $(Dual_2)$ it is obtained from $(Primal_1)$ by removing variables P_S with $S \notin \mathcal{H}$ (i.e., setting $P_S = 0$).

The new primal has polynomially many constraints and variables, so can be solved in polynomial time. From the arguments above, we have that its optimal value Y_2^* satisfies $Y - \epsilon \leq Y_2^* \leq Y^* \leq Y/\alpha_{GW}$. Recalling that Y^* is the optimal value for the original LP, we see that Y_2^* is a α_{GW}-approximation (with any polynomial precision ϵ).

5 Hardness of Approximation

In this section we show that assuming the Unique Games Conjecture, one cannot approximate *AFTcut* and *OFTcut* within a factor greater than α_{GW}. Formally, we prove the following:

▶ **Theorem 1.3.** *Assuming the Unique Games Conjecture and NP $\not\subseteq$ BPP, there is no polynomial time $(\alpha_{GW} + \epsilon)$-approximation algorithm for fault tolerant Max-Cut in unweighted graphs, for any constant $\epsilon > 0$. This holds for both adaptive and oblivious adversaries.*

In both cases, given an unweighted instance G of Max-Cut, we construct an unweighted graph G', as follows: we take the disjoint union of G with a star with $n = |V|$ leaves and a center u^*, and add an edge joining u^* to an arbitrary vertex $v_1 \in V$. This completes the construction of G'. Clearly, this is a polynomial construction.

We show for each kind of adversary how to translate a given (approximate) solution to *AFTcut* or *OFTcut* in G' into a solution to Max-Cut in G, which would imply the corresponding inapproximability results, using the fact that Max-Cut is hard to approximate within a factor better than α_{GW} [39]. We only present the proof for *OFTcut*, leaving *AFTcut* to the full version. We use the following simple observation.

▶ **Observation 5.1.** *Let $S \subseteq V$ be a cut in G, and $S' = S \cup \{u^*\}$. It holds that in G', u^* is a critical vertex of S', i.e., $\varphi(S') = C_{S'-u^*,G'-u^*}$. For every cut $S'' \subseteq V'$, we have $C_{S''-u^*,G'} = C_{S'' \cap V,G}$.*

The proof follows from the fact that for every vertex $v \in V'$, $d_{S'}(v) \leq n \leq d_{S'}(u^*)$, and that all edges in $G' - u^*$ belong to G.

We show first that the optimal values for *OFTcut* in G' and Max-Cut in G are equal.

▶ **Lemma 5.2.** *Let $\mathcal{D}^*$ be the distribution of an optimal OFTcut in G', and S^*_{mc} be an optimal Max-Cut in G. It holds that $\mu(\mathcal{D}^*, G') = C_{S^*_{mc},G}$.*

Proof. Let $\widetilde{S} = S^*_{mc} \cup \{u^*\}$, and let $\mathcal{D}$ be the distribution that assigns probability 1 to $\widetilde{S}$ and probability 0 to all other cuts. By Observation 5.1, u^* is a critical vertex, hence for every vertex $v \in G'$, we have $\mathop{\mathbb{E}}_{S \sim \mathcal{D}}[C_{S-u^*,G'-u^*}] = C_{\widetilde{S}-u^*,G'-u^*} \leq C_{\widetilde{S}-v,G'-v} = \mathop{\mathbb{E}}_{S \sim \mathcal{D}}[C_{S-v,G'-v}]$. Using Observation 5.1 again, we have $\mu(\mathcal{D}^*, G') \geq \mu(\mathcal{D}, G') \geq C_{\widetilde{S}-u^*,G'-u^*} = C_{S^*_{mc},G}$. Next, by Observation 5.1, we have $C_{S-u^*,G'-u^*} = C_{S \cap V,G} \leq C_{S^*_{mc},G}$, for every cut S from the support of $\mathcal{D}^*$, which implies that $\mu(\mathcal{D}^*, G') \leq \mathop{\mathbb{E}}_{S \sim \mathcal{D}^*}[C_{S-u^*,G'-u^*}] \leq C_{S^*_{mc},G}$. This completes the proof. ◀

Proof of Theorem 1.3 for an Oblivious Adversary. Assume, for a contradiction, that we have an α-approximation algorithm for $OFTcut$, for $\alpha = \alpha_{GW} + \epsilon > \alpha_{GW}$. We design a randomized approximation algorithm for Max-Cut. Let G be an input to Max-Cut. Construct the graph G' as described above. Let $\mathcal{D}$ be the distribution of an α-approximate $OFTcut$ in G'. By Lemma 5.2, we have $\mathop{\mathbb{E}}_{S\sim\mathcal{D}}[C_{S-u^*,G'-u^*}] \geq \mu(\mathcal{D},G') \geq \alpha \cdot C_{S^*_{mc},G}$, where S^*_{mc} is a Max-Cut in G. By Observation 5.1, it holds that $C_{S_{ft}-u^*,G'-u^*} = C_{S_{ft}\cap V,G} \leq C_{S^*_{mc}}$ for every cut S_{ft} in the support of $\mathcal{D}$. Letting $p = \mathbb{P}\left[C_{S-u^*,G'-u^*} \geq (\alpha-\epsilon/2)C_{S^*_{mc}}\right]$, we have

$$\alpha \cdot C_{S^*_{mc},G} \leq \mathop{\mathbb{E}}_{S\sim\mathcal{D}}[C_{S-u^*,G'-u^*}] \leq p \cdot C_{S^*_{mc},G} + (1-p)\cdot(\alpha-\epsilon/2)C_{S^*_{mc},G} \ ,$$

implying that $p \geq \epsilon/2$. Thus, for a random cut S_{ft} sampled from $\mathcal{D}$, it holds that $S_{ft}\cap V$ is an $(\alpha-\epsilon/2)$-approximation to Max-Cut, with probability $\epsilon/2$, where $\alpha-\epsilon/2 > \alpha_{GW}$. This contradicts to our assumption about the Unique Games Conjecture and $NP \not\subseteq BPP$. ◀

6 Discussion

Our work leaves several open questions regarding fault tolerant Max-Cut. An immediate question is to bridge the (rather small) gap between our approximation of $(0.8780-\epsilon)$ and our hardness of α_{GW} for k-$AFTcut$.

The central bottleneck is that Simultaneous Max-Cut, a main ingredient in our algorithm, has hardness of approximation that is slightly below α_{GW} and equals $(\alpha_{GW}-\delta)$ (where $\delta \geq 10^{-5}$) [11]. Thus, either one finds a different algorithm for k-$AFTcut$ that does not rely on Simultaneous Max-Cut and achieves an approximation of α_{GW}, or one can extend the hardness result of [11] to k-$AFTcut$ and thus rule out an approximation of α_{GW} for k-$AFTcut$. Another question is what approximation factors can be obtained for $AFTcut$ on general weighted graphs.

Another interesting question is how to deal with a non-constant number of faults, for both of the adversaries. Since the number of all possible cases of failure is not polynomial, a new approach may be needed. There are techniques that are used to deal with a non-constant number of faults, e.g., failure sampling, that is presented in [24]. It would be interesting to see whether these techniques can be used for fault tolerant Max-Cut as well.

One more important and intriguing open question is what happens in other fault tolerant problems when an oblivious adversary is considered. We are unaware of previous algorithms for an oblivious adversary in the fault-tolerance literature. Since an oblivious adversary is arguably more realistic in its nature, and since it is likely that one can get improved algorithms for this case, pursuing this line of research could be crucial for many additional fundamental problems involving fault tolerance.

References

1 Noga Alon, Shiri Chechik, and Sarel Cohen. Deterministic combinatorial replacement paths and distance sensitivity oracles. In Christel Baier, Ioannis Chatzigiannakis, Paola Flocchini, and Stefano Leonardi, editors, *46th International Colloquium on Automata, Languages, and Programming, ICALP 2019, July 9-12, 2019, Patras, Greece*, volume 132 of *LIPIcs*, pages 12:1–12:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019. doi:10.4230/LIPIcs.ICALP.2019.12.

2 Noga Alon and Assaf Naor. Approximating the cut-norm via grothendieck's inequality. *SIAM J. Comput.*, 35(4):787–803, 2006.

3 Eric Angel, Evripidis Bampis, and Laurent Gourvès. Approximation algorithms for the bi-criteria weighted MAX-CUT problem. *Discret. Appl. Math.*, 154(12):1685–1692, 2006. doi:10.1016/j.dam.2006.02.008.

4 Takao Asano and David P Williamson. Improved approximation algorithms for MAX SAT. *Journal of Algorithms*, 42(1):173–202, 2002.

5 Per Austrin, Siavosh Benabbas, and Konstantinos Georgiou. Better balance by being biased: A 0.8776-approximation for Max Bisection. *ACM Trans. Algorithms*, 13(1):2:1–2:27, 2016.

6 Dmitrii Avdiukhin, Slobodan Mitrovic, Grigory Yaroslavtsev, and Samson Zhou. Adversarially robust submodular maximization under knapsack constraints. In Ankur Teredesai, Vipin Kumar, Ying Li, Rómer Rosales, Evimaria Terzi, and George Karypis, editors, *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, KDD 2019, Anchorage, AK, USA, August 4-8, 2019*, pages 148–156. ACM, 2019. doi:10.1145/3292500.3330911.

7 Adi Avidor, Ido Berkovitch, and Uri Zwick. Improved approximation algorithms for MAX NAE-SAT and MAX SAT. In Thomas Erlebach and Giuseppe Persiano, editors, *Approximation and Online Algorithms, Third International Workshop, WAOA 2005, Palma de Mallorca, Spain, October 6-7, 2005, Revised Papers*, volume 3879 of *Lecture Notes in Computer Science*, pages 27–40. Springer, 2005. doi:10.1007/11671411_3.

8 Surender Baswana, Keerti Choudhary, Moazzam Hussain, and Liam Roditty. Approximate single source fault tolerant shortest path. In Artur Czumaj, editor, *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2018, New Orleans, LA, USA, January 7-10, 2018*, pages 1901–1915. SIAM, 2018. doi:10.1137/1.9781611975031.124.

9 Surender Baswana, Keerti Choudhary, and Liam Roditty. Fault tolerant reachability for directed graphs. In Yoram Moses, editor, *Distributed Computing - 29th International Symposium, DISC 2015, Tokyo, Japan, October 7-9, 2015, Proceedings*, volume 9363 of *Lecture Notes in Computer Science*, pages 528–543. Springer, 2015. doi:10.1007/978-3-662-48653-5_35.

10 Surender Baswana, Keerti Choudhary, and Liam Roditty. Fault tolerant subgraph for single source reachability: generic and optimal. In Daniel Wichs and Yishay Mansour, editors, *Proceedings of the 48th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2016, Cambridge, MA, USA, June 18-21, 2016*, pages 509–518. ACM, 2016. doi:10.1145/2897518.2897648.

11 Amey Bhangale and Subhash Khot. Simultaneous max-cut is harder to approximate than max-cut. In Shubhangi Saraf, editor, *35th Computational Complexity Conference, CCC 2020, July 28-31, 2020, Saarbrücken, Germany (Virtual Conference)*, volume 169 of *LIPIcs*, pages 9:1–9:15. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPIcs.CCC.2020.9.

12 Amey Bhangale, Subhash Khot, Swastik Kopparty, Sushant Sachdeva, and Devanathan Thiruvenkatachari. Near-optimal approximation algorithm for simultaneous max-cut. In Artur Czumaj, editor, *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2018, New Orleans, LA, USA, January 7-10, 2018*, pages 1407–1425. SIAM, 2018. doi:10.1137/1.9781611975031.93.

13 Amey Bhangale, Swastik Kopparty, and Sushant Sachdeva. Simultaneous approximation of constraint satisfaction problems. In Magnús M. Halldórsson, Kazuo Iwama, Naoki Kobayashi, and Bettina Speckmann, editors, *Automata, Languages, and Programming - 42nd International Colloquium, ICALP 2015, Kyoto, Japan, July 6-10, 2015, Proceedings, Part I*, volume 9134 of *Lecture Notes in Computer Science*, pages 193–205. Springer, 2015. doi:10.1007/978-3-662-47672-7_16.

14 Davide Bilò, Luciano Gualà, Stefano Leucci, and Guido Proietti. Fault-tolerant approximate shortest-path trees. *Algorithmica*, 80(12):3437–3460, 2018. doi:10.1007/s00453-017-0396-z.

15 Greg Bodwin, Michael Dinitz, Merav Parter, and Virginia Vassilevska Williams. Optimal vertex fault tolerant spanners (for fixed stretch). In Artur Czumaj, editor, *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2018, New Orleans, LA, USA, January 7-10, 2018*, pages 1884–1900. SIAM, 2018. doi:10.1137/1.9781611975031.123.

16 Greg Bodwin, Fabrizio Grandoni, Merav Parter, and Virginia Vassilevska Williams. Preserving distances in very faulty graphs. In Ioannis Chatzigiannakis, Piotr Indyk, Fabian Kuhn, and Anca Muscholl, editors, *44th International Colloquium on Automata, Languages, and Programming, ICALP 2017, July 10-14, 2017, Warsaw, Poland*, volume 80 of *LIPIcs*, pages 73:1–73:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2017. doi:10.4230/LIPIcs.ICALP.2017.73.

17 Greg Bodwin and Shyamal Patel. A trivial yet optimal solution to vertex fault tolerant spanners. In Peter Robinson and Faith Ellen, editors, *Proceedings of the 2019 ACM Symposium on Principles of Distributed Computing, PODC 2019, Toronto, ON, Canada, July 29 - August 2, 2019*, pages 541–543. ACM, 2019. doi:10.1145/3293611.3331588.

18 Austin Buchanan, Je Sang Sung, Sergiy Butenko, and Eduardo L. Pasiliao. An integer programming approach for fault-tolerant connected dominating sets. *INFORMS J. Comput.*, 27(1):178–188, 2015. doi:10.1287/ijoc.2014.0619.

19 Keren Censor-Hillel, Noa Marelly, Roy Schwartz, and Tigran Tonoyan. Fault tolerant max-cut. *CoRR*, arXiv:2105.01138, 2021. arXiv:2105.01138.

20 Moses Charikar, Venkatesan Guruswami, and Anthony Wirth. Clustering with qualitative information. *J. Comput. Syst. Sci.*, 71(3):360–383, 2005.

21 Shiri Chechik and Sarel Cohen. Near optimal algorithms for the single source replacement paths problem. In Timothy M. Chan, editor, *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2019, San Diego, California, USA, January 6-9, 2019*, pages 2090–2109. SIAM, 2019. doi:10.1137/1.9781611975482.126.

22 Shiri Chechik and Ofer Magen. Near optimal algorithm for the directed single source replacement paths problem. In Artur Czumaj, Anuj Dawar, and Emanuela Merelli, editors, *47th International Colloquium on Automata, Languages, and Programming, ICALP 2020, July 8-11, 2020, Saarbrücken, Germany (Virtual Conference)*, volume 168 of *LIPIcs*, pages 81:1–81:17. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPIcs.ICALP.2020.81.

23 Shiri Chechik and David Peleg. Robust fault tolerant uncapacitated facility location. *Theor. Comput. Sci.*, 543:9–23, 2014. doi:10.1016/j.tcs.2014.05.013.

24 Michael Dinitz and Robert Krauthgamer. Directed spanners via flow-based linear programs. In Lance Fortnow and Salil P. Vadhan, editors, *Proceedings of the 43rd ACM Symposium on Theory of Computing, STOC 2011, San Jose, CA, USA, 6-8 June 2011*, pages 323–332. ACM, 2011. doi:10.1145/1993636.1993680.

25 Michael Dinitz and Robert Krauthgamer. Fault-tolerant spanners: better and simpler. In Cyril Gavoille and Pierre Fraigniaud, editors, *Proceedings of the 30th Annual ACM Symposium on Principles of Distributed Computing, PODC 2011, San Jose, CA, USA, June 6-8, 2011*, pages 169–178. ACM, 2011. doi:10.1145/1993806.1993830.

26 Uriel Feige and Michel Goemans. Approximating the value of two power proof systems, with applications to Max 2-SAT and Max DiCut. In *istcs*, page 0182. IEEE, 1995.

27 M. R. Garey, David S. Johnson, and Larry J. Stockmeyer. Some simplified np-complete graph problems. *Theor. Comput. Sci.*, 1(3):237–267, 1976. doi:10.1016/0304-3975(76)90059-1.

28 Mohsen Ghaffari and Merav Parter. Near-optimal distributed algorithms for fault-tolerant tree structures. In Christian Scheideler and Seth Gilbert, editors, *Proceedings of the 28th ACM Symposium on Parallelism in Algorithms and Architectures, SPAA 2016, Asilomar State Beach/Pacific Grove, CA, USA, July 11-13, 2016*, pages 387–396. ACM, 2016. doi:10.1145/2935764.2935795.

29 Michel X. Goemans and David P. Williamson. Improved approximation algorithms for maximum cut and satisfiability problems using semidefinite programming. *J. ACM*, 42(6):1115–1145, 1995. doi:10.1145/227683.227684.

30 Fabrizio Grandoni and Virginia Vassilevska Williams. Improved distance sensitivity oracles via fast single-source replacement paths. In *53rd Annual IEEE Symposium on Foundations of Computer Science, FOCS 2012, New Brunswick, NJ, USA, October 20-23, 2012*, pages 748–757. IEEE Computer Society, 2012. doi:10.1109/FOCS.2012.17.

31 Martin Grötschel, László Lovász, and Alexander Schrijver. *The Ellipsoid Method*, pages 64–101. Springer Berlin Heidelberg, Berlin, Heidelberg, 1993. doi:10.1007/978-3-642-78240-4_4.

32 Sudipto Guha, Adam Meyerson, and Kamesh Munagala. A constant factor approximation algorithm for the fault-tolerant facility location problem. *J. Algorithms*, 48(2):429–440, 2003. doi:10.1016/S0196-6774(03)00056-7.

33 Manoj Gupta and Shahbaz Khan. Multiple source dual fault tolerant BFS trees. In Ioannis Chatzigiannakis, Piotr Indyk, Fabian Kuhn, and Anca Muscholl, editors, *44th International Colloquium on Automata, Languages, and Programming, ICALP 2017, July 10-14, 2017, Warsaw, Poland*, volume 80 of *LIPIcs*, pages 127:1–127:15. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2017. doi:10.4230/LIPIcs.ICALP.2017.127.

34 Johan Håstad. Some optimal inapproximability results. *J. ACM*, 48(4):798–859, 2001. doi:10.1145/502090.502098.

35 Kamal Jain, Mohammad Mahdian, and Mohammad R. Salavatipour. Packing steiner trees. In *Proceedings of the Fourteenth Annual ACM-SIAM Symposium on Discrete Algorithms, January 12-14, 2003, Baltimore, Maryland, USA*, pages 266–274. ACM/SIAM, 2003. URL: http://dl.acm.org/citation.cfm?id=644108.644154.

36 Kamal Jain and Vijay V. Vazirani. An approximation algorithm for the fault tolerant metric facility location problem. *Algorithmica*, 38(3):433–439, 2004. doi:10.1007/s00453-003-1070-1.

37 Richard M. Karp. Reducibility among combinatorial problems. In Raymond E. Miller and James W. Thatcher, editors, *Proceedings of a symposium on the Complexity of Computer Computations, held March 20-22, 1972, at the IBM Thomas J. Watson Research Center, Yorktown Heights, New York, USA*, The IBM Research Symposia Series, pages 85–103. Plenum Press, New York, 1972. doi:10.1007/978-1-4684-2001-2_9.

38 Subhash Khot. On the power of unique 2-prover 1-round games. In John H. Reif, editor, *Proceedings on 34th Annual ACM Symposium on Theory of Computing, May 19-21, 2002, Montréal, Québec, Canada*, pages 767–775. ACM, 2002. doi:10.1145/509907.510017.

39 Subhash Khot, Guy Kindler, Elchanan Mossel, and Ryan O'Donnell. Optimal inapproximability results for MAX-CUT and other 2-variable csps? *SIAM J. Comput.*, 37(1):319–357, 2007. doi:10.1137/S0097539705447372.

40 Subhash Khot and Oded Regev. Vertex cover might be hard to approximate to within 2-ϵ. *J. Comput. Syst. Sci.*, 74(3):335–349, 2008. doi:10.1016/j.jcss.2007.06.019.

41 Christos Levcopoulos, Giri Narasimhan, and Michiel H. M. Smid. Improved algorithms for constructing fault-tolerant spanners. *Algorithmica*, 32(1):144–156, 2002. doi:10.1007/s00453-001-0075-x.

42 Michael Lewin, Dror Livnat, and Uri Zwick. Improved rounding techniques for the MAX 2-sat and MAX DI-CUT problems. In William J. Cook and Andreas S. Schulz, editors, *Integer Programming and Combinatorial Optimization, 9th International IPCO Conference, Cambridge, MA, USA, May 27-29, 2002, Proceedings*, volume 2337 of *Lecture Notes in Computer Science*, pages 67–82. Springer, 2002. doi:10.1007/3-540-47867-1_6.

43 Sanjeev Mahajan and H. Ramesh. Derandomizing approximation algorithms based on semidefinite programming. *SIAM J. Comput.*, 28(5):1641–1663, 1999. doi:10.1137/S0097539796309326.

44 Shiro Matuura and Tomomi Matsui. 63-approximation algorithm for MAX DICUT. In Michel X. Goemans, Klaus Jansen, José D. P. Rolim, and Luca Trevisan, editors, *Approximation, Randomization and Combinatorial Optimization: Algorithms and Techniques, 4th International Workshop on Approximation Algorithms for Combinatorial Optimization Problems, APPROX 2001 and 5th International Workshop on Randomization and Approximation Techniques in Computer Science, RANDOM 2001 Berkeley, CA, USA, August 18-20, 2001, Proceedings*, volume 2129 of *Lecture Notes in Computer Science*, pages 138–146. Springer, 2001. doi:10.1007/3-540-44666-4_17.

45 Elchanan Mossel, Ryan O'Donnell, and Krzysztof Oleszkiewicz. Noise stability of functions with low influences: invariance and optimality. In *Foundations of Computer Science, (FOCS)*, pages 21–30, 2005.

46 James B. Orlin, Andreas S. Schulz, and Rajan Udwani. Robust monotone submodular function maximization. In Quentin Louveaux and Martin Skutella, editors, *Integer Programming and Combinatorial Optimization - 18th International Conference, IPCO 2016, Liège, Belgium, June 1-3, 2016, Proceedings*, volume 9682 of *Lecture Notes in Computer Science*, pages 312–324. Springer, 2016. doi:10.1007/978-3-319-33461-5_26.

47 Merav Parter. Vertex fault tolerant additive spanners. In Fabian Kuhn, editor, *Distributed Computing - 28th International Symposium, DISC 2014, Austin, TX, USA, October 12-15, 2014. Proceedings*, volume 8784 of *Lecture Notes in Computer Science*, pages 167–181. Springer, 2014. doi:10.1007/978-3-662-45174-8_12.

48 Merav Parter. Dual failure resilient BFS structure. In Chryssis Georgiou and Paul G. Spirakis, editors, *Proceedings of the 2015 ACM Symposium on Principles of Distributed Computing, PODC 2015, Donostia-San Sebastián, Spain, July 21 - 23, 2015*, pages 481–490. ACM, 2015. doi:10.1145/2767386.2767408.

49 Merav Parter. Distributed constructions of dual-failure fault-tolerant distance preservers. In Hagit Attiya, editor, *34th International Symposium on Distributed Computing, DISC 2020, October 12-16, 2020, Virtual Conference*, volume 179 of *LIPIcs*, pages 21:1–21:17. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPIcs.DISC.2020.21.

50 Merav Parter and David Peleg. Sparse fault-tolerant BFS trees. *CoRR*, abs/1302.5401, 2013. arXiv:1302.5401.

51 Merav Parter and David Peleg. Fault tolerant approximate BFS structures. In Chandra Chekuri, editor, *Proceedings of the Twenty-Fifth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2014, Portland, Oregon, USA, January 5-7, 2014*, pages 1073–1092. SIAM, 2014. doi:10.1137/1.9781611973402.80.

52 Merav Parter and David Peleg. Fault tolerant BFS structures: A reinforcement-backup tradeoff. In Guy E. Blelloch and Kunal Agrawal, editors, *Proceedings of the 27th ACM on Symposium on Parallelism in Algorithms and Architectures, SPAA 2015, Portland, OR, USA, June 13-15, 2015*, pages 264–273. ACM, 2015. doi:10.1145/2755573.2755590.

53 Prasad Raghavendra and Ning Tan. Approximating csps with global cardinality constraints using SDP hierarchies. In *Symposium on Discrete Algorithms (SODA)*, pages 373–387. SIAM, 2012.

54 Liam Roditty and Uri Zwick. Replacement paths and k simple shortest paths in unweighted directed graphs. *ACM Trans. Algorithms*, 8(4):33:1–33:11, 2012. doi:10.1145/2344422.2344423.

55 Shay Solomon. From hierarchical partitions to hierarchical covers: optimal fault-tolerant spanners for doubling metrics. In David B. Shmoys, editor, *Symposium on Theory of Computing, STOC 2014, New York, NY, USA, May 31 - June 03, 2014*, pages 363–372. ACM, 2014. doi:10.1145/2591796.2591864.

56 Chaitanya Swamy. Correlation clustering: Maximizing agreements via semidefinite programming. In *Symposium on Discrete Algorithms*, SODA, pages 526–527, 2004.

57 Chaitanya Swamy and David B. Shmoys. Fault-tolerant facility location. *ACM Trans. Algorithms*, 4(4):51:1–51:27, 2008. doi:10.1145/1383369.1383382.

58 Luca Trevisan, Gregory B. Sorkin, Madhu Sudan, and David P. Williamson. Gadgets, approximation, and linear programming. *SIAM J. Comput.*, 29(6):2074–2097, 2000. doi:10.1137/S0097539797328847.

59 Jiao Zhou, Zhao Zhang, Shaojie Tang, Xiaohui Huang, and Ding-Zhu Du. Breaking the $O(\ln n)$ barrier: An enhanced approximation algorithm for fault-tolerant minimum weight connected dominating set. *INFORMS J. Comput.*, 30(2):225–235, 2018. doi:10.1287/ijoc.2017.0775.

Algorithms, Reductions and Equivalences for Small Weight Variants of All-Pairs Shortest Paths

Timothy M. Chan
University of Illinois at Urbana-Champaign, IL, USA

Virginia Vassilevska Williams
MIT, CSAIL, Cambridge, MA, USA

Yinzhan Xu
MIT, Cambridge, MA, USA

Abstract

All-Pairs Shortest Paths (APSP) is one of the most well studied problems in graph algorithms. This paper studies several variants of APSP in unweighted graphs or graphs with small integer weights.

APSP with small integer weights in undirected graphs [Seidel'95, Galil and Margalit'97] has an $\tilde{O}(n^\omega)$ time algorithm, where $\omega < 2.373$ is the matrix multiplication exponent. APSP in directed graphs with small weights however, has a much slower running time that would be $\Omega(n^{2.5})$ even if $\omega = 2$ [Zwick'02]. To understand this $n^{2.5}$ bottleneck, we build a web of reductions around directed unweighted APSP. We show that it is *fine-grained equivalent* to computing a rectangular Min-Plus product for matrices with integer entries; the dimensions and entry size of the matrices depend on the value of ω. As a consequence, we establish an equivalence between APSP in directed unweighted graphs, APSP in directed graphs with small ($\tilde{O}(1)$) integer weights, All-Pairs Longest Paths in DAGs with small weights, cRed-APSP in undirected graphs with small weights, for any $c \geq 2$ (computing all-pairs shortest path distances among paths that use at most c red edges), $\#_{\leq c}$APSP in directed graphs with small weights (counting the number of shortest paths for each vertex pair, up to c), and approximate APSP with additive error c in directed graphs with small weights, for $c \leq \tilde{O}(1)$.

We also provide fine-grained reductions from directed unweighted APSP to All-Pairs Shortest Lightest Paths (APSLP) in undirected graphs with $\{0, 1\}$ weights and $\#_{\bmod c}$APSP in directed unweighted graphs (computing counts mod c), thus showing that unless the current algorithms for APSP in directed unweighted graphs can be improved substantially, these problems need at least $\Omega(n^{2.528})$ time.

We complement our hardness results with new algorithms. We improve the known algorithms for APSLP in directed graphs with small integer weights (previously studied by Zwick [STOC'99]) and for approximate APSP with sublinear additive error in directed unweighted graphs (previously studied by Roditty and Shapira [ICALP'08]). Our algorithm for approximate APSP with sublinear additive error is optimal, when viewed as a reduction to Min-Plus product. We also give new algorithms for variants of #APSP (such as $\#_{\leq U}$APSP and $\#_{\bmod U}$APSP for $U \leq n^{\tilde{O}(1)}$) in unweighted graphs, as well as a near-optimal $\tilde{O}(n^3)$-time algorithm for the original #APSP problem in unweighted graphs (when counts may be exponentially large). This also implies an $\tilde{O}(n^3)$-time algorithm for Betweenness Centrality, improving on the previous $\tilde{O}(n^4)$ running time for the problem. Our techniques also lead to a simpler alternative to Shoshan and Zwick's algorithm [FOCS'99] for the original APSP problem in undirected graphs with small integer weights.

2012 ACM Subject Classification Theory of computation → Shortest paths

Keywords and phrases All-Pairs Shortest Paths, Fine-Grained Complexity, Graph Algorithm

Digital Object Identifier 10.4230/LIPIcs.ICALP.2021.47

Category Track A: Algorithms, Complexity and Games

Related Version *Full Version*: https://arxiv.org/abs/2102.06181

Funding *Timothy M. Chan*: Supported by NSF Grant CCF-1814026.
Virginia Vassilevska Williams: Supported by an NSF CAREER Award, NSF Grants CCF-1528078, CCF-1514339 and CCF-1909429, a BSF Grant BSF:2012338, a Google Research Fellowship and a Sloan Research Fellowship.
Yinzhan Xu: Supported by NSF Grant CCF-1528078.

1 Introduction

All-Pairs Shortest Paths (APSP) is one of the oldest and most studied problems in graph algorithms. The fastest known algorithm for general n-node graphs runs in $n^3/2^{\Theta(\sqrt{\log n})}$ [31]. In unweighted graphs, or graphs with small integer weights, faster algorithms are known.

For APSP in *undirected* unweighted graphs (u-APSP), Seidel [21] and Galil and Margalit [12, 13] gave $\tilde{O}(n^\omega)$ time algorithms where $\omega \leq 2.373$ is the exponent of matrix multiplication [2, 27, 16]; the latter algorithm works for graphs with small integer weights[1] in $[\pm c_0]$ for $c_0 = \tilde{O}(1)$. The hidden dependence on c_0 was improved by Shoshan and Zwick [22].

For *directed* unweighted graphs or graphs with weights in $[\pm c_0]$, the fastest APSP algorithm is by Zwick [34], running in $O(n^{2.529})$ time. This running time is achieved using the best known bounds for rectangular matrix multiplication [17] and would be $\Omega(n^{2.5})$ even if $\omega = 2$.

There is a big discrepancy between the running times for undirected and directed APSP. One might wonder, why is this? Are directed graphs inherently more difficult for APSP, or is there some special graph structure we can uncover and then use it to develop an $\tilde{O}(n^\omega)$ time algorithm for directed APSP as well? (Note that matrix multiplication seems necessary for APSP since APSP is known to capture Boolean matrix multiplication.)

The first contribution in this paper is a fine-grained equivalence between directed unweighted APSP (u-APSP) and a certain rectangular version of the Min-Plus product problem.

The *Min-Plus product* of an $n \times m$ matrix A by an $m \times p$ matrix B is the matrix C with entries $C[i,j] = \min_{k=1}^{m}(A[i,k] + B[k,j])$. Let us denote by $\mathrm{M}^\star(n_1, n_2, n_3 \mid M)$ the problem of computing the Min-Plus product of an $n_1 \times n_2$ matrix by an $n_2 \times n_3$ matrix where both matrices have integer entries in $[M]$. Let $\mathcal{M}^\star(n_1, n_2, n_3 \mid M)$ be the best running time for $\mathrm{M}^\star(n_1, n_2, n_3 \mid M)$.

Zwick's algorithm [34] for u-APSP can be viewed as making a logarithmic number of calls to the Min-Plus product $\mathrm{M}^\star(n, n/L, n \mid L)$ for all $1 \leq L \leq n$ that are powers of $3/2$. The running time of Zwick's algorithm is thus, within polylogarithmic factors, $\max_L \mathcal{M}^\star(n, n/L, n \mid L)$.

Let $\mathcal{M}(a, b, c)$ denote the running time of the fastest algorithm to multiply an $a \times b$ by a $b \times c$ matrix over the integers. Let $\omega(a, b, c)$ be the smallest real number r such that $\mathcal{M}(n^a, n^b, n^c) \leq O(n^{r+\varepsilon})$ for all $\varepsilon > 0$.

The best known upper bound for the Min-Plus product running time $\mathcal{M}^\star(n, n/L, n \mid L)$ is the minimum of $O(n^3/L)$ (the brute force algorithm) and $\tilde{O}(L \cdot \mathcal{M}(n, n/L, n))$ [3]. For $L = n^{1-\ell}$, $\mathcal{M}^\star(n, n/L, n \mid L)$ is thus at most $\tilde{O}(\min\{n^{2+\ell}, n^{1-\ell+\omega(1,\ell,1)}\})$. Over all $\ell \in [0, 1]$, the runtime is maximized at $\tilde{O}(n^{2+\rho})$ where ρ is such that $\omega(1, \rho, 1) = 1 + 2\rho$.

Hence in particular, the running time of Zwick's algorithm is $\tilde{O}(n^{2+\rho})$. This running time has remained unchanged (except for improvements on the bounds on ρ) for almost 20 years. The current best known bound on ρ is $\rho < 0.529$, and if $\omega = 2$, then $\rho = 1/2$.

[1] In this paper, $[\pm c_0] = \{-c_0, \ldots, c_0\}$ and $[c_0] = \{0, \ldots, c_0\}$. The $\tilde{O}$ notation hides polylogarithmic factors (although conditions of the form $c_0 = \tilde{O}(1)$ may be relaxed to $c_0 \leq n^{o(1)}$ if we allow extra $n^{o(1)}$ factors in the $\tilde{O}$ time bounds).

Our first result is that u-APSP is sub-$n^{2+\rho}$ fine-grained equivalent to $\mathrm{M}^\star(n, n^\rho, n \mid n^{1-\rho})$:

▶ **Theorem 1.** *If $\mathrm{M}^\star(n, n^\rho, n \mid n^{1-\rho})$ is in $O(n^{2+\rho-\varepsilon})$ time for $\varepsilon > 0$, then u-APSP can also be solved in $O(n^{2+\rho-\varepsilon'})$ time for some $\varepsilon' > 0$. If u-APSP can be solved in $O(n^{2+\rho-\varepsilon})$ time for some $\varepsilon > 0$, then $\mathrm{M}^\star(n, n^\rho, n \mid n^{1-\rho})$ can also be solved in $O(n^{2+\rho-\varepsilon})$ time.*

The Min-Plus product of two $n \times n$ matrices with *arbitrary* integer entries is known to be equivalent to APSP with *arbitrary* integer entries [10], so that their running times are the same, up to constant factors. All known algorithms for directed unweighted APSP (including [34, 3] and others), make calls to Min-Plus product of rectangular matrices with integer entries that can be as large as say $n^{0.4}$. It is completely unclear, however, why a problem in *unweighted* graphs such as u-APSP should require the computation of Min-Plus products of matrices with such large entries. Theorem 1 surprisingly shows that it does. Moreover, it shows that unless we can improve upon the known approaches for Min-Plus product computation, there will be no way to improve upon Zwick's algorithm for u-APSP. The latter is an algebraic problem in disguise.

The main proof of Theorem 1 is simple – what is remarkable are the numerous consequences on equivalences and conditional hardness that follow from this idea. We first use Theorem 1 to build a class of problems that are all equivalent to u-APSP, via $(n^{2+\rho}, n^{2+\rho})$-fine-grained reductions (see [29] for a survey of fine-grained complexity). In particular, if $\omega = 2$ (or more generally when $\omega(1, \frac{1}{2}, 1) = 2$), these are all problems that are $n^{2.5}$-fine-grained equivalent.

Recall that in the *All-Pairs Longest Paths (APLP)* problem, we want to output for every pair of vertices s, t the weight of the longest path from s to t. While APLP is NP-hard in general, it is efficiently solvable in DAGs. In the *cRed-APSP* problem, for a given graph in which some edges can be colored red, we want to output for every pair of vertices s, t the weight of the shortest path from s to t that uses at most c red edges. For convenience, we call all non-red edges blue.

We use the following convention for problem names: the prefix "u-" is for unweighted graphs; the prefix "$[c_0]$-" is for graphs with weights in $[c_0]$ (similarly for "$[\pm c_0]$-" and for other ranges). Input graphs are directed unless stated otherwise.

▶ **Theorem 2.** *The following problems either all have $O(n^{2+\rho-\varepsilon})$ time algorithms for some $\varepsilon > 0$, or none of them do, assuming that $c_0 = \tilde{O}(1)$:*

- $\mathrm{M}^\star(n, n^\rho, n \mid n^{1-\rho})$,
- *u-APSP,*
- *$[\pm c_0]$-APSP for directed graphs without negative cycles,*
- *u-APLP for DAGs,*
- *$[\pm c_0]$-APLP for DAGs,*
- *u-cRed-APSP for* undirected *graphs for any $2 \le c \le \tilde{O}(1)$.*

Interestingly, while u-2Red-APSP in undirected graphs above is equivalent to u-APSP and hence improving upon its $\tilde{O}(n^{2+\rho})$ runtime would be difficult, we show that u-1Red-APSP in undirected graphs can be solved in $\tilde{O}(n^\omega)$ time via a modification of Seidel's algorithm, and hence there is a seeming jump in complexity in u-cRed-APSP from $c = 1$ to $c = 2$.

Besides the above equivalences we provide some interesting reductions from u-APSP to other well-studied matrix product and shortest paths problems.

Lincoln, Polak and Vassilevska W. [18] reduce u-APSP to some matrix product problems such as All-Edges Monochromatic Triangle and the $(\min, \max)$-Product studied in [24, 26] and [25, 9] respectively. Using the equivalence of u-APSP and $\mathrm{M}^\star(n, n/\ell, n \mid \ell^{1-p})$, we can reduce

u-APSP to another matrix product called Min Witness Equality Product (MinWitnessEq), where we are given $n \times n$ integer matrices A and B, and are required to compute $\min\{k \in [n] : A[i,k] = B[k,j]\}$ for every pair of (i,j). This can be viewed as a merge of the Min Witness product [8] [2] and Equality Product problems [15, 28].

Another natural variant of APSP is the problem of *approximating* shortest path distances. Zwick [34] presented an $\tilde{O}(n^\omega \log M)$ time algorithm for computing a $(1+\varepsilon)$-multiplicative approximation for all pairwise distances in a directed graph with integer weights in $[M]$, for any constant $\varepsilon > 0$.[3] This is essentially optimal assuming no $o(n^\omega)$ time algorithm can multiple $n \times n$ Boolean matrices since any such approximation algorithm can be used to multiply Boolean matrices.

An arguably better notion of approximation is to provide an additive approximation, i.e. outputting for every u, v an estimate $D'[u,v]$ for the distance $D[u,v]$ such that $D[u,v] \leq D'[u,v] \leq D[u,v] + E$, where E is an error that can depend on u and v.

At ICALP'08, Roditty and Shapira [20] studied the following variant: given an unweighted directed graph and a constant $p \in [0,1]$, compute for all u, v an estimate $D'[u,v]$ with $D[u,v] \leq D'[u,v] \leq D[u,v] + D[u,v]^p$. They gave an algorithm with running time $\tilde{O}(\max_\ell \min\{n^3/\ell, \mathcal{M}^\star(n, n/\ell^{1-p}, n \mid \ell^{1-p})\})$. For example, for $p = 0$, this matches the time complexity of Zwick's exact algorithm for u-APSP; for $p = 1$, this matches Zwick's $\tilde{O}(n^\omega)$-time algorithm with constant multiplicative approximation factor. For $p = 0.5$, with the current rectangular matrix multiplication bounds [17], the running time is $O(n^{2.447})$.

We obtain an improved running time:

▶ **Theorem 3.** *For any $p \in [0,1]$, given a directed unweighted graph, one can obtain additive $D[u,v]^p$ approximations to all distances $D[u,v]$ in time $\tilde{O}(\max_\ell \mathcal{M}^\star(n, n/\ell, n \mid \ell^{1-p}))$.*

The improvement over Roditty and Shapira's running time is substantial. For example, for all $p \geq 0.415$, the time bound is $O(n^{2.373})$ (the current matrix multiplication running time), whereas their algorithm only achieves $O(n^{2.373})$ for $p = 1$. Our result also answers one of Roditty and Shapira's open question (on whether $\tilde{O}(n^\omega)$ time is possible for any $p < 1$), if $\omega > 2$.

The new algorithm is also *optimal* (ignoring logarithmic factors) in a strong sense, as our reduction technique shows that for all ℓ, $\mathrm{M}^\star(n, n/\ell, n \mid \ell^{1-p})$ can be tightly reduced to the additive $D[u,v]^p$ approximation of APSP. In particular, u-APSP with constant additive error is fine-grained equivalent to exact u-APSP.

The *All-Pairs Lightest Shortest Paths (APLSP)* problem studied in [6, 33] asks to compute for every pair of vertices s, t the distance from s to t (with respect to the edge weights) and the smallest number of edges over all shortest paths from s and t. Traditional shortest-path algorithms can be easily modified to find the lightest shortest paths, but not the faster matrix-multiplication-based algorithms. Our reduction for u-cRed-APSP can be easily modified to reduce $\mathrm{M}^\star(n, n^\rho, n \mid n^{1-\rho})$ to $\{0,1\}$-APLSP in undirected graphs, which can be viewed as a conditional lower bound of $n^{2+\rho-o(1)}$ for the latter problem.

▶ **Corollary 4.** *If $\{0,1\}$-APLSP in undirected graphs is in $O(n^{2+\rho-\varepsilon})$ time for $\varepsilon > 0$, then so is $\mathrm{M}^\star(n, n^\rho, n \mid n^{1-\rho})$.*

---

[2] Recently, there has been renewed interest in studying the Min Witness product, due to a breakthrough [14] on the All-Pairs LCA in DAGs problem, which was one of the original motivations for studying Min Witness.

[3] Bringmann et al. [5] considered the more unusual setting of very large M, where the $\log M$ factor is to be avoided.

The fastest known algorithm to date for $\{0,1\}$-APLSP, or more generally, $[c_0]$-APLSP for $c_0 = \tilde{O}(1)$, for directed or undirected graphs is by Zwick [33] from STOC'99 and runs in $O(n^{2.724})$ time with the current best bounds for rectangular matrix multiplication (the running time would be $\tilde{O}(n^{8/3})$ if $\omega = 2$). Chan [6] (STOC'07) improved this running time to $\tilde{O}(n^{(3+\omega)/2}) \leq O(n^{2.687})$ but only if the weights are *positive*, i.e., for $([c_0] - \{0\})$-APLSP (and so his result does not hold for $\{0,1\}$-APLSP).

Both Zwick's and Chan's algorithms solve a more general problem, Lex_2-APSP, in which one is given a directed graph where each edge e is given two weights $w_1(e), w_2(e)$ and one wants to find for every pair of vertices u, v the lexicographic minimum over all u-v paths π of $(\sum_{e\in\pi} w_1(e), \sum_{e\in\pi} w_2(e))$. Then APLSP is Lex_2-APSP when all w_2 weights are 1, and the related *All-Pairs Shortest Lightest Paths (APSLP)* problem is when all w_1 weights are 1.

To complement the conditional lower bound for APLSP, and hence Lex_2-APSP, we present new algorithms for $[c_0]$-Lex_2-APSP for $c_0 = \tilde{O}(1)$, both (slightly) improving Chan's running time and also allowing zero weights, something that Chan's algorithm couldn't support.

▶ **Theorem 5.** *$[c_0]$-Lex_2-APSP can be solved in $O(n^{2.66})$ time for any $c_0 = \tilde{O}(1)$.*

If $\omega = 2$, the above running time would be $\tilde{O}(n^{2.5})$, improving Zwick's previous $\tilde{O}(n^{8/3})$ bound [33] and matching our conditional lower bound $n^{2+\rho-o(1)}$. For undirected graphs with *positive* weights in $[c_0] - \{0\}$, we further improve the running time to $O(n^{2.58})$ under the current matrix multiplication bounds.

We next consider the natural problem, *#APSP*, of counting the number of shortest paths for every pair of vertices in a graph. This problem needs to be solved, for example, when computing the so-called *Betweenness Centrality (BC)* of a vertex. BC is a well-studied measure of vertex importance in social networks. If we let $C[s,t]$ be the number of shortest paths between s and t, and $C_v[s,t]$ be the number of shortest paths between s and t that go through v, then $\text{BC}(v) = \sum_{s,t\neq v} C_v[s,t]/C[s,t]$ and the BC problem is to compute $\text{BC}(v)$ for a given graph and a given node v.

Prior work [4] showed that #APSP and BC in m-edge n-node unweighted graphs can be computed in $O(mn)$ time via a modification of Breadth-First Search (BFS).[4] However, all prior algorithms assumed a model of computation where adding two integers of arbitrary size takes constant time. In the more realistic word-RAM model (with $O(\log n)$ bit words), these algorithms would run in $\tilde{\Theta}(mn^2)$ time, as there are explicit examples of graphs with m edges (for any m, a function of n) for which the shortest paths counts have $\Theta(n)$ bits.[5] In particular, the best running time in terms of n so far has been $\tilde{O}(n^4)$.

We provide the first genuinely $\tilde{O}(n^3)$ time algorithm for #APSP, and thus Betweenness Centrality, in directed unweighted graphs.

▶ **Theorem 6.** *u-#APSP can be solved in $\tilde{O}(n^3)$ time by a combinatorial algorithm.*

This runtime cannot be improved since there are graphs for which the output size is $\Omega(n^3)$.

Since the main difficulty of the #APSP problem comes from the counts being very large, it is interesting to consider variants that mitigate this. Let $U \leq n^{\tilde{O}(1)}$. Let $\#_{\bmod U}$APSP be the problem of computing all pairwise counts modulo U. Let $\#_{\leq U}$APSP be the problem of

[4] Brandes presented further practical improvements as well.

[5] One example is an $(n/3+2)$-layered graph where the first $n/3$ layers have 2 vertices each and the last 2 layers have $n/6$ vertices each. The i-th layer and the $(i+1)$-th layer are connected by a complete bipartite graph for each $1 \leq i \leq n/3$, while the last two layers are connected by $O(m)$ edges.

computing for every pair of nodes u, v the minimum of their count and U (think of U as a "cap"). Finally, let $\#_{\text{approx-}U}$APSP be the problem of computing a $(1 + 1/U)$-approximation of all pairwise counts (think of keeping the $\log U$ most significant bits of each count).

We obtain the following result for u-$\#_{\leq U}$APSP in directed graphs:

▶ **Theorem 7.** *u-$\#_{\leq U}$APSP (in directed graphs) can be solved in $n^{2+\rho}\text{polylog } U \leq n^{2+\rho+o(1)}$ time.*

Furthermore, for any $U \geq 2$, if u-$\#_{\leq U}$APSP can be solved in $O(n^{2+\rho-\varepsilon})$ time for some $\varepsilon > 0$, then so can u-APSP (with randomization). For any $2 \leq U \leq \tilde{O}(1)$, the converse is true as well.

Thus, we get a conditionally optimal algorithm for u-$\#_{\leq U}$APSP. For $2 \leq U \leq \tilde{O}(1)$, the theorem above gives a fine-grained equivalence between u-$\#_{\leq U}$APSP and u-APSP; in particular, for $U = 2$, the problem corresponds to testing *uniqueness* for the shortest path of each pair. (For large U, however, it is not a fine-grained equivalence since the algorithm for u-$\#_{\leq U}$APSP does not go through Min-Plus product, but rather directly uses fast matrix multiplication.)

Our algorithm from Theorem 7 is based on Zwick's algorithm for u-APSP. We show that one can also modify Seidel's algorithm for u-APSP in undirected graphs to obtain $\tilde{O}(n^\omega)$ time algorithms for u-$\#_{\leq U}$APSP and u-$\#_{\bmod U}$APSP in undirected graphs.

▶ **Theorem 8.** *u-$\#_{\leq U}$APSP and u-$\#_{\bmod U}$APSP in undirected graphs can be solved in $\tilde{O}(n^\omega \log U)$ time.*

Furthermore, we show that u-$\#_{\text{approx-}U}$APSP in undirected graphs can be solved in $O(n^{2.58}\text{polylog } U)$ time, somewhat surprisingly, by a slight modification of our undirected Lex_2-APSP algorithm (despite the apparent dissimilarity between the two problems).

Paper Organization and Techniques. In Section 3, we show the web of reductions around u-APSP, proving Theorem 1, Theorem 2, the hardness of additive $D[u, v]^p$ approximate u-APSP and the hardness of u-$\#_{\leq U}$APSP in Theorem 7.

In Section 4, we give our algorithms for approximating APSP with additive errors, proving Theorem 3. In Section 5, we describe our algorithms for Lex_2-APSP. Due to space limitation, we defer our algorithms for various versions of #APSP to the full paper (including an algorithm for u-$\#_{\leq U}$APSP to complete the proof of Theorem 7, the proof of Theorem 8, and the algorithm for u-$\#_{\text{approx-}U}$APSP). An exception is our near-cubic algorithm for exact u-#APSP (proof of Theorem 6), which is simple and is described in Section 6.

For approximating APSP with additive error, we propose an interesting *two-phase* variant of Zwick's algorithm [34]. Zwick's algorithm computes distance products of $n \times (n/\ell)$ with $(n/\ell) \times n$ matrices for ℓ in a geometric progression. Our idea is to do less during the first phase, computing products of $(n/\ell) \times (n/\ell)$ with $(n/\ell) \times n$ matrices instead. We complete the work during a second phase. The observation is that for the APSP approximation problem, we can afford to perform the distance computation in the first phase *exactly*, but use approximation to speed up the second phase. The resulting approximation algorithm is even simpler than Roditty and Shapira's previous (slower) algorithm [20].

Our Lex_2-APSP algorithm for directed graphs also uses this two-phase approach, but in a more sophisticated way to control the size of the numbers in the rectangular matrix products. A number of interesting new ideas are needed.

To further illustrate the power of this two-phase approach, we also show (in the full version) how the idea can lead to an alternative $\tilde{O}(c_0 n^\omega)$ time algorithm for the standard $[c_0]$-APSP problem for undirected graphs, rederiving Shoshan and Zwick's result [22] in an arguably simpler way. This may be of independent interest (as Shoshan and Zwick's algorithm has complicated details).

Our Lex_2-APSP algorithm for *undirected* graphs uses small dominating sets for high-degree vertices, an idea of Aingworth et al. [1]. Originally, this idea was for developing combinatorial algorithms for *approximate* shortest paths that avoid matrix multiplication. Interestingly, we show that this idea can be combined with (rectangular) matrix multiplication to compute *exact* Lex_2 shortest paths.

2 Preliminaries

The computation model of all algorithms and reductions in this paper is the word-RAM model with $O(\log n)$ bit words.

We let $\mathcal{M}(n_1, n_2, n_3)$ denote the best known running time for multiplying an $n_1 \times n_2$ by an $n_2 \times n_3$ matrix over the integers. We use $\omega(a, b, c)$ to denote the rectangular matrix multiplication exponent, i.e. the smallest real number z such that $\mathcal{M}(n^a, n^b, n^c) \leq O(n^{z+\varepsilon})$ for all $\varepsilon > 0$. In particular, let $\omega = \omega(1, 1, 1)$. It is known that $\omega \in [2, 2.373)$ [2, 27, 16]. The best known bounds for $\omega(a, b, c)$ are in [17].

Let $\mathcal{M}^\star(n_1, n_2, n_3 \mid \ell_1, \ell_2)$ be the time to compute the Min-Plus product of an $n_1 \times n_2$ matrix A with an $n_2 \times n_3$ matrix B, where all finite entries of A are from $[\ell_1]$ and all finite entries of B are from $[\ell_2]$. Let us also denote $\mathcal{M}^\star(n_1, n_2, n_3 \mid \ell) := \mathcal{M}^\star(n_1, n_2, n_3 \mid \ell, \ell)$. It is known [3] that $\mathcal{M}^\star(n_1, n_2, n_3 \mid \ell) \leq \tilde{O}(\ell \cdot \mathcal{M}(n_1, n_2, n_3))$. This algorithm in [3] first replaces each entry e in both matrices A, B by $(n_2 + 1)^e$, then uses fast rectangular matrix multiplication to compute the product of the new matrices A, B. Since each arithmetic operation takes $\tilde{O}(\ell)$ time, the running time follows.

More generally, let $\mathcal{M}^\star(n_1, n_2, n_3 \mid m_1, m_2, m_3 \mid \ell_1, \ell_2)$ be the time to compute m_3 given entries of the Min-Plus product of an $n_1 \times n_2$ matrix A with an $n_2 \times n_3$ matrix B, where A has at most m_1 finite entries, all from $[\ell_1]$, and B has at most m_2 finite entries, all from $[\ell_2]$.

3 Directed APSP and Rectangular Min-Plus with Bounded Weights

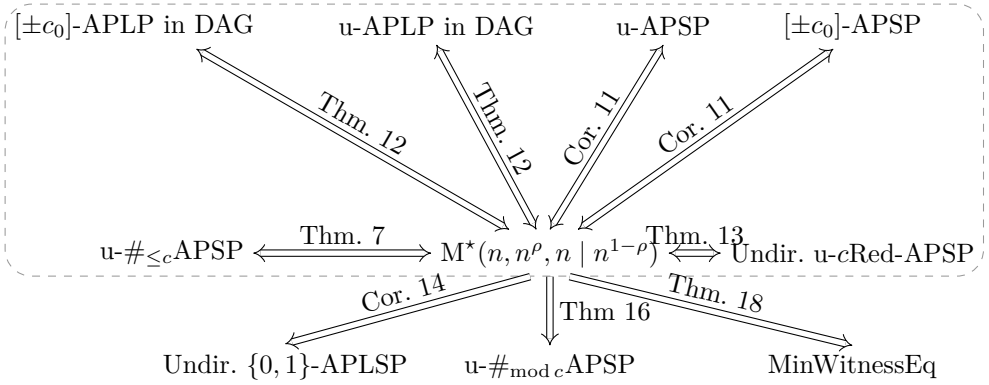


Figure 1 The web of (a subset of) the reductions in this paper. All reductions are $(n^{2+\rho}, n^{2+\rho})$-fine grained reductions, where ρ is such that $\omega(1, \rho, 1) = 1 + 2\rho$. The problems in the bounding box are sub $n^{2+\rho}$-equivalent. Here, $c_0 = \tilde{O}(1)$, and $2 \leq c \leq \tilde{O}(1)$. For the current best bounds on rectangular matrix multiplication [17], ρ is roughly 0.529.

Here we consider the All-Pairs Shortest Paths (APSP) problem in unweighted directed graphs, or more generally in directed graphs with integer weights in $[\pm c_0]$ with $c_0 = \tilde{O}(1)$ and no negative cycles. Zwick [34] showed that this problem in n-node graphs can be solved in time $\tilde{O}(n^{2+\rho})$ time where ρ is such that $\omega(1, \rho, 1) = 1 + 2\rho$. For the current best bounds on rectangular matrix multiplication [17], ρ is roughly 0.529.

Zwick's algorithm can be viewed as a reduction to rectangular Min-Plus matrix multiplication. The algorithm proceeds in stages, for each ℓ from 0 to $\log_{3/2}(n^{1-\rho})$.

In stage ℓ, up to logarithmic factors, one needs to compute the Min-Plus product of two matrices A_ℓ and B_ℓ where A_ℓ has dimensions $n \times n/(3/2)^\ell$ and B_ℓ has dimensions $n/(3/2)^\ell \times n$ and both matrices have entries bounded by $(3/2)^\ell$. Intuitively, this computes the pairwise distances that are roughly $(3/2)^\ell$. After stage $\log_{3/2}(n^{1-\rho})$, the algorithm also runs Dijkstra's algorithm[6] to and from $\tilde{O}(n^\rho)$ nodes S sampled randomly and uses $\tilde{O}(n^{2+\rho})$ extra time to complete the computation of the distances by considering for every $u, v \in V$, $\min_{s \in S}\{D[u, s] + D[s, v]\}$. This can also be viewed as using the brute-force algorithm to compute the Min-Plus products when $(3/2)^\ell \geq n^{1-\rho}$.

The total running time is within logarithmic factors of

$$n^{2+\rho} + \sum_{\ell=0}^{\log_{3/2}(n^{1-\rho})} \mathcal{M}^\star(n, n/(3/2)^\ell, n \mid (3/2)^\ell),$$

where $\mathcal{M}^\star(n_1, n_2, n_3 \mid M)$ is the Min-Plus product running time for matrices with entries in $\{0, \ldots, M\}$ and dimensions $n_1 \times n_2$ by $n_2 \times n_3$. With the known bounds for Min-Plus product, $\mathcal{M}^\star(n, n^\tau, n \mid M) \leq \tilde{O}(Mn^{\omega(1,\tau,1)})$, and the running time of Zwick's algorithm becomes $\tilde{O}(n^{2+\rho} + n^{1-\rho+\omega(1,\rho,1)})$, which is $\tilde{O}(n^{2+\rho})$ when $\omega(1, \rho, 1) = 1 + 2\rho$.

If $\omega = 2$, then ρ is 1/2 and the running time of Zwick's algorithm becomes $\tilde{O}(n^{2.5})$. This running time is a seeming barrier for the APSP problem in directed graphs.

In the full version we prove the following technical theorem which rephrases Zwick's algorithm [34] as a reduction.

▶ **Theorem 9.** *Let ρ be the solution to $\omega(1, \rho, 1) = 1 + 2\rho$. If the Min-Plus product of an $n \times n^\rho$ matrix by an $n^\rho \times n$ matrix where both matrices have integer entries bounded by $n^{1-\rho}$ (denoted as $\mathrm{M}^\star(n, n^\rho, n \mid n^{1-\rho})$) can be computed in $O(n^{2+\rho-\epsilon})$ time for some $\epsilon > 0$, then APSP in directed n node graphs with integer edge weights in $[\pm c_0]$ for $c_0 = \tilde{O}(1)$ can be solved in $O(n^{2+\rho-\epsilon'})$ time for $\epsilon' > 0$.*

If $\omega = 2$, the above theorem statement becomes: If the Min-Plus problem of an $n \times \sqrt{n}$ matrix by a $\sqrt{n} \times n$ matrix where both matrices have integer entries bounded by $\sqrt{n}$ can be computed in $O(n^{2.5-\delta})$ time for some $\delta > 0$, then APSP in directed n node graphs with integer edge weights in $[\pm c_0]$ for $c_0 = \tilde{O}(1)$ can be solved in $O(n^{2.5-\delta'})$ time for $\delta' > 0$.

We will show a reduction in the reverse direction as well, showing that rectangular Min-Plus product with suitably bounded entries can be reduced back to unweighted directed APSP.

▶ **Theorem 10.** *For any fixed $k \in (0, 1)$, $\mathrm{M}^\star(n, n^k, n \mid n^{1-k})$ can be reduced in $O(n^2)$ time to APSP in a directed unweighted graph with $O(n)$ vertices.*

[6] If there are negative weights, one also needs to run single source shortest paths (SSSP) from a node, as in Johnson's algorithm and then reweight the edges so that they are nonnegative. SSSP can be solved in $\tilde{O}((m + n^{1.5}) \log^2(c_0)) = \tilde{O}(n^2)$ time [23].

A consequence of Theorem 9, and the fact that u-APSP is a special case of $[\pm c_0]$-APSP for directed graphs without negative cycles, is the following equivalence.

▶ **Corollary 11.** *Let ρ be such that $\omega(1,\rho,1) = 1+2\rho$. Then u-APSP, $[\pm c_0]$-APSP for directed graphs without negative cycles, and $\mathrm{M}^\star(n, n^\rho, n \mid n^{1-\rho})$ are sub-$n^{2+\rho}$ fine-grained equivalent for $c_0 = \tilde{O}(1)$.*

In particular, if $\omega = 2$, APSP in directed unweighted graphs is sub-$n^{2.5}$ fine-grained equivalent to the Min-Plus problem of an $n \times \sqrt{n}$ matrix by a $\sqrt{n} \times n$ matrix where both entries have integer entries bounded by $\sqrt{n}$.

Proof of Theorem 10. Let A be an $n \times n^k$ matrix and let B be an $n^k \times n$ matrix, both with entries in $\{1, \ldots, n^{1-k}\}$.

We will create a directed graph as follows. Let I be a set of n nodes, which represent the rows of A. Let J be a set of n nodes, which represent the columns of B.

For every $p \in [n^k]$ corresponding to a column of A (or row of B), create a path of $2n^{1-k}+1$ nodes:

$$X(p) := x_{p,n^{1-k}} \to x_{p,n^{1-k}-1} \to \ldots \to x_{p,0} \to y_{p,1} \to y_{p,2} \to \ldots \to y_{p,n^{1-k}}.$$

For every $i \in [n]$ and $p \in [n^k]$, consider $t = A[i,p] \in [n^{1-k}]$. Add an edge from $i \in I$ to $x_{p,t}$. Similarly, for every $j \in [n]$ and $p \in [n^k]$, consider $t' = B[p,j] \in [n^{1-k}]$. Add an edge from $y_{p,t'}$ to $j \in J$.

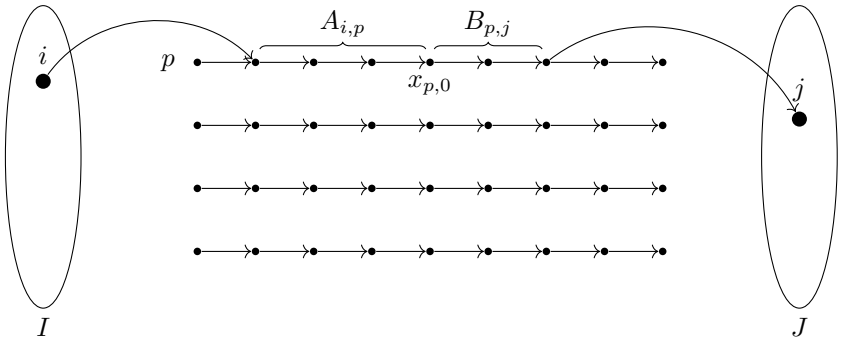


Figure 2 Sketch of the construction in proof of Theorem 10. For each vertex i and path p, we add an edge from i to a vertex on the path p whose distance to the middle point $x_{p,0}$ on the path is $A_{i,p}$. For each path p and vertex j, we add an edge from a vertex on the path whose distance from the middle point $x_{p,0}$ on the path is $B_{p,j}$ to vertex j.

Now, consider some $i \in [n], p \in [n^k], j \in [n]$ and $A[i,p] + B[p,j]$. If we consider the path consisting of $(i, x_{p,A[i,p]})$, $(y_{p,B[p,j]}, j)$ and the subpath of $X(p)$ between $x_{p,A[i,p]}$ and $y_{p,B[p,j]}$, its length is exactly $2 + A[i,p] + B[p,j]$. Also, any path from i to j is of this form. Thus, the shortest path from $i \in I$ to $j \in J$ in the created graph is exactly of length $2 + \min_p\{A[i,p] + B[p,j]\}$, and thus computing APSP in the directed unweighted graph we have created computes the Min-Plus product of A and B.

The number of vertices in the graph is $O(n^k \cdot n^{1-k}) = O(n)$. ◀

One consequence of Corollary 11 is that u-APSP and computing the predecessor matrix in unweighted directed APSP are also sub-$n^{2+\rho}$ fine-grained equivalent. It was known that Zwick's algorithm [34] can compute the predecessor matrix for unweighted directed APSP, which can also be viewed as a sub-$n^{2+\rho}$ time reduction from computing the predecessor

matrix to $\mathrm{M}^\star(n, n^\rho, n \mid n^{1-\rho})$. Also, if we can compute the predecessor matrix for the graph constructed in the above proof, we would know which path $X(p)$ the shortest path from i to j uses, which in turn solves $\mathrm{M}^\star(n, n^\rho, n \mid n^{1-\rho})$. Thus, computing the predecessor matrix for unweighted directed APSP is sub-$n^{2+\rho}$ fine-grained equivalent to $\mathrm{M}^\star(n, n^\rho, n \mid n^{1-\rho})$, and thus also equivalent to u-APSP.

Zwick's algorithm is general enough to apply to some variants of APSP. One example is the All-Pairs Longest Paths (APLP) problem in DAGs. To compute APLP in a DAG, we first negate the weight of every edge, then the problem becomes APSP, on which we can directly apply Zwick's algorithm. Therefore, Zwick's algorithm show reductions from u-APLP and $[\pm c_0]$-APLP in DAGs to $\mathrm{M}^\star(n, n^\rho, n \mid n^{1-\rho})$.

Perhaps more surprisingly, the other direction of the reduction also holds. Therefore, APLP in DAG and APSP in graphs with weights bounded by $\tilde{O}(1)$ are sub-$O(n^{2+\rho})$ equivalent.

▶ **Theorem 12.** *Let ρ be such that $\omega(1, \rho, 1) = 1 + 2\rho$. Then u-APLP in DAGs, $[\pm c_0]$-APLP in DAGs and $\mathrm{M}^\star(n, n^\rho, n \mid n^{1-\rho})$ are sub-$n^{2+\rho}$ fine-grained equivalent.*

The proof of Theorem 12 follows from the same approach and is deferred to the full version.

All problems shown equivalent to u-APSP above are problems on directed graphs. One natural question is that whether some problems on undirected graphs are also in this equivalence class, or whether we can show some undirected graph problems require $n^{2+\rho-o(1)}$ time if we assume problems in this equivalence class also require $n^{2+\rho-o(1)}$ time. To answer these questions, we first consider the u-cRed-APSP problem.

▶ **Theorem 13.** *Let ρ be such that $\omega(1, \rho, 1) = 1 + 2\rho$. u-cRed-APSP for $2 \le c = \tilde{O}(1)$ and $\mathrm{M}^\star(n, n^\rho, n \mid n^{1-\rho})$ are sub-$n^{2+\rho}$ fine-grained equivalent.*

The proof of Theorem 13 uses a similar graph construction and is in the full version.

By slightly modifying the proof of Theorem 13, we can show conditional hardness for APLSP on undirected graphs where the edge weights are in $\{0, 1\}$. The proof is in the full version.

▶ **Corollary 14.** *Let ρ be such that $\omega(1, \rho, 1) = 1 + 2\rho$. Suppose $\mathrm{M}^\star(n, n^\rho, n \mid n^{1-\rho})$ requires $n^{2+\rho-o(1)}$ time. Then APLSP on undirected graphs where the edge weights can be $\{0, 1\}$ also requires $n^{2+\rho-o(1)}$ time.*

Using similar ideas we also show hardness for *Vertex-Weighted APSP* in undirected graphs, where the vertex weights may be large. (The current best algorithms for Vertex-Weighted APSP for directed graphs [6, 32] had running time about $O(n^{2.85})$; the bound is $\tilde{O}(n^{11/4})$ if $\omega = 2$. No better algorithms were known in the undirected graphs – which our conditional lower bound attempts to explain.) The proof is in the full version.

▶ **Corollary 15.** *Let ρ be such that $\omega(1, \rho, 1) = 1 + 2\rho$. Suppose $\mathrm{M}^\star(n, n^\rho, n \mid n^{1-\rho})$ requires $n^{2+\rho-o(1)}$ time. Then vertex-weighted APSP on undirected graphs where the vertex weights are in $[O(n^{1-\rho})]$ also requires $n^{2+\rho-o(1)}$ time.*

The conditional hardness for u-$\#_{\mathrm{mod}\, U}$APSP and u-$\#_{\le U}$APSP for any $U \ge 2$ can be proved by combining our graph construction with randomized techniques for a unique variant of Min-Plus product; see the proof in the full version.

▶ **Theorem 16.** *Let ρ be such that $\omega(1, \rho, 1) = 1 + 2\rho$. Suppose $\mathrm{M}^\star(n, n^\rho, n \mid n^{1-\rho})$ requires $n^{2+\rho-o(1)}$ time (with randomization). Then u-$\#_{\mathrm{mod}\, U}$APSP and u-$\#_{\le U}$APSP for any $U \ge 2$ requires $n^{2+\rho-o(1)}$ time.*

In Section 4, we will give an algorithm for approximating APSP with sublinear additive errors. Using the same technique as our reductions from Rectangular Min-Plus product to APSP problems, we can show a conditional lower bound for this problem.

▶ **Theorem 17.** *Given a directed unweighted graph $G = (V, E)$ with n vertices and a function $f > 0$ where $\frac{\ell}{f(\ell)}$ is nondecreasing. Suppose we can approximate the shortest-path distance $D[u, v]$ with additive error $f(D[u, v])$, for all $u, v \in V$ in $T(n)$ time, then $\max_{1\le\ell\le n} \mathcal{M}^\star\left(n, n/\ell, n \mid \frac{\ell}{f(\ell)}\right) \le O(T(n))$.*

Proof. Fix any $1 \le \ell \le n$. First, note that $\mathcal{M}^\star\left(n, n/\ell, n \mid \frac{\ell}{f(\ell)}\right) = \Theta\left(\mathcal{M}^\star\left(n, n/\ell, n \mid \frac{\ell}{Cf(\ell)}\right)\right)$ for any constant C. Here, we take $C = 12$ to be a large enough constant.

Suppose we are given an $n \times n/\ell$ matrix A and an $n/\ell \times n$ matrix B, whose entries are positive integers bounded by $\frac{\ell}{12f(\ell)}$, and we want to compute their Min-Plus product $A \star B$. We use a similar reduction as the one in the proof of Theorem 10, but stretching the length of the middle paths. Specifically, we create vertex set I of size n, vertex set J of size n, and n/ℓ paths of the form $X(p) := x_{p,\frac{\ell}{3f(\ell)}} \rightsquigarrow \cdots \rightsquigarrow x_{p,0} \rightsquigarrow y_{p,0} \rightsquigarrow \cdots \rightsquigarrow y_{p,\frac{\ell}{3f(\ell)}}$. From $x_{p,i}$ to $x_{p,i-1}$ and $y_{p,j}$ to $y_{p,j+1}$, we embed paths of length $6f(\ell)$; from $x_{p,0}$ to $y_{p,0}$, we embed a path of length $\ell - 2$. Similar to previous reductions, for every $i \in [n] = I$ and $p \in [n/\ell]$, we add an edge from i to $x_{p,A[i,p]}$; for every $j \in [n] = J$ and $p \in [n/\ell]$, we add an edge from j to $y_{p,B[p,j]}$. Then the distance from $i \in I$ to $j \in J$ in this graph equals $\ell + 6f(\ell)(A \star B)[i, j]$.

Since $0 \le (A \star B)[i, j] \le \frac{\ell}{6f(\ell)}$, we must have $\ell \le \ell + 6f(\ell)(A \star B)[i, j] \le 2\ell$. Since $\frac{\ell}{f(\ell)}$ is nondecreasing, we must have $f(t\ell) \le tf(\ell)$ for any $t \ge 1$, and thus $f(\ell + 6f(\ell)(A \star B)[i, j]) \le 2f(\ell)$. Therefore, an $f(\ell + 6f(\ell)(A \star B)[i, j])$-additive approximation of APSP can determine that the distance from $i \in I$ to $j \in J$ is in $\ell + 6f(\ell)(A \star B)[i, j] \pm 2f(\ell)$, from which we can compute $(A \star B)[i, j]$ easily since $(A \star B)[i, j]$ must be an integer. ◀

Finally, we give a reduction from u-APSP to Min Witness Equality, where we are given $n \times n$ integer matrices A and B, and are required to compute $\min\{k \in [n] : A[i, k] = B[k, j]\}$ for every pair of (i, j). Reductions from u-APSP to matrix product problems are considered by Lincoln et al. [18], where they show reductions from u-APSP to the All-Edges Monochromatic Triangle problem and $(\min, \max)$-product problem, but their techniques do not seem to apply to Min Witness Equality.

The proof of the following theorem is deferred to the full version.

▶ **Theorem 18.** *Let ρ be such that $\omega(1, \rho, 1) = 1 + 2\rho$. Suppose $\mathrm{M}^\star(n, n^\rho, n \mid n^{1-\rho})$ requires $n^{2+\rho-o(1)}$ time. Then Min Witness Equality requires $n^{2+\rho-o(1)}$ time.*

4 Additive Approximation Algorithms for APSP

In this section, we give an algorithm for approximate APSP with additive errors in directed unweighted graphs, to match the lower bound that we have just proved in Theorem 17 (ignoring logarithmic factors). Namely, our algorithm achieves running time $\tilde{O}(\max_\ell \mathcal{M}^\star(n, n/\ell, n \mid \ell^{1-p}))$, which improves Roditty and Shapira's previous algorithm [20] with running time $\tilde{O}(\max_\ell \min\{n^3/\ell, \mathcal{M}^\star(n, n/\ell^{1-p}, n \mid \ell^{1-p})\})$.

Let $D[u, v]$ denote the shortest-path distance from u to v.

Overview. The new algorithm is a variation of Zwick's exact u-APSP algorithm [34], and is actually simpler than Roditty and Shapira's algorithm. The idea is to compute as many as the shortest-path distances *exactly* as we can in $\tilde{O}(n^\omega)$ time in an initial phase. In the second phase, we apply rectangular matrix multiplication to submatrices computed from the first phase, where entries are approximated by rounding and rescaling.

Preliminaries. For every ℓ that is a power of $3/2$, let $R_\ell \subseteq V$ be a subset of $\tilde{O}(n/\ell)$ vertices that hits all shortest paths of length $\ell/2$ [34]. (For example, a random sample works with high probability.) We may assume that $R_{(3/2)^i} \supseteq R_{(3/2)^{i+1}}$ (because otherwise, we can add $R_{(3/2)^j}$ to $R_{(3/2)^i}$ for all $j > i$ and the size bound would still hold). For subsets $S_1, S_2 \subseteq V$, let $D(S_1, S_2)$ denote the submatrix of D containing the entries for $(u, v) \in S_1 \times S_2$.

Phase 1. We first solve the following subproblem: compute $D[u, v]$ (exactly) for all $(u, v) \in R_\ell \times V$ with $D[u, v] \le \ell$, and similarly for all $(u, v) \in V \times R_\ell$ with $D[u, v] \le \ell$.

Suppose we have already computed $D[u, v]$ for all $(u, v) \in R_{2\ell/3} \times V$ with $D[u, v] \le 2\ell/3$, and similarly for all $(u, v) \in V \times R_{2\ell/3}$ with $D[u, v] \le 2\ell/3$.

We take the Min-Plus product $D(R_\ell, R_{2\ell/3}) \star D(R_{2\ell/3}, V)$. For each $(u, v) \in R_\ell \times V$, if its output entry is smaller than the current value of $D[u, v]$, we reset $D[u, v]$ to the smaller value. Similarly, we take the Min-Plus product $D(V, R_{2\ell/3}) \star D(R_{2\ell/3}, R_\ell)$. For each $(u, v) \in V \times R_\ell$, if its output entry is smaller than the current value of $D[u, v]$, we reset $D[u, v]$ to the smaller value. We reset all entries greater than ℓ to ∞.

To justify correctness, observe that for any shortest path π of length between $2\ell/3$ and ℓ, the middle $(2\ell/3)/2 = \ell/3$ vertices must contain a vertex of $R_{2\ell/3}$, which splits π into two subpaths each of length at most $\ell/2 + \ell/6 \le 2\ell/3$.

We do the above for all ℓ's that are powers of $3/2$. The total cost is

$$\tilde{O}\left(\max_\ell \mathcal{M}^\star(n/\ell, n/\ell, n \mid \ell)\right) \le \tilde{O}\left(\max_\ell \ell \cdot \mathcal{M}^\star(n/\ell, n/\ell, n)\right) \le \tilde{O}\left(\max_\ell \ell^2 (n/\ell)^\omega\right) = \tilde{O}(n^\omega).$$

Phase 2. Next we approximate all shortest-path distances $D[u, v]$ where $D[u, v]$ is between $2\ell/3$ and ℓ, with additive error $O(f(\ell))$ for a given function f, as follows:

We compute the Min-Plus product $D(V, R_{2\ell/3}) \star D(R_{2\ell/3}, V)$, keeping only entries bounded by $O(\ell)$. As we allow additive error $O(f(\ell))$, we round entries to multiples of $f(\ell)$. This takes $\tilde{O}(\mathcal{M}^\star(n, n/\ell, n \mid \frac{\ell}{f(\ell)}))$ time.

To justify correctness, observe as before that in any shortest path π of length between $2\ell/3$ and ℓ, some vertex in $R_{2\ell/3}$ splits the path into two subpaths of length at most $2\ell/3$.

We repeat for all ℓ's that are powers of $3/2$. The total cost is $\tilde{O}\left(\max_\ell \mathcal{M}^\star(n, n/\ell, n \mid \frac{\ell}{f(\ell)})\right)$.

Standard techniques for generating witnesses for matrix products can be applied to recover the shortest paths (e.g., see [11, 34]).

▶ **Theorem 19.** *Given a directed unweighted graph $G = (V, E)$ with n vertices and a function f where $\frac{\ell}{f(\ell)}$ is nondecreasing, we can approximate the shortest-path distance $D[u, v]$ with additive error $O(f(D[u, v]))$ for all $u, v \in V$, in $\tilde{O}\left(\max_\ell \mathcal{M}^\star(n, n/\ell, n \mid \frac{\ell}{f(\ell)})\right)$ time.*

▶ Remark. For $f(\ell) = \ell^p$, we can upper-bound the running time by

$$\begin{aligned}\tilde{O}\left(\max_\ell \mathcal{M}^\star(n, n/\ell, n \mid \ell^{1-p})\right) &\le \tilde{O}\left(L^{1-p} \cdot \mathcal{M}(n, n/L, n) + n^3/L\right) \\ &\le \tilde{O}\left(L^{1-p}(n^{2+o(1)} + n^\omega / L^{(\omega-2)/(1-\alpha)}) + n^3/L\right)\end{aligned}$$

for any choice of L, where α is the rectangular matrix multiplication exponent (satisfying $\omega(1,1,\alpha)=2$). For example, we can set $L=n^{3-\omega}$, and for $p>1-\min\{\frac{\omega-2}{1-\alpha},\frac{\omega-2}{3-\omega}\}$, get optimal $\tilde{O}(n^\omega)$ running time. In fact, with the current rectangular matrix multiplication bounds we get $\tilde{O}(n^{2.373})$ time for $p\geq 0.415\geq(\omega(1,0.373,1)-2\cdot 0.373-1)/(1-0.373)$. Roditty and Shapira [20] specifically asked whether there exists $p<1$ for which $\tilde{O}(n^\omega)$ time is possible; we have thus answered their question affirmatively if $\omega>2$.

▶ Remark. For directed graphs with weights from $[c_0]$, the running time is

$$\tilde{O}\left(c_0 n^\omega+\max_\ell \mathcal{M}^\star(n,n/\ell,n\mid c_0\tfrac{\ell}{f(\ell)})\right).$$

5 Algorithms for All-Pairs Lightest Shortest Paths

In this section, we describe algorithms for the following problem, which includes both All-Pairs Lightest Shortest Paths (APLSP) and Shortest Lightest Paths (APSLP) as special cases:

▶ **Problem 20. (Lex_2-APSP)** *We are given a graph $G=(V,E)$ with n vertices, where each edge $(u,v)\in E$ has a "primary" weight $w_1(u,v)$ and a "secondary" weight $w_2(u,v)$. For every pair of vertices $u,v\in V$, we want to find a path π from u to v that minimizes $(\sum_{e\in\pi}w_1(e),\sum_{e\in\pi}w_2(e))$ lexicographically.*

Let $D[u,v]$ be the lexicographical minimum of $(\sum_{e\in\pi}w_1(e),\sum_{e\in\pi}w_2(e))$. Let $D_1[u,v]$ be the minimum of $\sum_{e\in\pi}w_1(e)$ (the shortest-path distance) and let $D_2[u,v]$ be the second coordinate of $D[u,v]$. APLSP corresponds to the case when all secondary edge weights are 1, whereas APSLP corresponds to the case when all primary edge weights are 1.

The following lemma, which will be important in the analysis of our Lex_2-APSP algorithm, bounds the complexity of Min-Plus product of an $n_1\times n_2$ matrix A and an $n_2\times n_3$ matrix B in the case when the finite entries of A come from a small range $[\ell_1]$ (but the finite entries of B may come from a large range $[\ell_2]$). The bound can be made sensitive to the number m_2 of finite entries of B and the number m_3 of output entries we want. The lemma is a variant of [6, Theorem 3.5] (the basic approach originates from Matoušek's dominance algorithm [19], but this variant requires some extra ideas). It also generalizes and improves (using rectangular matrix multiplication) Theorem 1.2 in [30].

▶ **Lemma 21.** $\mathcal{M}^\star(n_1,n_2,n_3\mid\ell_1,\ell_2)=\tilde{O}\left(\min_t(\mathcal{M}^\star(n_1,n_2,n_2n_3/t\mid\ell_1)+tn_1n_3)\right)$. *More generally,* $\mathcal{M}^\star(n_1,n_2,n_3\mid m_1,m_2,m_3\mid\ell_1,\ell_2)=\tilde{O}\left(\min_t(\mathcal{M}^\star(n_1,n_2,m_2/t\mid\ell_1)+tm_3)\right)$.

Proof. Divide each column of B into groups of t entries by rank: the first group contains the t smallest elements, the second group contains the next t smallest, etc. (ties in ranks can be broken arbitrarily). Each column may have at most t leftover entries. The total number of groups is at most m_2/t.

For each $i\in[n_1]$ and $j'\in[m_2/t]$, let $C[i,j']$ be true iff there exists $k\in[n_2]$ such that $A[i,k]<\infty$ and group j' contains an element with row index k. Computing C reduces to taking a Boolean matrix product and has cost $O(\mathcal{M}(n_1,n_2,m_2/t))$.

For each $i\in[n_1]$ and $j'\in[m_2/t]$, suppose that group j' is part of column j and the maximum element in group j' is x; let $\widehat{C}[i,j']=\min_{k:B[k,j]\in[x,x+\ell_1]}(A[i,k]+B[k,j])$. Since entries in A are from the range $[\ell_1]\cup\{\infty\}$, and we only keep a size ℓ_1+1 range of values for matrix B, computing $\widehat{C}$ reduces to taking a Min-Plus product with entries in $[\ell_1]$ (after shifting) and has cost $O(\mathcal{M}^\star(n_1,n_2,m_2/t\mid\ell_1))$.

To compute the output entry at each of the m_3 positions (i, j), we find the group j' in column j with the smallest rank such that $C[i, j']$ is true. Let x be the maximum element in group j'. The answer $\min_k(A[i, k] + B[k, j])$ is at most $x + \ell_1$. Thus, the answer is defined by an index k that (i) corresponds to an element in group j', or (ii) corresponds to a leftover element in column j, or (iii) has $B[k, j] \in [x, x + \ell_1]$. Cases (i) and (ii) can be handled by linear search in $O(t)$ time; case (iii) is handled by looking up $\widehat{C}[i, j']$. The total time to compute m_3 output entries is $O(tm_3)$. ◀

5.1 $[c_0]$-Lex$_2$-APSP

Let $c_0 = \tilde{O}(1)$. For directed graphs, Zwick [33] presented a variant of his u-APSP algorithm that solves $[c_0]$-Lex$_2$-APSP (and thus $[c_0]$-APLSP and $[c_0]$-ALPSP) in time $\tilde{O}(\max_\ell \mathcal{M}^\star(n, n/\ell, n \mid \ell^2)) \leq \tilde{O}(\min_L(L^2\mathcal{M}(n, n/L, n) + n^3/L))$. This is $O(n^{2.724})$ by the current bounds on rectangular matrix multiplication [17] (and is $\tilde{O}(n^{8/3})$ if $\omega = 2$).

Chan [6] gave a faster algorithm for $([c_0] - \{0\})$-Lex$_2$-APSP (and in fact a special case of Vertex-Weighted APSP that includes $([c_0] - \{0\})$-Lex$_k$-APSP for an arbitrary constant k) in time $\tilde{O}(n^{(3+\omega)/2})$, which is $O(n^{2.687})$ by the current matrix multiplication exponent (and is $\tilde{O}(n^{2.5})$ if $\omega = 2$). Zwick's algorithm works even when zero primary weights are allowed, but Chan's algorithm does not (part of the difficulty is that the secondary distance of a path may be much larger than the primary distance). A more general version of Chan's algorithm [6] can handle zero primary weights (and $[c_0]$-Lex$_k$-APSP for constant k) but has a worse time bound of $\tilde{O}(n^{(9+\omega)/4})$, which can be slightly reduced using rectangular matrix mutiplication [32].

We describe an $O(n^{2.6581})$-time algorithm to solve $[c_0]$-Lex$_2$-APSP for directed graphs, which can handle zero weights and is faster than Zwick's $O(n^{2.724})$-time algorithm; it is also slightly faster than Chan's algorithm. The algorithm uses rectangular matrix multiplication (without which the running time would be $\tilde{O}(n^{(\omega+3)/2})$). It should be noted that Chan's previous algorithm can't be easily sped up using rectangular matrix multiplication, besides being inapplicable when there are zero primary weights.

Overview. The new algorithm can be viewed as an interesting variant of Zwick's u-APSP algorithm [34]. Zwick's algorithm uses rectangular Min-Plus products of dimensions around $n \times n/\ell$ and $n/\ell \times n$, in geometrically increasing parameter ℓ. Our algorithm proceeds in two phases. In both phases, we use the rectangular products of dimensions around $n/\ell \times n/\ell$ and $n/\ell \times n$. In the first phase, we consider ℓ in increasing order; in the second, we consider ℓ in decreasing order. In these Min-Plus products, entries of the first matrix in each product come from a small range; this enables us to use Lemma 21.

Preliminaries. Let L be a parameter to be set later. Let $\lambda[u, v]$ denote the length of a lexicographical shortest path between u and v. In this section, the *length* of a path refers to the number of edges in the path.

For every ℓ that is a power of $3/2$, as in Section 4, let $R_\ell \subseteq V$ be a subset of $\tilde{O}(n/\ell)$ vertices that hits all shortest paths of length $\ell/2$ [33, 34]. We may assume that $R_{(3/2)^i} \supseteq R_{(3/2)^{i+1}}$ (as before). Set $R_1 = V$.

For $S_1, S_2 \subseteq V$, let $D(S_1, S_2)$ denote the submatrix of D containing the entries for $(u, v) \in S_1 \times S_2$.

Phase 1. We first solve the following subproblem for a given $\ell \leq L$: compute $D[u,v]$ for all $(u,v) \in R_\ell \times V$ with $\lambda[u,v] \leq \ell$, and similarly for all $(u,v) \in V \times R_\ell$ with $\lambda[u,v] \leq \ell$. (We don't know $\lambda[u,v]$ in advance. More precisely, if $\lambda[u,v] \leq \ell$, the computed value should be correct; otherwise, the computed value is only guaranteed to be an upper bound.)

Suppose we have already computed $D[u,v]$ for all $(u,v) \in R_{2\ell/3} \times V$ with $\lambda[u,v] \leq 2\ell/3$, and similarly for all $(u,v) \in V \times R_{2\ell/3}$ with $\lambda[u,v] \leq 2\ell/3$.

We take the Min-Plus product $D(R_\ell, R_{2\ell/3}) \star D(R_{2\ell/3}, V)$ (where elements are compared lexicographically). For each $(u,v) \in R_\ell \times V$, if its output entry is smaller than the current value of $D[u,v]$, we reset $D[u,v]$ to the smaller value. Similarly, we take the Min-Plus product $D(V, R_{2\ell/3}) \star D(R_{2\ell/3}, R_\ell)$. For each $(u,v) \in V \times R_\ell$, if its output entry is smaller than the current value of $D[u,v]$, we reset $D[u,v]$ to the smaller value. We reset all entries greater than $c_0\ell$ to ∞.

To justify correctness, observe that for any shortest path π of length between $2\ell/3$ and ℓ, the middle $(2\ell/3)/2 = \ell/3$ vertices must contain a vertex of $R_{2\ell/3}$, which splits π into two subpaths each of length at most $\ell/2 + \ell/6 \leq 2\ell/3$.

To take the product, we map each entry $D[u,v]$ of $D(R_{2\ell/3}, V)$ to a number $D_1[u,v] \cdot c_0\ell + D_2[u,v] \in [\tilde{O}(\ell^2)]$. It is more efficient to break the product into ℓ separate products, by putting entries of $D(R_\ell, R_{2\ell/3})$ with a common D_1 value into one matrix. Then after shifting, the finite entries of each such matrix are in $[\tilde{O}(\ell)]$. (The entries of $D(R_{2\ell/3}, V)$ are still in $[\tilde{O}(\ell^2)]$.) Hence, the computation takes time $\tilde{O}(\ell \cdot \mathcal{M}^\star(n/\ell, n/\ell, n \mid \ell, \ell^2))$.

We do the above for all $\ell \leq L$ that are powers of $3/2$ (in increasing order).

Phase 2. Next we solve the following subproblem for a given $\ell \leq L$: compute $D[u,v]$ for all $(u,v) \in R_{2\ell/3} \times V$ with $\lambda[u,v] \leq L$.

Suppose we have already computed $D[u,v]$ for all $(u,v) \in R_\ell \times V$ with $\lambda[u,v] \leq L$.

We take the Min-Plus product $D(R_{2\ell/3}, R_\ell) \star D(R_\ell, V)$, keeping only entries bounded by $\tilde{O}(\ell)$ in the first matrix and $\tilde{O}(L)$ in the second matrix. For each $(u,v) \in V \times R_\ell$, if its output entry is smaller than the current value of $D[u,v]$, we reset $D[u,v]$ to the smaller value.

To justify correctness, recall that for $(u,v) \in R_{2\ell/3} \times V$, if $\lambda[u,v] < 2\ell/3$, then $D[u,v]$ is already computed in Phase 1. On the other hand, in any shortest path π of length between $2\ell/3$ and L, the first $\ell/2$ vertices of the path must contain a vertex of R_ℓ.

To take the product, we map each entry $D[u,v]$ of $D(R_{2\ell/3}, V)$ to a number $D_1[u,v] \cdot c_0 L + D_2[u,v] \in [\tilde{O}(\ell L)]$. As before, it is better to perform ℓ separate products, by putting entries of $D(R_{2\ell/3}, R_\ell)$ with a common D_1 value into one matrix. Then after shifting, the finite entries of each such matrix are in $[\tilde{O}(\ell)]$. (The entries of $D(R_{2\ell/3}, V)$ are still in $[\tilde{O}(\ell L)]$.) Hence, the computation takes time $\tilde{O}(\ell \cdot \mathcal{M}^\star(n/\ell, n/\ell, n \mid \ell, \ell L))$.

We do the above for all $\ell \leq L$ that are powers of $3/2$ (in decreasing order).

Last step. By the end of Phase 2 (when ℓ reaches 1), we have computed $D[u,v]$ for all (u,v) with $\lambda[u,v] \leq L$. To finish, we compute $D[u,v]$ for all (u,v) with $\lambda[u,v] > L$, as follows:

We run Dijkstra's algorithm $O(|R_L|)$ times to compute $D[u,v]$ for all $(u,v) \in R_L \times V$ and for all $(u,v) \in V \times R_L$. This takes $O(|R_L|n^2) = \tilde{O}(n^3/L)$ time. We then compute $D(V, R_L) \star D(R_L, V)$ by brute force in $O(|R_L|n^2) = \tilde{O}(n^3/L)$ time.

Correctness follows since every shortest path of length more than L must pass through a vertex in R_L.

As before, standard techniques for generating witnesses for matrix products can be applied to recover the shortest paths [11, 34].

Total time. The cost of Phase 2 dominates the cost of Phase 1. By Lemma 21, the total cost is

$$\tilde{O}\left(\max_{\ell \le L} \ell \cdot \mathcal{M}^\star(n/\ell, n/\ell, n \mid \ell, \ell L) \,+\, n^3/L\right)$$
$$\leq \quad \tilde{O}\left(\max_{\ell \le L} \ell \cdot \min_t \left(\mathcal{M}^\star(n/\ell, n/\ell, n^2/(\ell t) \mid \ell) + t n^2/\ell\right) \,+\, n^3/L\right).$$

We set $t = n/L$ and obtain

$$\tilde{O}(\max_{\ell \le L} \ell^2 \cdot \mathcal{M}(n/\ell, n/\ell, Ln/\ell) \,+\, n^3/L).$$

Intuitively, the maximum occurs when $\ell = 1$, and so we should choose L to minimize $\tilde{O}(\mathcal{M}(n, n, Ln) + n^3/L)$. With the current bounds on rectangular matrix multiplication [17], we choose $L = n^{0.342}$ and get running time $O(n^{2.6581})$. (Formally, we can verify this time bound using the convexity of the function $2x + \omega(1-x, 1-x, 1.342-x)$.)

▶ **Theorem 22.** *$[c_0]$-Lex$_2$-APSP (and thus $[c_0]$-APLSP and $[c_0]$-APSLP) can be solved in $O(n^{2.6581})$ time for any $c_0 = \tilde{O}(1)$.*

Remarks. Without rectangular matrix multiplication, the above still gives a time bound of $\tilde{O}(Ln^\omega + n^3/L)$, yielding $\tilde{O}(n^{(3+\omega)/2})$.

The same algorithm works even with negative weights (i.e., for $[\pm c_0]$-Lex$_2$-APSP), like Zwick's previous algorithm [33], assuming no negative cycles.

In the full version, we describe an alternative algorithm that has the same running time, though it does not allow zero primary edge weights (or negative weights).

5.2 Undirected $([c_0] - \{0\})$-Lex$_2$-APSP

A natural question is whether APLSP or APSLP is easier for undirected graphs. We now describe a faster $O(n^{2.58})$-time algorithm for $([c_0] - \{0\})$-Lex$_2$-APSP for undirected graphs. Zero primary weights are not allowed, but zero secondary weights are. (In particular, the algorithm can solve $[c_0]$-APSLP, when all primary weights are 1.)

Overview. We follow an idea of Aingworth et al. [1], to divide into two cases: when the source vertex has high degree or low degree. For high-degree vertices, there exists a small dominating set, and so these vertices can be covered by a small number of "clusters"; sources in the same cluster are close together, and so distances from one fixed source give us good approximation to distances from other sources in the same cluster, by the triangle inequality (since the graph is undirected). On the other hand, for low-degree vertices, the relevant subgraph is sparse, which enables faster algorithms. Originally, Aingworth et al.'s approach was intended for the design of approximation algorithms (with $O(1)$ additive error for unweighted graphs). We will adapt it to find *exact* shortest paths. (Chan [7] previously had also applied Aingworth et al.'s approach to exact APSP, but the goal there was in logarithmic-factor speedup, which was quite different.) In order to handle the high-degree case for Lex$_2$-APSP, we need further ideas to use approximate primary shortest-path distances to compute exact lexicographical shortest-path distances; in particular, we will need Min-Plus products on secondary distances (as revealed in the proof of Lemma 23 below). The combination of Aingworth et al.'s approach with matrix multiplication appears new, and interesting in our opinion.

Preliminaries. We first compute $D_1[u,v]$ for all (u,v) by running a known $[c_0]$-APSP algorithm on the primary distances in $O(n^\omega)$ time [3, 21].

Assume that we have already computed $D[u,v]$ for all (u,v) with $D_1[u,v] \le 2\ell/3$ for a given ℓ. We want to compute $D[u,v]$ for all (u,v) with $D_1[u,v] \le \ell$.

Define $D_2^{(\ell)}[u,v] = D_2[u,v]$ if $D_1[u,v] = \ell$, and $D_2^{(\ell)}[u,v] = \infty$ otherwise. For subsets $S_1, S_2 \subseteq V$, let $D_2^{(\ell)}(S_1,S_2)$ denote the submatrix of $D_2^{(\ell)}$ containing the entries for $(u,v) \in S_1 \times S_2$.

▶ **Lemma 23.** *Let $G=(V,E)$ be an undirected graph with edge weights in $[c_0]-\{0\}$. Assume that we have already computed $D[u,v]$ for all (u,v) with $D_1[u,v] \le 2\ell/3$. Given a set S of vertices that are within primary distance $c = \tilde{O}(1)$ from each other, we can compute $D[u,v]$ for all $u \in S$ and $v \in V$ with $D_1[u,v] \le \ell$ in $O(\mathcal{M}^\star(|S|, n/\ell, n \mid \ell))$ total time.*

Proof. Fix $s \in S$. Let $V_i = \{v \in V : D_1[s,v] \in i \pm c\}$. Note that $\sum_i |V_i| = \tilde{O}(n)$. Also note that if $u \in S$ and $D_1[u,v] = i$, then we must have $v \in V_i$ (by the triangle inequality, because the graph is undirected).

Pick an index $m \in [0.4\ell, 0.6\ell]$ with $|V_{m-c_0} \cup \cdots \cup V_m| = \tilde{O}(n/\ell)$.

For $i \le m$, we have already computed $D_2^{(i)}(S,V_i)$.

For $i = m+1, \ldots, \ell$, we will compute $D_2^{(i)}(S,V_i)$ as follows: For each $\Delta \in [c_0]$, we take the Min-Plus product $D_2^{(m-\Delta)}(S,V_{m-\Delta}) \star D_2^{(i-m+\Delta)}(V_{m-\Delta},V_i)$. Note that $D_2^{(i-m+\Delta)}(V_{m-\Delta},V_i)$ is already known, since $i-m+\Delta < 2\ell/3$. We take the minimum over all $\Delta \in [c_0]$ for those $(u,v) \in S \times V_i$ with $D_1[u,v] = i$.

Instead of doing the product individually for each i, it is more efficient to combine all the matrices $D_2^{(i-m+\Delta)}(V_{m-\Delta},V_i)$ over all $i > m$. This gives a single matrix (per Δ) with $|V_{m-\Delta}| = \tilde{O}(n/\ell)$ rows and $\sum_{i>m} |V_i| = \tilde{O}(n)$ columns. So, the entire product can be computed in $O(\mathcal{M}^\star(|S|, n/\ell, n \mid \ell))$ time. ◀

Let L be a parameter to be set later. Let V_{high} be the set of all vertices of degree more than n/L, and V_{low} be the set of all vertices of degree at most n/L.

Phase 1. We will first compute $D[u,v]$ for all $u \in V_{\text{high}}$ and $v \in V$ with $D_1[u,v] \le \ell$, as follows:

Let $X \subseteq V$ be a dominating set for V_{high} of size $\tilde{O}(L)$, such that every vertex in V_{high} is in the (closed) neighborhood of some vertex in X. Such a dominating set can be constructed (for example, by the standard greedy algorithm) in $\tilde{O}(n^2)$ time [1].

Let $X = \{x_1, x_2, \ldots, x_{\tilde{O}(L)}\}$. For each $x_i \in X$, we divide $N(x_i) \setminus \left(\bigcup_{j<i} N(x_j)\right)$ – its neighborhood excluding previous neighborhoods – into groups of size $O(n/L)$. The total number of groups is $\tilde{O}(L)$, and the groups cover all vertices in V_{high}. For each such group, we apply Lemma 23 (with $c = 2c_0$). The total time is $\tilde{O}(L \cdot \mathcal{M}^\star(n/L, n/\ell, n \mid \ell))$.

Phase 2. Next, for each $u \in V_{\text{low}}$, we will compute $D[u,v]$ for all $v \in V$ with $D_1[u,v] \le \ell$, as follows:

Define a graph G_u containing all edges (x,y) with $x \in V_{\text{low}}$ or $y \in V_{\text{low}}$; for each $z \in V_{\text{high}}$, we add an extra edge (u,z) with weight $D[u,z]$, which has been computed in Phase 1. Then the lexicographical shortest-path distance from u to v in G_u matches the lexicographical shortest-path distance in G, because if $\langle u_1, \ldots, u_k\rangle$ is a lexicographical shortest path in G with $u_1 = u$, and i is the largest index with $u_i \in V_{\text{high}}$ (set $i=1$ if none exists), then $\langle u_1, u_i, \ldots, u_k\rangle$ is a path in G_u. We run Dijkstra's algorithm on G_u from the source u. Since G_u has $O(n^2/L)$ edges, this takes $\tilde{O}(n^2/L)$ time per u. The total over all u is $\tilde{O}(n^3/L)$.

As before, standard techniques for generating witnesses for matrix products can be applied to recover the shortest paths [11, 34].

Total time. We do the above for all ℓ's that are powers of 3/2. The overall cost is

$$\begin{aligned}
&\tilde{O}\left(\max_{\ell} L\cdot\mathcal{M}^{\star}(n/L,n/\ell,n\mid\ell)\ +\ n^3/L\right)\\
&\leq\ \tilde{O}\left(\max_{\ell} L\cdot\min\left\{n^3/(L\ell),\ \ell\cdot\mathcal{M}(n/L,n/\ell,n)\right\}\ +\ n^3/L\right)\\
&=\ \tilde{O}\left(\max_{\ell\leq L} L\ell\cdot\mathcal{M}(n/L,n/\ell,n)\ +\ n^3/L\right)\ =\ \tilde{O}(L^2\cdot\mathcal{M}(n/L,n/L,n)\ +\ n^3/L).
\end{aligned}$$

With the current bounds on rectangular matrix multiplication, we choose $L=n^{0.4206}$ and get running time $O(n^{2.5794})$.

▶ **Theorem 24.** *($[c_0]-\{0\}$)-Lex$_2$-APSP (and thus -APLSP and -APSLP) for undirected graphs can be solved in $O(n^{2.5794})$ time for any $c_0=\tilde{O}(1)$.*

▶ Remarks. Without rectangular matrix multiplication, the above still gives a time bound of $\tilde{O}(L^3(n/L)^{\omega}+n^3/L)$, yielding $\tilde{O}(n^{2+1/(4-\omega)})$.

One could adapt the algorithm to solve Undirected ($[c_0]-\{0\}$)-Lex$_k$-APSP for a larger constant k, but the running time appears worse than the bound $\tilde{O}(n^{(3+\omega)/2})$ by Chan [6] (because of the need to compute a Min-Plus product between matrices with larger entries in Lemma 23).

6 Exact u-#APSP

We defer most of our algorithms for #APSP to the full paper. An exception is our algorithm for exact u-#APSP, which is simple and is described below. Interestingly, some of our #APSP algorithms are obtained by modifying our Lex$_2$-APSP algorithms, even though the #APSP and Lex$_2$-APSP problems appear very different.

For exact counts that could be exponentially large, we will describe a combinatorial $\tilde{O}(n^3)$-time algorithm to solve u-#APSP for directed unweighted graphs, in the standard word RAM model (with $(\log n)$-bit words). The idea behind the algorithm is actually related to the Lex$_2$-APSP algorithm in Section 5.2, but simplified with $L=1$ and without matrix multiplication and dominating sets.

Recall that the goal is to compute the number $C[u,v]$ of shortest paths from u to v, for all $u,v\in V$ for a given directed unweighted graph $G=(V,E)$.

We first compute $D[u,v]$ for all $u,v\in V$ in $O(n^3)$ time by known APSP algorithms. There are of course faster APSP algorithms for directed unweighted graphs, but we use the slower $O(n^3)$ time algorithm to keep the whole algorithm combinatorial.

Assume we have already computed $C[u,v]$ for all u,v with $D[u,v]\leq 2\ell/3$ for a given ℓ. Fix a source vertex $s\in V$. We will compute $C[s,v]$ for all v with $D[s,v]\leq\ell$, as follows:

Let $V_i=\{v\in V: D[s,v]=i\}$. Note that $\sum_i|V_i|=n$, so there exist an index $m\in[0.4\ell,0.6\ell]$ with $|V_m|=O(n/\ell)$.

For $i\leq m$, we have already computed $C[s,v]$ for all $v\in V_i$.

For $i=m+1,\ldots,\ell$, by setting $C[s,v]=\sum_{u\in V_m:D[u,v]=i-m}C[s,u]\cdot C[u,v]$, we compute $C[s,v]$ for all $v\in V_i$. Note that $C[s,u]$ and $C[u,v]$ have been computed from the previous iteration, since $i-m<2\ell/3$. The total number of arithmetic operations is $O(\sum_i|V_i|\cdot|V_m|)=O(n^2/\ell)$. Since the counts are bounded by $O(n^{\ell})$ and are $\tilde{O}(\ell)$-bit numbers, the total cost is $\tilde{O}(n^2/\ell\cdot\ell)=\tilde{O}(n^2)$.

We do this for every source $s \in V$. The overall cost is $\tilde{O}(n^3)$.
We do the above for all ℓ's that are powers of 3/2. The final time bound is $\tilde{O}(n^3)$.

▶ **Theorem 25.** *u-#APSP can be solved in $\tilde{O}(n^3)$ time.*

▶ Remarks. This is worst-case optimal up to polylogarithmic factors, as the total number of bits in the answers could be $\Omega(n^3)$.

Recall the Betweenness Centrality of a vertex v is defined as $\mathrm{BC}(v) = \sum_{s,t \neq v} C_v[s,t]/C[s,t]$ where $C_v[s,t]$ is the number of shortest paths between s and t that go through v. As an immediate corollary, we can compute the Betweenness Centrality of a given vertex exactly in a directed unweighted graph in $\tilde{O}(n^3)$ time.

▶ **Corollary 26.** *The betweenness centrality of a vertex can be computed in $\tilde{O}(n^3)$ time in a directed unweighted graph.*

References

1 Donald Aingworth, Chandra Chekuri, Piotr Indyk, and Rajeev Motwani. Fast estimation of diameter and shortest paths (without matrix multiplication). *SIAM J. Comput.*, 28(4):1167–1181, 1999.

2 Josh Alman and Virginia Vassilevska Williams. A refined laser method and faster matrix multiplication. In *Proceedings of the 32nd Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, page to appear, 2021.

3 Noga Alon, Zvi Galil, and Oded Margalit. On the exponent of the all pairs shortest path problem. *J. Comput. Syst. Sci.*, 54(2):255–262, 1997.

4 Ulrik Brandes. A faster algorithm for betweenness centrality. *The Journal of Mathematical Sociology*, 25(2):163–177, 2001.

5 Karl Bringmann, Marvin Künnemann, and Karol Wegrzycki. Approximating APSP without scaling: equivalence of approximate min-plus and exact min-max. In *Proceedings of the 51st Annual ACM Symposium on Theory of Computing (STOC)*, pages 943–954, 2019.

6 Timothy M. Chan. More algorithms for all-pairs shortest paths in weighted graphs. *SIAM J. Comput.*, 39(5):2075–2089, 2010. doi:10.1137/08071990X.

7 Timothy M. Chan. All-pairs shortest paths for unweighted undirected graphs in $o(mn)$ time. *ACM Trans. Algorithms*, 8(4):34:1–34:17, 2012. doi:10.1145/2344422.2344424.

8 Artur Czumaj, Miroslaw Kowaluk, and Andrzej Lingas. Faster algorithms for finding lowest common ancestors in directed acyclic graphs. *Theor. Comput. Sci.*, 380(1-2):37–46, 2007.

9 Ran Duan and Seth Pettie. Fast algorithms for (max, min)-matrix multiplication and bottleneck shortest paths. In *Proceedings of the 20th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 384–391, 2009.

10 Michael J Fischer and Albert R Meyer. Boolean matrix multiplication and transitive closure. In *Proceedings of the 12th Annual Symposium on Switching and Automata Theory (SWAT)*, pages 129–131, 1971.

11 Zvi Galil and Oded Margalit. Witnesses for Boolean matrix multiplication and for transitive closure. *J. Complex.*, 9(2):201–221, 1993.

12 Zvi Galil and Oded Margalit. All pairs shortest distances for graphs with small integer length edges. *Inf. Comput.*, 134(2):103–139, 1997. doi:10.1006/inco.1997.2620.

13 Zvi Galil and Oded Margalit. All pairs shortest paths for graphs with small integer length edges. *J. Comput. Syst. Sci.*, 54(2):243–254, 1997. doi:10.1006/jcss.1997.1385.

14 Fabrizio Grandoni, Giuseppe F Italian, Aleksander Łukasiewicz, Nikos Parotsidis, and Przemysław Uznański. All-pairs lca in dags: Breaking through the $o(n^{2.5})$ barrier. In *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 273–289. SIAM, 2021.

15 Karim Labib, Przemysław Uznański, and Daniel Wolleb-Graf. Hamming distance completeness. In *Proceedings of the 30th Annual Symposium on Combinatorial Pattern Matching (CPM)*, pages 14:1–14:17, 2019.

16 François Le Gall. Powers of tensors and fast matrix multiplication. In *Proceedings of the 39th International Symposium on Symbolic and Algebraic Computation (ISSAC)*, pages 296–303, 2014.

17 Francois Le Gall and Florent Urrutia. Improved rectangular matrix multiplication using powers of the Coppersmith-Winograd tensor. In *Proceedings of the 29th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1029–1046, 2018.

18 Andrea Lincoln, Adam Polak, and Virginia Vassilevska Williams. Monochromatic triangles, intermediate matrix products, and convolutions. In *Proceedings of the 11th Innovations in Theoretical Computer Science Conference (ITCS)*, pages 53:1–53:18, 2020.

19 Jiří Matoušek. Computing dominances in E^n. *Inf. Process. Lett.*, 38(5):277–278, 1991.

20 Liam Roditty and Asaf Shapira. All-pairs shortest paths with a sublinear additive error. In *Proceedings of the 35th International Colloquium on Automata, Languages and Programming (ICALP), Part I*, volume 5125 of *Lecture Notes in Computer Science*, pages 622–633. Springer, 2008.

21 Raimund Seidel. On the all-pairs-shortest-path problem in unweighted undirected graphs. *J. Comput. Syst. Sci.*, 51(3):400–403, 1995.

22 Avi Shoshan and Uri Zwick. All pairs shortest paths in undirected graphs with integer weights. In *Proceedings of the 40th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 605–614, 1999.

23 Jan van den Brand, Yin-Tat Lee, Danupon Nanongkai, Richard Peng, Thatchaphol Saranurak, Aaron Sidford, Zhao Song, and Di Wang. Bipartite matching in nearly-linear time on moderately dense graphs. In *2020 IEEE 61st Annual Symposium on Foundations of Computer Science (FOCS)*, pages 919–930. IEEE, 2020.

24 Virginia Vassilevska, Ryan Williams, and Raphael Yuster. Finding the smallest H-subgraph in real weighted graphs and related problems. In *Proceedings of the 33rd International Colloquium on Automata, Languages and Programming (ICALP), Part I*, volume 4051 of *Lecture Notes in Computer Science*, pages 262–273, 2006.

25 Virginia Vassilevska, Ryan Williams, and Raphael Yuster. All-pairs bottleneck paths for general graphs in truly sub-cubic time. In *Proceedings of the 39th Annual ACM Symposium on Theory of Computing (STOC)*, pages 585–589, 2007.

26 Virginia Vassilevska, Ryan Williams, and Raphael Yuster. Finding heaviest H-subgraphs in real weighted graphs, with applications. *ACM Trans. Algorithms*, 6(3):44:1–44:23, 2010.

27 Virginia Vassilevska Williams. Multiplying matrices faster than Coppersmith-Winograd. In *Proceedings of the 44th ACM Symposium on Theory of Computing (STOC)*, pages 887–898, 2012.

28 Virginia Vassilevska Williams. Problem 2 on problem set 2 of CS367, October 15, 2015. URL: `http://theory.stanford.edu/~virgi/cs367/hw2.pdf`.

29 Virginia Vassilevska Williams. On some fine-grained questions in algorithms and complexity. In *Proceedings of the International Congress of Mathematicians (ICM)*, pages 3447–3487, 2018.

30 Virginia Vassilevska Williams and Yinzhan Xu. Truly subcubic min-plus product for less structured matrices, with applications. In *Proceedings of the 31st Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 12–29, 2020.

31 R. Ryan Williams. Faster all-pairs shortest paths via circuit complexity. *SIAM J. Comput.*, 47(5):1965–1985, 2018. `doi:10.1137/15M1024524`.

32 Raphael Yuster. Efficient algorithms on sets of permutations, dominance, and real-weighted APSP. In *Proceedings of the 20th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 950–957, 2009. URL: `http://dl.acm.org/citation.cfm?id=1496770.1496873`.

33 Uri Zwick. All pairs lightest shortest paths. In *Proceedings of the 31st Annual ACM Symposium on Theory of Computing (STOC)*, pages 61–69, 1999.

34 Uri Zwick. All pairs shortest paths using bridging sets and rectangular matrix multiplication. *J. ACM*, 49(3):289–317, 2002.

An Almost Optimal Edit Distance Oracle

Panagiotis Charalampopoulos
The Interdisciplinary Center Herzliya, Israel

Paweł Gawrychowski
University of Wrocław, Poland

Shay Mozes
The Interdisciplinary Center Herzliya, Israel

Oren Weimann
University of Haifa, Israel

Abstract

We consider the problem of preprocessing two strings S and T, of lengths m and n, respectively, in order to be able to efficiently answer the following queries: Given positions i, j in S and positions a, b in T, return the optimal alignment score of $S[i\,..\,j]$ and $T[a\,..\,b]$. Let $N = mn$. We present an oracle with preprocessing time $N^{1+o(1)}$ and space $N^{1+o(1)}$ that answers queries in $\log^{2+o(1)} N$ time. In other words, we show that we can efficiently query for the alignment score of every pair of substrings after preprocessing the input for almost the same time it takes to compute just the alignment of S and T. Our oracle uses ideas from our distance oracle for planar graphs [STOC 2019] and exploits the special structure of the alignment graph. Conditioned on popular hardness conjectures, this result is optimal up to subpolynomial factors. Our results apply to both edit distance and longest common subsequence (LCS).

The best previously known oracle with construction time and size $\mathcal{O}(N)$ has slow $\Omega(\sqrt{N})$ query time [Sakai, TCS 2019], and the one with size $N^{1+o(1)}$ and query time $\log^{2+o(1)} N$ (using a planar graph distance oracle) has slow $\Omega(N^{3/2})$ construction time [Long & Pettie, SODA 2021]. We improve both approaches by roughly a $\sqrt{N}$ factor.

2012 ACM Subject Classification Theory of computation → Pattern matching; Theory of computation → Shortest paths

Keywords and phrases longest common subsequence, edit distance, planar graphs, Voronoi diagrams

Digital Object Identifier 10.4230/LIPIcs.ICALP.2021.48

Category Track A: Algorithms, Complexity and Games

Funding *Panagiotis Charalampopoulos*: Supported by Israel Science Foundation grant 592/17.
Shay Mozes: Partially supported by Israel Science Foundation grant 592/17.
Oren Weimann: Partially supported by Israel Science Foundation grant 592/17.

1 Introduction

String alignment is arguably the most popular problem in combinatorial pattern matching. Given two strings S and T of length m and n, the problem asks to compute the similarity between the strings according to some similarity measure. The two most popular similarity measures are edit distance and longest common subsequence (LCS). In both cases, the classical solution is essentially the same: Compute the shortest path from vertex $(0, 0)$ to vertex (m, n) in the so called *alignment graph* of the two strings. As taught in almost every elementary course on algorithms, computing this shortest path (and hence the optimal alignment of the two strings) can easily be done in $\mathcal{O}(N)$ time where $N = mn$, via dynamic programming. Interestingly, this time complexity cannot be significantly improved assuming popular conjectures such as the strong exponential time hypothesis (SETH) [1, 6, 8]. In fact,

by now we seem to have a rather good understanding of the complexity of this problem for different similarity measures and taking other parameters than the length of the strings into account, see [8].

Substring queries. A natural direction after having determined the complexity of a particular problem on strings is to consider the more general version in which we need to answer queries on substrings of the input string. This has been done for alignment [34, 36], pattern matching [24, 28], approximate pattern matching [17], dictionary matching [14, 15], compression [24], periodicity [27, 28], counting palindromes [33], longest common substring [3], computing minimal and maximal suffixes [5, 26], and computing the lexicographically k-th suffix [4].

Alignment oracles. Consider the shortest path from vertex (i, a) to vertex (j, b) in the alignment graph. It corresponds to the optimal alignment of two substrings: the substring of S between indices i and j and the substring of T between indices a and b. An *alignment oracle* is a data structure that, after preprocessing, can report the optimal alignment score of any two substrings of S and T. That is, given positions i, j in S and positions a, b in T, the oracle returns the optimal alignment score of $S[i\,..\,j]$ and $T[a\,..\,b]$ (or equivalently, the $(i-1, a-1)$-to-(j, b) distance in the alignment graph).

Tiskin [36, 37] considered a restricted variant of the problem, in which the queries are either the entire string S vs. a substring of T or a prefix of S vs. a suffix of T. For such queries, Tiskin gave an $\tilde{\mathcal{O}}(n+m)$-size oracle, that can be constructed in $\tilde{\mathcal{O}}(N)$ time, and answers queries in $\mathcal{O}(\log N/\log\log N)$ time [36]. For the general problem, Sakai [34] (building on Tiskin's work [36]) showed how to construct in $\mathcal{O}(N)$ time an alignment oracle with $\mathcal{O}(n+m)$ query time. In this work we show that, perhaps surprisingly, obtaining such an oracle can be done essentially for free! That is, at almost the same time it takes to compute just the alignment of S and T. More formally, our main result is:

▶ **Theorem 1.** *For two strings of lengths m and n, with $N = mn$, we can construct in $N^{1+o(1)}$ time an alignment oracle achieving either of the following tradeoffs:*

- *$N^{1+o(1)}$ space and $\log^{2+o(1)} N$ query time,*
- *$N \log^{2+o(1)} N$ space and $N^{o(1)}$ query time.*

Planar distance oracles and Voronoi diagrams. The starting point of our work is the recent developments in *distance oracles* for planar graphs. A distance oracle is a compact representation of a graph that allows to efficiently query the distance between any pair of vertices. Indeed, since the alignment graph is a planar graph, the state-of-the-art distance oracle for planar graphs of Long and Pettie [30] (which builds upon [13, 19, 21]) is an alignment oracle with space $N^{1+o(1)}$ and query time $\mathcal{O}(\log^{2+o(1)} N)$. However, the construction time of this oracle is $\Omega(N^{3/2})$. Our main contribution is an improved $N^{1+o(1)}$ construction time when the underlying graph is not just a planar graph but an alignment graph.

Our oracle has the same recursive structure as the planar graph oracles in [13, 21, 30] (in fact, the alignment graph, being a grid, greatly simplifies several technical, but standard, difficulties of the recursive structure). These oracles (inspired by Cabello's use of Voronoi diagrams for the diameter problem in planar graphs [10]) use the recursive structure in order to apply (at different levels of granularity) an efficient mechanism for *point location on Voronoi diagrams*. At a high level, a *Voronoi diagram* with respect to a subset S of vertices (called sites) is a partition of the vertices into $|S|$ parts (called Voronoi cells), where the cell of site $s \in S$ contains all vertices that are closer to s than to any other site in S. A *point*

location query, given a vertex v, returns the site s such that v belongs to the Voronoi cell of s. Our main technical contribution is a polynomially faster construction of the point location mechanism when the underlying graph is an alignment graph. We show that, in this case, the special structure of the Voronoi cells facilitates point location via a non-trivial divide and conquer. Unlike the planar oracles, which use planar duality to represent Voronoi diagrams, the representation and point location mechanisms we develop in this paper are novel and achieve the same query time, while being arguably simpler than those of Long and Pettie.[1]

It is common that techniques are originally developed for pattern matching problems (and in particular alignment problems) and later extended to planar graphs. A concrete example is the use of Monge matrices and unit-Monge matrices. However, it is much less common that techniques are first developed for planar graphs (in our case, the use of Voronoi diagrams) and only then translated to pattern matching problems.

Conditional lower bounds. Any lower bound on the time required to compute an optimal alignment of two strings directly implies an analogous lower bound for the sum of the preprocessing time and the query time of an alignment oracle. In particular, the existence of an oracle for which this sum is $\mathcal{O}(N^{1-\epsilon})$, for a constant $\epsilon > 0$, would refute SETH [6, 8].

In the *Set Disjointness* problem, we are given a collection of m sets $A_1, A_2, \ldots, A_m$ of total size M for preprocessing. We then need to report, given any query pair A_i, A_j, whether $A_i \cap A_j = \emptyset$. The Set Disjointness conjecture [18, 22, 32] states that any data structure with constant query time must use $M^{2-o(1)}$ space. Goldstein et al. [22] stated the following stronger conjecture.

▶ **Conjecture 2** (Strong Set Disjointness Conjecture [22]). *Any data structure for the Set Disjointness problem that answers queries in time t must use space $M^2/(t^2 \cdot \log^{\mathcal{O}(1)} M)$.*

The following theorem implies that, conditioned on the above conjecture, our alignment oracle is optimal up to subpolynomial factors; its proof is identical to that of [3, Theorem 1] as explained in [12].

▶ **Theorem 3** ([3, 12]). *An alignment oracle for two strings of length at most n with query time t must use $n^2/(t^2 \cdot \log^{\mathcal{O}(1)} n)$ space, assuming the Strong Set Disjointness Conjecture.*

Even though the main point of interest is in oracles that achieve fast (i.e. constant, polylogarithmic, or subpolynomial) query-time, the above lower bound suggests to study other tradeoffs of space vs. query-time. In Section 5 we show oracles with space sublinear in N. More formally, we prove the following theorem.

▶ **Theorem 4.** *Given two strings of lengths m and n with $N = mn$, integer alignment weights upper-bounded by w, and a parameter $r \in [\sqrt{N}, N]$ we can construct in $\tilde{\mathcal{O}}(N)$ time an $\tilde{\mathcal{O}}(Nw/\sqrt{r} + m + n)$-space alignment oracle that answers queries in time $\tilde{\mathcal{O}}(\sqrt{N} + r)$.*

For example, if the alignment weights are constant integers, by setting $r = \sqrt{N}$ we obtain an $\mathcal{O}(N^{3/4} + m + n)$-space oracle that answers queries in time $\tilde{\mathcal{O}}(\sqrt{N})$.

[1] We believe that our efficient construction can also be made to work, for alignment graphs, with the dual representation of Voronoi diagrams used in [13, 21, 30], but we think the new representation makes the presentation more approachable as it exploits the structure of the alignment graph more directly.

Other related works. When the edit distance is known to be bounded by some threshold k, an efficient edit distance oracle can be obtained via the Landau-Vishkin algorithm [29]. Namely, after $\mathcal{O}(n+m)$ time preprocessing of the input strings (oblivious to the threshold k), given a substring of S, a substring of T, and a threshold k, in $\mathcal{O}(k^2)$ time one can decide whether the edit distance of these substrings is at most k, and if so, return it.

Further, after an $\mathcal{O}(n+m)$-time preprocessing, given a substring X of S and a substring Y of T, the starting positions of substrings of Y that are at edit distance at most k from X can be returned in $\mathcal{O}(k^4 \cdot |Y|/|X|)$ time [17].

In a recent work [16] on dynamic string alignment, it was shown that, in the case where the alignment weights are small integers, two strings of total length at most n can be maintained under edit operations in $\tilde{\mathcal{O}}(n)$ time per operation so that an alignment of any pair of substrings can be queried in $\tilde{\mathcal{O}}(n)$ time.

2 Preliminaries

The alignment oracle presented in this paper applies to both edit distance and longest common subsequence (LCS). To simplify the presentation we focus on LCS but the extension to edit distance is immediate.

The LCS of two strings S and T is a longest string that is a subsequence of both S and T. We denote the length of an LCS of S and T by $\texttt{LCS}(S,T)$.

▶ **Example 5.** An LCS of $S = \texttt{acbcddaaea}$ and $T = \texttt{abbbccdec}$ is $\texttt{abcde}$; $\texttt{LCS}(S,T) = 5$.

For strings S and T, of lengths m and n respectively (we will assume that $n \geq m$), the alignment graph G of S and T is a directed acyclic graph of size $N = \mathcal{O}(mn)$. For every $0 \leq x \leq m$ and $0 \leq y \leq n$, the alignment graph G has a vertex (x,y) and the following unit-length edges (defined only if both endpoints exist):

- $((x,y),(x+1,y))$ and $((x,y),(x,y+1))$,
- $((x,y),(x+1,y+1))$, present if and only if $S[x] = T[y]$.

Intuitively, G is an $(m+1) \times (n+1)$ grid graph augmented with diagonal edges corresponding to matching letters of S and T. See Figure 1. We think of the vertex $(0,0)$ as the top-left vertex of the grid and the vertex (m,n) as the bottom-right vertex of the grid. We shall refer to the rows and columns of G in a natural way. It is easy to see that $\texttt{LCS}(S,T)$ equals $n+m$ minus the length of the shortest path from $(0,0)$ to (m,n) in G.

Multiple-source shortest paths. Given a planar graph with N vertices and a distinguished face h, the multiple-source shortest paths (MSSP) data structure represents all shortest path trees rooted at the vertices of h. It can be constructed in $\mathcal{O}(N \log N)$ time, requires $\mathcal{O}(N \log N)$ space, and can report the distance between any vertex u of h and any other vertex v in the graph in $\mathcal{O}(\log N)$ time. These bounds were first obtained for alignment graphs [35] and then extended to arbitrary planar graphs [11, 25].

The MSSP data structure can be augmented at no asymptotic overhead (cf. [23, Section 5]), to allow for the following. First, to report a shortest u-to-v path ρ in time $\mathcal{O}(|\rho| \log\log \Delta)$, where Δ is the maximum degree of a vertex in G. Second, to support the following queries in $\mathcal{O}(\log N)$ time [21]: Given two vertices $u, v \in G$ and a vertex x of h report whether u is an ancestor of v in the shortest path tree rooted at x, and whether u occurs before v in a preorder traversal of this tree. (We consider shortest path trees as ordered trees with the order inherited from the planar embedding.)

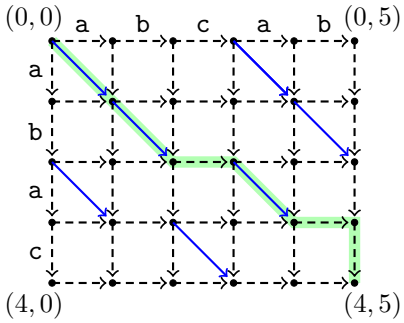


Figure 1 The alignment graph for $S = \texttt{abac}$ and $T = \texttt{abcab}$. We represent the horizontal and vertical edges by dashed black arrows, and the diagonal edges by blue arrows. A lowest scoring $(0, 0)$-to-$(4, 5)$ path is highlighted in green, it has weight 6 and corresponds to the LCS $\texttt{aba}$ of length $3 = 9 - 6 = |T| + |S| - 6$.

Recursive decomposition. We assume without loss of generality that the length of each of the two strings is a power of 2, and hence the alignment graph is a $(2^a + 1) \times (2^b + 1)$ grid. We consider a recursive decomposition $\mathcal{A}$ of G such that in each level all pieces are of the same rectangular shape. At each level, each piece will be of size $(2^c + 1) \times (2^d + 1)$ for non-negative integers c and d. Consider a piece P of size $(2^c + 1) \times (2^d + 1)$, with $c + d \neq 0$. Assuming without loss of generality that $c \geq d$, in the next level we will partition P to two pieces, each of size $(2^{c-1} + 1) \times (2^d + 1)$, that share the middle row of P. See Figure 2. We view $\mathcal{A}$ as a binary tree and identify a piece P with the node corresponding to it in $\mathcal{A}$.

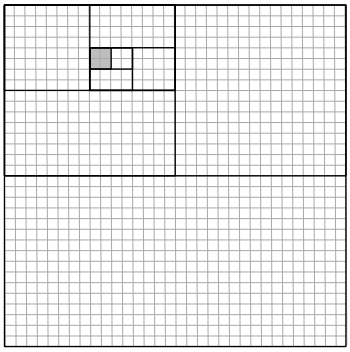


Figure 2 Illustration of pieces in a recursive decomposition of the alignment graph. Diagonal edges are not show to avoid clutter. All rectangular pieces that contain the gray square form a single root-to-leaf path in the tree $\mathcal{A}$.

Consider a piece $P \in \mathcal{A}$. The set ∂P of *boundary vertices* of a piece P consists of those vertices who have neighbours that are not in P. We call the vertices in $P \setminus \partial P$ the *internal vertices* of P. We denote by $\lrcorner(P)$ the set of boundary vertices of P that are either rightmost or bottommost in P, and by $\ulcorner(P)$ the set of boundary vertices of P that are either leftmost or topmost in P. We consider each of $\lrcorner(P)$ and $\ulcorner(P)$ to be ordered, such that adjacent vertices are consecutive and the earliest vertex is the bottom-left one. Therefore, whenever convenient, we refer to subsets of $\lrcorner(P)$ and $\ulcorner(P)$ as sequences. We define the *outside* of P, denoted by P^{out}, to be the set of vertices of $G \setminus (P \setminus \partial P)$ that are reachable from some vertex in P, i.e. the vertices of G that are not internal in P and are to the right or below some vertex of P. Note that any path from a vertex $u \in P$ to a vertex $v \in P^{out}$ must contain at least one vertex from $\lrcorner(P)$, and any path from a vertex $u \notin P$ to a vertex $u \in P$ must contain at least one vertex from $\ulcorner(P)$.

For any $r \in [1, nm]$, an *r-division* of G is a decomposition of G to pieces of size $\mathcal{O}(r)$, each with $\mathcal{O}(\sqrt{r})$ boundary vertices. Clearly, such a decomposition can be retrieved from $\mathcal{A}$, with all nodes being of the same depth. In particular, we will use recursive $(r_t, \ldots, r_1)$-divisions, where for every $i < t$, each piece of the r_i-division must be contained in some piece of the r_{i+1}-division. By convention, we will have r_t being a single piece consisting of the entire graph G. Such a recursive division can be materialized as follows: First, we select the appropriate depth of $\mathcal{A}$ for each r_i-division and mark all nodes of this depth. Then, we contract every edge of $\mathcal{A}$ of the form $(\mathsf{parent}(u), u)$ such that u is unmarked. Such a recursive $(r_t, \ldots, r_1)$-division can thus also be represented by a tree, which we will denote by $\mathcal{T}$.

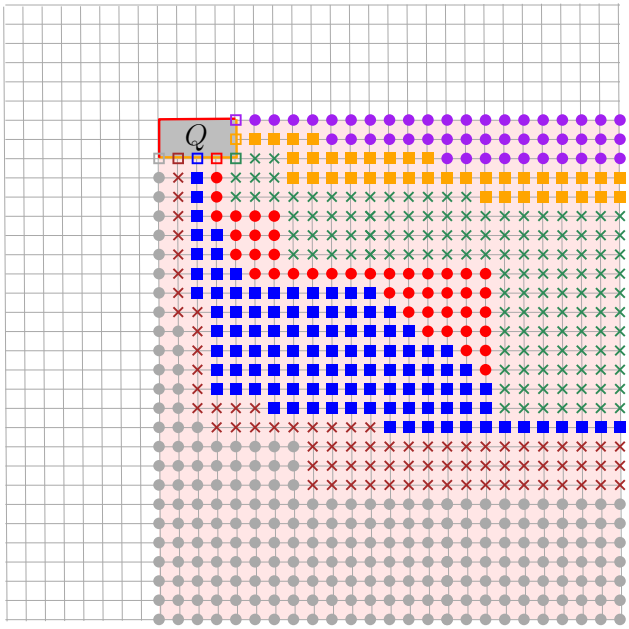


Figure 3 A piece Q (shaded gray). $\ulcorner(Q)$ is indicated by a red line, and $\lrcorner(Q)$ by an orange line. Q^{out} is shaded pink. The Voronoi diagram for Q^{out} with sites $\lrcorner(Q)$ (boxes) is also illustrated. Each site has a distinct color. Vertices in each Voronoi cell are indicated by a matching color.

Voronoi diagrams. Let H be a directed planar graph with real edge-lengths, and no negative-length cycles. Let h be a face of H, and let S be the set of vertices (called *sites*) of h. Each site $s \in S$ has a weight $\omega(s) \geq 0$ associated with it. The additively weighted distance $d^{\omega}(s, v)$ between a site $s \in S$ and a vertex $v \in H$ is defined as $\omega(s)$ plus the length of the shortest s-to-v path in H.

The *additively weighted Voronoi diagram* $\mathsf{VD}(S, \omega)$ of H is a partition of the vertices of H into pairwise disjoint sets, one set $\mathsf{Vor}(s)$ for each site $s \in S$. The set $\mathsf{Vor}(s)$, called the *Voronoi cell* of s, contains all vertices of H that are closer (w.r.t. $d^{\omega}(.,.)$) to s than to any other site in S. If $v \in \mathsf{Vor}(s)$ then we call s the site of v, and say that v belongs to the site s. Throughout the paper, we will only consider additively weighted Voronoi diagrams for the outside P^{out} of a piece $P \in \mathcal{A}$ with sites $S \subseteq \lrcorner(P)$. We next discuss the structure of such Voronoi diagrams.

We resolve ties between sites in favor of the site $s = (x, y)$ for which $(\omega(s), x, y)$ is lexicographically largest. Since the alignment graph is planar, this guarantees that the vertices in $\mathsf{Vor}(s)$ are spanned by a subtree of a shortest paths tree rooted at s: for every

vertex $v \in \mathsf{Vor}(s)$, for any vertex u on a shortest s-to-v path, we must have $u \in \mathsf{Vor}(s)$. Hence, each Voronoi cell is a simply connected region of the plane. The structure of the alignment graph dictates that any shortest path is monotone in the sense that it only goes right and/or down. This property immediately implies the following lemma.

▶ **Lemma 6.** *For any $a \leq c$ and $d \leq f$, if $u = (a, f)$ and $v = (c, d)$ both belong to $\mathsf{Vor}(s)$ then every vertex $w = (b, e)$ with $a \leq b \leq c$ and $d \leq e \leq f$ also belongs to $\mathsf{Vor}(s)$.*

Proof. Suppose $w = (b, e)$ belongs to $\mathsf{Vor}(s')$ for some $s' \neq s$. Since shortest paths only go right and down, the shortest s'-to-w path must cross either the s-to-u path or the s-to-v path, which is a contradiction. ◀

Lemma 6 together with the fact that $\mathsf{Vor}(s)$ is connected implies the following characterization of the structure of $\mathsf{Vor}(s)$, which roughly says that $\mathsf{Vor}(s)$ has the form of a double staircase, as illustrated in Figure 3.

▶ **Corollary 7.** *For any row a and any site s, the vertices of row a that belong to $\mathsf{Vor}(s)$ form a contiguous interval of columns $[i_a, j_a]$. Furthermore, the sequences i_a and j_a are monotone non-decreasing, $i_a \leq i_{a+1} \leq j_a + 1$, and $j_a \leq j_{a+1}$.*

▶ **Corollary 8.** *There is a rightmost vertex $s_{\searrow}$ in $\mathsf{Vor}(s)$ that is also a bottommost one.*

▶ **Corollary 9.** *For every $v \in \mathsf{Vor}(s)$, $\mathsf{Vor}(s)$ contains a path from v to $s_{\searrow}$.*

Our representation of a Voronoi diagram for P^{out} with sites $\lrcorner(P)$ consists of the following. For each $s \in \lrcorner(P)$, we store:

1. the rightmost bottommost vertex $s_{\searrow}$ of $\mathsf{Vor}(s)$, defined in Corollary 8,
2. a vertex $\mathsf{last}(s, s_{\searrow})$ on a shortest s-to-$s_{\searrow}$ path, whose definition will be given later.

3 The Alignment Oracle

In this section, we describe our oracle and prove that its space and query time are as in Theorem 1. In the next section we will show how to construct the oracle in $N^{1+o(1)}$ time.

Consider a recursive $(r_t, \ldots, r_1)$-division of G for some $N = r_t > \cdots > r_1 = \mathcal{O}(1)$ to be specified later. Recall that our convention is that the r_t-division consists of G itself. Further, we set the r_1-division to consist of pieces of size 2×2. We also consider an r_0-division, in which each vertex v of G is a singleton piece. Let us denote the set of pieces of the r_i-division by $\mathcal{R}_i$. Let $\mathcal{T}$ denote the tree representing this recursive $(r_t, \ldots, r_0)$-division, where each singleton piece $\{v\}$ at level 0 is attached as a child of a piece P at level 1 such that $v \in \lrcorner(P)$ – this is well-defined for all singletons apart from $\{(0, 0)\}$ and $\{(m, n)\}$, each of which is attached to the single level-1 piece containing it.

The oracle consists of the following. For each $0 \leq i \leq t - 1$, for each piece $P \in \mathcal{R}_i$ whose parent in $\mathcal{T}$ is $Q \in \mathcal{R}_{i+1}$:

1. If $i > 0$, we store an MSSP with sources $\lrcorner(P)$ for the graph obtained from P by flipping the orientation of all edges; we call this the *reverse MSSP of P*.
2. If $i > 0$, we store an MSSP with sources $\lrcorner(P)$ for $Q \setminus (P \setminus \partial P)$.
3. If $i < t - 1$, for each vertex $u \in \lrcorner(P)$ we store $\mathsf{VD}(u, Q)$: the Voronoi diagram for Q^{out} with sites $\lrcorner(Q)$ and additive weights the distances in G from u to these sites.

To complete the description of the oracle it remains to specify the definition of $\mathsf{last}(s, s_{\searrow})$. Before doing so, let us distinguish, for a (source) vertex u and a (target) vertex v, two levels of the recursive division that are of interest. Let R_0 be the singleton piece $\{u\}$. Let

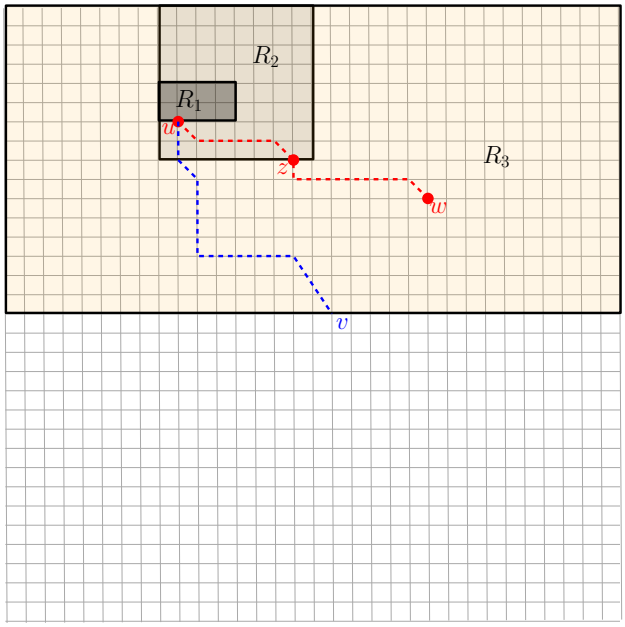


Figure 4 A vertex u and all the pieces of a recursive $(r_4, r_3, r_2, r_1, r_0)$-division that contain u. The piece R_0 consists of just u, and the piece R_4 is the entire alignment graph. Note that $u \in \lrcorner(R_1)$. In this example, $\mathsf{lev}(u) = 1$, $\mathsf{anc}(u, w) = 3$, and $\mathsf{anc}(u, v) = 4$ (because $v \in \partial R_3$). A u-to-w shortest path ρ is shown in dashed red. $\mathsf{last}(u, w)$ is the last vertex of ρ that belongs to $\lrcorner(R_2)$, which is the vertex z. The distance from u to w is $\mathsf{dist}(u, z) + \mathsf{dist}(z, w)$. $\mathsf{dist}(u, z)$ is stored in the reverse MSSP of R_2. $\mathsf{dist}(z, w)$ is stored in the MSSP of $R_3 \setminus (R_2 \setminus \partial R_2)$. Similarly, a u-to-v shortest path ρ is shown in dashed blue. Because $v \in \partial R_3$, $\mathsf{last}(u, v)$ is v itself.

$R_1, R_2, \ldots, R_t$ be the ancestors of R_0 in $\mathcal{T}$. Note that $u \in \lrcorner(R_i)$ for a non-empty prefix of the sequence of ancestors $R_0, R_1, \ldots, R_t$. Let $\mathsf{lev}(u) = \operatorname{argmax}_i\{u \in \lrcorner(R_i)\}$. Further, let $\mathsf{anc}(u, v) = \operatorname{argmin}_i\{v \in R_i \setminus \partial R_i\}$. Note that $\mathsf{anc}(u, v)$ is well defined since $v \in R_t = G$ and $\partial G = \emptyset$. Also note that if v is reachable from u then $\mathsf{lev}(u) < \mathsf{anc}(u, v)$. This is because all vertices in $R_{\mathsf{lev}(u)} \setminus \partial R_{\mathsf{lev}(u)}$ are unreachable from u.

Denote $H = R_{\mathsf{anc}(u,v)-1}$. We define $\mathsf{last}(u, v)$ as any boundary vertex of $\lrcorner(H)$ that lies on a shortest u-to-v path ρ. The idea behind this definition is that $\mathsf{last}(u, v)$ partitions this u-to-v path into a prefix and a suffix, each of which is represented in one of the MSSP data structures stored for H; The prefix of ρ ending at $\mathsf{last}(u, v)$ is represented in the shortest path tree rooted at $\mathsf{last}(u, v)$ in the reverse MSSP of H. The suffix of ρ starting at $\mathsf{last}(u, v)$ is represented in the shortest path tree rooted at $\mathsf{last}(u, v)$ in the MSSP for $H' \setminus (H \setminus \partial H)$ with sources $\lrcorner(H)$, where $H' = R_{\mathsf{anc}(u,v)}$ is the parent of H in $\mathcal{T}$. This allows us to efficiently compute $\mathsf{dist}(u, v)$ (the u-to-v distance) given $\mathsf{last}(u, v)$. See Figure 4. This concludes the description of the oracle.

▶ **Lemma 10.** *The oracle occupies space* $\mathcal{O}\left(N \log^2 N + N \log N \cdot \sum_{i=0}^{t-1} r_{i+1}/r_i\right)$.

Proof. The reverse MSSPs over all pieces of $\mathcal{A}$ require $\mathcal{O}(N \log^2 N)$ space, since $\sum_{P \in \mathcal{A}} |P| = \mathcal{O}(N \log N)$ and since the reverse MSSP of a piece P requires space $\mathcal{O}(|P| \log |P|)$.

For each $i \in (0, t-1]$, for each of the $\mathcal{O}(N/r_i)$ pieces in $\mathcal{R}_i$, we store an MSSP of size $\mathcal{O}(r_{i+1} \log r_{i+1})$. For each $i \in [0, t-1)$, for each of the $\mathcal{O}(N/r_i)$ pieces in $\mathcal{R}_i$, we store $\mathcal{O}(\sqrt{r_i})$ Voronoi diagrams each of size $\mathcal{O}(\sqrt{r_{i+1}})$. The stated bound follows. ◀

Query. We now describe how to answer a distance query $\mathsf{dist}(u, v)$. First, note that if u and v are in the same piece P in $\mathcal{R}_1$ we can report $\mathsf{dist}(u, v)$ in $\mathcal{O}(1)$ time using brute force.

Let R_0 be the singleton piece $\{u\}$. As before, let $R_1, R_2, \ldots, R_t$ be the ancestors of R_0 in $\mathcal{T}$. Since the distance query originates from an LCS query, v must be reachable from u. Let $\ell = \mathsf{lev}(u)$ and $h = \mathsf{anc}(u, v)$. We then have $u \in \lrcorner(R_\ell)$ and $v \in R_\ell^{out}$. We will answer $\mathsf{dist}(u, v)$ by identifying $\mathsf{last}(u, v)$, which lies on $\lrcorner(R_{h-1})$. As explained above, we can then obtain $\mathsf{dist}(u, v)$ from the MSSP data structures stored for R_{h-1}. See Algorithm 1.

Algorithm 1 DIST(u, v).

```
if u and v belong to the same piece in R_1 then
    return the answer by brute force
w ← GetLast(u, v)
return dist(u, w) + dist(w, v)
```

We now show how to implement the procedure GETLAST for finding $\mathsf{last}(u, v)$; see Algorithm 2 for a pseudocode. First, note that if $h = \ell + 1$ we can simply return u as $\mathsf{last}(u, v)$. Hence, in what follows, we assume that $h > \ell + 1$. The procedure GETLAST proceeds in iterations for $i = \ell+1, \ldots, h-1$. At the beginning of the iteration with value i, the procedure has a subset W_{i-1} of $\lrcorner(R_{i-1})$ such that some $w \in W_{i-1}$ belongs to a shortest u-to-v path. We initially set $W_\ell := \{u\}$, which trivially satisfies the requirement for $i = \ell + 1$. For each $w \in W_{i-1}$ the iteration uses a procedure GETNEXTCANDIDATES that adds at most two vertices of $\lrcorner(R_i)$ to W_i. The guarantee is that, if w is a vertex of $\lrcorner(R_{i-1})$ that belongs to a shortest u-to-v path, then at least one of the two added vertices also belongs to a shortest u-to-v path. Since $|W_i| \leq 2|W_{i-1}|$, at the end of the last iteration (the one for $h - 1$), we have a subset W_{h-1} of at most 2^t vertices of $\lrcorner(R_{h-1})$, one of which can be returned as $\mathsf{last}(u, v)$. To figure out which one, for each such vertex w, we use the MSSP data structures to compute $\mathsf{dist}(u, w) + \mathsf{dist}(w, v)$, and return a vertex for which the minimum is attained.

Algorithm 2 GETLAST(u, v).

```
ℓ ← lev(u)
h ← anc(u, v)
W_ℓ ← {u}
for i = ℓ + 1 to h − 1 do
    W_i ← ∅
    for each w ∈ W_{i−1} do
        W_i ← W_i ∪ GetNextCandidates(w, i, v)
return argmin_{w∈W_{h−1}} dist(u, w) + dist(w, v)
```

It remains to describe the procedure GETNEXTCANDIDATES. Consider any $w \in W_{i-1}$. To reduce clutter, let us denote R_i by Q. Since $w \in \lrcorner(R_{i-1})$, the Voronoi diagram $\mathsf{VD}(w, Q)$ for Q^{out} is stored by the oracle. The procedure GETNEXTCANDIDATES finds two sites of $\mathsf{VD}(w, Q)$, one of which is the site of v. Indeed, if w is a vertex on a u-to-v shortest path then the site of v in $\mathsf{VD}(w, Q)$ is a vertex of $\lrcorner(Q)$ on a shortest u-to-v path.

Let $(x_\lrcorner, y_\lrcorner)$ be the bottom-right vertex of Q. Every vertex $v = (x_v, y_v)$ of $Q^{out} \setminus \partial Q$ either has $x_v > x_\lrcorner$ or $y_v > y_\lrcorner$. We describe the case when $x_v > x_\lrcorner$; the case when $y_v > y_\lrcorner$ is analogous. Let Γ denote the set of $s \to s_\searrow$ paths ρ_s stored in $\mathsf{VD}(w, Q)$ according to the order of the sites s along $\lrcorner(Q)$. For every row $x > x_\lrcorner$, let Γ_x denote the subset of paths in Γ that intersect row x, ordered according to the order of the intersection vertices along row x.

▶ **Lemma 11.** *For every x, x' with $x > x'$, Γ_x is a subsequence of $\Gamma_{x'}$.*

Proof. This is a direct consequence of the fact that each path ρ_s goes monotonically down and right, and from the fact that ρ_s and $\rho_{s'}$ are disjoint for $s \neq s'$. ◀

We define the set of *critical rows* to be all rows x such that $(x, y) = s_{\searrow}$ for some $y \in [0, n]$ and $s \in \lrcorner(Q)$. Lemma 11 implies that if we consider the evolution of the sequences Γ_x as x increases from $x_{\lrcorner}$ to m as a dynamic process, changes occur only at critical rows. More precisely, if the row of $s_{\searrow}$ is x for some site s, then $\rho_s \in \Gamma_x$ but $\rho_s \notin \Gamma_{x+1}$. We can therefore maintain the sequences Γ_x in a persistent binary search tree. (A binary search tree can be made partially persistent at no extra asymptotic cost in the update and search times using a general technique for pointer-machine data structures of bounded degree [9].) Initially, the BST stores the sequence $\Gamma_{x_{\lrcorner}+1}$. Then, we go over the critical rows in increasing order, and remove the path ρ_s from the BST when we reach the row of $s_{\searrow}$.

For any $x_{\lrcorner} < x \leq m$, we can access the BST representation of Γ_x by finding the predecessor x' of x among the critical rows, and accessing the persistent BST at time x'.

Let $v = (x, y)$. We say that v is right (left) of a path $\rho_s \in \Gamma_x$ if y is greater (smaller) than any vertex of ρ_s at row x. We will either find a path $\rho_s \in \Gamma_x$ to which v belongs, or identify the last path $\rho_s \in \Gamma_x$ such that v is right of ρ_s. In the former case the site of v is s, and in the latter case the site of v is either s or the successor of s in Γ_x.

Recall that (1) the MSSP data structure, given a root vertex r and two vertices w, z, can determine in $\mathcal{O}(\log N)$ time whether w is left/right/ancestor/descendant of z in the shortest path tree rooted at r, (2) for each shortest path ρ_s represented in $\mathsf{VD}(w, Q)$, the representation contains $\mathsf{last}(s, s_{\searrow})$, and (3) the prefix of ρ_s ending at $\mathsf{last}(s, s_{\searrow})$ is represented in the shortest path tree rooted at $\mathsf{last}(s, s_{\searrow})$ in the reverse MSSP of H (recall that $H = R_{\mathsf{anc}(u,v)-1}$), while the suffix of ρ_s starting at $\mathsf{last}(s, s_{\searrow})$ is represented in the shortest path tree rooted at $\mathsf{last}(s, s_{\searrow})$ in the MSSP for $H' \setminus (H \setminus \partial H)$, where H' is the parent of H.

We perform binary search on Γ_x to identify the path ρ_s such that either $v \in \rho_s$ or ρ_s is the last path of Γ_x that is left of v. Focus on a step of the binary search that considers a path ρ_s. Denote $\mathsf{last}(s, s_{\searrow}) = (x_b, y_b)$. If $y < y_b$, we query the MSSP structure that contains the prefix of ρ_s, and otherwise we query the MSSP data structure that contains the suffix of ρ_s. In either case, the query either returns that v is on ρ_s or tells us whether v is left or right of ρ_s. In the former case we conclude that the site of v is s. In the latter case we continue the binary search accordingly. Each step of the binary search takes $\mathcal{O}(\log n)$ time. Note that $\log n = \mathcal{O}(\log N)$. Thus, the binary search takes $\mathcal{O}(\log^2 N)$ time, and when it terminates we have a site s that is either the site of v or the site such that ρ_s is the last path of Γ_x that is left of v. This implies that the site of v is either s or the successor of s in Γ_x, and concludes the description of GetNextCandidates.

▶ **Lemma 12.** *The oracle answers distance queries in time $\mathcal{O}(2^t \log^2 N)$.*

Proof. First, $\mathsf{lev}(u)$ and $\mathsf{anc}(u, v)$ can be (naively) computed in $\mathcal{O}(\log N)$ time by going over the ancestors of $\{u\}$ in $\mathcal{T}$: for each (rectangular) ancestor piece R of $\{u\}$, in $\mathcal{O}(1)$ time, we can retrieve the coordinates of R's corners and check whether $v \in R \setminus \partial R$ using v's coordinates. Overall, for a $\mathsf{dist}(u, v)$ query, we make $\mathcal{O}(2^t)$ calls to GetNextCandidates, each requiring $\mathcal{O}(\log^2 N)$ time, for a total of $\mathcal{O}(2^t \log^2 N)$ time. Finally, we make $\mathcal{O}(2^t)$ queries to the MSSP data structures, requiring $\mathcal{O}(2^t \log N)$ time in total. ◀

▶ Remark 13. Since our query procedure computes $\mathsf{last}(u, v)$, and we have MSSP data structures that capture the u-to-$\mathsf{last}(u, v)$ and the $\mathsf{last}(u, v)$-to-v shortest paths, an optimal alignment can be returned in time proportional to the total length of the two substrings.

By setting the r_i's appropriately, we obtain the following tradeoffs, which are identical to those of Pettie and Long for arbitrary planar graphs [30].

▶ **Proposition 14.** *For two strings of lengths m and n, with $N = mn$, there is an alignment oracle achieving either of the following tradeoffs:*
- $N \log^{2+o(1)} N$ *space and* $N^{o(1)}$ *query time,*
- $N^{1+o(1)}$ *space and* $\log^{2+o(1)} N$ *query time.*

Proof. The space of the oracle is $\mathcal{O}\left(N \log^2 N + N \log N \cdot \sum_{i=0}^{t-1} r_{i+1}/r_i\right)$ by Lemma 10. We will choose r_i's for $i \geq 1$ to be a geometric progression with common ratio p to be specified below. In that case, $t = \mathcal{O}(\log_p N)$ and the space becomes $\mathcal{O}(N \log^2 N + N \log N \cdot p \log_p N)$. First, let us set $p = N^{1/g(N)}$ for some $g(N)$ which is $\omega(\log N / \log\log N)$ and $o(\log N)$. Then, $p = 2^{\log N/g(N)} = 2^{o(\log\log N)} = \log^{o(1)} N$. We get $\mathcal{O}(N \log N \cdot 2^{\log N/g(N)} \log N) = N \log^{2+o(1)} N$ space and $N^{o(1)}$ query time. Second, let us set $p = N^{1/f(N)}$, for some $f(N)$ which is $\omega(1)$ and $o(\log\log N)$. We get $N^{1+o(1)}$ space and $\log^{2+o(1)} N$ query time. ◀

The following observation will prove useful in the efficient construction algorithm of the oracle that will be presented in the next section.

▶ **Observation 15.** *The query algorithm for $\mathsf{dist}(u, v)$ takes $\mathcal{O}(2^{t-\mathsf{lev}(u)} \log^2 N)$ time and uses only Voronoi diagrams $\mathsf{VD}(u, Q)$ for $Q \in \mathcal{R}_i$ with $i > \mathsf{lev}(u)$.*

4 An Efficient Construction Algorithm

In this section, we present an algorithm for constructing the alignment oracle in $N^{1+o(1)}$ time (thus completing the proof of Theorem 1). The computation of the recursive decomposition, the recursive $(r_t, \ldots, r_0)$-division and all of the MSSP structures stored for all pieces in $\mathcal{A}$ can be done in $\mathcal{O}(N \log^2 N)$ time. It therefore only remains to analyze the time it takes to construct all the representations of Voronoi diagrams stored by the oracle.

Consider some additively weighted Voronoi diagram for Q^{out} with sites a subsequence U of $\lrcorner(Q)$ – we will only build Voronoi diagrams with $U = \lrcorner(Q)$, but during the analysis, we will also consider Voronoi diagrams with sites $U \subseteq \lrcorner(Q)$. In what follows, when we talk about a piece $H \neq Q$, we will really mean its intersection with Q^{out}, assuming that it is non-empty. Similarly, when we talk about ∂H, $\ulcorner(H)$, and $\lrcorner(H)$ we will really mean the intersection of ∂H, $\ulcorner(H)$, and $\lrcorner(H)$ with Q^{out}, respectively. See Figure 5 for an illustration.

▶ **Lemma 16.** *Let $X \in \{\ulcorner(H), \lrcorner(H)\}$. Let $u, v \in X$ belong to distinct Voronoi cells. If u precedes v (in X) then the site $s_u \in U$ of u precedes (in U) the site $s_v \in U$ of v.*

Proof. For any vertex $a \in X$, that belongs to the cell of a site s_a, all vertices in the shortest s_a-to-a path belong to $\mathsf{Vor}(s_a)$. Towards a contradiction, suppose that s_u succeeds s_v in U. By the planarity of the graph and the fact that paths can go only down and right it follows that the shortest s_u-to-u path must cross the shortest s_v-to-v path in some vertex b. Then $b \in \mathsf{Vor}(s_u) \cap \mathsf{Vor}(s_v)$, which is a contradiction as Voronoi cells are disjoint. ◀

The above lemma means that the vertices of each of $\ulcorner(H)$ and $\lrcorner(H)$ can be partitioned into maximal contiguous intervals of vertices belonging to the Voronoi cell of the same site in U. When we say that we compute the partition of $\ulcorner(H)$ or $\lrcorner(H)$ with respect to U, we mean that we compute these intervals, specified by their endpoints, and, for each interval, the site from U to which the vertices of the interval belong.

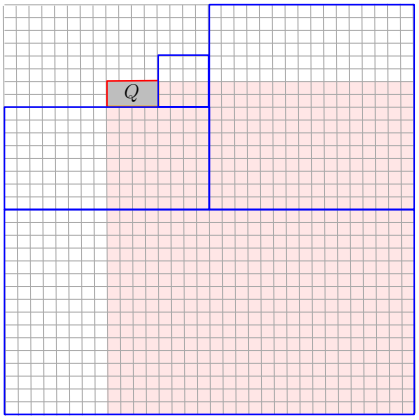


Figure 5 Covering Q^{out} (shaded pink) with siblings of ancestors of Q in $\mathcal{A}$ (blue boxes). When we refer to a piece H (e.g. a blue box), we only refer to the portion of H that belongs to Q^{out}.

The following simple observation allows us to compute partitions using binary search. It says that a piece H contains $s_{\searrow}$ if and only if s is the site of some vertex in $\ulcorner(H)$, and s is not the site of any vertex in $\lrcorner(H)$.

▶ **Lemma 17.** *For any $s \in S$ and any level ℓ, there is a unique level-ℓ piece $H \in \mathcal{A}$ for which $\mathsf{Vor}(s) \bigcap \ulcorner(H) \neq \emptyset$ and $\mathsf{Vor}(s) \bigcap \lrcorner(H) = \emptyset$, and this piece contains $s_{\searrow}$.*

Proof. By Corollary 9, for all $v \in \mathsf{Vor}(s)$, there exists a v-to-$s_{\searrow}$ path all of whose vertices are in $\mathsf{Vor}(s)$. Hence, for every level-ℓ piece H for which $\mathsf{Vor}(s) \bigcap \ulcorner(H) \neq \emptyset$ and $s_{\searrow} \notin H$, we must also have that $\mathsf{Vor}(s) \bigcap \lrcorner(H) \neq \emptyset$. We can thus focus on the at most four level-ℓ pieces that contain $s_{\searrow}$. It is readily verified that the bottom-right of those pieces is the only one for which $\mathsf{Vor}(s) \bigcap \lrcorner(H) = \emptyset$. ◀

▶ Remark 18. The condition in the statement of Lemma 17 is equivalent to: $s_{\searrow} \in H \setminus \lrcorner(H)$.

Lemma 17 provides a criterion on the partitions of $\ulcorner(H)$ and $\lrcorner(H)$ for determining whether a piece H contains any vertex $s_{\searrow}$. The following lemma describes a binary search procedure, Partition, which gets as input a sequence U of candidate sites, and returns the partition of $\ulcorner(H)$ or $\lrcorner(H)$; i.e., the subsequence of sites in U whose Voronoi cells contain the vertices of $\ulcorner(H)$ or $\lrcorner(H)$. The procedure Partition will be a key element in the overall construction algorithm. The following lemma describes an implementation of Partition using distance queries $\mathsf{dist}(u, v)$ with $u \in \lrcorner(Q)$ and $v \in \partial H$. We will ensure that such queries can be answered efficiently whenever Partition is called by the main algorithm.

▶ **Lemma 19.** *Given a sequence $U \subseteq \lrcorner(Q)$ of sites and their additive weights, we can perform the procedure Partition (that computes a partition of $\ulcorner(H)$ or $\lrcorner(H)$ w.r.t. U) in the time required by $\mathcal{O}(|U| \cdot \log n)$ distance queries $\mathsf{dist}(u, v)$ with $u \in \lrcorner(Q)$ and $v \in \partial H$.*

Proof. We will only prove the statement for $\ulcorner(H)$ as the case of $\lrcorner(H)$ is analogous. We start with a single interval, which is all of $\ulcorner(H)$. We will call an interval *active* if we have not concluded that all of its vertices belong to the same Voronoi cell. For each active interval L, we have a set C_L of *candidate sites*. Thus, initially, the single interval $\ulcorner(H)$ is active, and U is the set of its candidate sites.

The algorithm proceeds by divide and conquer. As long as we have an active interval L, we perform the following: we compute the site $u \in C_L$ with the minimum additively weighted distance to the midpoint of L. This is done in the time required by C_L distance queries of the form specified in the statement of the lemma. Then, we split L at this midpoint: for the

left part of L the set of candidate sites is now $\{v \in C_L : v \leq u\}$, while for the right part of L the set of candidate sites is now $\{v \in C_L : v \geq u\}$. If either of these two sets is of size 1, the corresponding interval becomes inactivate. That is, we recurse on at most two active intervals of roughly half the length. In the end, in a left-to-right pass, we merge consecutive intervals all of whose vertices belong to the same Voronoi cell.

Let us now analyze the time complexity of the above algorithm. First, observe that the sequences of candidates of any two intervals at the same level of the recursion are internally disjoint. Thus, each site is a candidate for at most two active intervals at the same recursive level. Second, at level j of the recursion, the length of every active interval is $\mathcal{O}(|\ulcorner(H)|/2^j)$. Hence, the total time required to process all intervals is proportional to the time required by $\mathcal{O}(|U| \cdot \log n)$ distance queries $\mathsf{dist}(u, v)$, with $u \in \lrcorner(Q)$ and $v \in \partial H$. ◀

We now present the algorithm for computing the representations of the Voronoi diagrams stored by the oracle. The algorithm performs the computation in order of decreasing levels of the recursive $(r_t, \ldots, r_0)$-division.

Consider some level i, and assume that we have already computed all Voronoi diagrams $\mathsf{VD}(u, R)$ for pieces $R \in \bigcup_{j>i} \mathcal{R}_j$. Consider any piece $P \in \mathcal{R}_{i-1}$. Let $Q \in \mathcal{R}_i$ be the parent of P in $\mathcal{T}$. Our goal is to compute, for every $u \in \lrcorner(P)$, the representation of $\mathsf{VD}(u, Q)$, the Voronoi diagram of Q^{out} with sites $\lrcorner(Q)$ and additive weights $\mathsf{dist}(u, \lrcorner(Q))$. Recall that this representation consists of the vertices $s_{\searrow}$ and $\mathsf{last}(s, s_{\searrow})$ for every site $s \in \lrcorner(Q)$. We would like to compute this representation in time roughly proportional to its size $|\lrcorner(Q)|$. By Observation 15, using the already computed parts of the oracle for levels $j > i$, we can already answer any distance query $\mathsf{dist}(s, v)$ for any $s \in \lrcorner(Q)$ and any $v \in Q^{out}$ in $\mathcal{O}(2^t \log^2 n)$ time. These are precisely the distance queries required for computing partitions of pieces H in Q^{out} w.r.t. sites in $\lrcorner(Q)$ (Lemma 19).

The computation is done separately for each $u \in \lrcorner(P)$. First, we compute the additive weights $\mathsf{dist}(u, \lrcorner(Q))$ in $\mathcal{O}(|\lrcorner(Q)| \cdot \log n)$ time using the MSSP data structure stored for $Q \setminus (P \setminus \partial P)$ with sites $\lrcorner(P)$. Next, we cover Q^{out} using $\mathcal{O}(\log N)$ pieces from $\mathcal{A}$ that are internally disjoint from Q (i.e. they may only share boundary vertices). These pieces are the $\mathcal{O}(\log N)$ siblings of the (weak) ancestors of Q in $\mathcal{A}$ that have a non-empty intersection with Q^{out} (see Figure 5). Notice that these pieces are in $\mathcal{A}$ but not necessarily in $\mathcal{T}$.

We shall find the vertices $s_{\searrow}$ of $\mathsf{VD}(u, Q)$ in each such piece H separately. We invoke Partition on $\ulcorner(H)$ and on $\lrcorner(H)$, and use Lemma 17 to determine whether H contains any vertices $s_{\searrow}$. If so, we zoom in on each of the two child pieces of H in $\mathcal{A}$ until, after $\mathcal{O}(\log N)$ steps, we get to a constant-size piece, in which we can find $s_{\searrow}$ by brute force. Note, however, that we are aiming for a running time that is roughly proportional to $|\lrcorner(Q)|$, but that the running time of Partition depends on the number of sites U w.r.t. which we partition. This is problematic since, e.g., when H contains $s_{\searrow}$ just for a single site s, we can only afford to invest $\tilde{\mathcal{O}}(1)$ time in locating $s_{\searrow}$ in H. In this case, computing the partition w.r.t. $|\lrcorner(Q)|$ is too expensive. Even computing the partition just w.r.t. the sites whose Voronoi cell has non-empty intersection with H, which is bounded by $|\ulcorner(H)|$, is too expensive. To overcome this problem we will show that it suffices to compute the partition w.r.t. a smaller sequence of sites, whose size is proportional to the number of sites s with $s_{\searrow}$ in H (actually in $H \setminus \lrcorner(H)$), rather than to the size of H or of $\ulcorner(H)$. We call such a sequence a *safe* sequence of sites for H, which we now define formally. Recall that the Voronoi diagram $\mathsf{VD}(u, Q)$ of Q^{out} has sites $\lrcorner(Q)$. Let U be a subsequence of $\lrcorner(Q)$. Consider the Voronoi diagram VD' of Q^{out} whose sites are the elements of U (with the same additive distances as in $\mathsf{VD}(u, Q)$). We say that U is *safe for* H if and only if the sets $\{(s, s_{\searrow}) : s \text{ is a site and } s_{\searrow} \in H \setminus \lrcorner(H)\}$ are identical for VD' and $\mathsf{VD}(u, Q)$.

▶ **Observation 20.** *A sequence that is safe for H is also safe for any child H' of H in $\mathcal{A}$.*

We will discuss the details of safe sequences after first providing the pseudocode of the procedure Zoom for finding the vertices $s_{\searrow}$ in a piece H.

Algorithm 3 Zoom(U, ω, H).

Input: The additive weight $\omega(s)$ for each $s \in \lrcorner(Q)$, a piece H in $\mathcal{A}$ that is internally disjoint from Q and has a non-empty intersection with Q^{out}, and a sequence $U \subseteq \lrcorner(Q)$ that is safe for H.

Output: All vertices $s_{\searrow}$ for $s \in \lrcorner(Q)$ that belong to H.

1: **if** $|H| = 4$ **then**
2: Find all vertices $s_{\searrow}$ in H for all $s \in U$ by computing $\omega(s) + \mathsf{dist}(s, v)$ for all $s \in U$ and all v adjacent to some vertex of H.
3: **for** each child H' of H in $\mathcal{A}$ **do**
4: $Z \leftarrow$ Partition$(U, \omega, \ulcorner(H'))$
5: $Z' \leftarrow$ Partition$(U, \omega, \lrcorner(H'))$
6: $L \leftarrow Z \setminus Z'$
7: **if** $L \neq \emptyset$ **then**
8: $V \leftarrow Z \setminus \{z \in Z \setminus L :$ both the predecessor and successor of z in Z are not in $L\}$
9: Zoom(V, ω, H')

The procedure Zoom takes as input a piece H and a sequence $U \subseteq \lrcorner(Q)$ that is safe for H. In order to compute $\mathsf{VD}(u, Q)$, we call the procedure Zoom$(\lrcorner(Q), \omega, H)$ for each of the $\mathcal{O}(\log N)$ pieces H that we use to cover Q^{out}. Clearly, in each of those initial $\mathcal{O}(\log N)$ calls $\lrcorner(Q)$ is a safe sequence of the respective piece. For each child H' of H in $\mathcal{A}$, we check whether the condition of Lemma 17 is satisfied in lines 4-7. Note that since U is a safe sequence for H, U is also a safe sequence for H' and hence our computation of the set $L = \{s \in \lrcorner(Q) : s_{\searrow} \in H' \setminus \lrcorner(H')\}$ is correct. If L is non-empty, we recurse on H' (cf. line 7). In line 8, we construct a safe sequence V for H', of size proportional to $|L|$ and then, in line 9, we call Zoom(V, ω, H'). In order to prove the correctness of procedure Zoom, it remains to show that V is indeed safe. The following two lemmas show this (see also Figure 6).

▶ **Lemma 21.** *Let U be safe for a piece H, such that* Partition$(U, \omega, \ulcorner(H)) = U$. *Suppose that there are three elements u_1, u_2, u_3 of U that appear consecutively (in this order) in both* Partition$(U, \omega, \ulcorner(H))$ *and* Partition$(U, \omega, \lrcorner(H))$. *Then, $U \setminus \{u_2\}$ is also safe for H.*

Proof. To avoid confusion we denote the Voronoi diagram of Q^{out} with sites U by VD and the one with sites $U \setminus \{u_2\}$ by VD'. We denote the Voronoi cells of VD by $\mathsf{Vor}(\cdot)$, and those of VD' by $\mathsf{Vor}'(\cdot)$. Note that for every $u \in U \setminus \{u_2\}$, $\mathsf{Vor}(u) \subseteq \mathsf{Vor}'(u)$. By Lemma 17, in VD, $u_{1\searrow}, u_{2\searrow}, u_{3\searrow} \notin H \setminus \lrcorner(H)$. Hence, in VD', $u_{1\searrow}, u_{3\searrow} \notin H \setminus \lrcorner(H)$.

▷ Claim. Every vertex y of $\mathsf{Vor}(u_2) \cap H$ belongs either to $\mathsf{Vor}'(u_1)$ or to $\mathsf{Vor}'(u_3)$.

Proof. Consider the last vertex z_1 of $\lrcorner(H)$ that is in $\mathsf{Vor}(u_1)$, and the first vertex z_3 of $\lrcorner(H)$ that is in $\mathsf{Vor}(u_3)$. Let ρ_1 be a shortest u_1-to-z_1 path, and ρ_3 be a shortest u_3-to-z_3 path. Note that all vertices of ρ_1 belong to $\mathsf{Vor}(u_1)$ and all vertices of ρ_3 belong to $\mathsf{Vor}(u_3)$. Consider any vertex y of $\mathsf{Vor}(u_2) \cap H$. The vertex y lies to the right of ρ_1 and to the left of ρ_3. In VD', the vertices of ρ_1 belong to $\mathsf{Vor}'(u_1)$ and the vertices of ρ_3 belong to $\mathsf{Vor}'(u_3)$. Hence, by Lemma 16, in VD', y can only belong to a site $s \neq u_2$ that is weakly between u_1 and u_3. Since u_1, u_2, u_3 appear consecutively in Partition$(U, \omega, \ulcorner(H)) = U$, the only such sites are u_1 and u_3, and the claim follows. ◁

By the above claim, the sets $\{(s, s_{\searrow}) : s$ is a site and $s_{\searrow} \in H \setminus \lrcorner(H)\}$ are identical for VD and VD'. Since U is safe for H, so is $U \setminus \{u_2\}$. ◀

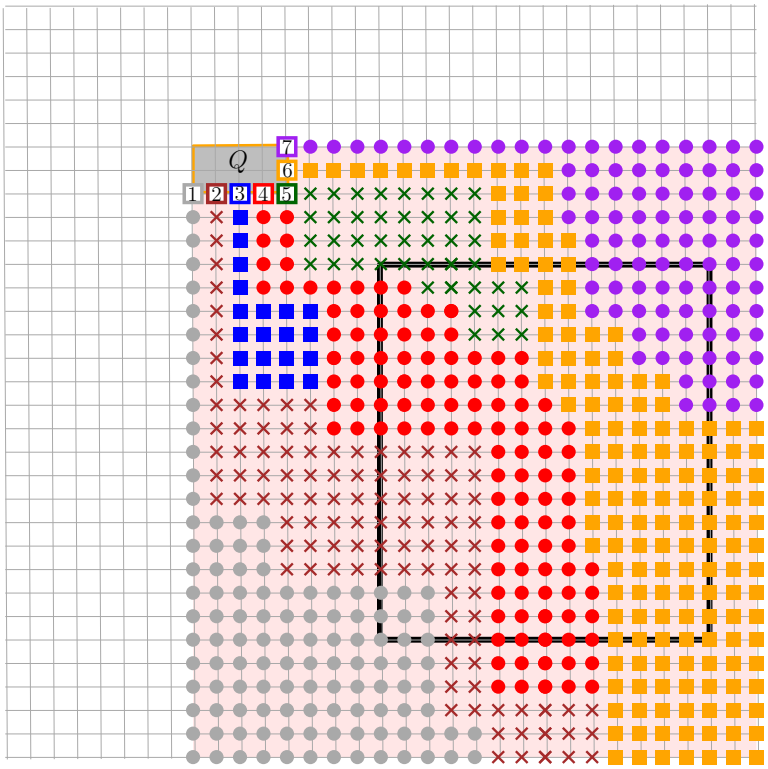


Figure 6 Illustration for Lemma 21. Part of Q^{out} (pink) for some piece Q (gray) is shown. A piece H is indicated by a black rectangle. The sites of $\lrcorner(Q)$ are numbered 1 through 7. The partition of $\ulcorner(H)$ w.r.t. $\lrcorner(Q)$ is $U = (1, 2, 4, 5, 6, 7)$. Hence, U is safe for H. The partition of $\lrcorner(H)$ w.r.t. $\lrcorner(Q)$ is $(1, 2, 4, 6, 7)$. Since sites $1, 2, 4$ are consecutive in both partitions, $(1, 4, 5, 6, 7)$ is also safe for H. Further, $\lrcorner(Q)$ is also clearly safe for H. However, $(1, 3, 4, 5, 6, 7)$ may not be safe for H, as $3_{\searrow}$ could be in $H \setminus \partial H$ in the Voronoi diagram with sites $1, 3, 4, 5, 6, 7$ and the same additive weights.

▶ **Lemma 22.** *Suppose that U is safe for H. Then, in each recursive call $\textsc{Zoom}(V, \omega, H')$ made by procedure $\textsc{Zoom}(U, \omega, H)$ for a child H' of H in $\mathcal{A}$, V is a safe sequence for H'.*

Proof. Since U is a safe sequence for H, U is also a safe sequence for H' (cf. Observation 20). Hence, L is the set of sites s such that $s_{\searrow} \in H' \setminus \lrcorner(H')$.

Z is the set of sites in U whose Voronoi cells have non-empty intersection with H'. In line 8, we remove from Z all vertices of $Z \setminus L$ that are not preceded or succeeded by a vertex in L. Therefore, by considering the removal of these sites one at a time, and directly applying Lemma 21 to each such removal, the resulting sequence V is safe for H'. ◀

This establishes the correctness of our construction algorithm. Let us now analyze its time complexity. Initially, we make $\mathcal{O}(\log N)$ calls to $\textsc{Zoom}(U, \omega, H)$, each with $U = \lrcorner(Q)$. In each recursive call, for a child H' of a piece H, the set U of sites is of size proportional to the size of the set $\{s \in \lrcorner(Q) : s_{\searrow} \in H' \setminus \lrcorner(H')\}$. Note that in each level of the tree $\mathcal{A}$ each $s \in \lrcorner(Q)$ is an element of exactly one such set. Hence, each $s \in \lrcorner(Q)$ contributes to $\mathcal{O}(\log N)$ calls to Zoom: the $\mathcal{O}(\log N)$ initial ones and at most one more per level of $\mathcal{A}$ (which is of depth $\mathcal{O}(\log N)$).

Thus, by Lemma 19, computing $s_{\searrow}$ for all $s \in \lrcorner(Q)$ reduces to $\mathcal{O}(|\lrcorner(Q)| \cdot \log^2 N)$ distance queries $\mathsf{dist}(u, v)$, with $u \in \lrcorner(Q)$ and $v \in Q^{out}$. We can answer each such query with the portion of the oracle that has already been computed in $\mathcal{O}(2^t \log^2 N)$ time. Now, recall that

a $\mathsf{dist}(s, s_{\searrow})$ query also computes $\mathsf{last}(s, s_{\searrow})$, and hence these values can also be retrieved in $\mathcal{O}(2^t \log^2 N)$ time. Thus, $\mathsf{VD}(u, Q)$, which is of size $\mathcal{O}(|\lrcorner(Q)|)$ can be computed in time $\mathcal{O}(|\lrcorner(Q)| \cdot 2^t \log^4 N)$, which is $|\lrcorner(Q)| \cdot N^{o(1)}$ for both choices of t in Proposition 14. Therefore, the time to compute $\mathsf{VD}(u, Q)$ for all pieces is $N^{o(1)} \cdot \sum_Q |\lrcorner(Q)| = N^{o(1)} \cdot \sum_{i=0}^{t-1} \frac{N}{r_i}\sqrt{r_i}$, which is $N^{1+o(1)}$ for both choices of t. This concludes the proof of Theorem 1.

5 Tradeoffs with $o(N)$ space

In this section we prove Theorem 4. Recall that for this result we consider integer alignment weights upper-bounded by w: A weight w_{match} for aligning a pair of matching letters, w_{mis} for aligning a pair of mismatching letters, and w_{del} for letters that are not aligned. One may assume without loss of generality that $2w_{match} > 2w_{mis} \geq w_{del}$ [36]. Given w_{match}, w_{mis} and w_{del}, we define $w'_{match} = 0$, $w'_{mis} = w_{match} - w_{mis}$ and $w'_{del} = \frac{1}{2}w_{match} - w_{del}$. These weights are also upper-bounded by w. Then, a shortest path (of length W) in the alignment grid with respect to the new weights, corresponds to a highest scoring path with respect to the original weights (of score $\frac{1}{2}(m+n)w_{match} - W$).

FR-Dijkstra. We define the *dense distance graph* (DDG) of a piece P as a directed bipartite graph with vertices ∂P and an edge from every vertex $u \in \ulcorner(P)$ to every vertex $v \in \lrcorner(P)$ with weight equal to the length of the shortest u-to-v path in P. We denote this graph as DDG_P.[2] DDG_P can be computed in time $\mathcal{O}((|\partial P|^2 + |P|) \log |P|) = \mathcal{O}(|P| \log |P|)$ using the MSSP data structure. In their seminal paper, Fakcharoenphol and Rao [20] designed an efficient implementation of Dijkstra's algorithm on any union of DDGs – this algorithm is nicknamed FR-Dijkstra. FR-Dijkstra exploits the fact that, due to planarity, the adjacency matrix of each DDG can be decomposed into Monge matrices (defined formally in equation (1) below). In our case, since each DDG is a bipartite graph, the entire adjacency matrix is itself Monge (this will be shown below). Let us now give an interface for FR-Dijkstra that is convenient for our purposes.

▶ **Theorem 23** ([20, 23, 31]). *Dijkstra's algorithm can be run on the union of a set of DDGs with $\mathcal{O}(M)$ vertices in total (with multiplicities) and an arbitrary set of $\mathcal{O}(M)$ extra edges in the time required by $\mathcal{O}(M \log^2 M)$ accesses to edges of this union.*

▶ Remark 24. In our case, the runtime of the algorithm encapsulated in the above theorem can be improved to $\mathcal{O}(M \log\log(nw))$. One of the two $\mathcal{O}(\log M)$ factors stems from the decomposition of the adjacency matrix into Monge submatrices, which is not necessary in our case. The second $\mathcal{O}(\log M)$ comes from the use of binary heaps. In our case, these heaps store integers in $\mathcal{O}(nw)$ and can be thus implemented with $\mathcal{O}(\log\log(nw))$ update and query times using an efficient predecessor structure [38, 39].

A warmup. Let us first show how to construct in $\tilde{\mathcal{O}}(N)$ time an $\tilde{\mathcal{O}}(N)$-size oracle that answers queries in $\tilde{\mathcal{O}}(\sqrt{N})$ time using well-known ideas [20]. We will then improve the size of the data structure by efficiently storing the computed DDGs.

Let us consider an r-division of G, for an r to be specified later. Further, consider the tree $\mathcal{A}'$, obtained from the recursive decomposition tree $\mathcal{A}$ by deleting all descendants of pieces in the r-division. For each piece $P \in \mathcal{A}'$, we compute and store DDG_P. In each of

[2] For general planar graphs, the DDG of a piece is usually defined as a complete directed graph on ∂P.

the $\mathcal{O}(\log N)$ levels of $\mathcal{A}$, for some value y, we have $\mathcal{O}(N/y)$ pieces, each with $\mathcal{O}(y)$ vertices and $\mathcal{O}(\sqrt{y})$ boundary vertices. Hence, both the construction time and the space occupied by these DDGs are $\tilde{\mathcal{O}}(N)$.

We next show how to compute the weight of an optimal alignment of $S[i\,..\,j]$ and $T[a\,..\,b]$, i.e. compute the shortest path ρ from $u = (i-1, a-1)$ to $v = (j, b)$, where $i < j$ and $a < b$. If u and v belong to $P \setminus \partial P$ for a piece P of the r-division, then both $S[i\,..\,j]$ and $T[a\,..\,b]$ are of length $\mathcal{O}(\sqrt{r})$, and we can hence run the textbook dynamic programming algorithm which requires $\mathcal{O}(r)$ time. Henceforth, we consider the complementary case.

Let P_u and P_v be the distinct r-division pieces that contain u and v, respectively. Further, let Q be the lowest common ancestor of P_u and P_v in $\mathcal{A}'$. For $z \in \{u, v\}$, let Q_z be the child of Q that contains z. The set of vertices $Q_u \cap Q_v$ are denoted by $\mathsf{sep}(Q)$ – which stands for separator. Observe that ρ must contain at least one vertex from $\mathsf{sep}(Q)$. Consider the set that consists of P_z and the siblings of weak ancestors of P_z in $\mathcal{A}'$ that are descendants of Q, and call it the *cone of* P_z. The cone of P_z covers Q_z and its elements are pairwise internally disjoint. See Figure 7 for an illustration. Now, observe, that any shortest path ρ between a vertex of ∂P_z and a vertex of $\mathsf{sep}(Q)$ can be partitioned into subpaths $\rho_1, \ldots, \rho_k$ such that each ρ_i lies entirely within some piece R_i in the cone of P_z and both ρ_i's endpoints are boundary vertices of R_i. Using these two observations, we can compute a shortest u-to-v path by running FR-Dijkstra on the cones of P_u and P_v, and, possibly, the following extra edges. In the case where the source u (resp. target v) is not a boundary vertex, we include $\mathcal{O}(\sqrt{r})$ additional edges: for each boundary vertex x of P_u (resp., P_v), an edge from u to x (resp., from x to v) with length equal to that of the shortest path from u to x (resp. from x to v). The weights of such edges can be computed in $\mathcal{O}(r)$ time using dynamic programming. Thus, a query can be answered in time $\tilde{\mathcal{O}}(\sqrt{N} + r)$. By setting $r = \sqrt{N}$ we get the promised complexities.

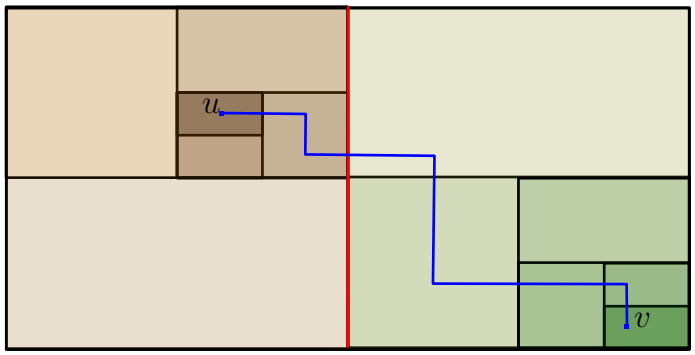


Figure 7 The piece Q is shown. $\mathsf{sep}(Q)$ is denoted by red, while a shortest u-to-v path is shown in blue. The pieces in the cone of P_u are shaded by brown, while the pieces in the cone of P_z are shaded by pink.

The tradeoff. We can now describe the entire tradeoff of Theorem 4. We assume that $r > w^2$, since otherwise $Nw/\sqrt{r} = \Omega(N)$ and Theorem 4 is satisfied by the warmup solution. For a piece P, we will show how to store DDG_P in $\mathcal{O}(|\partial P| \cdot w) = \mathcal{O}(w\sqrt{|P|})$ instead of $\mathcal{O}(|P|)$ space. Our representation will allow retrieving the length of any edge of DDG_P in $\tilde{\mathcal{O}}(1)$ time. Our approach closely follows ideas from [2].

For the remainder, we deviate from our ordering convention of $\ulcorner(P)$; the first vertex is now the top-right vertex of P, and the last is the bottom-left one. $\lrcorner(P)$ is ordered as before where the first vertex is the bottom-left one and the last is the top-right one. We denote

the i-th vertex of $\ulcorner(P)$ by v_i and the j-th vertex of $\lrcorner(P)$ by u_j. Note that we can infer whether any vertex u_j is reachable from a vertex v_i in $\mathcal{O}(1)$ time. For ease of presentation we would like the weights of all edges of DDG_P to be finite. To achieve this, for each edge between two vertices of $\ulcorner(P)$, we introduce (in P) an artificial edge with weight w in the opposite direction. It is readily verified that all v_i-to-u_j distances that were finite before the introduction of such edges remain unchanged. This is because the shortest path between two vertices of this modified graph that lie on the same column (resp. row) consists solely of vertical (resp. horizontal) edges.

Let M be the adjacency matrix of DDG_P with entry $M[i,j]$ storing the distance from v_i to u_j, and let $k = |\ulcorner(P)| = |\lrcorner(P)|$. Matrix M satisfies the Monge property, namely:

$$M[i+1,j] - M[i,j] \leq M[i+1,j+1] - M[i,j+1] \tag{1}$$

for any $i \in [1,k-1]$ and $j \in [1,k-1]$. This is because the shortest v_i-to-u_j and v_{i+1}-to-u_{j+1} paths must necessarily cross.

In addition, for any fixed $j \in [1,k]$, for all $i \in [1,k-1]$, we have

$$|M[i+1,j] - M[i,j]| \leq w. \tag{2}$$

This is because edges v_iv_{i+1} and $v_{i+1}v_i$ both have weight at most w. This implies that $M[i,j] \leq M[i+1,j]+w$, as a shortest v_i-to-u_j path cannot be longer than the concatenation of the edge v_iv_{i+1} with a shortest v_{i+1}-to-u_j path. Similarly, we have $M[i+1,j] \leq M[i,j]+w$.

Our representation of M is as follows, and fairly standard [2, 16, 36]. We define a $(k-1)\times(k-1)$ matrix P, satisfying

$$P[i,j] = M[i,j] + M[i+1,j+1] - M[i,j+1] - M[i+1,j].$$

Equations (1) and (2) imply that, for any $i \in [1,k-1]$, the sequence $M[i+1,j] - M[i,j]$ is nondecreasing and contains only values in $[-w,w]$. Hence, P has $\mathcal{O}(kw)$ non-zero entries. Now, observe that

$$\sum_{r\geq i, c\geq j} P[r,c] = M[i,j] + M[k,k] - M[i,k] - M[k,j]. \tag{3}$$

We store the last row and column of M. By (3), this means that retrieving $M[i,j]$ boils down to computing $\sum_{r\geq i,c\geq j} P[r,c]$. We view the non-zero entries of P as points in the plane and build in $\tilde{\mathcal{O}}(kw)$ time an $\tilde{\mathcal{O}}(kw)$-size 2D-range tree over them [7], which can return $\sum_{r\geq i,c\geq j} P[r,c]$ for any i,j in $\tilde{\mathcal{O}}(1)$ time. The overall space required by our representation of DDG_P is thus $\tilde{\mathcal{O}}(kw) = \tilde{\mathcal{O}}(|\partial P|\cdot w)$, and any entry of M can be retrieved in $\tilde{\mathcal{O}}(1)$ time.

In total, over all $\mathcal{O}(N/r)$ pieces of the r-division, the space required is $\tilde{\mathcal{O}}((N/r)\cdot\sqrt{r}\cdot w) = \tilde{\mathcal{O}}(Nw/\sqrt{r})$. This level dominates the other levels of the decomposition, as the sizes of pieces, as well as their boundaries, decrease geometrically in each root-to-leaf path. Note that, for the dynamic programming part of the query algorithm, we can simply store the strings, which take $\mathcal{O}(m+n)$ space. This concludes the proof of Theorem 4.

References

1. Amir Abboud, Arturs Backurs, and Virginia Vassilevska Williams. Tight hardness results for LCS and other sequence similarity measures. In *IEEE 56th Annual Symposium on Foundations of Computer Science, FOCS*, pages 59–78, 2015.
2. Amir Abboud, Pawel Gawrychowski, Shay Mozes, and Oren Weimann. Near-optimal compression for the planar graph metric. In *29th Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2018*, pages 530–549. SIAM, 2018.
3. Amihood Amir, Panagiotis Charalampopoulos, Solon P. Pissis, and Jakub Radoszewski. Dynamic and internal longest common substring. *Algorithmica*, 82(12):3707–3743, 2020.
4. Maxim Babenko, Paweł Gawrychowski, Tomasz Kociumaka, and Tatiana Starikovskaya. Wavelet trees meet suffix trees. In *26th Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2015*, pages 572–591, 2015.
5. Maxim A. Babenko, Pawel Gawrychowski, Tomasz Kociumaka, Ignat I. Kolesnichenko, and Tatiana Starikovskaya. Computing minimal and maximal suffixes of a substring. *Theoretical Computer Science*, 638:112–121, 2016.
6. Arturs Backurs and Piotr Indyk. Edit distance cannot be computed in strongly subquadratic time (unless SETH is false). *SIAM Journal on Computing*, 47(3):1087–1097, 2018.
7. Jon Louis Bentley. Decomposable searching problems. *Information Processing Letters*, 8(5):244–251, 1979.
8. Karl Bringmann and Marvin Künnemann. Quadratic conditional lower bounds for string problems and dynamic time warping. In *56th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2015*, pages 79–97, 2015.
9. Gerth Stølting Brodal. Partially persistent data structures of bounded degree with constant update time. *Nordic Journal of Computing*, 3(3):238–255, 1996.
10. Sergio Cabello. Subquadratic algorithms for the diameter and the sum of pairwise distances in planar graphs. *ACM Transactions on Algorithms*, 15(2):21:1–21:38, 2019.
11. Sergio Cabello, Erin W. Chambers, and Jeff Erickson. Multiple-source shortest paths in embedded graphs. *SIAM Journal on Computing*, 42(4):1542–1571, 2013.
12. Panagiotis Charalampopoulos. *Data Structures for Strings in the Internal and Dynamic Settings*. PhD thesis, King's College London, 2020.
13. Panagiotis Charalampopoulos, Paweł Gawrychowski, Shay Mozes, and Oren Weimann. Almost optimal distance oracles for planar graphs. In *51st Annual ACM SIGACT Symposium on Theory of Computing, STOC 2019*, pages 138–151, 2019.
14. Panagiotis Charalampopoulos, Tomasz Kociumaka, Manal Mohamed, Jakub Radoszewski, Wojciech Rytter, Juliusz Straszyński, Tomasz Waleń, and Wiktor Zuba. Counting distinct patterns in internal dictionary matching. In *31st Annual Symposium on Combinatorial Pattern Matching, CPM 2020*, pages 8:1–8:15, 2020.
15. Panagiotis Charalampopoulos, Tomasz Kociumaka, Manal Mohamed, Jakub Radoszewski, Wojciech Rytter, and Tomasz Waleń. Internal dictionary matching. In *30th International Symposium on Algorithms and Computation, ISAAC 2019*, pages 22:1–22:17, 2019.
16. Panagiotis Charalampopoulos, Tomasz Kociumaka, and Shay Mozes. Dynamic string alignment. In *31st Annual Symposium on Combinatorial Pattern Matching, CPM 2020*, pages 9:1–9:13, 2020.
17. Panagiotis Charalampopoulos, Tomasz Kociumaka, and Philip Wellnitz. Faster approximate pattern matching: A unified approach. In *61st Annual IEEE Symposium on Foundations of Computer Science, FOCS 2020*, pages 978–989, 2020.
18. Hagai Cohen and Ely Porat. On the hardness of distance oracle for sparse graph. *CoRR*, 2010. `arXiv:1006.1117`.
19. Vincent Cohen-Addad, Søren Dahlgaard, and Christian Wulff-Nilsen. Fast and compact exact distance oracle for planar graphs. In *58th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2017*, pages 962–973, 2017.

20 Jittat Fakcharoenphol and Satish Rao. Planar graphs, negative weight edges, shortest paths, and near linear time. *Journal of Computer and System Sciences*, 72(5):868–889, 2006.

21 Paweł Gawrychowski, Shay Mozes, Oren Weimann, and Christian Wulff-Nilsen. Better tradeoffs for exact distance oracles in planar graphs. In *29th Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2018*, pages 515–529, 2018.

22 Isaac Goldstein, Tsvi Kopelowitz, Moshe Lewenstein, and Ely Porat. Conditional lower bounds for space/time tradeoffs. In *15th International Symposium Algorithms and Data Structures, WADS 2017*, pages 421–436, 2017.

23 Haim Kaplan, Shay Mozes, Yahav Nussbaum, and Micha Sharir. Submatrix maximum queries in Monge matrices and partial Monge matrices, and their applications. *ACM Transactions on Algorithms*, 13(2):26:1–26:42, 2017.

24 Orgad Keller, Tsvi Kopelowitz, Shir Landau Feibish, and Moshe Lewenstein. Generalized substring compression. *Theoretical Computer Science*, 525:42–54, 2014.

25 Philip N. Klein. Multiple-source shortest paths in planar graphs. In *16th Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2005*, pages 146–155, 2005.

26 Tomasz Kociumaka. Minimal suffix and rotation of a substring in optimal time. In *27th Annual Symposium on Combinatorial Pattern Matching, CPM 2016*, pages 28:1–28:12, 2016.

27 Tomasz Kociumaka, Jakub Radoszewski, Wojciech Rytter, and Tomasz Waleń. Efficient data structures for the factor periodicity problem. In *19th International Symposium on String Processing and Information Retrieval, SPIRE 2012*, pages 284–294, 2012.

28 Tomasz Kociumaka, Jakub Radoszewski, Wojciech Rytter, and Tomasz Waleń. Internal pattern matching queries in a text and applications. In *26th Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2015*, pages 532–551, 2015.

29 Gad M. Landau and Uzi Vishkin. Fast parallel and serial approximate string matching. *Journal of Algorithms*, 10(2):157–169, 1989. doi:10.1016/0196-6774(89)90010-2.

30 Yaowei Long and Seth Pettie. Planar distance oracles with better time-space tradeoffs. In *32nd Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2021*, pages 2517–2536, 2021.

31 Shay Mozes and Christian Wulff-Nilsen. Shortest paths in planar graphs with real lengths in $O(n \log^2 n / \log \log n)$ time. In *18th Annual European Symposium on Algorithms, ESA 2010*, pages 206–217, 2010.

32 Mihai Pătraşcu and Liam Roditty. Distance oracles beyond the Thorup-Zwick bound. *SIAM Journal on Computing*, 43(1):300–311, 2014.

33 Mikhail Rubinchik and Arseny M. Shur. Counting palindromes in substrings. In *24th International Symposium on String Processing and Information Retrieval, SPIRE 2017*, pages 290–303, 2017.

34 Yoshifumi Sakai. A substring-substring LCS data structure. *Theoretical Computer Science*, 753:16–34, 2019.

35 Jeanette P. Schmidt. All highest scoring paths in weighted grid graphs and their application to finding all approximate repeats in strings. *SIAM Journal on Computing*, 27(4):972–992, 1998.

36 Alexander Tiskin. Semi-local string comparison: algorithmic techniques and applications, 2007. arXiv:0707.3619.

37 Alexander Tiskin. Semi-local longest common subsequences in subquadratic time. *Journal of Discrete Algorithms*, 6(4):570–581, 2008.

38 Peter van Emde Boas. Preserving order in a forest in less than logarithmic time and linear space. *Information Processing Letters*, 6(3):80–82, 1977.

39 Dan E. Willard. Log-logarithmic worst-case range queries are possible in space $\Theta(N)$. *Information Processing Letters*, 17(2):81–84, 1983.

Faster Algorithms for Rooted Connectivity in Directed Graphs

Chandra Chekuri
University of Illinois at Urbana-Champaign, IL, USA

Kent Quanrud
Purdue University, West Lafayette, IN, USA

Abstract

We consider the fundamental problems of determining the rooted and global edge and vertex connectivities (and computing the corresponding cuts) in *directed* graphs. For rooted (and hence also global) edge connectivity with small integer capacities we give a new randomized Monte Carlo algorithm that runs in time $\tilde{O}\left(n^2\right)$. For rooted edge connectivity this is the first algorithm to improve on the $\Omega(n^3)$ time bound in the dense-graph high-connectivity regime. Our result relies on a simple combination of sampling coupled with sparsification that appears new, and could lead to further tradeoffs for directed graph connectivity problems.

We extend the edge connectivity ideas to rooted and global vertex connectivity in directed graphs. We obtain a $(1+\epsilon)$-approximation for rooted vertex connectivity in $\tilde{O}(nW/\epsilon)$ time where W is the total vertex weight (assuming integral vertex weights); in particular this yields an $\tilde{O}\left(n^2/\epsilon\right)$ time randomized algorithm for unweighted graphs. This translates to a $\tilde{O}(\kappa nW)$ time exact algorithm where κ is the rooted connectivity. We build on this to obtain similar bounds for global vertex connectivity.

Our results complement the known results for these problems in the low connectivity regime due to work of Gabow [8] for edge connectivity from 1991, and the very recent work of Nanongkai et al. [23] and Forster et al. [6] for vertex connectivity.

2012 ACM Subject Classification Theory of computation → Graph algorithms analysis

Keywords and phrases rooted connectivity, directed graph, fast algorithm, sparsification

Digital Object Identifier 10.4230/LIPIcs.ICALP.2021.49

Category Track A: Algorithms, Complexity and Games

Related Version *Full Version*: `https://arxiv.org/abs/2104.07205`

Funding *Chandra Chekuri*: Supported in part by NSF grants CCF-1910149 and CCF-1907937.

Acknowledgements We thank the reviewers for their helpful comments.

1 Introduction

Let $G = (V, E)$ be a simple directed graph; that is, a directed graph with no parallel edges. Recall that G is *strongly connected* if there is a path from any vertex $a \in V$ to any vertex $b \in V$. The *edge connectivity* is the minimum number of edges that need to be removed so that G is *not* strongly connected. The corresponding set of edges is called the *minimum edge cut*. The *vertex connectivity* is the minimum number of vertices that need to be removed so that the remaining graph is not strongly connected or has only one vertex. The corresponding set of vertices is called the *minimum vertex cut*. These problems generalize to weighted settings where the edges and vertices are assigned positive weights and the goal is to find the minimum weight edge or vertex cut. Determining the edge and vertex connectivities and finding the corresponding minimum cuts are among the basic problems in graph algorithms.

This work obtains faster randomized algorithms for finding minimum edge and vertex cuts in directed graphs, especially in the dense setting. The algorithms are based on a simple technique which could be of independent interest.

Our interest is actually in the more general *rooted connectivity* problems. Let $r \in V$ be a fixed vertex, called the *root*. The *r-rooted edge connectivity* is the minimum number of edges that have to be removed so that there is some vertex in $V - r$ that r cannot reach. An algorithm for rooted edge connectivity easily implies an algorithm for edge connectivity, by fixing any root and returning the minimum of the rooted connectivity in G and the rooted connectivity in the graph obtained by reversing all the edges in G. Another important motivation for investigating rooted connectivity is the fundamental result by Edmonds [4] that the r-rooted edge connectivity equals the maximum number of edge-disjoint arboresences rooted at r. We refer the reader to [27, 7] for further connections in combinatorial optimization. Similarly, the *r-rooted vertex connectivity* is the minimum number of vertices (excluding r) that have to be removed so that r cannot reach some vertex in the residual graph. Algorithms for rooted vertex connectivity also lead to global vertex connectivity by a similar but somewhat more involved reduction. There is a long and rich history of developing algorithms for determining the edge and vertex connectivity. We first note that all of these connectivity and cut problems reduce to a polynomial number of (s, t)-cut and flow problems by standard reductions. Beyond (s, t)-flow, an interesting algorithmic landscape emerges with different running times depending on whether we are interested in edge or vertex cuts, directed or undirected graphs, and weighted or unweighted graphs.

Rooted and global edge-connectivity. We first discuss edge connectivity in directed graphs. Let λ denote either the rooted or global edge connectivity of the graph depending on the context. One can compute both via $O(n)$ (s, t)-minimum cut computations. For the simple and unweighted directed graph setting, Mansour and Schieber [21] improved on this and gave algorithms that run in $O(mn)$ time or in $O\big(\lambda^2 n^2\big)$ time for global connectivity. It was also observed by Alon (cf. [21]) that this approach can be parameterized by the minimum out-degree δ^+ to obtain a $O(n \log(\delta^+) \operatorname{EC}(m, n)/\delta^+)$ algorithm, where $\operatorname{EC}(m, n)$ denotes the running time for (s, t)-edge connectivity[1]. Gabow [8] then gave a $O\big(m\lambda \log\big(n^2/m\big)\big)$ for rooted connectivity in graphs with integer capacities. Gabow's algorithm is based on Edmonds' theorem described above. Gabow's algorithm is nearly linear time for sparse unweighted graphs, and remains the fastest algorithm for small λ for both rooted and global edge connectivity. It is interesting that Gabow's algorithm is not based on (s, t)-flow. For directed graphs with arbitrary edge capacities, Hao and Orlin [12] gave an $O\big(mn \log\big(n^2/m\big)\big)$ algorithm for rooted connectivity by adapting the push-relabel max flow algorithm; in fact their algorithm computes the (r, v)-connectivity for all $v \in V - r$. Thereafter there have been no direct running time improvements to rooted or global edge connectivity in directed graphs but we point out that there have been numerous breakthroughs in the running times for (s, t)-flow and connectivity [11, 24, 15, 19, 20, 18, 14, 28, 10]. In particular, starting with the work of Goldberg and Rao [11], the running time for (s, t)-flow is $o(mn)$ which breaks the flow-decomposition barrier. Motivated by these developments and several others, there has been a resurgence of interest and literature on faster graph algorithms for several fundamental problems. Despite these developments there has been no algorithm for rooted

[1] Depending on the context, we let $\operatorname{EC}(m, n)$ denote the running time for (s, t)-cut in either a simple or a weighted directed graph with m edges and n vertices.

Table 1 Running times for finding the minimum cut in unweighted directed graphs (i.e., $U = 1$). $\mathrm{EC}(m, n)$ denotes the running time of computing (s, t)-connectivity (in unweighted graphs). See also [27, §15.3a].

	$O(n\,\mathrm{EC}(m,n))$	Trivial. Also holds for rooted connectivity.
	$O(n\,\mathrm{EC}_\lambda(m,n))$	Matula [22]. Also holds for rooted connectivity.
	$O(mn)$, $O\big(\lambda^2 n^2\big)$	Mansour and Schieber [21]
*	$O\Big(\frac{n\log\delta}{\delta}\,\mathrm{EC}(m,n)\Big)$	Alon (cf. Mansour and Schieber [21]). δ is the minimum out-degree in the graph.
*	$O\big(m\lambda\log\big(n^2/m\big)\big)$	Gabow [8]. Also holds for rooted connectivity.
*	$\tilde{O}\big(n^2\big)$	Theorem 1. Randomized. Also holds for rooted connectivity.

edge-connectivity in simple directed graphs that is faster than $O(n^3)$ in the worst case. In this paper we obtain a nearly quadratic time algorithm which also applies to graphs with small integer capacities.[2]

▶ **Theorem 1.** *Let $G = (V, E)$ be a simple directed graph with m edges n, vertices, and integer edge weights $w : E \to [U]$. Then the minimum rooted r-cut can be computed with high probability in $\tilde{O}\big(n^2U\big)$ randomized time.*

This running time is particularly compelling when the rooted edge connectivity λ is high.

Rooted and global vertex-connectivity. We now consider (rooted) vertex connectivity in directed graphs. It is well known that for fast algorithms, global vertex connectivity is more involved than edge connectivity and the running times are more varied. While the rooted vertex connectivity can be reduced to computing $O(n)$ (s, t)-cuts, the global version, if done naively, would require $\Omega(n^2)$ calls to the (s, t)-cut problem since it is not obvious how to find a vertex that is not part of the minimum global vertex cut. There is a large body of literature and we highlight the leading (randomized) running times, where we state running times for randomized algorithms with high probability of success. Let κ denote the weight minimum vertex cut, where we assume the minimum weight of any vertex is 1. For large κ and general capacities, there is a randomized algorithm by Henzinger et al. [13] (extending the directed edge connectivity algorithm of [12]) that runs in $O(mn\log(n))$ time. For small values of κ in the unweighted setting, recent randomized algorithms by Forster et al. [6] based on *local connectivity* have obtained $\tilde{O}\big(m\kappa^2\big)$ and $\tilde{O}\big(n\kappa^3 + \kappa^{3/2}\sqrt{mn}n\big)$ running times. For more intermediate values of κ, there are also randomized $\tilde{O}\big(\kappa m^{2/3}n\big)$ and $\tilde{O}\big(\kappa m^{4/3}\big)$ time algorithms [23] as well as an $O(n^\omega + n\kappa^\omega)$ time algorithm [3], where $\omega \approx 2.3728596$ is the current exponent for fast matrix multiplication [1]. There is also recent interest in obtaining fast $(1 + \epsilon)$-approximation algorithms for minimum vertex cut [23, 6]. In particular [23] obtains a $O\big(n^\omega/\epsilon^2 + m\min\{\kappa, \sqrt{n}\}\big)$ running time (where ω denotes the exponent for matrix multiplication) and [6] obtains a randomized algorithm with running time $\tilde{O}(m\kappa/\epsilon)$. We obtain the following theorem.

[2] Here and throughout $\tilde{O}(\cdots)$ hides polylogarithmic factors in m and n. We note that the ideas introduced in this work are simple and the logarithmic factors they generate are easy to account for. However Theorem 1 also uses the recent (s, t)-flow algorithm of [28] with running time $\mathrm{EC}(m, n) = \tilde{O}\big(m + n^{1.5}\big)$, which has large logarithmic factors.

Table 2 A table of running times for finding the minimum vertex cut in unweighted directed graphs (i.e., $W = n$). $\mathrm{VC}(m, n)$ denotes the running time of computing (s, t)-vertex connectivity, and is at most $\tilde{O}\left(m + n^{1.5}\right)$ [29]. All randomized algorithms above are correct with high probability. See also [27, §15.2a] and [6].

$O\left(n^2\,\mathrm{VC}(m,n)\right)$	Trivial.
$O(n\,\mathrm{VC}(m,n)\log(n))$	Trivial. Randomized. $\kappa \leq .999n$
$O(\kappa n\,\mathrm{VC}(m,n))$	Podderyugin [25], Even and Tarjan [5]
$O(n^\omega + n\kappa^\omega)$	Cheriyan and Reif [3].
$O(\kappa mn)$, $O\left((\kappa^3+n)m\right)$	Henzinger et al. [13].
$O(mn\log(n))$	Henzinger et al. [13]. Randomized.
$O\left(\min\left\{\kappa^{5/2}, \kappa n^{3/4}\right\}m + mn\right)$	Gabow [9].
$\tilde{O}\left(\kappa m^{2/3}n\right)$, $\tilde{O}\left(\kappa m^{4/3}\right)$	Nanongkai et al. [23]. Randomized.
$\tilde{O}\left(m\kappa^2\right)$, $\tilde{O}\left(n\kappa^3 + \kappa^{3/2}m^{1/2}n\right)$	Forster et al. [6]. Randomized.
$\tilde{O}\left(n^2\kappa\right)$	Corollary 3. Randomized.

▶ **Theorem 2.** *Let $G = (V, E)$ be a directed graph with m edges, n vertices, and integer vertex weights $w : V \to \mathbb{N}$. Let $r \in V$ be a fixed root vertex. Let κ be the rooted vertex connectivity from r. Let $W = \sum_{v\in V} w(v)$ be the total weight of the graph. For any $\epsilon > 0$ a $(1+\epsilon)$-approximate rooted minimum vertex cut can be computed with high probability in $\tilde{O}(m + n(W-\kappa)/\epsilon)$ randomized time; for unit weights this is $\tilde{O}(m + n(n-\kappa)/\epsilon)$. The rooted connectivity can be computed with high probability in $\tilde{O}(m + \kappa n(W-\kappa))$ time.*

Note that $W \geq n$ in the above running times. We point out that the approximation algorithm's running time is independent of κ. This large κ regime has been challenging for previous approaches. The rooted connectivity algorithm, when combined with sampling and other ideas, gives the following theorem for global vertex connectivity. As we remarked, the reduction from global to rooted is not as clean for vertex connectivity as it is for edge connectivity.

▶ **Corollary 3.** *Let $G = (V, E)$ be a directed graph with m edges, n vertices, and integer vertex weights $w : V \to \mathbb{N}$. Let $W = \sum_{v\in V} w(v)$ be the total vertex weight of the graph. Let κ be the global vertex connectivity of G. There is a randomized algorithm that for any $\epsilon > 0$ outputs a $(1+\epsilon)$-approximate minimum vertex cut with high probability in time $\tilde{O}(nW/\epsilon)$. There is a $\tilde{O}(\kappa nW)$ time randomized algorithm that computes the (exact) minimum vertex cut with high probability. In particular, for unit weights, the running time is $\tilde{O}\left(\kappa n^2\right)$.*

1.1 Key ideas

Our algorithms are based on a simple but key idea that we briefly outline below. We focus on the edge-connectivity case since the idea for vertex connectivity is essentially the same with some modifications. We would like to take advantage of recent developments on fast algorithms for (s, t)-cut and reduce to solving a small number of such cut problems in a black box fashion (unlike the approach of [12] based on the properties of a specific flow

algorithm). For undirected graph global connectivity there has been very recent exciting progress by Li and Panigrahi [17] reducing to a *logarithmic* number of (s,t)-cuts. However, the technique makes strong use of the symmetry of the edge-cut function which are absent in the directed graph setting. In a different direction the work of Nanongkai, Saranurak, and Yingchareonthawornchai [23] and follow up improvements by Forster et al. [6], developed fast algorithms for global connectivity based on *local connectivity* and *randomization*. At a high-level they use sampling to identify two vertices s,t on the opposite sides of a cut and then reduce to (s,t)-cut, or they use a local-connectivity algorithm from each vertex $v \in V$. This approach is particularly well-suited for *small* connectivity.

For directed graph edge connectivity Gabow's algorithm with running time $\tilde{O}(m\lambda)$ is very good. In order to beat $O(n^3)$ in the worst case, the bottleneck is the dense graph regime with high connectivity. We have two main ideas that are particularly well suited to this regime. First, we focus on the rooted case even though it may appear to be more difficult than the global connectivity case. The global connectivity can be much smaller than the rooted connectivity; for instance the graph may not be strongly connected, in which case the global edge connectivity is 0, while the rooted connectivity for a particular root can still be $\Omega(n)$. Consider rooted connectivity from a given vertex r. In order to reduce to (s,t)-cut we would like to find a node t such that t is the sink side of a minimum r-cut. Let $T \subseteq V$ be a sink side of a minimum r-cut and hence $\lambda = |\delta^-(T)|$; here $\delta^-(T)$ denotes the set of edges entering T. If $|T|$ is large we can randomly sample a small number of vertices and we will succeed with good probability in finding a vertex from T. Therefore the difficult case is when $|T|$ is small and this is the setting in which we make our key observation: if the graph is *simple* (or edge capacities are small) and the sink side of a minimum r-cut is small (but not a singleton!), then T cannot have a high-degree vertex. How can we take advantage of this? Since we are working with the rooted problem, we can shrink all high-degree vertices into the root r! In other words we can *sparsify* the graph if the sink side is small and compensate for the higher sampling rate (and larger number of (s,t)-cut computations) we need to find a vertex on the sink side. Simple in retrospect, this tradeoff between sparsification and sampling rate coupled with guessing the size of the sink component gives us the overall algorithm with some additional technical work. We believe that our high-level idea will find use in other contexts when combined with other techniques.

Recent related work. Several recent papers have obtained results that are relevant to this work. Li et al. [16] obtained an $\tilde{O}\big(mn^{1-1/12+o(1)}\big)$ time algorithm for vertex connectivity in directed and unweighted graphs. A followup work by one of the authors of this paper obtained $(1+\epsilon)$-approximation algorithms for *weighted* graphs, for rooted and global, edge and vertex connectivity, with $\tilde{O}\big(n^2/\epsilon^{O(1)}\big)$ running times [26]. Improved exact algorithms for weighted edge connectivity have also been obtained very recently in [2].

2 Edge connectivity

In this section, we prove the main theorem for edge connectivity, Theorem 1. To this end, we will first introduce the main key lemma, called the *Rooted Sparsification Lemma*, in Section 2.1. In Section 2.2, we give a lemma that applies the Rooted Sparsification Lemma to give a faster algorithm when the number of vertices in the sink component is known to be in a fixed interval between 1 and n. Theorem 1 is then proven in Section 2.4, applying the ideas from Section 2.2 to each of a small family of intervals.

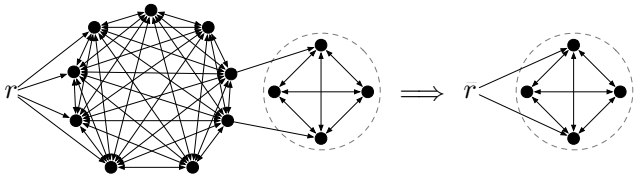


Figure 1 An example of the Rooted Sparsification Lemma in action. In particular, contracting the high in-degree vertices into r leaves the sink component of the minimum r-cut intact.

2.1 The Rooted Sparsification Lemma for Edge Connectivity

We introduce the key technical ingredient that we call the Rooted Sparsification Lemma. This lemma says that if the sink component of the minimum r-cut is small, then unless it is a singleton cut (which is easy to find directly), we can contract all vertices with high in-degree into the root while preserving the minimum rooted cut exactly. The result is a smaller and sparser graph in which we can find the minimum rooted cut faster. Later we will see that the running time saved by operating on a smaller graph makes up for the difficulty in identifying a vertex from a smaller sink component.

▶ **Lemma 4.** *Let $G = (V, E)$ be a simple directed graph with m edges, n vertices, and edge weights $w : E \to [1, U]$. Let $r \in V$ be a fixed root vertex. Let $k \in \mathbb{N}$. Consider the graph $\bar{G}$ obtained by contracting all vertices with weighted in-degree $\geq (1 + U)k$ into r. Let $\bar{r}$ denote the contracted vertex in $\bar{G}$. Then we have the following.*

1. *$\bar{G}$ is a multigraph with less than $(1 + U)nk$ edges.*
2. *If the minimum number of vertices in a sink component of a minimum r-cut has greater than 1 and less than or equal to k vertices, then the minimum r-cut and the minimum $\bar{r}$-cut are the same.*

Note that contraction cannot reduce the value of r-cut. An example illustrating the lemma is given in Figure 1. The proof is in two steps.

Small sinks make small cuts (except for singletons). The first step towards the Rooted Sparsification Lemma for edge connectivity is the following basic observation relating the connectivity to the number of vertices in the sink component of a minimum rooted cut. For simple graphs (i.e., $U = 1$), the following lemma says that *except for the case where the minimum rooted cut is achieved by a singleton*, the rooted connectivity is less than the number of vertices in the sink component of the cut. With capacities between 1 and U, we obtain a similar inequality except scaled by U. See Figure 2 for an illustration of the following lemma.

▶ **Lemma 5.** *Let $G = (V, E)$ be a simple directed graph with m edges, n vertices, and edge weights $w : E \to [1, U]$. Let $r \in V$ be a fixed root vertex. Let λ be the rooted edge connectivity from r. Let k be the minimum number of vertices in a sink component of a minimum r-cut. Then either $k = 1$ or $\lambda < Uk$.*

Proof. Let T be the set of vertices on the sink-side of a cut with λ edges. Suppose $k = |T| > 1$. Every vertex in T has weighted in-degree $> \lambda$. Consider all edges with head in T. Because G has capacities between 1 and U, of all the edges with head in T, at most $k(k-1)U$ total weight have their tail in T as well. Thus $\lambda > k\lambda - k(k-1)U$. Rearranging, we have $k(k-1)U > (k-1)\lambda$, hence $kU > \lambda$. ◀

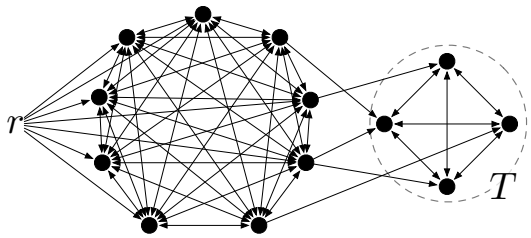


Figure 2 The set of vertices T has 4 vertices and there are 5 edges crossing into T. Lemma 5 implies that T cannot be the sink component of the minimum r-cut. Indeed, there are singleton cuts of degree 4 inside T.

▶ Remark 6. The above argument is simple and (unsurprisingly) we realized that a similar line of reasoning has appeared in previous work [21] (though towards a different algorithmic approach and not in the context of rooted connectivity).

Small sinks are sparse sinks. We now prove the Rooted Sparsification Lemma, Lemma 4. The high level argument is very simple and we first give an informal argument to emphasize the intuition. If the sink component of the minimum r-cut is small, then by Lemma 5, the minimum r-cut is also small. Suppose for the sake of discussion that the graph is simple (i.e., $U = 1$). If both the minimum r-cut and the sink component are small and the graph is simple, then every vertex in the sink component has small in-degree. The contrapositive implies that every high in-degree vertex is on the source side of the cut. Thus the high in-degree vertices can be safely contracted into the root.

Proof of the rooted sparsification lemma. Recalling the statement of the lemma, it is easy to see that contracting all vertices with weighted in-degree $\geq (1+U)k$ into r results in a multigraph $\bar{G}$ in which every vertex has weighted in-degree $< (1+U)k$, and hence there are at most $(1+U)nk$ edges total.

Let T be the sink component of a minimum r-cut. Observe that contracting into r cannot decrease the edge connectivity. If one can show that no vertices in T are contracted into $\bar{r}$, then T is the sink component of a minimum $\bar{r}$-cut as well.

By Lemma 5, the minimum r-cut has size $\lambda < Uk$. Because G is simple and T has $\leq k$ vertices, every vertex in T has in-degree less than $\lambda + k < (1+U)k$. Thus any contracted vertex is outside of T. This completes the proof. ◀

2.2 Rooted connectivity for a fixed range of component sizes

Applying the Rooted Sparsification Lemma usefully requires a fairly tight upper bound on the number of vertices in the sink component of the minimum r-cut. In this section, we assume we are given a lower bound k_1 and an upper bound k_2 on the number of vertices in the sink component, and develop algorithms for the minimum rooted cut in this parametrized regime. The running times are decreasing in k_1 and increasing in k_2; that is, they are better for tighter bounds on the number of vertices in the sink component.

▶ **Lemma 7.** *Let $G = (V, E)$ be a simple directed graph with m edges, n vertices, and edge weights $w : E \to [1, U]$. Let $r \in V$ be a fixed root vertex. Let λ be the rooted edge connectivity from r. Let $k_1, k_2 \in \mathbb{N}$ with $1 \leq k_1 \leq k_2 \leq n$. Suppose the sink component of the minimum r-cut has between k_1 and k_2 vertices. Then the minimum r-cut can be computed with constant probability in*

$$O\left(m + \frac{n}{k_1}(\mathrm{EC}(\min\{m, nk_2U\}, n))\right) \text{ time.}$$

Proof. We first consider the case $k_1 > 1$. By Lemma 4, we can reduce the number of edges to $m' = O(k_2 nU)$ while preserving the r-cut and retaining all k_1 or more vertices in the sink-side component. Let us sample $O(n/k_1)$ sink vertices t in the remaining graph uniformly at random, and compute the minimum (r,t)-cut for each. This takes $\mathrm{EC}(\min\{m,m'\},n) = \mathrm{EC}(\min\{m,k_2nU\},n)$ time for each instance, as desired. With constant probability, at least one sink is sampled out of the sink component of the minimum r-cut, which will return the minimum r-cut.

If $k_1 = 1$, then we must also address the possibility of a singleton cut. We apply the above for $k_1 = 2$ and compare the output to all of the singleton r-cuts, and output the smallest of these cuts. ◀

2.3 Rooted connectivity for small sink components

▶ **Lemma 8.** *Let $G = (V,E)$ be a simple directed graph with m edges, n vertices, and integer edge weights $w : E \to [U]$. Let $r \in V$ be a fixed root vertex. Let $k \in \mathbb{N}$ be a given parameter. There is a deterministic algorithm that runs in $O\big(m + nk^2U^2 \log(\max\{n/kU\})\big)$ time and returns an r-cut with the following guarantee. If the sink component of a minimum r-cut has at most k vertices, then the algorithm will return a minimum r-cut.*

Proof. If the sink side of the minimum cut has less than k vertices, then via Lemma 5, either a singleton induces a minimum r-cut, or the minimum r-cut has size $\lambda < Uk$. For the latter case, we apply the rooted sparsification lemma and reduce the graph to $O(nkU)$ edges while preserving the minimum r-cut. We apply Gabow's algorithm [8] to the sparsified graph and it runs $O\big(nk^2U^2 \log(\max\{1, n/kU\})\big)$ time, and either finds a minimum rooted cut or certifies that the r-cut value in the sparsified graph has value $\geq kU$. We compare the output with all singleton r-cuts. ◀

2.4 Algorithm for rooted edge connectivity

We now prove the main theorem for edge connectivity, Theorem 1. By Lemma 7, if the number of vertices in the sink component is known, then we can reduce very efficiently to (s,t)-connectivity by either sparsifying the graph (if the number is small) or easily guessing a sink (if the number is large). More generally, we can pursue both strategies relative to any given upper and lower bounds on the number of vertices in the sink component. Meanwhile, for small component sizes (that are not singletons), we can still sparsify the graph, while the cut size must be small, which combine to produce fast running times via [8] in Lemma 8. The only unknown is the number of vertices in the sink component. Here we guess the number of vertices up to a constant factor, which only requires enumerating a logarithmic number of guesses. We restate Theorem 1 for the sake of convenience.

▶ **Theorem 1.** *Let $G = (V,E)$ be a simple directed graph with m edges n, vertices, and integer edge weights $w : E \to [U]$. Then the minimum rooted r-cut can be computed with high probability in $\tilde{O}\big(n^2U\big)$ randomized time.*

Proof. Let $\ell \in [n]$ be a parameter to be determined. The sink component of the minimum r-cut either (a) is a singleton, (b) has at most ℓ vertices, or (c) has between 2^i and 2^{i+1} vertices for some $i \geq \lfloor \log \ell \rfloor$. For each of these categories we apply a subroutine and take the minimum of the cut values found.

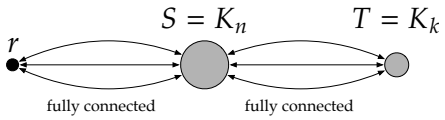
Singleton cuts are easy to evaluate in $O(m)$ time. Let $i_0 = \lfloor \log \ell \rfloor$ and $i_1 = \max\{\lceil \log m/nU \rceil, i_0 + 1\}$. For $i = i_0, \ldots, i_1 - 1$, let $k_i = 2^i$. Let $k_{i_1} = n$. For (b) we invoke Lemma 8 with maximum sink component k_{i_0}. To address (c), for each $i = i_0, \ldots, i_1 - 1$, we invoke Lemma 7 $O(\log n)$ times with lower bound k_i and upper bound k_{i+1} on the number of vertices in the sink component. We use $\mathrm{EC}(m, n) = \tilde{O}\left(m + n^{1.5}\right)$ [28]. The combined running time is $\tilde{O}\left(n^2 U + \frac{n^{2.5}}{\ell} + n\ell^2 U^2\right)$. For $\ell = \sqrt{n}/U$, this gives the claimed running time. ◀

3 Rooted and global vertex connectivity

In this section, we describe and analyze the approximation algorithms for rooted and global vertex connectivity. The high-level approach is similar to the previously discussed algorithm for edge connectivity. The first step, Lemma 10, is a variant of the Rooted Sparsification Lemma that applies to (approximate) vertex connectivity. It plays a similar role as its counterpart for edge connectivity, allowing one to sparsify the graph when the sink component of the minimum rooted vertex cut is small. The proof of Lemma 10 is given Section 3.1. We then give an algorithm specific to (roughly) the number of vertices in the sink component in Section 3.2. We use this algorithm as a subroutine in the final algorithm for approximate rooted connectivity in Section 3.4. In Section 3.5, we give the reduction from approximate global vertex connectivity to approximate rooted vertex connectivity. The exact global vertex connectivity algorithm for integer weights follows from an appropriate choice of error parameter.

3.1 Rooted sparsification for approximate vertex connectivity

Recall that a key idea in the algorithm for (rooted) edge connectivity was the Rooted Sparsification Lemma, which allows us to substantially decrease the number of edges when the sink component of the minimum rooted cut is small. Underlying the rooted sparsification lemma for edge connectivity was a direct relation between the size of the sink component and the weight of the minimum edge cut – Lemma 5 in Section 2.1. But this relation does not hold for vertex connectivity, even in unweighted and undirected graphs – even if the sink component is small, the vertex in-cut can be very large. For example, for arbitrarily large n and any fixed constant k, let $S = K_n$ be a clique of size n and let $T = K_k$ be a clique of size k. Add edges between all $s \in S$ and all $t \in T$. Let r be an additional root vertex connected to every vertex in S. Then T is the sink component of the minimum vertex r-cut. It has a constant number of vertices, k, while the size of the vertex cut, n, is arbitrarily large.



That said, we show that a useful sparsification is possible if we relax to *approximating* the rooted vertex connectivity, and qualify the lemma by the assumption that no singleton cut already represents a good approximation. To this end, let $u, v \in V$. We say that u is an *in-neighbor* of v if $(u, v) \in E$. We denote the set of in-neighbors of a vertex v by $N^-(v) \stackrel{\text{def}}{=} \{u \in V : (u, v) \in E\}$. The definition of in-neighbors naturally extends to sets of vertices; for $S \subset V$ we define $N^-(S) = (\cup_{v \in S} N^-(v)) \setminus S$. The *weighted in-degree* of v is defined as the total weight of all in-neighbors of v. Similarly we define the set of *out-neighbors* of a vertex v, denoted $N^+(v)$, as $N^+(v) \stackrel{\text{def}}{=} \{u \in V : (u, v) \in E\}$, and the weighted out-degree of v, denoted $\deg^+(v)$, as the sum of weights over $N^+(v)$.

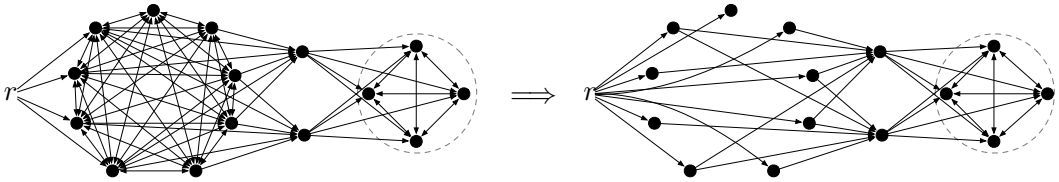


Figure 3 An example of the Rooted Sparsification Lemma for vertex connectivity in action. In the input graph on the left, minimum vertex r-cut has size 2 and the sink component (circled) has 4 vertices. The minimum in-degree (other than r) is 5. On the right hand side, all vertices with in-degree ≥ 9 have all their incoming edges replaced with a single edge from r. The minimum vertex r-cut is again 2 and the sink-component of the minimum r-cut remains unchanged.

Our first lemma gives an approximate relationship between the weight of the minimum weight rooted vertex cut and the weight of the sink component of the minimum weight rooted vertex cut.

▶ **Lemma 9.** *Let $\epsilon > 0$ be fixed. Let $G = (V, E)$ be a directed graph with m edges, n vertices, and vertex weights $w : V \to [1, \infty)$. Let $r \in V$ be a fixed root vertex. Let κ be the rooted vertex connectivity from r. Suppose the in-neighborhood of every non-root vertex has total weight greater tha $(1 + \epsilon)\kappa$. Then the minimum vertex r-cut has more than $\epsilon\kappa$ weight in the sink component.*

Proof. Let the minimum r-cut be of the form $N^-(S)$, where $S \subseteq V - r$. To prove the claim it suffices to show that $w(S) > \epsilon\kappa$.

For any vertex $v \in S$, by assumption, total weight of in-neighbors is more tha $(1 + \epsilon)\kappa$. At most κ weight of these in-neighbors are in the minimum vertex r-cut, $N^-(S)$. This implies that v has more than $\epsilon\kappa$ weight of in-neighors in S, and hence $\sum_{s \in S} w(s) > \epsilon\kappa + 1$ (where the extra 1 is for the weight of v). ◀

▶ **Lemma 10.** *Let $\epsilon > 0$ be fixed. Let $G = (V, E)$ be a directed graph with m edges, n vertices, and vertex weights $w : V \to [1, \infty)$. Let $r \in V$ be a fixed root vertex. Let κ be the rooted vertex connectivity from r. Suppose every vertex (excluding r) has weighted in-degree greater than $(1 + \epsilon)\kappa$. Let $k \in \mathbb{N}$. Consider the graph $\bar{G}$ obtained by replacing, for each vertex $v \in V$ with weighted in-degree $\geq (1 + 1/\epsilon)k$, all of the in-coming edges to v with a single edge from r to v. Then we have the following.*

1. *$\bar{G}$ has maximum weighted in-degree at most $(1 + 1/\epsilon)nk$.*
2. *$\bar{G}$ has at most $(1 + 1/\epsilon)nk$ edges.*
3. *If the sink component of a minimum vertex r-cut in G has weight $\leq k$, then the minimum vertex r-cut in G and $\bar{G}$ are the same.*

Proof. Let T be the sink component of a minimum rooted r-vertex cut, of minimum weight among such sink components. Suppose T has weight less than or equal to k. By Lemma 9, $\kappa < k/\epsilon$. Therefore any vertex v with weighted in-degree greater than $(1 + 1/\epsilon)k$ cannot be in T of the minimum rooted r-vertex cut. We claim that replacing the incoming edges to v does not decrease rooted vertex connectivities for r. As a thought experiment, suppose we make the replacement over two steps, where we first add the edge from r to v, and then remove the other incoming edges to v. The first step does not decrease vertex connectivities, and forces the rooted vertex connectivity from r to v to be $+\infty$. Removing the other incoming

edges to r does not effect the connectivity from r to v, so no other vertex connectivities from r are effected either. Over the two steps, then, we see that no rooted connectivities from r decrease.

On the other hand, since v is not in the sink of the minimum vertex r-cut, the rooted vertex connectivity of r does not change. ◀

3.2 Rooted vertex connectivity parametrized by sink component size

We now give an algorithm for rooted vertex connectivity parametrized by the weight of the vertices in the sink component of the minimum rooted cut. More precisely, we take as additional input two weights $k_1 \leq k_2$ and assume the sink component has weight between k_1 and k_2.

In the following, let $\mathrm{VC}(m, n)$ be the running time for vertex (s, t)-cut.

▶ **Lemma 11.** *Let $\epsilon > 0$ be fixed. Let $G = (V, E)$ be a directed graph with m edges, n vertices, and vertex weights $w : V \to [1, \infty)$. Let $r \in V$ be a fixed root vertex. Let κ be the rooted vertex connectivity from r. Let $W = \sum_{v \in V} w(v)$ be the total weight in the graph. Suppose every vertex (excluding r) has weighted in-degree greater than $(1 + \epsilon)\kappa$. Let $k_1, k_2 \in \mathbb{N}$ with $0 < k_1 < k_2$. Suppose also that the sink component of the minimum r-cut has between k_1 and k_2 total weight. Then the minimum r-cut can be computed with constant probability in*

$$O\left(m + \left(\frac{W - \kappa}{k_1}\right) \mathrm{VC}\left(\min\left\{m, \frac{k_2 n}{\epsilon}\right\}, n\right)\right)$$

randomized time.

Proof. By Lemma 10, in $O(m)$ time, we can reduce the number of edges to at most $O(k_2 n/\epsilon)$ without decreasing the rooted vertex connectivity. We sample vertices from $V' = V \setminus (\{r\} \cup N^+(r))$. Note that V' has weight at most $W - \deg^+(r) \leq W - \kappa$.

We sample $O\big((W - \deg^+(r))/k_1\big) \leq O((W - \kappa)/k_1)$ vertices $t \in V'$ independently in proportion to their weight. For each sampled vertex t, we compute the minimum (r, t)-vertex cut in the sparsified graph. With constant probability, one of these vertices t is in the sink component of the minimum vertex r-cut, and the minimum vertex (r, t)-cut is the minimum vertex r-cut. ◀

▶ Remark 12. The simple observation that the weight of $N^+(r)$ is at least κ is from [13].

3.3 Rooted vertex connectivity for small sink components

This section develops an approximation algorithm for rooted vertex connectivity specifically for the case where the sink component has small weight. The algorithm takes an upper bound k on the weight of the sink component, and guarantees an approximate minimum cut when there is a minimum rooted vertex cut where the sink component has weight at most k. The approach is inspired by the recent local connectivity algorithm of [6], and also integrates the rooted sparsification lemma. This algorithm is developed in two steps. The first step is a local cut algorithm that, given a vertex $t \in V$, searches for a small (r, t)-cut around t in time proportional to a given upper bound on the weight of the sink component. The second step first applies the rooted sparsification lemma, finds a vertex t in the sink component by random sampling, and runs the local cut algorithm for this choice of t.

The following lemma, which describes the local cut algorithm, is nearly identical to [6] except for two small modifications. First, we work with integral capacities, which does not change any arguments. Second is the inclusion of the root r which we want to keep on the opposite side of the local cut. The proof is included for the sake of completeness. In the following, the *in-volume* of a set of vertices T in a directed, edge-capacitated graph is the sum of weighted in-degrees over all vertices in T. Similarly the out-volume is defined as the sum of weighted out-degrees.

▶ **Lemma 13.** *Let $G = (V, E)$ be a directed graph with integral edge capacities. We assume that G is already available in memory in adjacency list format. Let $r, t \in V$, and let $\lambda, \ell > 0$ be given parameters. Then there is a randomized algorithm that runs in $O(\ell\lambda/\epsilon)$ time with the following guarantee.*

Let λ^* be the minimum capacity of all (r, t) cuts where the sink component has in-volume at most ℓ. If $\lambda^* < \lambda$, then with constant probability, the algorithm returns an (r, t)-cut of capacity at most $(1 + \epsilon)\lambda^*$.

Proof. Let T be the sink component of a minimum (r, t)-edge cut among those where the sink component has in-volume at most ℓ. We run a randomized variation of augmenting paths in the reversed graph G_{rev} where t is the source. Note that T now has out-volume at most ℓ in G_{rev}. We run the following subroutine for at most $1 + (1 + \epsilon)\lambda$ iterations, where each iteration routes one unit of flow from t to some chosen v.

Each iteration i runs DFS from t in the residual graph, until it either (a) visits r, (b) has traversed edges of total capacity at least $O(\ell/\epsilon)$, or (c) has explored all the edges reachable from t while failing to satisfy either (a) or (b). In event (a), we route one unit of flow to r. In event (b), we select one of the visited edges randomly in proportion to their capacity, and route one unit of flow to the endpoint of that edge. In either case, after routing, we update the residual graph by reverse (one unit of capacity) of each edge on the path from t to the selected sink. In event (c), we return the entire component of vertices reachable from t which induces an (r, t)-cut in the original graph. If, after $1 + (1 + \epsilon)\lambda$ iterations, we never reach event (c), then the algorithm terminates with failure.

We first argue that we return a $(1 + \epsilon)$-approximate (r, t)-cut with constant probability. We first point out that the total out-volume of T in the residual graph never increases, as we are reversing edges along edges along a path starting from t. Next, we observe that in each instance of event (b), where we randomly sample the endpoint of a visited edge as a sink, there is less than $\epsilon/2$ probability that this endpoint lies in T. This is because the graph search has traversed a total capacity of at least $O(\ell/\epsilon)$, and T has out-volume at most ℓ. That is, the out-volume of T represents at most an $(\epsilon/2)$-fraction of the searched edges.

Now, over the first $(1 + \epsilon)\lambda^*$ iterations, we expect to sample less than $\epsilon\lambda^*/2$ sinks from T. By Markov's inequality, we sample less than $\epsilon\lambda^*$ sinks from T over the first $(1 + \epsilon)\lambda^*$ iterations with probability at least $1/2$. In this event, if the algorithm did not find an (r, t)-cut within the first λ^* iterations, then we must have routed more than λ^* units of flow out of T – a contradiction. Thus the algorithm finds an (r, t)-cut within $(1 + \epsilon)\lambda^*$ iterations with probability at least $1/2$. Since this cut was obtained as the reachable set of t after routing at most $(1 + \epsilon)\lambda^*$ units of flow, the cut has capacity $\leq (1 + \epsilon)\lambda^*$.

It remains to prove the running time. Each iteration takes $O(\ell/\epsilon)$ time to traverse at most $O(\ell/\epsilon)$ edges. The algorithm runs for at most $O(\lambda)$ iterations. ◀

The next lemma presents the approximate rooted vertex cut algorithm that uses Lemma 13 as a subroutine. It also uses the rooted sparsification lemma to reduce the size of the graph and give stronger bounds on the volume of the sink component of the desired vertex cut.

▶ **Lemma 14.** *Let $\epsilon > 0$ be fixed. Let $G = (V, E)$ be a directed graph with m edges, n vertices, and integer vertex weights $w : V \to \mathbb{N}$. Let $r \in V$ be a fixed root vertex. Let $k \in \mathbb{N}$ and suppose that the sink component of the minimum r-cut has weight $\leq k$. Then a $(1+\epsilon)$-approximate minimum r-cut can be computed with high probability in $O\big(m + (W-\kappa)k^2 \log(n) \log(k)/\epsilon^3\big)$ randomized time.*

Proof. By Lemma 9, either a $(1+\epsilon)$-approximate minimum cut is induced by a singleton, or the minimum r-vertex cut has weight at most $O(k/\epsilon)$. The former is addressed by inspecting all singleton cuts. For the rest of the proof, let us assume the latter. By Lemma 10, we can sparsify the graph to have maximum weighted in-degree $O(k/\epsilon)$, hence at most $O(nk/\epsilon)$ total edges.

Let T be the sink component of the minimum r-cut, which has total vertex weight at most $O(k)$, and induces an r-cut with capacity $\kappa \leq O(k/\epsilon)$. Recall the standard auxiliary "split-graph" where vertex capacities are modeled as edge capacities. The high-level idea is to find a vertex $t \in T$ by random sampling and then apply Lemma 13 to the appropriate auxiliary vertices of r and t in the split graph.

To this end, we first bound the volume of the sink-component corresponding to T in the split-graph. We recall that the split graph splits each vertex v into an auxiliary "in-vertex" v^- and an auxiliary "out-vertex" v^+. For each v there is a new edge (v^-, v^+) with capacity equal to the vertex capacity of v. Each edge (u, v) is replaced with an edge (u^+, v^-) with capacity[3] equal to the vertex capacity of u. As a sink component, T maps to a vertex set T' in the split-graph consisting of (a) both copies v^- and v^+ of each vertex $v \in T$, and (b) the out-vertex v^+ of each vertex v in the vertex in-cut $N^-(T)$. For each vertex $v \in T$, v^- has (edge-)weighted in-degree equal to the vertex-weighted in-degree of v in the original graph, which is at most $O(k/\epsilon)$. This sums to $O(|T|k/\epsilon)$ over all $v \in T$. For each $v \in T$, v^+ has weighted in-degree equal to the vertex weight of T, which sums to the total vertex weight of T. Lastly, for each $v \in N^-(T)$, v^+ has weighted in-degree equal to the vertex weight of v. This sums to $\kappa \leq O(k/\epsilon)$ over all $v \in N^-(T)$. All summed up, the total in-volume of T' in the split-graph is at most $O(k/\epsilon)$ times the total vertex weight of T.

Suppose we had a constant factor estimate ℓ for the total vertex weight of T. Then we can sample $O\big((W - \deg^+(r)) \log(n)/\ell\big) \leq O((W-\kappa)\log(n)/\ell)$ vertices by weight from $V \setminus (\{r\} \cup N^+(r))$. With high probability, we sample $O(\log n)$ vertices from T. For each sampled vertex t we invoke Lemma 13 to find an (r,t)-cut, with upper bound $O(\ell k/\epsilon)$ on the volume of the sink component and $O(k/\epsilon)$ as the upper bound on the cut. With high probability, one of these calls returns a $(1+\epsilon)$-approximate cut. The total time, over all calls, would be $O\big((W-\kappa)\log(n)k^2/\epsilon^3\big)$.

Of course, we do not know the vertex weight of T *a priori*. However, we know that it is upper bounded by k, and let ℓ enumerate all powers of 2 between 1 and k. For each ℓ, run the process described above under the hypothesis that ℓ is a constant factor estimate for the total vertex weight of T. Each choice of ℓ takes $O\big((W-\kappa)\log(n)k^2/\epsilon^3\big)$ time. There are $O(\log(k))$ choices of ℓ. One of these choices of ℓ is a constant factor for the total volume of T and produces a $(1+\epsilon)$-approximate minimum (r,t)-cut with high probability. ◀

3.4 Rooted vertex connectivity

We now present the algorithm for approximate rooted vertex connectivity and prove Theorem 2. The algorithm combines the subroutine in Lemma 11 for logarithmically many ranges of weights, and Lemma 14 for sufficiently small weights. We restate Theorem 2 for the sake of convenience.

[3] Usually, this edge is set to capacity ∞, but either the weight of u or the weight of v are also valid.

▶ **Theorem 2.** *Let $G = (V, E)$ be a directed graph with m edges, n vertices, and integer vertex weights $w : V \to \mathbb{N}$. Let $r \in V$ be a fixed root vertex. Let κ be the rooted vertex connectivity from r. Let $W = \sum_{v \in V} w(v)$ be the total weight of the graph. For any $\epsilon > 0$ a $(1+\epsilon)$-approximate rooted minimum vertex cut can be computed with high probability in $\tilde{O}(m + n(W-\kappa)/\epsilon)$ randomized time; for unit weights this is $\tilde{O}(m + n(n-\kappa)/\epsilon)$. The rooted connectivity can be computed with high probability in $\tilde{O}(m + \kappa n(W-\kappa))$ time.*

Proof. Let $\kappa_0 = \epsilon\sqrt{n}$. Let $i_0 = \lfloor \log \kappa_0 \rfloor$, and let $i_1 = \max\{\lceil \log \epsilon m/n \rceil, i_0 + 1\}$ For each $i = \lfloor \log \kappa_0 \rfloor, \lfloor \log \kappa_0 \rfloor + 1, \ldots, i_1 - 1$, let $k_i = 2^i$. Let $k_{i_1} = W - \deg^+(r)$ where we recall that $\deg^+(r)$ is the weighted out-degree of r. For each i, we apply Lemma 11 with lower bound k_i and upper bound k_{i+1} on the weight of the sink component of the minimum vertex r-cut. We repeat this subroutine $O(\log n)$ times for each i to amplify the success probability from constant to high probability. We use $\mathrm{VC}(m, n) = \tilde{O}\big(m + n^{1.5}\big)$ [29]. We also apply Lemma 14 with $\epsilon\kappa_0$ has an upper bound on the sink component size. The set of all cuts obtained by these methods includes a $(1+\epsilon)$-approximate minimum r-cut with high probability, and we return the minimum of these cuts. The combined running time is

$$\tilde{O}\left(m + \frac{(W-\kappa)n}{\epsilon} + \frac{(W-\kappa)n^{1.5}}{\kappa_0} + (W-\kappa)\kappa_0^2/\epsilon^3\right) \leq \tilde{O}(m + (W-\kappa)n/\epsilon),$$

as desired. The exact bound follows by first using the approximation algorithm to obtain a constant factor estimate for κ, and then setting setting $\epsilon \leq 1/\kappa$. ◀

3.5 Global vertex connectivity

We now shift to global vertex connectivity and prove Corollary 3, which we address by reduction to the algorithm for rooted vertex connectivity above. We note that obtaining a root is slightly non-trivial because many vertices may be in the minimum weight vertex cut. We restate Corollary 3 for the sake of convenience.

▶ **Corollary 3.** *Let $G = (V, E)$ be a directed graph with m edges, n vertices, and integer vertex weights $w : V \to \mathbb{N}$. Let $W = \sum_{v \in V} w(v)$ be the total vertex weight of the graph. Let κ be the global vertex connectivity of G. There is a randomized algorithm that for any $\epsilon > 0$ outputs a $(1+\epsilon)$-approximate minimum vertex cut with high probability in time $\tilde{O}(nW/\epsilon)$. There is a $\tilde{O}(\kappa nW)$ time randomized algorithm that computes the (exact) minimum vertex cut with high probability. In particular, for unit weights, the running time is $\tilde{O}\big(\kappa n^2\big)$.*

Proof. Let κ denote the global vertex connectivity. If we sample a single vertex r in proportion to its weight, then with probability $1 - \kappa/W$, r is not in the minimum vertex cut. Then either the rooted vertex connectivity from r, or to r (i.e., from r in the graph G' with all the edges reversed), will give the rooted vertex cut. In principle we would like to apply Theorem 2 with root r in both orientations, which conditional on r not being in the minimum cut, succeeds with high probability. We amplify by repeating $L = O(\frac{W}{W-\kappa} \log n)$ times to obtain the high probability bound. Observe that the running time, via Theorem 2, is

$$\tilde{O}(mL + nW/\epsilon).$$

We would like to remove the mL factor.

To this end, observe that the m term arises from applying the rooted sparsification lemma for various estimates k of the weight of the sink component. Recall that for fixed k and ϵ, the sparsification lemma replaces, for every vertex v with in-degree $> O(k/\epsilon)$, all the incoming edges to v with a single edge from the root. Note that much of the sparsification lemma can be executed without r. In particular, we can remove all incoming edges to the high in-degree vertices without knowing r; once r is given, we add an edge from r to each of these vertices. The key point is that the first part, which takes $O(m)$ time, can be done once for all L sampled roots for each value of k. Thereafter, each of the L roots takes $O(n)$ to complete the sparsification for that root. This replaces the $\tilde{O}(mL)$ term with $\tilde{O}(nL)$, which is dominated by $\tilde{O}(nW/\epsilon)$.

For the exact algorithm, we first apply the approximation algorithm with $\epsilon = 1/2$ obtain a factor-2 approximation to κ within the claimed running time. We then apply the approximation algorithm again with $1/(2\kappa) \leq \epsilon \leq 1/\kappa$. ◀

References

1. Josh Alman and Virginia Vassilevska Williams. A refined laser method and faster matrix multiplication. In Dániel Marx, editor, *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms, SODA 2021, Virtual Conference, January 10 - 13, 2021*, pages 522–539. SIAM, 2021.
2. Ruoxu Cen, Jason Li, Danupon Nanongkai, Debmalya Panigrahi, and Thatchaphol Saranurak. Minimum cuts in directed graphs via $\sqrt{n}$ max-flows. *CoRR*, abs/2104.07898, 2021. `arXiv:2104.07898`.
3. Joseph Cheriyan and John H. Reif. Directed s–t numberings, rubber bands, and testing digraph k-vertex connectivity. *Comb.*, 14(4):435–451, 1994.
4. Jack Edmonds. Submodular functions, matroids, and certain polyhedra. In R. Guy, H. Hanani, N. Sauer, and J. Schönheim, editors, *Combinatorial Structures and Their Applications (Proceedings Calgary International Conference on Combinatorial Structures and Their Applications, Calgary, Alberta, 1969; , eds.)*, pages 69–87. Gordon and Breach, New York, 1970.
5. Shimon Even and Robert Endre Tarjan. Network flow and testing graph connectivity. *SIAM J. Comput.*, 4(4):507–518, 1975.
6. Sebastian Forster, Danupon Nanongkai, Liu Yang, Thatchaphol Saranurak, and Sorrachai Yingchareonthawornchai. Computing and testing small connectivity in near-linear time and queries via fast local cut algorithms. In Shuchi Chawla, editor, *Proceedings of the 2020 ACM-SIAM Symposium on Discrete Algorithms, SODA 2020, Salt Lake City, UT, USA, January 5-8, 2020*, pages 2046–2065. SIAM, 2020.
7. András Frank. *Connections in Combinatorial Optimization*. Oxford University Press, 2011.
8. Harold N. Gabow. A matroid approach to finding edge connectivity and packing arborescences. *J. Comput. Syst. Sci.*, 50(2):259–273, 1995.
9. Harold N. Gabow. Using expander graphs to find vertex connectivity. *J. ACM*, 53(5):800–844, 2006.
10. Yu Gao, Yang P. Liu, and Richard Peng. Fully dynamic electrical flows: Sparse maxflow faster than goldberg-rao. *CoRR*, abs/2101.07233, 2021. `arXiv:2101.07233`.
11. Andrew V. Goldberg and Satish Rao. Beyond the flow decomposition barrier. *J. ACM*, 45(5):783–797, 1998.
12. Jianxiu Hao and James B. Orlin. A faster algorithm for finding the minimum cut in a directed graph. *J. Algorithms*, 17(3):424–446, 1994.
13. Monika Rauch Henzinger, Satish Rao, and Harold N. Gabow. Computing vertex connectivity: New bounds from old techniques. *J. Algorithms*, 34(2):222–250, 2000.
14. Tarun Kathuria, Yang P. Liu, and Aaron Sidford. Unit capacity maxflow in almost $o(m^{4/3})$ time. In *61st IEEE Annual Symposium on Foundations of Computer Science, FOCS 2020, Durham, NC, USA, November 16-19, 2020*, pages 119–130. IEEE, 2020.

15 Yin Tat Lee and Aaron Sidford. Path finding methods for linear programming: Solving linear programs in $\tilde{O}(\sqrt{\text{rank}})$ iterations and faster algorithms for maximum flow. In *55th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2014, Philadelphia, PA, USA, October 18-21, 2014*, pages 424–433. IEEE Computer Society, 2014.

16 Jason Li, Danupon Nanongkai, Debmalya Panigrahi, Thatchaphol Saranurak, and Sorrachai Yingchareonthawornchai. Vertex connectivity in poly-logarithmic max-flows. To appear in ACM STOC„ 2021.

17 Jason Li and Debmalya Panigrahi. Deterministic min-cut in poly-logarithmic max-flows. In *IEEE 61st Annual Symposium on Foundations of Computer Science, FOCS 2020*. IEEE Computer Society, 2020.

18 Yang P. Liu and Aaron Sidford. Faster energy maximization for faster maximum flow. In Konstantin Makarychev, Yury Makarychev, Madhur Tulsiani, Gautam Kamath, and Julia Chuzhoy, editors, *Proccedings of the 52nd Annual ACM SIGACT Symposium on Theory of Computing, STOC 2020, Chicago, IL, USA, June 22-26, 2020*, pages 803–814. ACM, 2020.

19 Aleksander Madry. Navigating central path with electrical flows: From flows to matchings, and back. In *54th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2013, 26-29 October, 2013, Berkeley, CA, USA*, pages 253–262. IEEE Computer Society, 2013.

20 Aleksander Madry. Computing maximum flow with augmenting electrical flows. In Irit Dinur, editor, *IEEE 57th Annual Symposium on Foundations of Computer Science, FOCS 2016, 9-11 October 2016, Hyatt Regency, New Brunswick, New Jersey, USA*, pages 593–602. IEEE Computer Society, 2016.

21 Yishay Mansour and Baruch Schieber. Finding the edge connectivity of directed graphs. *J. Algorithms*, 10(1):76–85, 1989.

22 David W. Matula. Determining edge connectivity in $o(nm)$. In *28th Annual Symposium on Foundations of Computer Science, Los Angeles, California, USA, 27-29 October 1987*, pages 249–251. IEEE Computer Society, 1987. doi:10.1109/SFCS.1987.19.

23 Danupon Nanongkai, Thatchaphol Saranurak, and Sorrachai Yingchareonthawornchai. Breaking quadratic time for small vertex connectivity and an approximation scheme. In *Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing*, pages 241–252, 2019.

24 James B. Orlin. Max flows in $o(mn)$ time, or better. In Dan Boneh, Tim Roughgarden, and Joan Feigenbaum, editors, *Symposium on Theory of Computing Conference, STOC'13, Palo Alto, CA, USA, June 1-4, 2013*, pages 765–774. ACM, 2013.

25 V. D. Podderyugin. An algorithm for finding the edge connectivity of graphs. *Vopr. Kibern.*, 2:136, 1973.

26 Kent Quanrud. Fast approximations for rooted connectivity in weighted directed graphs. *CoRR*, abs/2104.06933, 2021. arXiv:2104.06933.

27 Alexander Schrijver. *Combinatorial Optimization - Polyhedra and Efficiency*. Springer, 2003.

28 Jan van den Brand, Yin Tat Lee, Yang P. Liu, Thatchaphol Saranurak, Aaron Sidford, Zhao Song, and Di Wang. Minimum cost flows, mdps, and ℓ_1-regression in nearly linear time for dense instances. *CoRR*, abs/2101.05719, 2021. arXiv:2101.05719.

29 Jan van den Brand, Yin Tat Lee, Danupon Nanongkai, Richard Peng, Thatchaphol Saranurak, Aaron Sidford, Zhao Song, and Di Wang. Bipartite matching in nearly-linear time on moderately dense graphs. *CoRR*, abs/2009.01802, 2020. arXiv:2009.01802.

Isolating Cuts, (Bi-)Submodularity, and Faster Algorithms for Connectivity

Chandra Chekuri
University of Illinois at Urbana-Champaign, IL, USA

Kent Quanrud
Purdue University, West Lafayette, IN, USA

Abstract

Li and Panigrahi [37], in recent work, obtained the first deterministic algorithm for the global minimum cut of a weighted undirected graph that runs in time $o(mn)$. They introduced an elegant and powerful technique to find *isolating cuts* for a terminal set in a graph via a small number of s-t minimum cut computations.

In this paper we generalize their isolating cut approach to the abstract setting of symmetric bisubmodular functions (which also capture symmetric submodular functions). Our generalization to bisubmodularity is motivated by applications to element connectivity and vertex connectivity. Utilizing the general framework and other ideas we obtain significantly faster randomized algorithms for computing global (and subset) connectivity in a number of settings including hypergraphs, element connectivity and vertex connectivity in graphs, and for symmetric submodular functions.

2012 ACM Subject Classification Theory of computation → Graph algorithms analysis

Keywords and phrases cuts, vertex connectivity, hypergraphs, fast algorithms, submodularity, bisumodularity, lattices, isolating cuts, element connectivity

Digital Object Identifier 10.4230/LIPIcs.ICALP.2021.50

Category Track A: Algorithms, Complexity and Games

Related Version *Full Version*: `https://arxiv.org/abs/2103.12908` [10]

Funding *Chandra Chekuri*: Supported in part by NSF grants CCF-1910149 and CCF-1907937.

Acknowledgements We thank the reviewers for their helpful comments.

1 Introduction

We investigate fast algorithms for several fundamental connectivity problems in (weighted) undirected graphs as well as their generalizations to the abstract setting of submodular and bisubmodular functions. The motivation for this work arose from the recent paper of [37] that described a new algorithmic approach for finding the *global minimum cut* in an undirected graph. For a graph $G = (V, E)$ with edge weights $w : E \to \mathbb{R}_{>0}$, the global minimum cut problem is to find the minimum weight subset of edges whose removal disconnects the graph; alternatively it is to find a set S, where $\emptyset \subsetneq S \subsetneq V$, that minimizes $w(\delta(S))$[1]. When G is unweighted, this is called the edge connectivity of the graph. There has been extensive work on algorithms for this problem, and its study has led to many important theoretical developments. Karger developed a near-linear time randomized algorithm [30] that runs in $O(m \log^3 n)$ time with some recent improvements in the log factors via better data structures [21, 42]. Here m is the number of edges and n is number of nodes in the

[1] For $A \subset V$, $\delta(A)$ denote the set of edges in G with exactly one end point in A. $w(\delta(A))$ is notation for $\sum_{e \in \delta(A)} w(e)$.

graph. However, the best deterministic algorithm until recently was $\tilde{O}(mn)$ via two very different approaches [24, 49]. Li and Panigrahi developed a new approach that improved this bound. Their algorithm runs in time $O(m^{1+o(1)})$ plus the time to compute $O(\text{polylog}(n))$ (s,t)-minimum cut computations in a graph with m edges and n nodes. Their approach uses the (s,t)-minimum cut algorithm as a black box.

Isolating cuts. A key technique in [37] is an algorithm to find *isolating cuts*. To describe this notion, let $R \subseteq V$ be subset of nodes that we call terminals. Given $r \in R$, a set $S \subseteq V$ is an isolating cut for r (with respect to R) if $S \cap R = \{r\}$. Consider the problem of finding, for *each* $r \in R$, a minimum weight isolating cut, that is; a cut $S_r \subseteq V$ where $S_r = \arg\min_{S \subseteq V, S \cap R = \{v\}} w(\delta(S))$. Note that if $R = V$ this is trivial since $S_r = \{r\}$ for each r. However, the problem is non-trivial when $R \subset V$ is a proper subset of V. A naive approach would require $|R|$ (s,t)-minimum cut computations. [37] described a simple and elegant procedure that computes all the isolating cuts for any given R in time proportional to $O(\log |R|)$ (s,t)-minimum cut computations. This, combined with simple random sampling, can be used to easily derive a randomized algorithm for global minimum cut that relies on $O(\text{polylog}(n))$ (s,t)-minimum cut computations. Note that even though the total time corresponds to $O(\text{polylog}(n))$ (s,t)-minimum cuts, the second phase of their algorithm requires computing $|R|$ (s,t)-minimum cuts, but in smaller graphs whose total size is $O(m)$ and thus can be folded into a single (s,t)-minimum cut on roughly the same input size as the original graph. Their algorithm gives a new randomized approach to global minimum cut; however, it does not lead to a faster algorithm than the existing near-linear time algorithm. Instead [37] focuses on deterministic running times and avoids random sampling by relying on several technical tools including deterministic expander decompositions to obtain a deterministic algorithm. We note, however, that the algorithm in [37] applies to the more general problem of finding the Steiner minimum cut: given $X \subseteq V$, the goal is to find a minimum cut separating a pair of nodes in X. See [25, 29] for applications.

Vertex and element connectivity. Our focus here is not on deterministic algorithms per se but rather on the applicability of the isolating cut approach to derive faster (randomized) algorithms in settings beyond edge connectivity. There has been tremendous recent and ongoing progress in fast algorithms for (s,t)-flow and cut problems and leveraging these algorithms for global connectivity is opened up by the new approach. In particular, an important motivating problem is to compute the global *(weighted) vertex connectivity* of a graph which has received substantial recent attention [17, 45]. In this setting we are given a graph $G = (V, E)$ with vertex weights $w : V \to \mathbb{R}_+$ and the goal is to find a minimum weight subset $S \subset V$ such that $G - S$ has at least two non-trivial connected components. However, as is well-known, vertex cuts/separators are not as easy to work with as edge cuts. Despite recent exciting progress via an approach based on local cuts and connectivity, the weighted case had not been addressed and the best known algorithms are from the work of Henzinger, Rao and Gabow [27]. Our starting point is the observation that the isolating cut approach of [37] relies only on the submodularity and symmetry of the edge-cut function of undirected graphs. Recall that a real-valued set function $f : 2^V \to \mathbb{R}$ is *submodular* iff $f(A) + f(B) \geq f(A \cup B) + f(A \cap B)$ for all $A, B \subseteq V$. A set function is symmetric if $f(A) = f(V \setminus A)$ for all $A \subseteq V$. The applicability of the isolation cut approach to symmetric submodular set functions already yields faster algorithms for hypergraph connectivity and several other problems that we describe subsequently. However, as we already remarked, vertex cuts do not lend themselves to this approach as vertex cuts, unlike undirected edge cuts, are simply not a symmetric submodular function.

When considering isolating cuts in the context of vertex connectivity one naturally encounters the notion of *element connectivity*, which has been found to have several important connections between edge and vertex connectivity. Element connectivity plays a key role in network design, and in fact, it was introduced by [28] to overcome the difficulty of working with vertex connectivity. We refer the reader to surveys and related papers on network design [16, 14, 15, 23, 33] for extensive literature on this topic. It also plays an important role in packing vertex disjoint Steiner trees and forests among others [13, 5, 1, 8]; [7] surveys this area. We now formally define element connectivity. The input is a graph $G = (V, E)$ and a partition of V into terminals T and non-terminals $N = V \setminus T$. The *elements* of G are the edges and non-terminals; that is, $E \cup N$. For two terminals s, t we define the element connectivity between s and t as the minimum number of elements whose removal disconnects s from t. We emphasize that element connectivity is defined *only* between the terminals. We can generalize this to the weighted setting where edges and non-terminals have non-negative weights. The global element connectivity of $G = (T \cup N, E)$ is the minimum element connectivity between any two terminals. [11] considered algorithms for computing (global) element connectivity. For global element connectivity they obtained an algorithm with running time $O(|T|)$ times the time for (s, t)-minimum cut computation.

Set-pairs and Bisubmodularity. Cuts for element and vertex connectivity do not fall into the setting of symmetric submodular set functions. A vertex separator S induces a partition of $V \setminus S$ into disjoint sets A, B that do not share an edge, and obviously $B \neq V \setminus A$ (for nonempty S). Nevertheless, one of the reasons for the tractability of element connectivity is that it does admit submodularity properties. The natural way to view its submodularity properties is via the more general notion of *bisubmodular* set functions. Given a ground set V a set-pair is (A, B) where $A, B \subseteq V$. Informally speaking a bisubmodular function f assigns a real-value to each set-pair (A, B) in a collection of set-pairs as to satisfy the inequality

$$f(X_1, Y_1) + f(X_2, Y_2) \geq f(X_1 \cup X_2, Y_1 \cap Y_2) + f(X_1 \cap X_2, Y_1 \cup Y_2)$$

for all set-pairs (X_1, Y_1) and (X_2, Y_2) on which it is defined. For this to make sense the collection of set-pairs needs to be closed under the above criss-crossed intersection and union operations for set-pairs. These binary operations can be understood more clearly as the meet and join of an appropriately defined lattice; we defer the formal definitions to Section 2. One can generalize the notion of cuts to set-pairs. Let (S, T) be a set-pair corresponding to a partition of a terminal set R. A set-pair (A, B) cuts (S, T) if $S \subseteq A$ and $T \subseteq B$. One can then define the f-minimum cut problem for (S, T): find the set-pair of minimum f value among all set-pairs that cut (S, T). With this definition in place the notions of global minimum cut for a terminal set $R \subseteq V$, and isolating cuts for R, naturally generalize. In this paper we show that the isolating cut approach of [37] generalizes to the class of symmetric bisubmodular set functions defined over appropriate collections of set-pairs.

1.1 Contributions and Results

We make two contributions at the high-level. The first is conceptual in generalizing the isolating cut approach to the (bi)submodular setting. The second is to apply this abstract framework with additional ideas to derive faster randomized algorithms for several fundamental problems. Together they yield a plethora of new running times for a diverse collection of connectivity problems, both abstract (optimizing over set functions in an oracle model) and concretely in graphs. The multiplicity of results is for the following combination of

reasons. First, by implementing the isolating cut approach at a higher level of abstraction, and abstaining from concrete specificities, we not only expose the isolating cut approach to new problems, but allow for the substitution of different domain specific black box subroutines that, within a domain, can have interesting tradeoffs. Second, and unlike the case of graph edge connectivity, the second phase of the isolating cut approach can often benefit from additional problem specific ideas, especially if one wants to take advantage of certain domain-specific algorithms that can be very powerful if applied carefully.

An important aspect of the isolating cut approach is that it inherently gives an algorithm for the subset connectivity version. In the following we will use m, n to refer to the number of edges and vertices in a given graph and use $\mathrm{EC}(m, n)$ to refer to the running time for computing a minimum (s, t)-cut in an edge-weighted directed graph, and $\mathrm{VC}(m, n)$ for the running time for computing a minimum (s, t)-cut in a vertex-weighted directed graph. We instantiate concrete running times for special cases when needed.

Connectivity of Bisubmodular functions. The precise statement that captures the general isolation cut property in bisubmodular set functions requires stating several technical definitions. Our main results for this are captured by Lemma 11, Lemma 12 and Theorem 13 which are better understood after the technical definitions. Here we state an informal theorem that captures these results.

▶ **Theorem 1** (Informal). *Let $f : \mathcal{V} \to \mathbb{R}$ be a symmetric bisubmodular function defined over a collection of set-pairs $\mathcal{V}$ over V. Let $R \subseteq V$. Suppose one has an oracle that given a partition (S, T) of R finds the f-minimum set-pair $(A, B) \in \mathcal{V}$ that cuts (S, T). In $O(\log |R|)$ calls to this oracle one can find for each $r \in R$ a set-pair $(X_r, X'_r) \in \mathcal{V}$ such that the following properties hold: (i) for each $r \in R$, (X_r, X'_r) is a $(r, R - r)$ separating set-pair, (ii) there is an f-minimum set-pair (Y_r, Y'_r) separating $(r, R - r)$ such that $Y_r \subseteq X_r$, $X'_r \subseteq Y'_r$ and (iii) $X_r \cap X_q = \emptyset$ for $r \neq q$. The total run time for finding the f-minimum isolating cut (Y_r, Y'_r) for each $r \in R$ can thus be bounded by the $O(\log |R|)$ cut computations and the total time to find the cuts inside each (X_r, X'_r).*

Symmetric submodular functions. We derive the following theorem as a corollary.

▶ **Theorem 2.** *Let $f : 2^V \to \mathbb{R}$ be a symmetric submodular function and $R \subseteq V$ and let $n = |V|$. Suppose there is an algorithm for submodular function minimization in the value oracle model in time $\mathrm{SFM}(n) = g_1(n)EO + g_2(n)$ where EO is the time for the evaluation oracle. Assuming that $g_1(n) = \Omega(n)$ and $g_2(n) = \Omega(n)$, a minimum f-cut that separates some two terminals in R can be found in $O\big(\mathrm{SFM}(n) \log^2(n)\big)$ time.*

▶ **Corollary 3.** *Let f be an integer valued symmetric submodular function with $|f(S)| \leq M$. Using the submodular function minimization algorithms of [35] one can find the global minimum cut of f with high probability in time $\tilde{O}(n^2 \log(nM)EO + n^3 \log^{O(1)}(nM))$.*

The preceding corollary should be compared to Queyranne's well-known combinatorial algorithm that uses $O(n^3\mathrm{EO})$ time [47]. The algorithm from [35] is not strongly polynomial but uses a factor $\tilde{\Omega}(n)$ fewer evaluation calls. Further, our randomized algorithm can handle minimum f-cut for a subset of terminals while Queyranne's algorithm does not generalize. In addition, the black box reduction can take advantage of future improvements to $\mathrm{SFM}(n)$ as well as for special cases as we will see next.

Hypergraph connectivity. A hypergraph $H = (V, E)$ consists of vertices V and hyperedges E where each hyperedge $e \in E$ is a subset of nodes; that is, $e \subseteq V$. We let $p = \sum_{e \in E} |e|$ denote the total size of H and let m, n denote number of hyperedges and vertices. The rank r of a hypergraph is the maximum edge size; graphs are rank 2 hypergraphs. The cut function of a hypergraph is symmetric and submodular and the global minimum cut question for edge connectivity naturally generalizes to hypergraphs. The best deterministic algorithm for this problem runs in $O(pn + n^2 \log n)$ time [32, 47, 41]. The best randomized algorithm runs in time $\tilde{O}(n^r)$ time with high probability in rank r hypergraphs [18] and this is better than $\tilde{O}(pn)$ only for very dense hypergraphs. Via sparsification one can also get an algorithm in unweighted hypergraphs that runs in time $O(p + \lambda n^2)$ where λ is the minimum cut value [12]. We obtain the following theorem that gives significantly better bounds in most settings of interest, and new tradeoffs, while also generalizing to subset minimum cut.

▶ **Theorem 4.** *Let $H = (V, E)$ be a weighted hypergraph with m edges, n nodes and total size $p = \sum_{e \in E} |e|$. Let $R \subseteq V$. The global minimum cut for R in H can be found with high probability in time $\tilde{O}(\mathrm{EC}(p, m + n))$ or in time $\tilde{O}(\sqrt{pn(m+n)^{1.5}})$.*

Now we state our algorithmic results for element connectivity and vertex connectivity that follow via the bisubmodularity framework and problem specific optimizations.

Element connectivity. The fastest known algorithm so far for global element connectivity is from [11] and runs in time $O(|T|\,\mathrm{EC}(m, n))$ for terminal set T, which can be $\Omega(n\,\mathrm{EC}(m, n))$. We obtain the following.

▶ **Theorem 5.** *Let $G = (T \cup N, E)$ be an instance of weighted element connectivity with $|T| = k$ terminals. The global element connectivity can be computed in*

$$\tilde{O}\left(\mathrm{EC}(m, n) + \max_{m_1, \ldots, m_k} \left\{ \sum_{i=1}^{k} \mathrm{EC}(m_i, n) : m_1 + \cdots + m_k \leq 2m \right\}\right)$$

time with high probability. The algorithm generalizes to subset element connectivity.

In particular, for $\mathrm{EC}(m, n)$ of the form $\mathrm{EC}(m, n) = \tilde{O}(m\,\mathrm{poly}(m, n))$, the running time above is $\tilde{O}(\mathrm{EC}(m, n))$. For instance, via [34], one obtains an $\tilde{O}(m\sqrt{n})$ time algorithm. However, recent breakthrough work of [51] showed that $\mathrm{EC}(m, n) = \tilde{O}(m + n^{1.5})$. This running time bound cannot be directly used in the preceding theorem. Using further ideas we obtain an improved running times that are encapsualted in the following theorem.

▶ **Theorem 6.** *Let $G = (T \cup N, E)$ be an instance of element connectivity with n nodes and m edges. Let $w : V \cup E \to [1..U]$ assign integer (or infinite) weights to each vertex and edge. The global element connectivity can be computed in randomized $\tilde{O}\big(m^{1+o(1)} n^{3/8} U^{1/4} + n^{1.5}\big)$ time or in $\tilde{O}\big(m^{1/2} n^{5/4}\big)$ time where $\tilde{O}(\cdots)$ hides $\mathrm{poly}(\log(n), \log(U))$-factors.*

Vertex connectivity. We now consider global vertex connectivity of both weighted and unweighted graphs. For simplicity we consider the interesting setting where there is a vertex separator of size less than $0.99W$ where W is the total vertex weight. We obtain new and faster randomized $(1 + \epsilon)$-approximation algorithms that improves upon the randomized $\tilde{O}(mn)$ exact algorithm of Henzinger, Rao and Gabow [27]. The algorithms are based on reducing, via sampling, to computing isolating element cuts. The running times we obtain are captured by the following theorem.

▶ **Theorem 7.** *Let $G = (V, E)$ be a* weighted *instance of vertex connectivity. There is a randomized algorithm that gives a $(1+\epsilon)$-approximation with high-probability in $\tilde{O}(\mathrm{EC}(m, n)/\epsilon)$ time; in particular there is a randomized algorithm that runs in time $\tilde{O}(m\sqrt{n}/\epsilon)$. For dense graphs there is a randomized algorithm that runs in $\tilde{O}(m^{1/2}n^{5/4}/\epsilon)$ time.*

There has been exciting recent work on faster algorithms for vertex connectivity via a local connectivity approach [45, 17]. The algorithms are limited to unweighted graphs while our theorem above gives the first constant factor approximation for weighted vertex connectivity in $o(mn)$ time. For unweighted graphs and graphs with small integer capacities we can obtain *exact* algorithms by setting $\epsilon = 1/\kappa$ where κ is the vertex connectivity. We obtain several different tradeoffs depending on m, n, κ. These can be found in Section 4.

Recent related work. There have been several recent papers on the use of isolating cuts and other approaches for connectivity problems in undirected as well as directed graphs. We refer the reader to some of these papers [36, 9, 46, 6]. Others have also observed that the isolating cut approach generalizes to symmetric submodular functions; see [43] for instance.

Organization. Section 2 describes the bisubmodularity framework and the abstract results at a high level; a more detailed description with several examples and formal proofs of the lemmas and theorems stated in Section 2 can be found in the full version [10]. Section 3 describes the algorithms for element-connectivity and Section 4 describes our algorithms for vertex connectivity. Section 5 describes the results for hypergraph connectivity. The proofs for Section 5 have been omitted due to space constraints and can be found in the full version [10].

2 Isolating Cuts, Symmetric Bisubmodular Functions, and Lattices

Our goal in this section to define the relevant machinery to explore and make explicit the generality of the isolating cut idea. As discussed in the introduction, this framework is motivated by the necessity of going beyond symmetric submodular set functions to capture concrete applications of interest such as element and vertex connectivity. Given the abstract nature of this discussion, in constrast to the concrete algorithmic applications, we have elected to give a brief and minimal discussion of the bisubmodular framework here. A more comprehensive description, including many more examples as well as the proofs of all lemmas and theorems stated here, can be found in the full version [10].

Let V be a finite set of elements. An ordered pair $(A, B) \in 2^V \times 2^V$ is a *set-pair* over V. For a family of set-pairs $\mathcal{V} \subseteq 2^V \times 2^V$ over V, we say that $\mathcal{V}$ is a *crossing lattice*[2] over V if it is closed under the following two operators.

$$(X_1, Y_1) \vee (X_2, Y_2) = (X_1 \cup X_2, Y_1 \cap Y_2).$$
$$(X_1, Y_1) \wedge (X_2, Y_2) = (X_1 \cap X_2, Y_1 \cup Y_2).$$

If $\mathcal{V}$ is closed under these operations, then $\mathcal{V}$ is a lattice under the partial order

$$(X_1, Y_1) \preceq (X_2, Y_2) \iff X_1 \subseteq X_2,\ Y_2 \subseteq Y_1.$$

[2] This notion is analogous to the definition of a crossing family of sets.

The binary operator $\vee$ returns the unique least upper bound of its arguments (a.k.a. the *meet*) and the binary operator $\wedge$ returns the unique greatest lower bound of its arguments (a.k.a. the *join*).

For a pair of sets $(X, Y) \in 2^V \times 2^V$, the *transpose* of (X, Y), denoted $(X, Y)^T$, is the reversed pair of sets $(X, Y)^T \stackrel{\text{def}}{=} (Y, X)$. A crossing lattice $\mathcal{V} \subseteq 2^V \times 2^V$ is *symmetric* if is closed under taking the transpose. We have the following identities relating the transpose with the lattice operations $\vee$ and $\wedge$. Observe that for $\mathcal{X}, \mathcal{Y} \in \mathcal{V}$, we have $\left(\mathcal{X}^T\right)^T = \mathcal{X}$, $(\mathcal{X} \vee \mathcal{Y})^T = \mathcal{X}^T \wedge \mathcal{Y}^T$, and $(\mathcal{X} \wedge \mathcal{Y})^T = \mathcal{X}^T \vee \mathcal{Y}^T$. Lastly, A crossing lattice $\mathcal{V} \subseteq 2^V \times 2^V$ is *pairwise disjoint* if $X \cap Y = \emptyset$ for all $(X, Y) \in \mathcal{V}$.

We now define an abstract, lattice-based notion of cuts that unifies the various different families of cuts of interest in graphs. Let V be a set. For two set-pairs $\mathcal{S} = (S, T) \in 2^V \times 2^V$ and $\mathcal{X} = (X, Y) \in 2^V \times 2^V$, we denote

$$\mathcal{S} \subseteq \mathcal{X} \stackrel{\text{def}}{\iff} S \subseteq X, T \subseteq Y.$$

If $\mathcal{S} \subseteq \mathcal{X}$, then we say that $\mathcal{X}$ *cuts* $\mathcal{S}$ or that $\mathcal{X}$ is an $\mathcal{S}$*-cut*. If $\mathcal{V}$ is a crossing lattice over V, $R \subset V$ is a subset, and $\mathcal{R}$ is a crossing lattice over R, then we say that $\mathcal{V}$ *separates* $\mathcal{R}$ if for every $\mathcal{S} \in \mathcal{R}$, there is an $\mathcal{S}$-cut $\mathcal{X} \in \mathcal{V}$. The following lemma observes that cuts are closed under the two lattice operations.

▶ **Lemma 8.** *Let V be a set and let $R \subseteq V$. Let $\mathcal{V} \subseteq 2^V \times 2^V$ be a crossing lattice over V and let $\mathcal{R} \subseteq 2^R \times 2^R$ be a crossing lattice over R. Suppose that $\mathcal{V}$ separates $\mathcal{R}$. Let $\mathcal{S}_1, \mathcal{S}_2 \in \mathcal{R}$, let $\mathcal{X}_1 \in \mathcal{V}$ be an $\mathcal{S}_1$-cut, and let $\mathcal{X}_2 \in \mathcal{V}$ be an $\mathcal{S}_2$-cut. Then $\mathcal{X}_1 \vee \mathcal{X}_2$ is an $\mathcal{S}_1 \vee \mathcal{S}_2$-cut and $\mathcal{X}_1 \wedge \mathcal{X}_2$ is an $\mathcal{S}_1 \wedge \mathcal{S}_2$-cut.*

Now, let $\mathcal{V}$ be a lattice. A real-valued function $f : \mathcal{V} \to \mathbb{R}$ is *submodular* if for all $\mathcal{X}, \mathcal{Y} \in \mathcal{V}$,

$$f(\mathcal{X}) + f(\mathcal{Y}) \geq f(\mathcal{X} \vee \mathcal{Y}) + f(\mathcal{X} \wedge \mathcal{Y}).$$

Bisubmodular functions can be interpreted as submodular functions over particular crossing lattices. There are at least two definitions of bisubmodular function in the literature. These definitions are similar and we discuss both.

In one definition (e.g., in [48]), a function $f : 2^V \times 2^V \to \mathbb{R}$ is called *bisubmodular* if for all $X_1, Y_1, X_2, Y_2 \subseteq V$, we have

$$f(X_1, Y_1) + f(X_2, Y_2) \geq f(X_1 \cup X_2, Y_1 \cap Y_2) + f(X_1 \cap X_2, Y_1 \cup Y_2). \tag{1}$$

A bisubmodular function $f : 2^V \times 2^V \to \mathbb{R}$ is submodular over the crossing lattice of all set-pairs, $\mathcal{V} = 2^V \times 2^V$.

Another definition (e.g., [4, 3, 2, 19]) of a bisubmodular function f is that $f(X_1, Y_1)$ is only defined for disjoint sets X_1 and Y_1, and otherwise satisfies inequality (1) for these inputs. In this version, f is bisubmodular iff it is a submodular function over the lattice of disjoint sets, $\mathcal{V} = \{(X, Y) : X, Y \subseteq V, X \cap Y = \emptyset\}$.

Now, let $\mathcal{V} \subseteq 2^V \times 2^V$ be a symmetric crossing lattice. A function $f : \mathcal{V} \to \mathbb{R}$ is *symmetric* if for all $\mathcal{X} \in \mathcal{V}$, $f(\mathcal{X}) = f\left(\mathcal{X}^T\right)$. This is a different definition than for symmetric submodular *set* functions and generalizes the (more standard) set-based definition. Both undirected edge cuts and vertex cuts are examples of symmetric submodular functions over appropriate symmetric crossing lattices.

There is an important relationship between the sets of terminals being separated and *minimal* minimum cuts that separate them, highlighted in the following lemma.

▶ **Lemma 9.** *Let V be a set and $R \subset V$. Let $\mathcal{V} \subseteq 2^V \times 2^V$ be a symmetric crossing lattice over V and let $\mathcal{R} \subseteq 2^R \times 2^R$ be a symmetric crossing lattice over R, such that $\mathcal{V}$ separates $\mathcal{R}$. Let $f : \mathcal{V} \to \mathbb{R}$ be a symmetric bisubmodular function. Consider the function $h : \mathcal{R} \to \mathcal{V}$ where $h(\mathcal{S})$ is defined as $\preceq$-minimum, f-minimum $\mathcal{S}$-cut. Then h is well-defined and carries the partial orders on $\mathcal{R}$ to $\mathcal{V}$; that is, $\mathcal{S}_1 \preceq \mathcal{S}_2 \implies h(\mathcal{S}_1) \preceq h(\mathcal{S}_2)$.*

The following is a particularly convenient form of Lemma 9, and the one applied directly in the sequel.

▶ **Lemma 10.** *Let V be a set and $R \subset V$. Let $\mathcal{V} \subseteq 2^V \times 2^V$ be a symmetric crossing lattice over V and let $\mathcal{R} \subseteq 2^R \times 2^R$ be a symmetric crossing lattice over R, such that $\mathcal{V}$ separates $\mathcal{R}$. Let $f : \mathcal{V} \to \mathbb{R}$ be a symmetric bisubmodular function. Let $\mathcal{S}_1, \ldots, \mathcal{S}_k \in \mathcal{R}$ and $\mathcal{X}_1, \ldots, \mathcal{X}_k \in \mathcal{V}$ such that for all $i \in [k]$, $\mathcal{X}_i$ is an f-minimum $\mathcal{S}_i$-cut. Then for any $\mathcal{S} \in \mathcal{R}$ such that $\mathcal{S} \preceq \mathcal{S}_i$ for all i, there is an f-minimum $\mathcal{X}$-cut with $\mathcal{X} \preceq \mathcal{X}_1 \wedge \cdots \wedge \mathcal{X}_k$.*

We now come to the issue of computing isolating cuts. We formalize this as follows. Let V be a set and $R \subset V$. Let $\mathcal{V} \subseteq 2^V \times 2^V$ be a symmetric and *pairwise disjoint* crossing lattice over V and let $\mathcal{R} \subseteq 2^R \times 2^R$ be the symmetric and *pairwise disjoint* crossing lattice over R consisting of all partitions of R; i.e., $\mathcal{R} = \{(S,T) : S \cup T = R,\ S \cap T = \emptyset\}$. Let $f : \mathcal{V} \to \mathbb{R}$ be a symmetric bisubmodular function. For each $r \in R$ we wish to find an f-minimum cut $\mathcal{Y}_r$ for the set-pair $(\{r\}, R - \{r\})$ (which we abbreviate as $(r, R - r)$ for notational simplicity). The main property that leads to efficiency is captured by the next lemma.

▶ **Lemma 11.** *Let V be a set and $R \subset V$. Let $\mathcal{V} \subseteq 2^V \times 2^V$ be a symmetric and* pairwise disjoint *crossing lattice over V and let $\mathcal{R} \subseteq 2^R \times 2^R$ be the symmetric and* pairwise disjoint *crossing lattice over R consisting of all partitions of R; i.e., $\mathcal{R} = \{(S,T) : S \cup T = R,\ S \cap T = \emptyset\}$. Let $f : \mathcal{V} \to \mathbb{R}$ be a symmetric bisubmodular function. Suppose we had access to an oracle that, given $\mathcal{S} \in \mathcal{R}$, returns a minimum $\mathcal{S}$-cut $\mathcal{W} \in \mathcal{V}$. Let $k = \lceil \log |R| \rceil$. Then with k calls to the oracle, one can compute k cuts $\mathcal{W}_1, \ldots, \mathcal{W}_k \in \mathcal{V}$ such that the following holds.*

For each $r \in R$, let $\mathcal{X}_r = \left(\bigwedge_{i:(r,V-r) \preceq \mathcal{W}_i} \mathcal{W}_i\right) \wedge \left(\bigwedge_{i:(r,V-r) \preceq \mathcal{W}_i^T} \mathcal{W}_i^T\right)$ be the intersection of cuts transposed to always include r in the first component. Then we have the following. (1) For all $r \in R$, $\mathcal{X}_r$ is an $(r, R-r)$-cut. (2) For all $r \in R$, there is a minimum $(r, R-r)$-cut $\mathcal{Y}_r$ such that $\mathcal{Y}_r \preceq \mathcal{X}_r$. (3) For any two distinct elements $r, q \in R$, $\mathcal{X}_r \wedge \mathcal{X}_q \preceq (\emptyset, R)$. (That is, the first components of the set pairs $\mathcal{X}_r$ are pairwise disjoint.)

Using the preceding lemma the problem of computing the f-minimum r-isolating cuts is reduced to finding such a cut in $\mathcal{X}_r$. The advantage, in terms of running time, is captured by the disjointness property: for distinct $r, q \in R$ we have $\mathcal{X}_r \wedge \mathcal{X}_q \preceq (\emptyset, R)$. For each r let $\mathcal{X}_r = (A_r, B_r)$. Thus we have $\sum_r |A_r| \leq |V|$. Given r and $\mathcal{X}_r$, the problem of computing the f-minimum cut $\mathcal{Y}_r \preceq \mathcal{X}_r$ can in several settings be reduced to solving a problem that depends only on $|A_r|$ and $|V|$. We capture this in the following lemma.

▶ **Lemma 12.** *Let V be a set and $R \subset V$. Let $\mathcal{V} \subseteq 2^V \times 2^V$ be a symmetric and* pairwise disjoint *crossing lattice over V and let $\mathcal{R} \subseteq 2^R \times 2^R$ be the symmetric and* pairwise disjoint *crossing lattice over R consisting of all partitions of R; i.e., $\mathcal{R} = \{(S,T) : S \cup T = R,\ S \cap T = \emptyset\}$. Let $f : \mathcal{V} \to \mathbb{R}$ be a symmetric bisubmodular function. Suppose we had access to an oracle that, given $\mathcal{S} \in \mathcal{R}$, returns a minimum $\mathcal{S}$-cut $\mathcal{W} \in \mathcal{V}$ and let $\mathrm{SM}(n)$ denote its running time where $n = |V|$. Moreover, suppose we have an oracle that given any $u \in R$ and $(A_u, B_u) \in \mathcal{V}$ with $u \in A_u$ outputs an f-minimum cut $\mathcal{Y}_u \preceq (A_u, B_u)$ in time $\mathrm{SMI}(|A_u|, n)$. Let $k = \lceil \log |R| \rceil$. Then, one can compute for each $r \in R$ an f-minimium r-isolating cut in in total time $O(k\,\mathrm{SM}(n) + \max_{0 \leq n_1, n_2, \ldots, n_{|R|} : \sum_i n_i = n} \sum_{i=1}^{|R|} \mathrm{SMI}(n_i, n))$.*

A simple random sampling approach combined with isolating cuts, as shown in [37] for edge cuts in graphs, yields the following theorem in a much more abstract setting.

▶ **Theorem 13.** *Let V be a set and $R \subset V$. Let $\mathcal{V} \subseteq 2^V \times 2^V$ be a symmetric and pairwise disjoint crossing lattice over V and let $\mathcal{R} \subseteq 2^R \times 2^R$ be the symmetric and pairwise disjoint crossing lattice over R consisting of all disjoint subsets of R; i.e., $\mathcal{R} = \{(S,T) : S,T \subseteq R,\, S \cap T = \emptyset\}$. Let $f : \mathcal{V} \to \mathbb{R}$ be a symmetric bisubmodular function. Suppose we had access to an oracle that, given $\mathcal{S} \in \mathcal{R}$, returns a minimum $\mathcal{S}$-cut $\mathcal{W} \in \mathcal{V}$ and let $\mathrm{SM}(n)$ denote its running time where $n = |V|$. Moreover, suppose we have an oracle that given any $u \in R$ and $(A_u, B_u) \in \mathcal{V}$ with $u \in A_u$ outputs an f-minimum cut $\mathcal{Y}_u \preceq (A_u, B_u)$ in time $\mathrm{SMI}(|A_u|, n)$. Then one can compute the minimum (nontrivial) $\mathcal{R}$-cut with constant probability in $O\Big(\mathrm{SM}(n)\log^2|R| + \max_{0\le n_1,n_2,\ldots,n_{|R|}:\sum_i n_i = n} \log(|R|)\sum_{i=1}^{|R|}\mathrm{SMI}(n_i,n)\Big)$ time.*

We derive the following corollary for symmetric submodular set functions.

▶ **Corollary 14.** *Let $f : 2^V \to \mathbb{R}$ be a symmetric submodular function and $R \subseteq V$ and let $n = |V|$. Suppose there is an algorithm for submodular function minimization in the value oracle model in time $\mathrm{SFM}(n) = g_1(n)EO + g_2(n)$ where EO is the time for the evaluation oracle. Assuming that $g_1(n) = \Omega(n)$ and $g_2(n) = \Omega(n)$, a minimum f-cut that separates some two terminals in R can be found in $O\big(\mathrm{SFM}(n)\log^2(n)\big)$ time.*

3 Element connectivity

Let $G = (V, E)$ be an undirected graph with m edges and n vertices. Let $T \subseteq V$ be a set of terminals and let $N = V \setminus T$ be the non-terminal set. For any two distinct terminals $u, v \in T$, the element connectivity between u and v is defined as the maximum number of paths from u to v that are edge-disjoint and vertex-disjoint in the non-terminal vertices $V \setminus T$. That is, only terminal vertices may be reused across paths. This notion can be easily generalized to the weighted setting where edges and non-terminals have non-negative weights/capacities. For any two terminals $s, t \in T$, we denote by $\kappa'(s,t)$ the element connectivity between them. One can compute $\kappa'(s,t)$ via a simple reduction to s-t maximum flow in a directed graph which takes $\mathrm{EC}(m,n)$ time. In this section we are concerned with the problem of computing the global element connectivity which is defined as $\kappa' = \min_{s,t\in T, s\neq t}\kappa'(s,t)$. In fact we are also interested in computing the more general problem of computing $\kappa'(R) = \min_{s,t\in R, s\neq t}\kappa'(s,t)$ where $R \subseteq T$; note that $\kappa' = \kappa'(T)$. Here we apply our general framework that obtains a randomized algorithm with running time $O(\mathrm{EC}(m,n)\log^2|R|)$. In addition to the global minimum cut for R, as we will see in the next section, finding all the isolating cuts can be used with other ideas for vertex connectivity.

Let $G = (T \uplus N, E)$ be an instance of a weighted element connectivity problem. Let $w : N \cup E \to \mathbb{R}_{\ge 0}$ assign weights to the elements. Let $R \subseteq T$ be a subset of terminals with $|R| \ge 2$. We reduce the problem of computing $\kappa'(R)$ to Theorem 13 as follows.

For ease of notation, let $\overline{V} = V \cup E$ denote the elements. Consider the family of pairs of sets $\mathcal{V} \subseteq 2^{\overline{V}} \times 2^{\overline{V}}$ defined as the set of pairs $(X, Y) \in \overline{V} \times \overline{V}$ with the following properties: (i) X and Y are disjoint, (ii) no edge in X is adjacent to a vertex in Y, and no edge in Y is adjacent to a vertex in X, and (iii) $T \subseteq X \cup Y$. $\mathcal{V}$ describes the disjoint sets that are element-wise disconnected and cover T. Clearly $\overline{V}$ is symmetric and pairwise disjoint. It is also straightforward to verify that $\overline{V}$ is a crossing lattice.

We define a function $f : \mathcal{V} \to \mathbb{R}$ by $f(X, Y) = \sum_{x \in \overline{V} - (X \cup Y)} w(x)$. $f(\mathcal{X})$ gives the total weight of elements that are not a member of either of the two sets in $\mathcal{X}$. This function f is submodular and in fact it is modular. One can easily verify that $f(X_1, Y_1) + f(X_2, Y_2) = f(X_1 \cup X_2, Y_1 \cap Y_2) + f(X_1 \cap X_2, Y_1 \cup Y_2)$. Thus $\mathcal{V}$ is a symmetric and pairwise disjoint crossing lattice, and $f : \mathcal{V} \to \mathbb{R}$ is a symmetric submodular function over $\mathcal{V}$.

Isolating (weighted) element cuts and global connectivity Let $R \subseteq T$ and let $\mathcal{R}$ be the crossing lattice consisting of all pairwise disjoint subsets of R. Given a partition of R into two sets (A, B), an f-minimum (A, B)-cut, which corresponds to the minimum element cut separating A from B, can be computed via directed (s, t)-maxflow, in $\mathrm{EC}(m, n)$-time. By Lemma 11, we can compute disjoint sets of elements $\left\{\overline{U}_r \subset \overline{V} : r \in R\right\}$ where for each $r \in R$, $\overline{U}_r$ contains the r-side component of a minimum $(r, R - r)$-element cut. Moreover, because the $\overline{U}_r$'s are obtained as intersections of sides of element cuts, for any distinct $r, q \in R$, there is no edge from $\overline{U}_r$ incident to a vertex from $\overline{U}_q$ and (symmetrically) vice-versa.

For each r, let $\overline{U}'_r \subset \overline{V}$ be the set of vertices outside $\overline{U}_r$ and incident to an edge in $\overline{U}_r$, and the edges outside $\overline{U}_r$ incident to vertices in $\overline{U}_r$. Informally speaking, $\overline{U}'_r$ is the "boundary" of $\overline{U}_r$ in an element connectivity sense. Let $n_r = \left|V \cap \left(\overline{U}_r \cup \overline{U}'_r\right)\right|$ be the number of vertices in $\overline{U}_r \cup \overline{U}'_r$ and let $m_r = \left|E \cap \left(\overline{U}_r \cup \overline{U}'_r\right)\right|$ be the number of edges in $\overline{U}_r \cup \overline{U}'_r$. Note that $\sum_r m_r \leq 2m$ since each edge can either appear in $\overline{U}_r$ for a unique choice of r or in $\overline{U}'_r$ for two choices of r.

To find an isolating cut for r we need to find the cheapest element cut contained in $\overline{U}_r$. We can do this via a flow computation as described below. For each r, consider the graph G_r where we first take the graph $\overline{U}_r \cup \overline{U}'_r$ and introduce an auxiliary vertex $\bar{t}$. We connect $\bar{t}$ to all vertices in $\overline{U}'_r$ with infinite capacity. For every edge $e \in \overline{U}'_r$ with exactly one endpoint in $\overline{U}_r$, we replace the opposite endpoint with $\bar{t}$. Observe that the minimum $(r, \bar{t})$-element cut in G_r coincides with the minimum $(r, R - r)$-element cut in G. G_r has $O(m_r)$ edges and $O(n_r)$ vertices, and the element $(r, \bar{t})$-cut problem can be solved in $\mathrm{EC}(m_r, n_r)$ time. Summing over all $r \in R$ gives the following theorem.

▶ **Theorem 15.** *Let $G = (T \cup N, E)$ be an instance of element connectivity with n nodes and m edges and let $R \subseteq T$. Let $w : V \cup E \to (-\infty, \infty]$ assign positive weight to each vertex and edge. Let $k = |T|$. Then one can compute, for all $r \in R$, the minimum weight element $(r, R - r)$-cut in $O\Big(\mathrm{EC}(m, n) \log k + \max_{m_1, \ldots, m_k} \Big\{\sum_{i=1}^{k} \mathrm{EC}(m_i, n) : m_1 + \cdots + m_k \leq 2m\Big\}\Big)$, where $\mathrm{EC}(m, n)$ is the running time for element (S, T)-cut with m edges and n vertices.*

With Theorem 15 in place we can reduce the global mincut problem for R to the isolating cut computation via sampling [37], and captured in the abstract setting Theorem 13, to obtain the following theorem to compute $\kappa'(R)$.

▶ **Theorem 16.** *Let $G = (T \cup N, E)$ be an instance of element connectivity with n nodes and m edges and let $R \subseteq T$. Let $w : V \cup E \to (-\infty, \infty]$ assign positive weight to each vertex and edge. Let $k = |R|$. Then one can compute $\kappa'(R)$ with constant probability in time $O((\mathrm{EC}(m, n) \log(k) + \max_{m_1, \ldots, m_n} \{\sum_{i=1}^{n} \mathrm{EC}(m_i, n) : m_1 + \cdots + m_n \leq 2m\}) \log n)$.*

3.1 Refined running times for element connectivity

Until recently, the leading running times for $\mathrm{EC}(m, n)$ (e.g., $\mathrm{EC}(m, n) = \tilde{O}(m\sqrt{n})$ [34]) plug directly into Theorem 16 to give running times of the form $\tilde{O}(\mathrm{EC}(m, n))$ to compute the global element connectivity. A recent breakthrough work by [50] has obtained a running

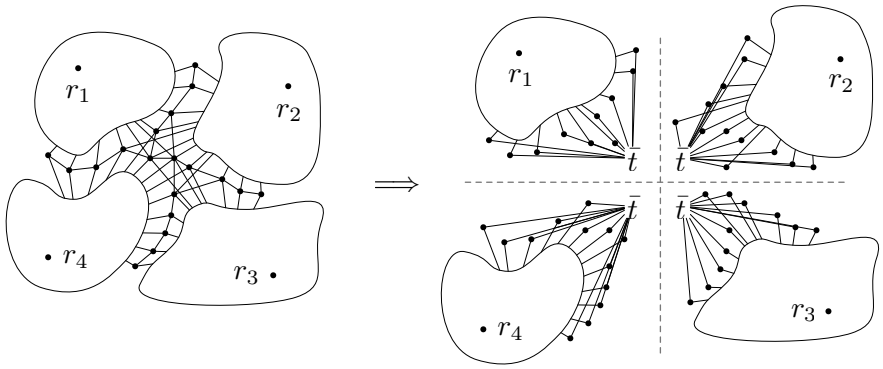


Figure 1 Applying the uncrossing framework to vertex connectivity reduces vertex isolating cuts to edge disjoint cut problems. Note that the separating vertices may appear in multiple subproblems, which is an obstruction towards a direct $\tilde{O}(\mathrm{EC}(m,n))$ overall running time for isolating vertex cuts.

time of $\mathrm{EC}(m,n) = \tilde{O}\big(m + n^{1.5}\big)$ for polynomially bounded and integral capacities. However, Theorem 16 does not directly benefit from this running time because the vertices are not partitioned across subproblems. See Figure 1 for an illustration in the concrete setting of vertex cuts. Consequently, plugging $\mathrm{EC}(m,n) = \tilde{O}\big(m + n^{1.5}\big)$ directly into Theorem 16 generates a running time of $\tilde{O}\big(m + n^{1.5}k\big)$, where $k = |T|$. The additional factor of k (to a certain extent) defeats the purpose of the isolating cuts framework.

In this section, we develop more advanced algorithms that take the isolating cut framework as a starting point, and incorporates additional ideas to take advantage of $\mathrm{EC}(m,n) = \tilde{O}\big(m + n^{1.5}\big)$. In addition to obtaining faster algorithms, these results point to a general algorithm design space where additional ideas can be introduced to obtain even better running times. The first algorithm we present leverages the fact that the edges are partitioned across subproblems, even if the vertices are not.

▶ **Theorem 17.** *Let $G = (T \cup N, E)$ be an instance of element connectivity with n nodes and m edges and let $R \subseteq T$. Let $w : V \cup E \to [1..U]$ assign integer (or infinite) weights to each vertex and edge. For $R \subseteq T$, the minimum R-isolating vertex cut can be computed in*

$$\tilde{O}\Big(m^{1+o(1)}n^{3/8}U^{1/4} + n^{1.5}\Big)$$

time.

Proof. Let $k = |R|$. We apply Theorem 15 and give concrete upper bounds using known upper bounds for $\mathrm{EC}(m,n)$. Let $m_1, \ldots, m_k \in \mathbb{N}$ with $m_1 + \cdots + m_k \le m$. Recall that $\mathrm{EC}(m,n) = \tilde{O}\big(m^{4/3+o(1)}U^{1/3}\big)$ by [31] and $\mathrm{EC}(m,n) = \tilde{O}\big(m + n^{3/2}\big)$ by [50]. Let $\alpha > 0$ be a parameter to be determined. We apply the first running time when $m_i < m\alpha/k$ and the second running time then $m_i \ge m\alpha/k$. At most k/α indices i have $m_i \ge m\alpha/k$. Thus,

$$\begin{aligned}\sum_{i=1}^{k}\mathrm{EC}(m_i,n) &= \sum_{i:m_i \ge m\alpha/k}\mathrm{EC}(m_i,n) + \sum_{i:m_i<m\alpha/k}\mathrm{EC}(m_i,n)\\ &\le \tilde{O}\left(m + \frac{k}{\alpha}n^{1.5} + \sum_{i:m_i<m\alpha/k} m_i^{4/3+o(1)}U^{1/3}\right)\\ &\overset{(a)}{\le} \tilde{O}\left(m + \frac{k}{\alpha}\left(n^{1.5} + \Big(\frac{\alpha m}{k}\Big)^{4/3+o(1)}U^{1/3}\right)\right).\end{aligned}$$

Here (a) is by convexity: the quantity $\sum_{i:m_i<m\alpha/k} m_i^{4/3+o(1)}$ subject to the condition that $\sum_i m_i \leq m$ is at most $(k/\alpha)(\alpha m/k)^{4/3+o(1)}$. Balancing terms at $\alpha = kn^{9/8}/m$, this gives an upper bound of $\tilde{O}\big(m^{1+o(1)}n^{3/8}U^{1/4}\big)$, hence the claimed running time. ◀

We point out that other running time tradeoffs between m and U can be obtained by instead applying the flow alogrithms from [39, 40].

The next theorem, which is particularly good for dense graphs, leverages the fact that while the vertices are not necessarily partitioned across subproblems, at least the "inner" vertex sets $\overline{U}_r \cap V$ are disjoint and all of the repeating "boundary" vertices are guaranteed to be outside the r-component of each $(r, R-r)$-minimum cut. The following algorithm balances a tradeoff between the recent algorithm with [51] with blocking flows [22]. In the application of blocking flows, we argue that with an appropriate construction of the auxiliary graph in the components given by the decomposition by isolating cuts, the maximum length of any augmenting paths is proportional to the number of inner vertices (rather than the total number of vertices) for that component.

▶ **Theorem 18.** *Let $G = (T \cup N, E)$ be an instance of element connectivity with n nodes and m edges and let $R \subseteq T$. Let $w : V \cup E \to [0, U]$ assign positive (or infinite) weights to each vertex and edge. For $R \subseteq T$, the minimum R-isolating cut can be computed in*

$$\tilde{O}\Big(m^{1/2}n^{5/4}\Big)$$

randomized time, where $\tilde{O}(\cdots)$ hides $\mathrm{poly}(\log(n), \log(U))$-factors.

Proof. We recall the construction from Theorem 15, adopting the same notation. In addition, for each r, let $\tilde{n}_r$ be the number of vertices in $\overline{U}_r$. Note that as the $\bar{U}_r$'s are disjoint, we have $\sum_{r\in R} \tilde{n}_r \leq n$.

For each r, we employ two different approaches to computing the minimum $(r, \bar{t})$-element cut. On one hand we can apply any max flow algorithm in $\mathrm{EC}(m_r, n_r)$ time. As remarked above we have $\mathrm{EC}(m_r, n_r) \leq \tilde{O}\big(m_r + n_r^{1.5}\big)$ by [50]. The second approach is to apply blocking flows with the following additional observations. Element connectivity can be modeled as maximum flow in undirected graphs with edge and vertex capacities, which in turn can be reduced to maximum flow in edge capacitated directed graphs. Recall the directed graph representation of vertex capacities, sometimes called the "split graph". We remind the reader that in the split graph, each non-terminal vertex $v \in V_r \setminus \big\{r, \bar{t}\big\}$ is split into two vertices – an "in-vertex" v^- and an "out-vertex" v^+ – and there is an edge (v^-, v^+) with capacity equal to $w(v)$. Each edge $(u, v) \in E_r$ is replaced with an edge (u^+, v^-) with the same capacity. From this split graph, we contract the edges $(v^+, \bar{t})$ for all $v \in V \cap \overline{U}'_r$, which is safe because $\bar{t}$ is the sink and each edge $(v^+, \bar{t})$ has infinite capacity. Now, in this directed auxiliary graph, we have $O(m_r)$ edges and $O(n_r)$ vertices. We now observe that the auxiliary vertices corresponding to $\overline{U}'_r \cap V$, $\Big\{v^- : v \in \overline{U}'_r \cap V\Big\}$, do not have any edges between them. Then any $(s, \bar{t})$ path in this graph or in any residual graph that may arise cannot have consecutive auxiliary vertices from $\overline{U}'_r$. Therefore, every augmenting path has length at most $2\tilde{n}_r$. In turn, $O(\tilde{n}_r)$ iterations of blocking flows suffice to find the minimum $(r, \bar{t})$ cut in G_r, which takes $O(m_r \log(m_r/n_r))$ time per iteration [22] and $O(m_r\tilde{n}_r \log(m_r/n_r))$ time overall.

Let $\alpha > 0$ be a parameter to be determined. We have $\tilde{n}_r \geq \alpha n/k$ for at most k/α vertices $r \in R$. We have

$$\begin{aligned} O\left(\sum_r \min\{\mathrm{EC}(m_r, n_r), m_r \tilde{n}_r \log(m)\}\right) &\leq \tilde{O}\left(\sum_{r:\tilde{n}_r \leq \alpha n/k} m_r \tilde{n}_r + \sum_{r:\tilde{n}_r \geq \alpha n/k} \left(m_r + n_r^{1.5}\right)\right) \\ &\leq \tilde{O}\left(m + \left(\frac{\alpha}{k}\right) mn + \left(\frac{k}{\alpha}\right) n^{1.5}\right) \\ &\overset{(a)}{\leq} \tilde{O}\left(m + m^{1/2} n^{5/4}\right) = \tilde{O}\left(m^{1/2} n^{5/4}\right), \end{aligned}$$

as desired. Here, in step (a), we substituted $\alpha = kn^{1/4}/m^{1/2}$. ◀

4 Vertex connectivity

In this section we consider the problem of computing the vertex connectivity in weighted and unweighted graphs. Let $G = (V, E)$ be an undirected graph with m edges and n vertices. Let $w : V \to [1, U]$ be positive vertex weights. Given distinct nodes $s, t \in V$ such that $st \notin E$, the minimum weight vertex separator between s and t can be computed via flow techniques. Recently there has been significant improvement in the running time of vertex capacitated flow to $\tilde{O}(m+n^{1.5})$ [51]. We use $\mathrm{VC}(m, n)$ to denote the complexity of computing such a separator. We let $\kappa(s, t)$ denote the weight of the separator between s, t with the understanding that $\kappa(s, t) = \infty$ if $\{s, t\} \in E$. Here we are interested in the minimum vertex weight separator of G which can be defined as $\min_{s,t \in V, s \neq t} \kappa(s, t)$.

Let $R \subset V$ such that R is an *independent set* in G; that is, no two vertices in R share an edge. One can then define $\kappa(R)$ to be $\min_{s,t \in R, s \neq t} \kappa(s, t)$. We observe that $\kappa(R)$ is the same as the element connectivity of R in the graph where R is the set of terminals and $V \setminus R$ are the non-terminals and edge weights are set to ∞; i.e., only vertices are allowed to be removed. We have already seen algorithms for element connectivity, which immediately convert to isolating cut algorithms for vertex connectivity. For instance, one can compute the minimum isolating cut in $\tilde{O}\left(m^{1+o(1)} n^{3/8} U^{1/4} + n^{1.5}\right)$ time with integral vertex weights between 1 and U, or in $\tilde{O}\left(\sqrt{m} n^{5/4}\right)$ time for polynomially bounded weights.

These running times for isolating cuts do not, however, immediately convert to running times for vertex cuts. To obtain the minimum vertex cut as an isolating cut, we must initialize the algorithm with a set of vertices R for which the minimum vertex cut is also an isolating cut. Let (S, T) be opposite sides of a minimum vertex cut $N(S) = N(T)$. Without loss of generality suppose S has weight less than or equal to T. We would like a set R that samples exactly one point from S, at least one point from T, and avoids $N(S)$ altogether. Even in the unweighted setting, uniform sampling is thwarted by the fact that $N(S)$ may be much larger than S, and it is difficult to hit S without hitting $N(S)$ too. In the following lemma, we observe that if we relax our problem to an $(1 + \epsilon)$-approximately minimum vertex cut, then we can sample a useful set R with reasonably good probability.

▶ **Lemma 19.** *Let $\epsilon > 0$ be fixed. Let $G = (V, E)$ be an undirected graph with m edges and n vertices. Let $w : V \to [1, U]$ be positive vertex weights and let $W = \sum_{v \in V} w(v)$ be the total weight. Let κ be the weight of the minimum weight vertex cut. Suppose the minimum weighted degree is greater than $(1 + \epsilon)\kappa$. Then one can compute a randomized independent set $R \subset V$ such that the minimum vertex cut is an R-isolating set with probability at least $\Omega((\epsilon/\log(nU)) \max\{\epsilon, (1 - \kappa/W)\})$.*

Proof. For ease of notation, let

$$\epsilon_0 = \max\left\{\epsilon, \frac{1}{2}\left(1 - \frac{\kappa}{W}\right)\right\}$$

Let (S, T) be opposite sides of the minimum vertex cut $N(S) = N(T)$. Without loss of generality suppose $w(S) \leq w(T)$ where we use the notation $w(A)$ to denote the total weight of vertices in A, that is, $w(A) = \sum_{v \in A} w(v)$. Since $N(S)$ is the minimum weight vertex separator we have $w(N(S)) = \kappa$. We would like a independent set $R \subset V$ that has exactly one point from S, at least one point from T, and avoids $N(S)$ altogether. Then (S, T) would isolate the lone vertex in $R \cap S$ from $R - r \subseteq T$, as desired. We can achieve this via a sampling procedure that we described below.

First we claim that $w(S) \geq \epsilon\kappa$. Fix an arbitrary vertex $v \in S$. By assumption, $w(N(v)) \geq (1 + \epsilon)\kappa$. Since $N(v) \subseteq S \cup N(S)$ and $S \cap N(S) = \emptyset$, we have $w(S) \geq \epsilon\kappa$.

Let μ be any value in the range $[2\max\{w(S), \kappa\}, 4\max\{w(S), \kappa\}]$. Since $\max\left\{\sum_{v \in S} w(v), \kappa\right\}$ lies in the range $[1, \text{poly}(n, U)]$, we can sample a value μ that lies in the above range with probability $\Omega(1/\log(nU))$ by randomly picking a power of 2 in the range $[1, \text{poly}(n, U)]$. Once we fix μ, let R be a random subset of vertices obtained by independently sampling each vertex v with probability $w(v)/\mu$. Then, as long as R has an adjacent pair of vertices, we remove one of them from R. We claim that the intial sample for R has one point from S, no points from $N(S)$, and at least one point from T with probability $\geq \Omega(\epsilon\epsilon_0)$. If so, then since S and T are independent from one another, dropping vertices in the second phase will not remove any vertices from S, and retain at least one vertex in T, as desired. Observe that the three events are independent.

The probability that R avoids $N(S)$ is $\prod_{v \in N(S)}(1 - w(v)/\mu)$. Since $w(N(S)) = \kappa$ and $\mu \geq 2\kappa$, for any $v \in N(S)$, $w(v)/\mu \leq 1/2$. For $x \in (0, 1/2]$ the inequality $(1 - x) \geq e^{-2x}$ holds. Hence, $\prod_{v \in N(S)}(1 - w(v)/\mu) \geq \prod_{v \in N(S)} e^{-2w(v)/\mu} \geq e^{-2\kappa/\mu} \geq 1/e$.

Recall that $w(S) \geq \epsilon\kappa$ and hence the probability that R samples exactly one vertex from S is

$$\sum_{v \in S} \frac{w(v)}{\mu} \prod_{u \in S - \{v\}} (1 - \frac{w(u)}{\mu}) \geq \sum_{v \in S} \frac{w(v)}{\mu} e^{-2(w(S) - w(v))/\mu} \geq \frac{1}{e} \sum_{v \in S} \frac{w(v)}{\mu} \geq \frac{\epsilon}{4e}.$$

In the preceding set of inequalities we used the fact that $1 - x \geq e^{-2x}$ for $x \in [0, 1/2]$ since $w(S)/\mu \leq 1/2$. In the final inequality we used the fact that $w(S) \geq \epsilon\kappa$ which implies that $w(S) \geq \epsilon\mu/4$.

We claim that $w(T) \geq \epsilon_0\mu/4$. Assuming the claim, the probability that R samples at least one vertex from T is $\geq 1 - e^{-w(T)/\mu} \geq 1 - e^{\epsilon_0/4} = \Omega(\epsilon_0)$. To see the claim, recall that $w(T) \geq w(S) \geq \epsilon\kappa$. We also have $w(S) + w(T) + w(N(S)) = W$ which implies that $w(T) \geq \frac{1}{2}(W - \kappa) \geq \frac{1}{2}(1 - \frac{\kappa}{W})W \geq \frac{1}{2}(1 - \frac{\kappa}{W})w(S)$. Since $\mu \leq 4\max\{w(S), \kappa\}$, we have $w(T) \geq \epsilon_0\mu/4$.

Thus, given μ lies in the range $[2\max\{w(S), \kappa\}, 4\max\{w(S), \kappa\}]$ which happens with probability $\Omega(1/\log(nU))$ we have the desired sample R with probability $\Omega(\epsilon \cdot \epsilon_0)$. ◀

4.1 Approximate vertex connectivity

By combining the isolating cut algorithms with the sampling lemma above (for the case where no singleton already induces a good enough vertex cut), we obtain the following approximation algorithm for vertex connectivity. We point out that in the running time below, the trailing factor $(\min\{1/\epsilon, W/(W - \kappa)\})$ is simply a constant except in the relatively

uninteresting setting where the minimum weight vertex cut is almost all of the weight of the graph. In the regime of interest, the following is a $\tilde{O}(1/\epsilon)$ factor greater than the running time to compute an isolating vertex cut.

▶ **Theorem 20.** *Let $\epsilon > 0$ be fixed. Let $G = (V, E)$ be an undirected graph with m edges and n vertices. Let $w : V \to \mathbb{R}_{>0}$ be positive vertex weights and let $W = \sum_{v \in V} w(v)$ be the total weight. Let κ be the weight of the minimum weight vertex cut. Then a minimum vertex cut can be computed with high probability in $\tilde{O}((1/\epsilon)\,\mathrm{IsoVC}(m, n)\min\{(1/\epsilon), W/(W-\kappa)\})$ randomized time, where $\mathrm{IsoVC}(m, n)$ is the running time to compute the minimum isolating vertex cut in a weighted graph of m edges and n vertices.*

Proof. Let

$$\ell = \tilde{O}\left(\frac{1}{\epsilon}\min\left\{\frac{1}{\epsilon}, \frac{W}{W-\kappa}\right\}\right)$$

The algorithm first repeats the following subroutine $O(\ell)$ times. This subroutine first generates an set $R \subset V$ by Lemma 19, and then it computes a minimum R-isolating cut. It compares the ℓ isolating cuts generated above with the singleton cuts in the graph, returning the minimum overall.

We argue that the algorithm returns a $(1+\epsilon)$-approximate minimum weight cut with high probability by the following simple analysis. In one case, some singleton cut is an approximate minimum cut, in which case the algorithm always succeeds. In the second case, the minimum weighted degree is at least an $(1+\epsilon)$-multiplicative factor greater than the vertex connectivity. In that case, the minimum weight vertex cut is a minimum R-isolating cut for at least one of the random sets R with high probability, in which case we return the minimum weight vertex cut. ◀

We briefly compare our bound above to previous work. As mentioned previously Henzinger, Rao and Gabow [27] obtain a randomized algorithm that gives the exact vertex connectivity in $\tilde{O}(mn)$ time for weighted graphs. We obtain a $(1+\epsilon)$-approximation in $\tilde{O}(m\sqrt{n}/\epsilon)$ time or in $\tilde{O}\big(m^{1/2}n^{5/4}/\epsilon\big)$ time; other bounds are outlined in previous subsection. We are thus able to obtain substantially faster algorithm if we settle for a small approximation. There have been past works on approximation for vertex connectivity but as far as we know they have been limited to unweighted graphs. Henzinger [26] obtained a 2-approximation in $O(n^2\min(\sqrt{n}, \kappa))$. Forster et al. obtained a $(1+\epsilon)$-approximation in randomized time $\tilde{O}\big(m + n\kappa^2/\epsilon\big)$ which is near-linear for small connectivity, and combining various other results they improve upon Henzinger's result. We refer the reader to [17] for detailed bounds. Our running times are useful for the larger connectivity regime and we can obtain improved bounds in various other regimes of interest. We leave a more detailed comparison to a future version of the paper.

4.2 Exact vertex connectivity

Now, for integral weights, the approximation algorithm above gives the following exact algorithm for vertex connectivity by suitable choice of ϵ. Again we highlight that in the running time below, the trailing factor $(\min\left\{\kappa, \frac{W}{W-\kappa}\right\})$ is simply a constant except in the relatively uninteresting setting where κ is almost $\sum_{v \in V} w(v)$, in which case the remaining factors of $O(\kappa\,\mathrm{IsoVC}(m, n))$ are not as compelling anyway.

Table 1 A table of running times for finding the minimum vertex cut in an *unweighted* and undirected graph. $\mathrm{VC}(m,n)$ denotes the running time of computing (s,t)-vertex connectivity. $\mathrm{EC}(m,n)$ denotes the running time computing (s,t)-edge connectivity. See also [48, Section 15.2a].

$O\big(n^2\,\mathrm{VC}(\kappa n, n)\big)$	Combines trivial algorithm with sparsification [44].
$O(n\,\mathrm{VC}(\kappa n, n))$	Combines randomized trivial algorithm with sparsification [44]. $\kappa \leq .99n$
$O(n^\omega + n\kappa^\omega)$.	[38].
$\tilde{O}\big(\kappa n^2\big)$	[27]. Randomized.
$O\big(\min\big\{n^{3/4}, \kappa^{3/2}\big\}\kappa^2 n + \kappa n^2\big)$	[20].
$\tilde{O}\big(m + \kappa^{7/3} n^{4/3}\big)$	[45]. Randomized.
$\tilde{O}\big(m + n\kappa^3\big)$	[17]. Randomized.
$\tilde{O}\big(m + \kappa^{7/3} n^{4/3}\big)$	Corollary 22. Randomized. $\kappa \leq .99n$
$\tilde{O}\big(m + \kappa^2 n^{11/8+o(1)} + \kappa n^{3/2}\big)$	Corollary 22. Randomized. $\kappa \leq .99n$
$\tilde{O}\big(m + \kappa^{1.5} n^{7/4}\big)$	Corollary 22. Randomized. $\kappa \leq .99n$.

▶ **Theorem 21.** *Let $G = (V, E)$ be an undirected graph with m edges and n vertices. Let $w : V \to \mathbb{N}$ be integer vertex weights and let $W = \sum_{v \in V} w(v)$ be the total weight. Let κ be the weight of the minimum weight vertex cut. Then the minimum vertex cut can be computed with high probability in $\tilde{O}(\kappa\,\mathrm{IsoVC}(m,n)\min\{\kappa, W/(W-\kappa)\})$ randomized time, where $\mathrm{IsoVC}(m,n)$ is the running time to compute the minimum isolating vertex cut in a weighted graph of m edges and n vertices.*

Proof. For integral capacities, a $(1 + 1/(\kappa+1))$-approximation is an exact solution. Thus the result follows from Theorem 20. ◀

For the unweighted case, combining the above with sparsification [44] gives the following.

▶ **Corollary 22.** *Let $G = (V, E)$ be a simple unweighted graph. Then the minimum vertex cut can be computed with high probability in $\tilde{O}(m + \kappa\,\mathrm{IsoVC}(n\kappa, n)\min\{\kappa, n/(n-\kappa)\})$ randomized time, where $\mathrm{IsoVC}(m,n)$ is the running time to compute the minimum isolating vertex cut in a graph of m edges and n vertices.*

Proof. For unweighted graphs we can assume we know κ (via exponential search which adds an additional $O(\log \kappa)$ overhead). We apply the well-known linear-time sparsification algorithm of Nagamochi and Ibaraki [44] to reduce the number of edges to $O(n\kappa)$ and then run the algorithm in the preceding theorem on the sparsified graph which gives the claimed bound. ◀

The reduction from exact vertex connectivity to isolating vertex cut above, mixed with the algorithms for isolating vertex cuts, and optionally including the sparsification step from Corollary 22, produces a number of new running times that are optimal for different ranges of κ. In general, the running times obtained here have a lower dependence on κ then other

algorithms for vertex connectivity with a poly(κ) dependence (which is common for the unweighted setting), so the running times here are particularly good for moderate to large κ. For a more detailed comparison between the literature and new running times for the unweighted setting (where we restrict to unweighted for simplicity), see Table 1.

5 Hypergraph Connectivity

Let $H = (V, E)$ be a weighted hypergraph and let $R \subseteq V$. The cut function of a hypergraphs is symmetric and submodular. Given disjoint sets $S, T \subset V$ the minimum S-T cut in H can be computed in $\mathrm{EC}(p, m+n)$ time via standard reductions[3]. We can use Lemma 11 and Corollary 14 to understand the running time to compute R-connectivity in H. Up to logarithmic factors it suffices to estimate the time to find R-isolating cuts. Recall that the running time consists of two parts. The first part is $O(\log |R|)$ calls to S-T cut problem in H. After this we have the following situation. For each $r \in R$ we obtain a set $U_r \subset V$ such that $r \in R$ and $U_r \cap (R - r) = \emptyset$. Furthermore the sets U_r over $r \in R$ are pairwise disjoint. For each r the goal is to find a set $Y_r \subseteq U_r$ with minimum $w(\delta(Y_r))$ where $\delta(Y_r)$ is set of hyperedges crossing Y_r. Let $n_r = |U_r|$. We can compute Y_r by solving a cut problem in an auxiliary hypergraph G_r on $n_r + 1$ vertices obtained by shrinking $V \setminus U_r$ into a single vertex. Let p_r be the total size of the hyperedges in G_r. It is not hard to see that $\sum_{r \in R} p_r = O(p)$. Thus each cut problem in G_r can be computed in either $\mathrm{EC}(p_r, m + n_r + 1)$. This implies the following.

▶ **Theorem 23.** *The minimum isolating cuts over a set of vertices R of size $k = |R|$ in a hypergraph with m edges, n vertices, and total size p can be computed*

$$\tilde{O}\left(\mathrm{EC}(p, m+n) + \max_{n_1,\ldots,n_k,p_1,\ldots,p_k} \left\{ \sum_{i=1}^{k} \mathrm{EC}(p_i, m+n_i) : n_1 + \cdots + n_k \leq n,\ p_1 + \cdots + p_k \leq 2p \right\}\right)$$

time with high probability.

In particular $\mathrm{EC}(p, m+n)$ is $\tilde{O}(p\sqrt{m+n}\log U)$ [34] and for unweighted case we have $\mathrm{EC}(p, m+n) = \tilde{O}(p^{4/3})$ [39]. We can obtain two other run times for hypergraphs that provide different tradeoffs. These are obtained by more carefully solving the second part of the isolating cut framework, and transfer ideas from vertex connectivity to hypergraphs.

1. $\sqrt{pn(m+n)^{1.5}}$.
2. $\tilde{O}\left(p(m+n)^{\frac{3\alpha}{2(1+\alpha)}}\beta^{\frac{1}{1+\alpha}}\right)$ for any α, β where $\mathrm{EC}(m, n) \leq m^{1+\alpha}\beta$ (e.g., [31] gives $\mathrm{EC}(m, n) \leq \tilde{O}\left(m^{4/3}U^{1/3}\right)$, which we interpret as $\alpha = 1/3$ and $\beta = U^{1/3}$).

We sketch the proofs of theorems that obtain the preceding bounds.

▶ **Theorem 24.** *The minimum isolating cut in a hypergraph can be computed in*

$$\tilde{O}\left(\sqrt{pn(m+n)^{1.5}}\right)$$

randomized time.

The proof is omitted due to space constraints, and can be found in [10]. We mention that the approach is similar to the the algorithm for element isolating cuts that had a running time of $\tilde{O}\left(\sqrt{m}n^{5/4}\right)$.

[3] One can also reduce to computing s-t cut in a vertex capacitated undirected graph with p edges and $m + n$ nodes, although there does not seem to be any particular advantage with current running time bounds for $\mathrm{EC}(p, m+n)$.

▶ **Theorem 25.** *Suppose* $\mathrm{EC}(m,n) \leq \tilde{O}\left(m^{1+\alpha}\beta\right)$ *for fixed* $\alpha, \beta > 0$. *Then minimum isolating cuts can be computed in* $\tilde{O}\left(p(m+n)^{\frac{3\alpha}{2(1+\alpha)}}\beta^{\frac{1}{1+\alpha}} + p + (m+n)^{1.5}\right)$.

The proof is omitted due to space constraints, and can be found in [10]. We mention that the approach is similar to the algorithm for element isolating cuts that obtained a running time of $\tilde{O}\left(m^{1+o(1)}n^{3/8}U^{1/4} + n^{1.5}\right)$.

References

1 Ashkan Aazami, Joseph Cheriyan, and Krishnam Raju Jampani. Approximation algorithms and hardness results for packing element-disjoint steiner trees in planar graphs. *Algorithmica*, 63(1-2):425–456, 2012.

2 Kazutoshi Ando and Satoru Fujishige. On structures of bisubmodular polyhedra. *Math. Program.*, 74:293–317, 1996.

3 Kazutoshi Ando, Satoru Fujishige, and Takeshi Naitoh. A characterization of bisubmodular functions. *Discret. Math.*, 148(1-3):299–303, 1996.

4 André Bouchet. Greedy algorithm and symmetric matroids. *Math. Program.*, 38(2):147–159, 1987.

5 Gruia Călinescu, Chandra Chekuri, and Jan Vondrák. Disjoint bases in a polymatroid. *Random Structures & Algorithms*, 35(4):418–430, 2009.

6 Ruoxu Cen, Jason Li, Danupon Nanongkai, Debmalya Panigrahi, and Thatchaphol Saranurak. Minimum cuts in directed graphs via $\sqrt{n}$ max-flows. *CoRR*, abs/2104.07898, 2021. `arXiv:2104.07898`.

7 Chandra Chekuri. Some open problems in element connectivity, September 2015. URL: `http://chekuri.cs.illinois.edu/papers/elem-connectivity-open-probs.pdf`.

8 Chandra Chekuri and Nitish Korula. A graph reduction step preserving element-connectivity and packing steiner trees and forests. *SIAM Journal on Discrete Mathematics*, 28(2):577–597, 2014.

9 Chandra Chekuri and Kent Quanrud. Faster algorithms for rooted connectivity in directed graphs, 2021. To appear in ICALP,. `arXiv:2104.07205`.

10 Chandra Chekuri and Kent Quanrud. Isolating cuts, (bi-)submodularity, and faster algorithms for global connectivity problems. *CoRR*, abs/2103.12908, 2021. `arXiv:2103.12908`.

11 Chandra Chekuri, Thapanapong Rukkanchanunt, and Chao Xu. On element-connectivity preserving graph simplification. In Nikhil Bansal and Irene Finocchi, editors, *Algorithms - ESA 2015 - 23rd Annual European Symposium, Patras, Greece, September 14-16, 2015, Proceedings*, volume 9294 of *Lecture Notes in Computer Science*, pages 313–324. Springer, 2015.

12 Chandra Chekuri and Chao Xu. Minimum cuts and sparsification in hypergraphs. *SIAM Journal on Computing*, 47(6):2118–2156, 2018.

13 Joseph Cheriyan and Mohammad R Salavatipour. Packing element-disjoint steiner trees. *ACM Transactions on Algorithms (TALG)*, 3(4):47–es, 2007.

14 Joseph Cheriyan, Santosh Vempala, and Adrian Vetta. Network design via iterative rounding of setpair relaxations. *Combinatorica*, 26(3):255–275, 2006.

15 Julia Chuzhoy and Sanjeev Khanna. An $o(k^3 \log n)$-approximation algorithm for vertex-connectivity survivable network design. In *2009 50th Annual IEEE Symposium on Foundations of Computer Science*, pages 437–441. IEEE, 2009.

16 Lisa Fleischer, Kamal Jain, and David P Williamson. Iterative rounding 2-approximation algorithms for minimum-cost vertex connectivity problems. *Journal of Computer and System Sciences*, 72(5):838–867, 2006.

17 Sebastian Forster, Danupon Nanongkai, Liu Yang, Thatchaphol Saranurak, and Sorrachai Yingchareonthawornchai. Computing and testing small connectivity in near-linear time and queries via fast local cut algorithms. In Shuchi Chawla, editor, *Proceedings of the 2020 ACM-SIAM Symposium on Discrete Algorithms, SODA 2020, Salt Lake City, UT, USA, January 5-8, 2020*, pages 2046–2065. SIAM, 2020.

18 Kyle Fox, Debmalya Panigrahi, and Fred Zhang. Minimum cut and minimum k-cut in hypergraphs via branching contractions. In *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 881–896. SIAM, 2019.

19 Satoru Fujishige and Satoru Iwata. Bisubmodular function minimization. *SIAM J. Discret. Math.*, 19(4):1065–1073, 2005.

20 Harold N. Gabow. Using expander graphs to find vertex connectivity. *J. ACM*, 53(5):800–844, 2006.

21 Pawel Gawrychowski, Shay Mozes, and Oren Weimann. Minimum cut in $(m \log^2 n)$ time. In Artur Czumaj, Anuj Dawar, and Emanuela Merelli, editors, *47th International Colloquium on Automata, Languages, and Programming, ICALP 2020, July 8-11, 2020, Saarbrücken, Germany (Virtual Conference)*, volume 168 of *LIPIcs*, pages 57:1–57:15. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020.

22 Andrew V. Goldberg and Robert E. Tarjan. Finding minimum-cost circulations by successive approximation. *Math. Oper. Res.*, 15(3):430–466, 1990.

23 Anupam Gupta and Jochen Könemann. Approximation algorithms for network design: A survey. *Surveys in Operations Research and Management Science*, 16(1):3–20, 2011.

24 Jianxiu Hao and James B. Orlin. A faster algorithm for finding the minimum cut in a directed graph. *J. Algorithms*, 17(3):424–446, 1994.

25 Ramesh Hariharan, Telikepalli Kavitha, Debmalya Panigrahi, and Anand Bhalgat. An $\tilde{O}(mn)$ gomory-hu tree construction algorithm for unweighted graphs. In *Proceedings of the thirty-ninth annual ACM symposium on Theory of computing*, pages 605–614, 2007.

26 Monika Rauch Henzinger. A static 2-approximation algorithm for vertex connectivity and incremental approximation algorithms for edge and vertex connectivity. *Journal of Algorithms*, 24(1):194–220, 1997.

27 Monika Rauch Henzinger, Satish Rao, and Harold N. Gabow. Computing vertex connectivity: New bounds from old techniques. *J. Algorithms*, 34(2):222–250, 2000.

28 Kamal Jain, Ion Măndoiu, Vijay V Vazirani, and David P Williamson. A primal–dual schema based approximation algorithm for the element connectivity problem. *Journal of Algorithms*, 45(1):1–15, 2002.

29 Stephen Jue and Philip N. Klein. A near-linear time minimum steiner cut algorithm for planar graphs, 2019. `arXiv:1912.11103`.

30 David R Karger. Minimum cuts in near-linear time. *Journal of the ACM (JACM)*, 47(1):46–76, 2000.

31 Tarun Kathuria, Yang P. Liu, and Aaron Sidford. Unit capacity maxflow in almost \$o(mˆ{4/3})\$ time. In *61st IEEE Annual Symposium on Foundations of Computer Science, FOCS 2020, Durham, NC, USA, November 16-19, 2020*, pages 119–130. IEEE, 2020.

32 Regina Klimmek and Frank Wagner. A simple hypergraph min cut algorithm. Technical Report B 96-02, Bericht FU Berlin Fachbereich Mathematik und Informatik, 1996.

33 Guy Kortsarz and Zeev Nutov. Approximating minimum cost connectivity problems. In *Dagstuhl Seminar Proceedings*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2010.

34 Yin Tat Lee and Aaron Sidford. Path finding methods for linear programming: Solving linear programs in $\tilde{O}(\sqrt{\text{rank}})$ iterations and faster algorithms for maximum flow. In *55th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2014, Philadelphia, PA, USA, October 18-21, 2014*, pages 424–433. IEEE Computer Society, 2014.

35 Yin Tat Lee, Aaron Sidford, and Sam Chiu-wai Wong. A faster cutting plane method and its implications for combinatorial and convex optimization. In *2015 IEEE 56th Annual Symposium on Foundations of Computer Science*, pages 1049–1065. IEEE, 2015.

36 Jason Li, Danupon Nanongkai, Debmalya Panigrahi, Thatchaphol Saranurak, and Sorrachai Yingchareonthawornchai. Vertex connectivity in poly-logarithmic max-flows, 2021. To appear in ACM STOC,.

37 Jason Li and Debmalya Panigrahi. Deterministic min-cut in poly-logarithmic max-flows. In *61st IEEE Annual Symposium on Foundations of Computer Science, FOCS 2020, Durham, NC, USA, November 16-19, 2020*, pages 85–92. IEEE, 2020.

38 Nathan Linial, László Lovász, and Avi Wigderson. Rubber bands, convex embeddings and graph connectivity. *Comb.*, 8(1):91–102, 1988.

39 Yang P. Liu and Aaron Sidford. Faster energy maximization for faster maximum flow. In Konstantin Makarychev, Yury Makarychev, Madhur Tulsiani, Gautam Kamath, and Julia Chuzhoy, editors, *Proccedings of the 52nd Annual ACM SIGACT Symposium on Theory of Computing, STOC 2020, Chicago, IL, USA, June 22-26, 2020*, pages 803–814. ACM, 2020.

40 Aleksander Madry. Computing maximum flow with augmenting electrical flows. In Irit Dinur, editor, *IEEE 57th Annual Symposium on Foundations of Computer Science, FOCS 2016, 9-11 October 2016, Hyatt Regency, New Brunswick, New Jersey, USA*, pages 593–602. IEEE Computer Society, 2016.

41 Wai-Kei Mak and D.F. Wong. A fast hypergraph min-cut algorithm for circuit partitioning. *Integration, the VLSI Journal*, 30(1):1–11, 2000.

42 Sagnik Mukhopadhyay and Danupon Nanongkai. Weighted min-cut: sequential, cut-query, and streaming algorithms. In Konstantin Makarychev, Yury Makarychev, Madhur Tulsiani, Gautam Kamath, and Julia Chuzhoy, editors, *Proccedings of the 52nd Annual ACM SIGACT Symposium on Theory of Computing, STOC 2020, Chicago, IL, USA, June 22-26, 2020*, pages 496–509. ACM, 2020.

43 Sagnik Mukhopadhyay and Danupon Nanongkai. A note on isolating cut lemma for submodular function minimization, 2021. `arXiv:2103.15724`.

44 Hiroshi Nagamochi and Toshihide Ibaraki. A linear-time algorithm for finding a sparse k-connected spanning subgraph of a k-connected graph. *Algorithmica*, 7(5&6):583–596, 1992.

45 Danupon Nanongkai, Thatchaphol Saranurak, and Sorrachai Yingchareonthawornchai. Breaking quadratic time for small vertex connectivity and an approximation scheme. In *Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing*, pages 241–252, 2019.

46 Kent Quanrud. Fast approximations for rooted connectivity in weighted directed graphs. *CoRR*, abs/2104.06933, 2021. `arXiv:2104.06933`.

47 Maurice Queyranne. Minimizing symmetric submodular functions. *Mathematical Programming*, 82(1-2):3–12, 1998.

48 Alexander Schrijver. *Combinatorial Optimization - Polyhedra and Efficiency*. Springer, 2003.

49 Mechthild Stoer and Frank Wagner. A simple min-cut algorithm. *Journal of the ACM (JACM)*, 44(4):585–591, 1997.

50 Jan van den Brand, Yin Tat Lee, Yang P. Liu, Thatchaphol Saranurak, Aaron Sidford, Zhao Song, and Di Wang. Minimum cost flows, mdps, and ℓ_1-regression in nearly linear time for dense instances. *CoRR*, abs/2101.05719, 2021. `arXiv:2101.05719`.

51 Jan van den Brand, Yin Tat Lee, Danupon Nanongkai, Richard Peng, Thatchaphol Saranurak, Aaron Sidford, Zhao Song, and Di Wang. Bipartite matching in nearly-linear time on moderately dense graphs. *CoRR*, abs/2009.01802, 2020. `arXiv:2009.01802`.

Majority vs. Approximate Linear Sum and Average-Case Complexity Below NC^1

Lijie Chen
MIT, Cambridge, MA, USA

Zhenjian Lu
University of Warwick, Coventry, UK

Xin Lyu
Tsinghua University, Beijing, China

Igor C. Oliveira
University of Warwick, Coventry, UK

Abstract

We develop a general framework that characterizes strong average-case lower bounds against circuit classes $\mathcal{C}$ contained in NC^1, such as $\mathsf{AC}^0[\oplus]$ and ACC^0. We apply this framework to show:

- *Generic seed reduction*: Pseudorandom generators (PRGs) against $\mathcal{C}$ of seed length $\leq n-1$ and error $\varepsilon(n) = n^{-\omega(1)}$ can be converted into PRGs of *sub-polynomial* seed length.
- *Hardness under natural distributions*: If E (deterministic exponential time) is average-case hard against $\mathcal{C}$ under *some* distribution, then E is average-case hard against $\mathcal{C}$ under the *uniform* distribution.
- *Equivalence between worst-case and average-case hardness*: Worst-case lower bounds against $\mathsf{MAJ} \circ \mathcal{C}$ for problems in E are *equivalent* to strong average-case lower bounds against $\mathcal{C}$. This can be seen as a certain converse to the Discriminator Lemma [Hajnal et al., JCSS'93].

These results were not known to hold for circuit classes that do not compute majority. Additionally, we prove that classical and recent approaches to *worst-case* lower bounds against ACC^0 via communication lower bounds for NOF multi-party protocols [Håstad and Goldmann, CC'91; Razborov and Wigderson, IPL'93] and Torus polynomials degree lower bounds [Bhrushundi et al., ITCS'19] also imply *strong average-case hardness* against ACC^0 under the uniform distribution.

Crucial to these results is the use of *non-black-box* hardness amplification techniques and the interplay between ***Majority*** (MAJ) and ***Approximate Linear Sum*** ($\widetilde{\mathsf{SUM}}$) gates. Roughly speaking, while a MAJ gate outputs 1 when the sum of the m input bits is at least $m/2$, a $\widetilde{\mathsf{SUM}}$ gate computes a real-valued bounded weighted sum of the input bits and outputs 1 (resp. 0) if the sum is close to 1 (resp. close to 0), with the promise that one of the two cases always holds. As part of our framework, we explore ideas introduced in [Chen and Ren, STOC'20] to show that, for the purpose of proving lower bounds, a top layer MAJ gate is *equivalent* to a (weaker) $\widetilde{\mathsf{SUM}}$ gate. Motivated by this result, we extend the algorithmic method and establish stronger lower bounds against bounded-depth circuits with layers of MAJ and $\widetilde{\mathsf{SUM}}$ gates. Among them, we prove that:

- *Lower bound*: NQP does not admit fixed quasi-polynomial size $\mathsf{MAJ} \circ \widetilde{\mathsf{SUM}} \circ \mathsf{ACC}^0 \circ \mathsf{THR}$ circuits.

This is the first explicit lower bound against circuits with distinct layers of MAJ, $\widetilde{\mathsf{SUM}}$, and THR gates. Consequently, if the aforementioned equivalence between MAJ and $\widetilde{\mathsf{SUM}}$ as a *top gate* can be extended to *intermediate layers*, long sought-after lower bounds against the class $\mathsf{THR} \circ \mathsf{THR}$ of depth-2 polynomial-size threshold circuits would follow.

2012 ACM Subject Classification Theory of computation

Keywords and phrases circuit complexity, average-case hardness, complexity lower bounds

Digital Object Identifier 10.4230/LIPIcs.ICALP.2021.51

Category Track A: Algorithms, Complexity and Games

Related Version *Full Version*: `https://eccc.weizmann.ac.il/report/2021/040/` [11]

48th International Colloquium on Automata, Languages, and Programming (ICALP 2021).
Editors: Nikhil Bansal, Emanuela Merelli, and James Worrell; Article No. 51; pp. 51:1–51:20

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Funding Lijie Chen is supported by an IBM fellowship. This work received support from the Royal Society University Research Fellowship URF\R1\191059.

Acknowledgements We would like to thank William Hoza for asking whether PRGs for $\mathcal{C}$ imply average-case lower bounds against $\mathcal{C}$ under the uniform distribution. We are also grateful to Roei Tell for the observation that part of our equivalence theorem extends to the constant-error case as well.

1 Introduction

1.1 Overview

Establishing the *intractability of computations* and understanding the *power of randomness* in algorithms are among the most basic open problems in theoretical computer science. The theory of computational pseudorandomness provides a firm link between these two research directions. One of the most celebrated developments in this area is a proof that if E (deterministic exponential time $2^{O(n)}$) requires Boolean circuits of exponential size then $\mathsf{P} = \mathsf{BPP}$ [30, 43]. This result and its underlying techniques provide a robust mathematical theory that connects *worst-case lower bounds*, *average-case hardness*, and the construction of *pseudorandom generators*.

Unfortunately, a large part of this beautiful and far-reaching theory is not known to survive in *restricted* computational settings. For instance, while we know since the eighties that E cannot be $(1/2 + n^{-1/2+\Omega(1)})$-approximated by $\mathsf{AC}^0[\oplus]$ [40], it is an important open problem to obtain *strong* average-case hardness results of the form $1/2 + n^{-k}$ for all k and pseudorandom generators against this circuit class. The fact that existing connections between hardness and pseudorandomness do not apply in restricted settings is significant, given that known unconditional results and existing lower bound frontiers lie within weak sub-classes of NC^1, such as ACC^0.

Several works (e.g. [45, 23, 42, 36, 5, 22, 47, 29]) have investigated the difficulty of extending the hardness vs. randomness theory and its consequences to restricted circuit classes. Roughly speaking, these results show that standard "black-box" techniques to amplify computational hardness and construct pseudorandom generators *require* the underlying circuit class $\mathcal{C}$ to be closed under *majority*. However, obtaining lower bounds against circuit classes that are closed under majority is a notorious open problem. This leaves us in this unsatisfying situation where many benefits of the theory mentioned above only apply to settings where current circuit-analysis techniques do not hold. In other words, we have the following "lose-lose" scenario: above TC^0 we have no lower bounds, while below it we have lower bounds but no hardness amplification.

In this work, we explore *non-black-box* techniques to overcome this difficulty, obtaining a general connection between worst-case lower bounds, strong average-case hardness, and pseudorandomness for *weak circuit classes*. Our results build on recent ideas of Chen and Ren [14] employed in the context of the algorithmic method. Using our techniques, we are able to establish fundamental equivalences that were previously only known for circuit classes containing TC^0. As a consequence, the new results are widely applicable and can affect *current frontiers in circuit complexity theory*.

A crucial ingredient in our proofs is the interplay between Majority (MAJ) and Approximate Linear Sum ($\widetilde{\mathsf{SUM}}$) gates. Roughly speaking, while a MAJ gate outputs 1 when the sum of the m input bits is at least $m/2$, a $\widetilde{\mathsf{SUM}}$ gate computes a real-valued bounded weighted sum of the input bits and outputs 1 (resp. 0) if the sum is close to 1 (resp. close to 0), with

the promise that one of the two cases always holds. $\widetilde{\mathsf{SUM}}$ gates are significantly simpler than MAJ gates (e.g. MAJ has approximate degree [38] of order $\Omega(m)$), but still powerful enough to implement useful computations, such as hardness amplification for *specific* problems (a non-black-box element).

Complementing our results about the average-case complexity of restricted circuit classes, we obtain the first unconditional lower bounds against bounded-depth circuits with distinct layers of MAJ, $\widetilde{\mathsf{SUM}}$, and THR gates. These results suggest that further investigating the relation between MAJ and $\widetilde{\mathsf{SUM}}$ might be a path to lower bounds against depth-2 threshold circuits, a long-standing open problem in complexity theory (cf. [18, 9]).

1.2 Results and techniques

To begin with, we recall some definitions for linear sums of functions. Our notation is taken from previous work [50, 15, 14, 13] on lower bounds via the algorithmic method. Let $\mathcal{C}$ be a class of functions from $\{0,1\}^n \to \{0,1\}$.

$\mathsf{SUM} \circ \mathcal{C}$-circuits. A $\mathsf{SUM} \circ \mathcal{C}$-circuit $C\colon \{0,1\}^n \to \mathbb{R}$ is a circuit that can be written as $C(x) = \sum_{i=1}^{\ell} \alpha_i \cdot C_i(x)$, where each α_i is a real, and each $C_i \in \mathcal{C}$. Here ℓ is called the *sparsity* of C, and is denoted as $\mathsf{sparsity}(C)$. We also use $\mathsf{complexity}(C)$ to denote $\max(\ell, \sum_{i=1}^{\ell} |\alpha_i|)$. Furthermore, if a $\mathsf{SUM} \circ \mathcal{C}$-circuit C always outputs values in the interval $[0,1]$, we say it is a $[0,1]$-$\mathsf{SUM} \circ \mathcal{C}$-circuit.

$\widetilde{\mathsf{SUM}}_{\delta} \circ \mathcal{C}$-circuits. Let $\delta \in [0, 0.5)$. A $\widetilde{\mathsf{SUM}}_{\delta} \circ \mathcal{C}$-circuit $C\colon \{0,1\}^n \to \{0,1\}$ is defined by a $\mathsf{SUM} \circ \mathcal{C}$-circuit $L\colon \{0,1\}^n \to \mathbb{R}$ satisfying the following promise: for every $x \in \{0,1\}^n$, either $|L(x) - 1| \leq \delta$ or $|L(x)| \leq \delta$. (We stress that this promise is only required over inputs x to the $\mathsf{SUM} \circ \mathcal{C}$-circuit L, and not over all possible input values to the top SUM gate.) We say $C(x) = 1$ if $|L(x) - 1| \leq \delta$ and $C(x) = 0$ otherwise. The sparsity and the complexity of C is defined as the sparsity and the complexity of L, respectively.

For a circuit class $\mathcal{C}$, we use $\mathsf{SUM} \circ \mathcal{C}$, $[0,1]$-$\mathsf{SUM} \circ \mathcal{C}$, and $\widetilde{\mathsf{SUM}}_{\delta} \circ \mathcal{C}$ to denote the collection of such circuit families with at most $\mathrm{poly}(n)$ complexity. When $\mathcal{C}$ has a clear notion of complexity, such as circuit size, this also means that the involved $\mathcal{C}$-subcircuits are of polynomial size. In some statements we might refer to classes such as $\widetilde{\mathsf{SUM}}_{\delta} \circ \mathcal{C}[s]$ to emphasize a specific upper bound s on the complexities of $\mathcal{C}$-subcircuits and of the top gate.

Notation for standard concepts. A $\mathsf{MAJ}\colon \{0,1\}^m \to \{0,1\}$ gate $\mathsf{MAJ}(y_1, \ldots, y_m)$ outputs 1 if and only if $\sum_i y_i \geq m/2$. A $\mathsf{THR}\colon \{0,1\}^m \to \{0,1\}$ gate is described by weights $w_1, \ldots, w_m, \theta \in \mathbb{R}$ and outputs 1 if and only if $\sum_i w_i y_i \geq \theta$.

For a probability distribution $\mathcal{D}$ over $\{0,1\}^n$ and Boolean functions $f, g\colon \{0,1\}^n \to \{0,1\}$, we say that f is γ-approximated by g over $\mathcal{D}$ if $\Pr_{x \sim \mathcal{D}}[f(x) = g(x)] \geq \gamma$. For convenience, circuit lower bounds involving approximations of the form $1/2 + 1/n^{\omega(1)}$ might be informally referred to as *strong average-case lower bounds* or simply *strong correlation bounds*.

Our results refer to non-uniform circuit classes, and we use $\mathcal{C}_1 \circ \mathcal{C}_2$ to refer to circuit families consisting of a top circuit from $\mathcal{C}_1$ composed with bottom circuits from $\mathcal{C}_2$.[1]

We use $\mathcal{U}_n$ to denote the uniform distribution over $\{0,1\}^n$. A distribution $\mathcal{D}$ *ε-fools* a function $f\colon \{0,1\}^n \to \{0,1\}$ if $|\Pr[f(\mathcal{D}) = 1] - \Pr[f(\mathcal{U}_n) = 1]| \leq \varepsilon$. We say that a sequence $G_n\colon \{0,1\}^{\ell(n)} \to \{0,1\}^n$ is an infinitely often PRG against a circuit class $\mathcal{C}$ with error ε

[1] As usual, in the case of $\mathcal{C}_2 = \mathsf{ACC}^0$, where $\mathsf{ACC}^0 = \bigcup_{m \in \mathbb{N}} \mathsf{AC}^0[m]$ with m here representing the modulo, we require that each $\mathcal{C}_2$-subcircuit of a circuit D from $\mathcal{C}_1 \circ \mathcal{C}_2$ uses the same fixed m.

(i.e. ε-PRG) and seed length ℓ if G_n is computable in time $2^{O(\ell(n))}$ and for infinitely many values of n, the induced distribution $G_n(\mathcal{U}_{\ell(n)})$ $\varepsilon(n)$-fools each function $f\colon \{0,1\}^n \to \{0,1\}$ in $\mathcal{C}$.

1.2.1 Equivalences for worst-case and strong average-case hardness

Our first contribution is a general result that tightly connects worst-case lower bounds, strong average-case hardness, and pseudorandomness in *restricted computational models*.

▶ **Theorem 1** (Non-black-box equivalences for worst-case and strong average-case hardness)**.** *Let $\mathcal{C}$ be a circuit class contained in NC^1 that is closed under negations and under a bottom layer of juntas over $O(1)$ input bits. The following statements are equivalent:*

1. *There is $L \in \mathsf{E}$ such that $L \notin \widetilde{\mathsf{SUM}}_{1/3} \circ \mathcal{C}$.*
2. *There is $L \in \mathsf{E}$ and $\delta \geq 1/\mathsf{poly}(n)$ such that $L \notin \widetilde{\mathsf{SUM}}_{\delta} \circ \mathcal{C}$.*
3. *There is $L \in \mathsf{E}$ such that $L \notin \mathsf{MAJ} \circ \mathcal{C}$.*
4. *There is $L \in \mathsf{E}$ such that, for every $k \geq 1$, L cannot be computed by probabilistic $\mathcal{C}$-circuits with error $1/2 - 1/n^k$.*[2]
5. *There is $L \in \mathsf{E}$ and a distribution ensemble $\mathfrak{D}$ such that for every $k \geq 1$, L cannot be $(1/2 + n^{-k})$-approximated by $\mathcal{C}$ under $\mathfrak{D}$.*
6. *There is $L \in \mathsf{E}$ such that for every $k \geq 1$, L cannot be $(1/2 + n^{-k})$-approximated by $\mathcal{C}$ under the uniform distribution.*
7. *There is $L \in \mathsf{E}$ that cannot be approximated by $[0,1]$-$\mathsf{SUM} \circ \mathcal{C}$ within ℓ_1 distance $1/3$.*[3]
8. *There is $L \in \mathsf{E}$ and $\delta \geq 1/\operatorname{poly}(n)$ such that L cannot be approximated by $[0,1]$-$\mathsf{SUM} \circ \mathcal{C}$ within ℓ_1 distance δ.*
9. *There is an i.o. ε-PRG G against $\mathcal{C}$ with seed length $n-1$ and error $\varepsilon(n) \leq n^{-\omega(1)}$.*[4]
10. *For each $\gamma > 0$, there is an i.o. ε-PRG against $\mathcal{C}$ with seed length n^γ and $\varepsilon(n) \leq n^{-\omega(1)}$.*

This result can be applied to a variety of natural circuit classes, such as $\mathsf{AC}^0[\oplus]$, ACC^0, and constant-degree polynomial threshold functions (PTFs). We stress that while Theorem 1 requires the circuit class $\mathcal{C}$ to be contained in NC^1, in circuit complexity this is the most interesting case for the result. More precisely, for circuit classes that are above NC^1, it is well known that worst-case hardness for a problem in E can be converted into average-case hardness and PRGs. (Furthermore, NC^1 is closed under a top MAJ or $\widetilde{\mathsf{SUM}}$ gate.) We remark that Theorem 1, with appropriate modifications, can be adapted to other uniform complexity classes, such as $\mathsf{BPE} = \mathsf{BPTIME}[2^{O(n)}]$ and PSPACE. For simplicity, we restrict our discussion to E.

We observe that a connection between worst-case hardness and *weak* average-case hardness for functions in E has been established in [20], under the assumption that the circuit class $\mathcal{C}$ contains AC^0 and is closed under composition. In contrast to their work, we have a much weaker assumption on $\mathcal{C}$, and our setting of parameters allows us to obtain equivalences to PRGs and to derive consequences that do not follow from their results.

We now highlight three fundamental consequences of Theorem 1. Note that, while our proof employs $\widetilde{\mathsf{SUM}}$ gates in important ways, none of these results refer to such gates.

---

[2] Following standard terminology, a probabilistic $\mathcal{C}$-circuit F is simply a distribution of $\mathcal{C}$-circuits. We say that F computes a Boolean function g with error ε if for every input x we have $\Pr_F[F(x) \neq g(x)] \leq \varepsilon$.

[3] In other words, there is no family of circuits $F_n \in [0,1]$-$\mathsf{SUM} \circ \mathcal{C}$ such that $\mathsf{E}_{x \sim \{0,1\}^n}[|L(x) - F_n(x)|] \leq 1/3$ for all large n. This notion plays a crucial role in [13] and other related works.

[4] More precisely, for each choice of k, there is an infinite set $S_k \subseteq \mathbb{N}$ such that G fools circuits from $\mathcal{C}[n^k]$ on inputs of length $n \in S_k$ with error $\varepsilon(n) \leq n^{-k}$.

1. Seed reduction for PRGs. Perhaps surprisingly, the equivalence between Items 9 and 10 of Theorem 1 shows the existence of a *generic seed reduction phenomenon* for weak circuit classes. Thus to construct i.o. PRGs of sub-polynomial seed length for a class $\mathcal{C}$ satisfying the conditions of this result it is enough to construct a non-trivial i.o. PRG (i.e. of seed length $\leq n-1$) with small error. In particular, improving the error parameter of the PRG against $\mathsf{AC}^0[\oplus]$ described in [16] to inverse-super-polynomial would lead to major consequences for $\mathsf{AC}^0[\oplus]$-circuits.

2. Hardness under some distribution implies hardness under the uniform distribution. Theorem 1 also has important implications to our understanding of the average-case hardness of problems in E with respect to weak circuit classes. This is an immediate consequence of Items 5 and 6, which establish the result for strong average-case hardness of the form $1/2 + 1/n^{\omega(1)}$. In the full version of this paper [11], we observe that our techniques can also translate constant-error average-case hardness under an arbitrary distribution to constant-error average-case hardness under the uniform distribution. An interesting application of these results is that the existence of a PRG against $\mathcal{C}$, which was only known to imply hardness under some distribution (see e.g. Section 3 of [46]), also implies hardness with respect to the uniform distribution (which in turn is sufficient to construct PRGs).

3. Equivalence between worst-case and average-case hardness. The well-known Discriminator Lemma from Hajnal et al. [24] has found numerous applications in circuit complexity lower bounds. It shows that if a Boolean function f cannot be $(1/2+1/\operatorname{poly}(n))$-approximated by a class $\mathcal{C}$ then f is not in $\mathsf{MAJ} \circ \mathcal{C}$. In other words, one can lift an average-case lower bound against $\mathcal{C}$ to a worst-case lower bound against the stronger class $\mathsf{MAJ} \circ \mathcal{C}$. Interestingly, the equivalence between Items 3 and 6 in Theorem 1 shows that, for the purpose of proving lower bounds for a problem in E, a worst-case lower bound against $\mathsf{MAJ} \circ \mathcal{C}$ is actually *equivalent* to a strong average-case lower bound against $\mathcal{C}$. To our knowledge, this was previously unknown for weak computational models.[5]

A consequence of Theorem 1 relevant to the study of $\widetilde{\mathsf{SUM}}$ gates is that if $\mathsf{E} \not\subseteq \widetilde{\mathsf{SUM}}_{\delta} \circ \mathcal{C}$ for some $\delta(n) = 1/n^c$ then $\mathsf{E} \not\subseteq \widetilde{\mathsf{SUM}}_{1/3} \circ \mathcal{C}$.[6] Another interesting implication is that the average-case lower bounds against $[0,1]$-$\mathsf{SUM} \circ \mathcal{C}$ under ℓ_1 distance investigated in [13] are *necessary* and *sufficient* for strong average-case hardness against $\mathcal{C}$.

Next, we discuss some of the techniques behind Theorem 1.

Theorem 1: Techniques. As alluded to above, the proof of Theorem 1 relies on non-black-box hardness amplification techniques explored by Chen and Ren [14] and on a careful balance between the *strength* and *weakness* of $\widetilde{\mathsf{SUM}}$ gates. To give some intuition, we discuss the main ingredients behind a more direct proof of the following equivalence, which also explains the assumptions on the circuit class $\mathcal{C}$:

$$\text{Worst-case hardness against } \widetilde{\mathsf{SUM}} \circ \mathcal{C} \iff \text{i.o. PRGs against } \mathcal{C} \text{ with error } \varepsilon = n^{-\omega(1)}.$$

[5] We also remark that it was known [19, 28, 33] before that for general circuit class $\mathcal{C}$, weak average-case hardness against $\mathsf{MAJ} \circ \mathcal{C}$ implies strong average-case hardness against $\mathcal{C}$.

[6] We note that a simple error amplification technique for $\widetilde{\mathsf{SUM}}$ (see [11]) blows up the complexity of the involved $\widetilde{\mathsf{SUM}} \circ \mathcal{C}$-circuits to quasi-polynomial when amplifying from constant-error approximation to inverse polynomial. For this reason, it does not establish this implication.

While it is possible to show that a $\widetilde{\mathsf{SUM}}$ gate can be efficiently simulated by a MAJ gate,[7] the opposite simulation does not hold (e.g. consider approximate degree). In this sense, $\widetilde{\mathsf{SUM}}$ gates are indeed weak. Still, it is possible to show essentially that, for a certain *specific* NC^1-hard problem L contained in P, a $\widetilde{\mathsf{SUM}}$ gate of polynomial complexity can implement a hardness amplification proof: roughly speaking, a weak approximator circuit for L can be transformed into a correct circuit for L by incurring only a top $\widetilde{\mathsf{SUM}}$ gate overhead. This allows us to employ the following win-win analysis. Either the NC^1-hard problem L is $1/2 + n^{-k}$-hard against $\mathcal{C}$ on infinitely many input lengths for every choice of k, in which case an i.o. PRG against $\mathcal{C}$ can be constructed from L using standard techniques under the assumption that $\mathcal{C}$ is closed under bottom layer $O(1)$-juntas, or there is a choice of k such that L can be $1/2 + n^{-k}$ approximated by $\mathcal{C}$-circuits on large enough input lengths. The latter implies via the hardness amplification reconstruction routine that $L \in \widetilde{\mathsf{SUM}} \circ \mathcal{C}$, which in turns yields $\mathsf{NC}^1 \subseteq \widetilde{\mathsf{SUM}} \circ \mathcal{C}$ using the NC^1-hardness of L (which in fact admits ultra efficient reductions). Now under our assumption that $\mathcal{C} \subseteq \mathsf{NC}^1$, it is easy to see that $\mathsf{NC}^1 = \widetilde{\mathsf{SUM}} \circ \mathcal{C}$. As a consequence, a worst-case lower bound against $\widetilde{\mathsf{SUM}} \circ \mathcal{C}$ provides a worst-case lower bound against NC^1, and again, PRGs can be constructed from such an assumption via standard methods (since NC^1 admits black-box worst-case to average-case amplification).

For the other direction, we start with an i.o. PRG G against $\mathcal{C}$ that might have a large seed length but guarantees *low error* $\varepsilon(n) = n^{-\omega(1)}$. Here the important insight is that a low error PRG that fools $\mathcal{C}$ also fools linear combinations of functions in $\mathcal{C}$ with bounded coefficients. This implies that G fools $\widetilde{\mathsf{SUM}} \circ C$. Another standard argument shows that from such a PRG one can define a function in E that is worst-case hard against $\widetilde{\mathsf{SUM}} \circ C$.

We stress that two crucial ingredients of our equivalence theorem are the existence of the hard problem L mentioned above and the use of $\widetilde{\mathsf{SUM}}$ gates. The hard language L is actually a pair of problems CMD and DCMD with very useful structural properties (see Section 2.2). They have been explored in a few other works (e.g. [31, 4, 20, 1]), and are tightly connected to *decomposable randomized encodings*, which are well-studied in cryptography (see [3]). The fruitful interaction between these problems and $\widetilde{\mathsf{SUM}}$ gates was first noticed by [14] in the context of the algorithmic method and is a crucial ingredient in their proof that NQP is strongly average-case hard against ACC^0.

While the proof of Theorem 1 avoids the black-box "barrier" and applies to circuit classes that are not assumed to be closed under majority, our techniques come with certain limitations. As a consequence of our indirect analysis via a win-win argument, Theorem 1 does not provide almost-everywhere equivalences for some items and does not scale to large circuit size bounds above quasi-polynomial. These are important directions for future work.

Applications to ACC^0-circuits lower bound approaches. As a concrete application of Theorem 1 to current frontiers in circuit complexity, we explore its consequences to the average-case complexity of ACC^0. We use our framework to show that existing "combinatorial" approaches to worst-case lower bounds would also provide *strong average-case hardness* against ACC^0. Before stating this result, we briefly recall some concepts.

[7] It is possible to approximate all coefficients of the bounded linear sum using sums of powers of 2^i with $i \in \mathbb{Z}$, then multiply the linear sum by an appropriate power of 2 to obtain integer coefficients, and finally simulate the resulting sum by an appropriate THR gate with polynomial weights, which can be translated to a MAJ gate using duplicated input wires and by negating input variables if necessary.

Let $\mathbb{T} = \mathbb{R}/\mathbb{Z}$ be the one-dimensional torus. A *torus polynomial* [7] (see also [34]) is a real polynomial $p(x_1, \ldots, x_n)$ restricted to the domain $\{0,1\}^n$ and evaluated modulo one.[8] For the purpose of representing the output of a Boolean function $f\colon \{0,1\}^n \to \{0,1\}$ as a value in $\mathbb{T}$, we map the output bit $f(x)$ to $f(x)/2$. For $\delta < 1/4$, we say that f is δ-approximated by a degree-d torus polynomial if there is a degree-d real polynomial $p(x_1, \ldots, x_n)$ such that if $f(x) = 1$ then $p(x) - \lfloor p(x) \rfloor \in [1/2 - \delta, 1/2 + \delta]$ and if $f(x) = 0$ then $p(x) - \lfloor p(x) \rfloor \in [0, \delta] \cup [1 - \delta, 1)$. A recent approach proposed by [7] shows that ACC^0 lower bounds follow from torus polynomial degree lower bounds for approximating a Boolean function.

The number-on-forehead (NOF) multi-party communication model was introduced by [8], and work of [26, 41] show that explicit communication lower bounds in this model (even in the single-round model where all players simultaneously communicate to a referee) imply lower bounds against SYM^+-circuits, which are known to simulate ACC^0 [6].

▶ **Theorem 2** (Lifting worst-case ACC^0 lower bound approaches to strong correlation bounds)**.** *Consider the following statements:*

1. **Torus Polynomials:** *There is a language $L \in \mathsf{E}$ and a function $\delta(n) \geq 1/\operatorname{poly}(n)$ such that L does not have δ-approximation torus polynomials of degree $\operatorname{polylog}(n)$.*
2. **NOF Protocols:** *There is a language in E that does not admit (single-round) NOF multi-party protocols with $\operatorname{polylog}(n)$ parties of communication cost $\operatorname{polylog}(n)$.*

In each case, if the corresponding statement holds then there is a language in E that cannot be $(1/2 + 1/\operatorname{poly}(n))$-approximated under the uniform distribution by ACC^0.

As a consequence, lower bounds against these models provide i.o. PRGs of sub-polynomial seed length against ACC^0.

Theorem 2: Techniques. It is not hard to adapt classical techniques to show that if a Boolean function can be approximated by torus polynomials of bounded degree, then it can also be computed by NOF protocols of low complexity. For this reason, in order to prove Theorem 2 it is sufficient to obtain average-case hardness against ACC^0 from degree lower bounds for torus polynomials approximating Boolean functions.[9] To achieve this, we refine the argument of [7] and invoke our framework. In more detail, we show the stronger result that even functions families in $\widetilde{\mathsf{SUM}} \circ \mathsf{ACC}^0$ can be approximated by low-degree torus polynomials. This yields the result using the equivalence between Items 6 and 2 in Theorem 1.

To establish this claim, we make use of low degree "middle-bit polynomials" [21], a sub-class of SYM^+-circuits that is strong enough to simulate ACC^0. By a careful adaptation of the argument of [7], we are able to show that a linear sum (with bounded coefficients) of middle-bit polynomials with a special structure can be converted into a torus polynomial. The argument is somewhat subtle, and involves the manipulation of universal circuits for depth-d $\mathsf{ACC}^0[s]$ in order to enforce similar parameters for all middle-bit polynomials feeding the top $\widetilde{\mathsf{SUM}}$ gate. The details appear in the full version of this paper [11].

[8] By a value y (mod 1) we mean its fractional part given by $y - \lfloor y \rfloor$, where the floor function $\lfloor y \rfloor$ denotes the largest integer less than or equal to y. For instance, 1.37 (mod 1) is 0.37 and -2.21 (mod 1) is 0.79.

[9] Alternatively, earlier work on ACC^0 already showed that $\mathsf{MAJ} \circ \mathsf{ACC}^0$-circuits can be simulated by NOF protocols of low communication. Therefore, the NOF protocols part of Theorem 2 follows directly from our Theorem 1.

1.2.2 Lower bounds against circuits with layers of $\widetilde{\mathsf{SUM}}$ and MAJ gates

Observe that Theorem 1 (via Items 1, 2, and 3) establishes the following equivalence: for the purpose of proving circuit lower bounds for a function in E, a top layer MAJ gate is *equivalent* to a top layer $\widetilde{\mathsf{SUM}}$ gate. Given that $\widetilde{\mathsf{SUM}}$ is simpler than MAJ, and lower bounds against $\widetilde{\mathsf{SUM}} \circ \mathcal{C}$ offer a path to correlation bounds and PRGs against $\mathcal{C}$, obtaining a better understanding of $\widetilde{\mathsf{SUM}}$ gates in Boolean circuits might have significant benefits.

In this section, we explore *unconditional* lower bounds against circuits with layers of MAJ and $\widetilde{\mathsf{SUM}}$ gates. Our results are connected to the long-standing problem of showing explicit lower bounds against $\mathsf{THR} \circ \mathsf{THR}$, the class of polynomial-size depth-2 threshold circuits (where size is measured by number of gates). For convenience of the reader, we review below some results related to this frontier.

Threshold circuits. Recall that a threshold gate THR over m input bits is described by weights $w_1, \ldots, w_m, \theta \in \mathbb{R}$. It outputs 1 on an input $y \in \{0,1\}^m$ if and only if $\sum_i w_i y_i \geq \theta$. It is known that every such gate can be implemented with integer weights of magnitude $2^{O(m \log m)}$ (see [25]). In the context of polynomial size circuits, by duplicating input wires a MAJ gate can be equivalently defined as the restriction of a THR gate to polynomially bounded integer weights. It was shown that $\mathsf{MAJ} \circ \mathsf{THR} = \mathsf{MAJ} \circ \mathsf{MAJ}$ and $\mathsf{THR} \circ \mathsf{THR}$ is contained in $\mathsf{MAJ} \circ \mathsf{MAJ} \circ \mathsf{MAJ}$ [18]. Exponential lower bounds are known against $\mathsf{THR} \circ \mathsf{MAJ}$-circuits [17], and $\mathsf{THR} \circ \mathsf{MAJ}$ is strictly contained in $\mathsf{THR} \circ \mathsf{THR}$ [9]. Recently, [32] described a function in P that requires $\mathsf{THR} \circ \mathsf{THR}$-circuits of size (measured by the number of gates) nearly $n^{3/2}$. This is the strongest known lower bound against this class (see their work for extensions to other circuit size measures) for a function in P. It is also known that E^{NP} does not have $n^{2-\varepsilon}$-size $\mathsf{THR} \circ \mathsf{THR}$-circuits for every constant $\varepsilon > 0$ [2, 44].

$\mathsf{LTF}^s \circ \mathcal{C}$-circuits: An intermediary class between $\mathsf{MAJ} \circ \mathcal{C}$ and $\mathsf{THR} \circ \mathcal{C}$. In order to make progress toward showing super-polynomial lower bounds against $\mathsf{THR} \circ \mathsf{THR}$-circuits, we study a newly defined gate LTF^s whose power lies between MAJ and THR.[10] Let $\mathsf{SUM}^{\infty} \circ \mathcal{C}$ be the relaxation of $\mathsf{SUM} \circ \mathcal{C}$ to an *unrestricted* top SUM gate (i.e. the top gate can use arbitrary real coefficients that might not be polynomially bounded). For a given function s and a circuit class $\mathcal{C}$, we say that a function f admits a $\mathsf{LTF}^s \circ \mathcal{C}$-circuit of size S if there is a circuit $D \in \mathsf{SUM}^{\infty} \circ \mathcal{C}$ such that the following hold: (1) $f(x) = 1$ if and only if $D(x) \geq 0$; (2) $|D(x)| \in (1/s, s)$ for every $x \in \{0,1\}^n$; (3) the total size of the $\mathcal{C}$-subcircuits of D is at most S. Note that unrestricted weights are allowed in the top gate, but we are promised that on each input x the value $D(x)$ is neither too close to 0 nor too large in magnitude.[11]

We are able to extend the algorithmic method [49] to show that #SAT algorithms for a circuit class $\mathcal{C}$ imply worst-case lower bounds against $\mathsf{LTF}^s \circ \mathcal{C}$ and average-case lower bounds against $\widetilde{\mathsf{SUM}} \circ \mathcal{C}$. Let $\mathsf{NQP} = \mathsf{NTIME}[2^{\mathrm{polylog}(n)}]$ be the class of languages computable in non-deterministic quasi-polynomial time. We say that a circuit class $\mathcal{C}$ is *nice* if $\mathcal{C}$ is closed under negation, (bottom) projections, and a top AND gate of unbounded fan-in, and in addition $\mathcal{C}$-circuits of size s admit general circuits of depth $O(\log s)$. Examples of nice circuit classes include AC^0, ACC^0, and $\mathsf{AC}^0[\oplus] \circ \mathsf{THR}$.

[10] LTF denotes linear threshold function, another standard name for THR. We employ both names in this paper to make a clear distinction between the new gates and THR.

[11] Note that we only impose this constraint for each input x of the combined $\mathsf{SUM}^{\infty} \circ \mathcal{C}$-circuit, and not over all possible input strings for the top gate.

▶ **Theorem 3** (Stronger lower bounds from #SAT algorithms). *Let $\mathcal{C}$ be a nice circuit class. Suppose there is a constant $\varepsilon > 0$ such that, given a $\mathcal{C}$-circuit of size 2^{n^ε} over n input variables, its number of satisfying assignments can be deterministically computed in time 2^{n-n^ε}. Then the following statements hold:*

1. *For every constant $k > 0$, NQP does not have $\mathsf{LTF}^{2^{\log^k n}} \circ \mathcal{C}$-circuits of size $2^{\log^k n}$.*
2. *For every choice of constants $k > 0$ and $\delta \in (0, 0.5)$, NQP cannot be $(1/2 + 2^{-\log^k n})$-approximated by $\widetilde{\mathsf{SUM}}_\delta \circ \mathcal{C}$-circuits where both the sparsity of the top SUM-gate and the size of the bottom layer $\mathcal{C}$-circuits are at most $2^{\log^k n}$.*[12]

To our knowledge, these two circuit lower bound consequences are incomparable. By combining Theorem 3 with existing #SAT algorithms for $\mathcal{C} = \mathsf{ACC}^0 \circ \mathsf{THR}$-circuits [51], we obtain the following unconditional lower bounds.

▶ **Corollary 4** (Lower bounds against circuits with $\widetilde{\mathsf{SUM}}$, THR, and MAJ gates). *The following results hold:*

1. *For every constant $k > 0$, NQP does not admit $\mathsf{LTF}^{2^{\log^k n}} \circ \mathsf{ACC}^0 \circ \mathsf{THR}$-circuits of size $2^{\log^k n}$.*
2. *For every choice of constants $k > 0$ and $\delta \in (0, 0.5)$, NQP cannot be $(1/2 + 2^{-\log^k n})$-approximated by $\widetilde{\mathsf{SUM}}_\delta \circ \mathsf{ACC}^0 \circ \mathsf{THR}$-circuits where the top sum has sparsity $2^{\log^k n}$ and all $\mathsf{ACC}^0 \circ \mathsf{THR}$-subcircuits have size $2^{\log^k n}$.*
3. *For every choice of constants $k > 0$ and $\delta \in (0, 0.5)$, NQP cannot be computed by $\mathsf{MAJ} \circ \widetilde{\mathsf{SUM}}_\delta \circ \mathsf{ACC}^0 \circ \mathsf{THR}$-circuits where the top MAJ gate has fan-in $2^{\log^k n}$ and all $\widetilde{\mathsf{SUM}}_\delta \circ \mathsf{ACC}^0 \circ \mathsf{THR}$-subcircuits have size and sparsity $2^{\log^k n}$.*

To contrast these results with previous work, we note that [15, Theorem 15] gave a *worst-case* lower bound against $\widetilde{\mathsf{SUM}}_\delta \circ \mathsf{ACC}^0 \circ \mathsf{THR}$-circuits with any *constant* error δ less than $1/2$. Also, [14, Section 5.2] showed a strong average-case lower bound against $\widetilde{\mathsf{SUM}}_\delta \circ \mathsf{ACC}^0 \circ \mathsf{THR}$-circuits, where the top sum gate has *zero error* (i.e., $\delta = 0$). Consequently, Corollary 4 Item 2 simultaneously strengthens both results. On the other hand, Corollary 4 Item 3 shows the first lower bound against circuits combining layers of $\widetilde{\mathsf{SUM}}_{1/3}$, MAJ, and THR gates.

Before discussing our techniques in more detail, we mention an open problem and its connection to $\mathsf{THR} \circ \mathsf{THR}$ lower bounds. Recall that this class is contained in $\mathsf{MAJ} \circ \mathsf{MAJ} \circ \mathsf{MAJ}$. In light of the super-polynomial lower bound against $\mathsf{MAJ} \circ \widetilde{\mathsf{SUM}}_\delta \circ \mathsf{ACC}^0 \circ \mathsf{THR}$ from Corollary 4 Item 3, it would be very interesting to understand the relation between MAJ gates and $\widetilde{\mathsf{SUM}}$ gates appearing in *internal layers* of Boolean circuits. In particular, we note that if MAJ can be simulated by $\widetilde{\mathsf{SUM}}_{1/3} \circ \mathsf{ACC}^0$-circuits of quasi-polynomial size (or THR can be simulated by $\mathsf{MAJ} \circ \widetilde{\mathsf{SUM}}_\delta \circ \mathsf{ACC}^0$-circuits of quasi-polynomial size), then $\mathsf{NQP} \not\subseteq \mathsf{THR} \circ \mathsf{THR}$. On the other hand, if this is not the case, strong average-case lower bounds against ACC^0 follow from Theorem 1.

Theorem 3 and Corollary 4: Techniques. The proofs of the first two items of Corollary 4 are immediate from the corresponding items of Theorem 3 via the #SAT algorithm for $\mathcal{C} = \mathsf{ACC}^0 \circ \mathsf{THR}$ given by [51]. On the other hand, Item 3 of Corollary 4 can be established in different ways. The first proof is just a standard application of the Discriminator Lemma [24] together with the lower bound from Item 2. A second proof follows from Item 1, via a

[12] For the interested reader, we notice that the coefficients of the top $\widetilde{\mathsf{SUM}}$ gate can be unbounded in this lower bound.

simulation of a $\mathsf{MAJ} \circ \widetilde{\mathsf{SUM}}_\delta \circ \mathcal{C}$-circuit of quasi-polynomial complexity by a $\mathsf{LTF}^{2^{\log^k n}} \circ \mathsf{ACC}^0 \circ \mathsf{THR}$-circuit of size size $2^{\log^k n}$, for some constant k. This can be done by first reducing the error δ of each $\widetilde{\mathsf{SUM}}_\delta \circ \mathcal{C}$-subcircuit (see [11]), then rewriting the corresponding $\mathsf{MAJ} \circ \widetilde{\mathsf{SUM}}_\varepsilon$ top layers as an LTF^s gate via an appropriate collapse. We omit the details.

The proofs of Items 1 and 2 of Theorem 3 are essentially independent. We discuss each of them next, starting with Item 1.

An extension of the algorithmic method [49] obtained by [37] shows that SAT algorithms for a circuit class $\mathcal{C}$ of sub-exponential size circuits (satisfying minor closure conditions) that run in time 2^{n-n^ε} imply that $\mathsf{NQP} \not\subseteq \mathcal{C}$. In a more recent work that builds on [50], [15] established (in particular) that #SAT algorithms of similar running time provide the stronger lower bound $\mathsf{NQP} \not\subseteq \widetilde{\mathsf{SUM}} \circ \mathcal{C}$. Our proof of Item 1 of Theorem 3 relies on the latter result and on a win-win argument inspired by [14]. In more detail, and oversimplifying a bit, we argue that if a special NC^1-hard problem L (contained in NQP) is not in $\mathsf{LTF}^{2^{\log^k n}} \circ \mathcal{C}$, then we are done. Otherwise, we explore LTF^s gates and the special form of the NC^1-hardness of L to show that NC^1 can be simulated by $\widetilde{\mathsf{SUM}} \circ \mathcal{C}$-circuits of quasi-polynomial complexity. Given this lemma and the corresponding simulation, we can reduce the derivation of the desired lower bound to previous work, i.e., we invoke the aforementioned connection between #SAT algorithms and lower bounds against $\widetilde{\mathsf{SUM}} \circ \mathcal{C}$. This provides a language in NQP that is not in $\widetilde{\mathsf{SUM}} \circ \mathcal{C}$ of complexity $2^{\log^\ell n}$, where $\ell = \ell(k)$ is large enough. Now by simulating $\mathsf{LTF}^{2^{\log^k n}} \circ \mathcal{C}$-circuits using quasi-polynomial size Boolean formulas, and using the collapse of NC^1 to quasi-polynomial size $\widetilde{\mathsf{SUM}} \circ \mathcal{C}$, it is possible to argue that L is also hard against $\mathsf{LTF}^{2^{\log^k n}} \circ \mathcal{C}$.

The proof of Item 2 of Theorem 3 shares some similarities with the argument above, but the technical details are different. From a high-level perspective, we also employ a win-win argument, though this time it is based on the *average-case* complexity of the language L mentioned above. Moreover, we cannot rely on previous connections between #SAT algorithms and lower bounds in a *black-box* way. Given that explaining the relevant details would be fairly technical, we refer the interested reader to the full version of this paper [11]. We mention that a conceptual contribution is that while our proof of Theorem 3 Part 2 follows the strategy of previous works, such as [10, 15, 14], on obtaining lower bounds from meta-algorithms, it does not use PCPs of proximity (PCPP), which was a key ingredient in the proofs of those works. For this, we rely in part on a PCP stated in [48], combined with other ideas.

Organization. In Section 2 we introduce the necessary technical preliminaries for proving our results. In Section 3 we prove our main equivalence result (Theorem 1). Due to space constraints, the remaining proofs are deferred to the full version of this paper [11].

2 Preliminaries

2.1 Notation

We use $\mathbb{N}$ to denote the set of all non-negative integers and $\mathbb{N}_{\geq 1}$ to denote $\mathbb{N} \setminus \{0\}$. For every $n \in \mathbb{N}_{\geq 1}$, we let $\mathcal{U}_n$ denote the uniform distribution over $\{0,1\}^n$. For convenience, in some settings a Boolean function $f\colon \{0,1\}^n \to \{0,1\}$ will be viewed as a function with output in $\{-1,1\}$, where -1 and 1 are interpreted as True and False, respectively.

For a predicate $P(x)$, we use $\mathbb{1}_{P(x)}$ to denote its corresponding Boolean value on x. That is, $\mathbb{1}_{P(x)} = 1$ if $P(x)$ is true, and 0 otherwise. For a real v, we define $\mathrm{sign}(v) := (-1) \cdot \mathbb{1}_{v<0} + 1 \cdot \mathbb{1}_{v \geq 0}$.

For two strings $\alpha, \beta \in \{0,1\}^*$, we write $\alpha \circ \beta$ to denote the concatenation of α and β.

A projection of a function $f(x_1, \ldots, x_n)$ is a function $g(y_1, \ldots, y_m)$ with a projection mapping $P\colon \{0,1\}^m \to \{0,1\}^n$ such that $g(y_1, \ldots, y_m) = f(P(y_1, \ldots, y_m))$. By "projection" we mean that each output bit of $P(y_1, \ldots, y_m)$ is either an input bit y_i, its negation, or a constant.

Let a be a positive integer. For an arbitrary $\ell \geq 1$ and a function $h\colon \{0,1\}^\ell \to \{0,1\}$, we say that $h \in \mathsf{JUNTA}_a$ if the output of h depends on at most a input coordinates.

For a circuit class $\mathcal{C}$ and $s \geq 1$, we use $\widetilde{\mathsf{SUM}} \circ \mathcal{C}[s]$ to denote the class of $\widetilde{\mathsf{SUM}} \circ \mathcal{C}$-circuits where the top SUM gate has complexity at most s and the bottom layer $\mathcal{C}$-circuits have size at most s.

For a function $f\colon \{0,1\}^n \to \{0,1\}$, we let $f_\pm\colon \{0,1\}^n \to \{-1,1\}$ be the $\{-1,1\}$-version of f where we map the output of f from 0 to 1 and 1 to -1. Also, for a circuit class $\mathcal{C}$ where the circuits in $\mathcal{C}$ output values in $\{0,1\}$, we denote by $\mathcal{C}_\pm$ the $\{-1,1\}$-version of $\mathcal{C}$ where the circuits in $\mathcal{C}_\pm$ output values in $\{-1,1\}$.

2.2 A $\oplus$L-complete problem with good properties

The existence of $\oplus$L-complete problems with good reducibility properties will be important for us. (Recall that $\oplus$L is the class of problems solvable by a nondeterministic logspace Turing machine that accepts the input if the number of accepting paths is odd.) We define the following two problems, called Connected Matrix Determinant (CMD) and Decomposed Connected Matrix Determinant (DCMD):

▶ **Definition 5.** *An instance of* CMD *is an* $n \times n$ *matrix over* $\mathbb{F}_2$ *where the main diagonal and above may contain either* 0 *or* 1*, the second diagonal (i.e. the one below the main diagonal) contains* 1*, and other entries are* 0*. In other words, the matrix is of the following form (where* $*$ *represents any element in* $\mathbb{F}_2$*):*

$$\begin{pmatrix} * & * & * & \cdots & * & * \\ 1 & * & * & \cdots & * & * \\ 0 & 1 & * & \cdots & * & * \\ 0 & 0 & 1 & \cdots & * & * \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & 1 & * \end{pmatrix}.$$

The instance is an $(n(n+1)/2)$*-bit string specifying elements on and above the main diagonal. We define* $x \in \mathsf{CMD}$ *if and only if the determinant (over* $\mathbb{F}_2$*) of the matrix corresponding to* x *is* 1*.*

An instance of DCMD *is a string of length* $n^3(n+1)/2$*. For an input* x*,* $\mathsf{DCMD}(x)$ *is computed as follows: we partition* x *into blocks of length* n^2*, let* y_i $(1 \leq i \leq n(n+1)/2)$ *be the parity of the* i*-th block, and define* $\mathsf{DCMD}(x) := \mathsf{CMD}(y_1 \circ y_2 \circ \cdots \circ y_{n(n+1)/2})$*.*

The precise definitions of CMD and DCMD are not important here, as we only need the following two important results about them.

▶ **Theorem 6** ([4, 20])**.** *There is a function* $P : \{0,1\}^{n(n+1)/2} \times \{0,1\}^{O(n^4)} \to \{0,1\}^{n^3(n+1)/2}$ *such that the following hold.*

- *For any input* $x \in \{0,1\}^{n(n+1)/2}$*, the random variable* $P(x, \mathcal{U}_{O(n^4)})$ *is uniformly distributed in* $\{0,1\}^{n^3(n+1)/2}$*.*
- *For any* $x \in \{0,1\}^{n(n+1)/2}$ *and* $r \in \{0,1\}^{O(n^4)}$*, let* $P(x,r) = y$*, then* $\mathsf{CMD}(x) = \mathsf{DCMD}(y) \oplus r_0$*, where* r_0 *is the first bit of* r*.*
- *For each fixed randomness* r*,* $P(x,r)$ *is a projection over* x*, computable in polynomial time given* r*.*

▶ **Theorem 7** ([31]). CMD *is* $\oplus$L*-complete under projections.*

Observe that if CMD is in a circuit class $\mathcal{C}$ closed under projections then all problems in (non-uniform) NC^1 are also in $\mathcal{C}$, given that the problem of evaluating an input Boolean formula is solvable with logarithmic space.

We refer the reader to the full version of [14] for a self-contained exposition of these problems and their relevant properties, including pointers to related work.

2.3 Pseudorandomness

We need the following Hardness vs. Randomness framework for constructing PRGs.

▶ **Lemma 8** (Hardness vs. Randomness [39], see also [14, Appendix E.3] for the proof). *There is a function* $G\colon \{0,1\}^* \times \{0,1\}^* \to \{0,1\}^*$ *such that the following holds. Let* n, ℓ, a *be integers such that* $a \le \ell$*, and* $t = O\big(\ell^2 \cdot n^{1/a}/a\big)$*. Let* $\mathcal{C}$ *be a circuit class closed under negation. For any function* $Y\colon \{0,1\}^\ell \to \{0,1\}$ *represented as a length-*2^ℓ *truth table, if* Y *cannot be* $(1/2 + \varepsilon/n)$*-approximated by* $\mathcal{C} \circ \mathsf{JUNTA}_a$*-circuits where the top circuit has size* S*, then for every circuit* $C \in \mathcal{C}$ *of size* S*,*

$$\left| \mathop{\mathbf{Pr}}_{z \sim \{0,1\}^t}[C(G(Y,z)) = 1] - \mathop{\mathbf{Pr}}_{x \sim \{0,1\}^n}[C(x) = 1] \right| \le \varepsilon.$$

Moreover, the function G *is computable in* $\mathrm{poly}(n, 2^t)$ *time.*

The following simple fact says PRGs imply worst-case hardness.

▶ **Proposition 9** (Worst-case hardness from PRGs). *Let* $\mathcal{F}$ *be a class of functions. If there is an i.o.* ε*-PRG* $G\colon \{0,1\}^r \to \{0,1\}^n$ *with seed length* $r(n)$ *against* $\mathcal{F}_n$*, where* $\varepsilon < 1 - 2^{r(n)-n}$*, then there is a language* $L \in E$ *such that* L *cannot be computed by* $\mathcal{F}$*.*

Proof. Please see the full version [11] for details. ◀

2.4 Hardness amplification

The following result allows us to amplify hardness against NC^1.

▶ **Lemma 10** (Hardness amplification against NC^1, see e.g. [43, 20]). *Suppose there is a language* $L \in \mathsf{E}$ *such that* $L \notin \mathsf{NC}^1$*. Then there is a language* $L' \in \mathsf{E}$ *such that for every constant* $k \ge 1$*,* L' *cannot be* $(1/2 + 1/n^k)$*-approximated by formulas of size* n^k*.*

The following notion of ℓ_1-approximation by SUM-circuits plays a crucial role in some recent results on average-case lower bounds via the algorithmic method (e.g. [13, 12, 27]).

▶ **Definition 11** (ℓ_1-approximation by SUM-circuits). *Let* $\delta \in (0,1)$ *and let* $\mathcal{C}$ *be a circuit class. We say that a function* $f\colon \{0,1\}^n \to \{0,1\}$ *is approximated by a* $[0,1]$-$\mathsf{SUM} \circ \mathcal{C}$*-circuit* C *within* ℓ_1 *distance* δ *if*

$$\mathop{\mathbf{E}}_{x \sim \mathcal{U}_n}[\,|f(x) - C(x)|\,] \le \delta.$$

For functions $f, g\colon \{0,1\}^n \to \mathbb{R}$, we let $\langle f, g \rangle := \mathbf{E}_{x \in \{0,1\}^n}[f(x) \cdot g(x)]$.

▶ **Proposition 12.** *Let* $\delta \in (0,1)$*,* $f\colon \{0,1\}^n \to \{0,1\}$*, and* $\mathcal{C}$ *be a circuit class.*

1. *If* f *can be approximated by* $[0,1]$-$\mathsf{SUM} \circ \mathcal{C}$*-circuits of complexity* s *within* ℓ_1 *distance* δ*, then there is a* $\mathsf{SUM} \circ \mathcal{C}_\pm$*-circuit* C *of complexity* $O(s)$ *such that* $\|C\|_\infty \le 1$ *and* $\langle f_\pm, C \rangle \ge 1 - 2\delta$*.*

2. *If there is a* $\mathsf{SUM} \circ \mathcal{C}_{\pm}$*-circuit* C *of complexity* s *such that* $\|C\|_{\infty} \leq 1$ *and* $\langle f_{\pm}, C\rangle \geq 1 - 2\delta$*, then* f *can be approximated by* $[0,1]$-$\mathsf{SUM} \circ \mathcal{C}$*-circuits of complexity* $O(s)$ *within* ℓ_1 *distance* δ*.*

Proof. Please see the full version [11] for details. ◀

Given a set X and a Boolean function $f\colon X \to \{-1,1\}$, for and integer $t \geq 1$ and $X^t = X \times \ldots \times X$ (t times) we let $f^{\oplus t}\colon X^t \to \{-1,1\}$ be the Boolean function defined as $f^{\oplus t}(x_1,\ldots,x_t) \coloneqq \prod_{i\in[t]} f(x_i)$. We will need the following XOR lemma from [13].

▶ **Theorem 13** ([35] and [13, Lemma 3.8], see also [12, Lemma 1.7])**.** *Let* $\mathcal{F}$ *be a class of Boolean functions that is closed under negation and restriction. For every* $\delta, \varepsilon \in (0,1)$ *and every function* $f\colon \{0,1\}^n \to \{-1,1\}$*, if*

$$\langle f, C\rangle \leq 1 - \delta$$

for every $\mathsf{SUM} \circ \mathcal{F}$*-circuit* C *where the top* SUM *has complexity* $10 \cdot n/\varepsilon^2$ *and* $\|C\|_{\infty} \leq 1$*, then*

$$\langle f^{\oplus t}, D\rangle \leq (1-\delta)^t + \varepsilon/\delta$$

for any Boolean function $D\colon \{0,1\}^{tn} \to \{-1,1\}$ *in* $\mathcal{F}$*.*

3 Equivalences for worst-case and strong average-case lower bounds

In this section, we prove our equivalence results for worst-case hardness, strong average-case hardness and pseudorandomness.

▶ **Reminder of Theorem 1.** *Let* $\mathcal{C}$ *be a circuit class that satisfies the following:*

- $\mathcal{C}$ *is closed under negation and projection.*
- $\mathcal{C}$ *is closed under a bottom layer of juntas over* $O(1)$ *input bits. That is*

$$\bigcup_{k\geq 1} \mathcal{C}[n^k] \circ \mathsf{JUNTA}_k \subseteq \bigcup_{k\geq 1} \mathcal{C}[n^k].$$

- $\bigcup_{k\geq 1} \mathcal{C}[n^k] \subseteq \mathsf{NC}^1$.

Then the following statements are equivalent:

1. *There is* $L \in \mathsf{E}$ *such that for every* $k \geq 1$*,* $L \notin \widetilde{\mathsf{SUM}}_{1/3} \circ \mathcal{C}[n^k]$*.*
2. *There is* $L \in \mathsf{E}$ *and* $\delta \geq 1/\mathsf{poly}(n)$ *such that for every* $k \geq 1$*,* $L \notin \widetilde{\mathsf{SUM}}_{\delta} \circ \mathcal{C}[n^k]$*.*
3. *There is* $L \in \mathsf{E}$ *such that for every* $k \geq 1$*,* $L \notin \mathsf{MAJ} \circ \mathcal{C}[n^k]$*.*
4. *There is* $L \in \mathsf{E}$ *such that, for every* $k \geq 1$*,* L *cannot be computed by a probabilistic* $\mathcal{C}[n^k]$*-circuit with error* $1/2 - 1/n^k$*.*
5. *There is* $L \in \mathsf{E}$ *and a distribution* $\mathcal{D}$ *such that for every* $k \geq 1$*,* L *cannot be* $(1/2 + n^{-k})$*-approximated by* $\mathcal{C}[n^k]$ *under* $\mathcal{D}$*.*
6. *There is* $L \in \mathsf{E}$ *such that for every* $k \geq 1$*,* L *cannot be* $(1/2 + n^{-k})$*-approximated by* $\mathcal{C}[n^k]$ *under the uniform distribution.*
7. *There is* $L \in \mathsf{E}$ *such that for every* $k \geq 1$*,* L *cannot be approximated by* $[0,1]$-$\mathsf{SUM} \circ \mathcal{C}[n^k]$ *within* ℓ_1 *distance* $1/3$*.*
8. *There is* $L \in \mathsf{E}$ *and* $\delta \geq 1/\operatorname{poly}(n)$ *such that for every* $k \geq 1$*,* L *cannot be approximated by* $[0,1]$-$\mathsf{SUM} \circ \mathcal{C}[n^k]$ *within* ℓ_1 *distance* δ*.*

9. *There is an i.o. ε-PRG G against $\mathcal{C}$ with seed length $n-1$ and error $\varepsilon(n) \leq n^{-\omega(1)}$. In other words, for each choice of k, there is an infinite set $S_k \subseteq \mathbb{N}$ such that G fools circuits from $\mathcal{C}[n^k]$ on inputs of length $n \in S_k$ with error $\varepsilon(n) \leq n^{-k}$.*
10. *For every $\gamma > 0$, there is an i.o. ε-PRG against $\mathcal{C}$ with seed length n^γ and $\varepsilon(n) \leq n^{-\omega(1)}$.*

Proof. We will first show Item 2 ⇒ Item 6 ⇒ Item 10 ⇒ Item 1 ⇒ Item 2, establishing the equivalence of Items 1, 2, 6, and 10. We then show Item 6 ⇒ Item 5 ⇒ Item 4 ⇒ Item 1, which adds Items 4 and 5 to the list of equivalent items. Next, we show Item 6 ⇒ Item 3 ⇒ Item 4, which adds Item 3, and Item 10 ⇒ Item 9 ⇒ Item 2, which adds Item 9. Finally, we show Item 6 ⇒ Item 7 ⇒ Item 8 ⇒ Item 6, adding Items 7 and 8 to the list and completing the proof.

Item 2 ⇒ Item 6. We consider two cases. If DCMD cannot be $\left(1/2 + 1/n^k\right)$-approximated by $\mathcal{C}[n^k]$ for every $k \geq 1$ under the uniform distribution, then we are done.

Now consider the case that there is some $k \geq 1$ such that DCMD can be $\left(1/2 + 1/n^k\right)$-approximated by $\mathcal{C}[n^k]$. By the random self-reducibility of $\mathsf{DCMD}/\mathsf{CMD}$ (see Theorem 6 and also [14, Section 3]), for any $\delta = 1/\operatorname{poly}(n)$, CMD can be computed by a $\widetilde{\mathsf{SUM}}_\delta \circ \mathcal{C}$-circuit where the top SUM-gate has polynomial complexity and the bottom-layer $\mathcal{C}$-circuits have polynomial size. By Theorem 7, for every polynomial-size parity branching program B, there is a projection $p\colon \{0,1\}^n \to \{0,1\}^{n^{O(1)}}$ such that for every $x \in \{0,1\}^n$, $B(x) = \mathsf{CMD}(p(x))$. Since $\mathcal{C}$ is closed under projection, this means that every polynomial-size parity branching program has a $\widetilde{\mathsf{SUM}}_\delta \circ \mathcal{C}$-circuit of polynomial complexity and size, which then implies that every function in NC^1 also has such a $\widetilde{\mathsf{SUM}}_\delta \circ \mathcal{C}$-circuit. On the other hand, by Item 2, there is a function $L \in \mathsf{E}$ that has no $\widetilde{\mathsf{SUM}}_\delta \circ \mathcal{C}$-circuit of polynomial complexity and size, so L is not in NC^1. Using hardness amplification against NC^1 (Lemma 10), it follows that there is a function in E that is strongly average-case hard against NC^1, which by assumption contains polynomial-size $\mathcal{C}$-circuits.

Item 6 ⇒ Item 10. We construct the PRG using the hardness vs. randomness framework. Consider Lemma 8 with the following setting of parameters: $a := 2/\gamma$ and $\ell := n^{\gamma/4}$. Let $G_{L_\ell}\colon \{0,1\}^t \to \{0,1\}^n$ be the PRG defined as $G_{L_\ell}(z) := G(L_\ell, z)$, where $L \in \mathsf{E}$ is the language from Item 6. Note that the seed length t is at most $O\left(\ell^2 \cdot n^{1/a}/a\right) \leq n^\gamma$ and G_{L_ℓ} can be computed in time $\operatorname{poly}(n, 2^t) = 2^{O(n^\gamma)}$. Let $k \geq 1$ be any constant and consider any ℓ-variate $\mathcal{C} \circ \mathsf{JUNTA}_a$-circuit C where the top circuit has size $n^k = \ell^{4k/\gamma}$. Since $\mathcal{C}$ is closed under a bottom layer of juntas, we have that $C \in \mathcal{C}\left[\ell^{k'}\right]$ for some large enough $k' > 4k/\gamma$. Also, let $\varepsilon = 1/n^k$, which implies $\varepsilon/n = 1/n^{k+1} = 1/\ell^{4(k+1)/\gamma} \geq 1/\ell^{k'}$. From Item 6, we have that L_ℓ cannot be $\left(1/2 + 1/\ell^{k'}\right)$-approximated by any circuit from $\mathcal{C}[\ell^{k'}]$, for infinitely many values of ℓ. Then by Lemma 8, we conclude that G_{L_ℓ} $\left(1/n^k\right)$-fools any circuit from $\mathcal{C}[n^k]$, for infinitely many values of n.

Item 10 ⇒ Item 1. Let $G\colon \{0,1\}^r \to \{0,1\}^n$ be an i.o. PRG as in Item 10, where $r \leq n-2$. That is, for each choice of k', G fools circuits from $\mathcal{C}[n^{k'}]$ on input length n with error $\varepsilon(n) \leq n^{-k'}$, for infinitely many values of n.

Let $k \geq 1$ and let $C \in \widetilde{\mathsf{SUM}}_{1/3} \circ \mathcal{C}[n^k]$. By Proposition 9, it suffices to show that G is an i.o. $\left(< \frac{3}{4}\right)$-PRG against C. Let $\tilde{C}$ be the corresponding linear sum for C. That is,

$$\tilde{C}(x) := \sum_i \alpha_i \cdot C_i(x),$$

where $C_i \in \mathcal{C}[n^k] \subseteq \mathcal{C}[n^{k':=k+1}]$ and $\sum_i |\alpha_i| \leq n^k$. Since $\tilde{C}$ (1/3)-approximates C in a pointwise manner, we have

$$\left|\mathbf{E}[C(\mathcal{U})] - \mathbf{E}[\tilde{C}(\mathcal{U})]\right| \leq 1/3 \text{ and } \left|\mathbf{E}[C(G)] - \mathbf{E}[\tilde{C}(G)]\right| \leq 1/3.$$

Therefore, if we can show that

$$\left|\mathbf{E}[\tilde{C}(\mathcal{U})] - \mathbf{E}[\tilde{C}(G)]\right| \leq \delta,$$

for some $\delta < 1/12$ (infinitely often), then G δ'-fools C (infinitely often), where $\delta' = 2/3 + \delta < 3/4$. We have

$$\begin{aligned}
\left|\mathbf{E}[\tilde{C}(\mathcal{U})] - \mathbf{E}[\tilde{C}(G)]\right| &= \left|\mathbf{E}\left[\sum_i \alpha_i \cdot C_i(\mathcal{U})\right] - \mathbf{E}\left[\sum_i \alpha_i \cdot C_i(G)\right]\right| \\
&= \left|\sum_i \alpha_i \cdot \mathbf{E}[C_i(\mathcal{U})] - \mathbf{E}[C_i(G)]\right| \\
&\leq \max_i |\mathbf{E}[C_i(\mathcal{U})] - \mathbf{E}[C_i(G)]| \cdot \sum_i |\alpha_i| \\
&\leq n^{-k'} \cdot n^k \leq 1/n,
\end{aligned}$$

as desired.

Item 1 ⇒ Item 2. This implication is straightforward.

Item 6 ⇒ Item 5 ⇒ Item 4. The first implication is obvious. The contrapositive of the second implication follows from an averaging argument.

Item 4 ⇒ Item 1. It suffices to show that for every $k \geq 1$, every function in $\widetilde{\mathsf{SUM}}_{1/3} \circ \mathcal{C}[n^k]$ has a probabilistic $\mathcal{C}[n^k]$-circuit with error $1/2 - 1/n^{O(k)}$.

For the simplicity of presentation, we will consider Boolean functions that take inputs from $\{0,1\}^n$ and output values in $\{-1,1\}$. Let $f_{\pm}\colon \{0,1\}^n \to \{-1,1\} \in \widetilde{\mathsf{SUM}}_{1/3} \circ \mathcal{C}_{\pm}[n^k]$. Then there is a linear sum of $\mathcal{C}_{\pm}[n^k]$-circuits

$$f_1(x) := \sum_i \alpha_i \cdot \left(\frac{1 - C_i(x)}{2}\right),$$

where $C_i\colon \{0,1\}^n \to \{-1,1\} \in \mathcal{C}_{\pm}[n^k]$ and $\sum_i |\alpha_i| \leq n^k$, such that

- if $f_{\pm}(x) = 1$, then $f_1(x) \leq 1/3$, and
- if $f_{\pm}(x) = -1$, then $f_1(x) \geq 2/3$.

Next, let

$$f_2(x) := 1/2 - f_1(x).$$

It is easy to see that

- if $f_{\pm}(x) = 1$, then $f_2(x) \geq 1/6$, and
- if $f_{\pm}(x) = -1$, then $f_2(x) \leq -1/6$.

Now note that since $\mathcal{C}_{\pm}$ is closed under negation, f_2 can be written as

$$f_2(x) := \sum_j \beta_j \cdot D_j(x),$$

where for each j, $D_j\colon \{0,1\}^n \to \{-1,1\} \in \mathcal{C}_{\pm}[n^k]$, $\beta_j \geq 0$, and $T := \sum_j \beta_j \leq n^{O(k)}$. Finally, let

$$f_3(x) := \frac{f_2(x)}{T}.$$

Let $\mathcal{D}$ be the probabilistic $\mathcal{C}_{\pm}[n^k]$-circuit where D_j is sampled with probability β_j/T. Then for every x we have $\mathbf{E}_{\mathcal{D}}[\mathcal{D}(x)] = f_3(x)$. Moreover, if $f_{\pm}(x) = 1$, then

$$\begin{aligned}\frac{1}{6T} &\leq \mathop{\mathbf{E}}_{\mathcal{D}}[\mathcal{D}(x)] \\ &= \mathop{\mathbf{Pr}}_{\mathcal{D}}[\mathcal{D}(x) = 1] - \mathop{\mathbf{Pr}}_{\mathcal{D}}[\mathcal{D}(x) = -1] \\ &= \mathop{\mathbf{Pr}}_{\mathcal{D}}[\mathcal{D}(x) = 1] - (1 - \mathop{\mathbf{Pr}}_{\mathcal{D}}[\mathcal{D}(x) = 1]) \\ &= 2 \cdot \mathop{\mathbf{Pr}}_{\mathcal{D}}[\mathcal{D}(x) = 1] - 1,\end{aligned}$$

which implies

$$\mathop{\mathbf{Pr}}_{\mathcal{D}}[\mathcal{D}(x) = 1] \geq \frac{1}{2} + \frac{1}{12T}.$$

Similarly, we can show that if $f_{\pm}(x) = -1$, then

$$\mathop{\mathbf{Pr}}_{\mathcal{D}}[\mathcal{D}(x) = -1] \geq \frac{1}{2} + \frac{1}{12T}.$$

Therefore, $\mathcal{D}$ is a probabilistic $\mathcal{C}_{\pm}[n^k]$-circuit for $f_{\pm}$ with error $1/2 - 1/n^{O(k)}$.

Item 6 ⇒ Item 3. This follows from the standard Discriminator Lemma [24].

Item 3 ⇒ Item 4. We will show that for every $k \geq 1$, every function that has a probabilistic $\mathcal{C}[n^k]$-circuit with error $1/2 - 1/n^k$ is contained in $\mathsf{MAJ}_{n^{O(k)}} \circ \mathcal{C}[n^{O(k)}]$.

Let $f\colon \{0,1\}^n \to \{0,1\}$ and $\mathcal{D}$ be the probabilistic $\mathcal{C}[n^k]$-circuit for f with error $1/2 - 1/n^k$. That is, for every x,

$$\mathop{\mathbf{Pr}}_{\mathcal{D}}[\mathcal{D}(x) = f(x)] \geq 1/2 + 1/n^k.$$

By the Chernoff bound, if we sample $t := O(n^{2k} \cdot n)$ circuits $C_1, \ldots, C_t$ from $\mathcal{D}$, then

$$\mathop{\mathbf{Pr}}_{C_1,\ldots,C_t \sim \mathcal{D}}\left[\mathop{\mathbf{Pr}}_{i \in [t]}[C_i(x) = f(x)] \geq 1/2 + 1/(2n^k)\right] \geq 1 - 2^{-2n}.$$

By a union bound over $x \in \{0,1\}^n$, there exist t circuits $C_1, \ldots, C_t$ such that for every x,

$$\mathop{\mathbf{Pr}}_{i \in [t]}[C_i(x) = f(x)] \geq 1/2 + 1/(2n^k).$$

Therefore, by taking the majority of these t circuits, we obtain a $\mathsf{MAJ}_{n^{O(k)}} \circ \mathcal{C}[n^{O(k)}]$-circuit that computes f.

Item 10 ⇒ Item 9 ⇒ Item 2. This first implication is obvious. The proof of the second implication is essentially the same as that of "Item 10 ⇒ Item 1". From Item 9, we get an i.o. PRG with seed length $n - 1$ that $\left(< \frac{1}{2}\right)$-fools $\widetilde{\mathsf{SUM}}_{\delta} \circ \mathcal{C}$-circuits for some $\delta = 1/\operatorname{poly}(n)$, which by Proposition 9 implies Item 2. We omit the details here.

Item 6 ⇒ Item 7. Let $L\colon \{0,1\}^* \to \{0,1\}$ be the language from Item 6. For the sake of contradiction, suppose there is a $k \geq 1$ such that L can be approximated by $[0,1]$-$\mathsf{SUM} \circ \mathcal{C}[n^k]$-circuits within ℓ_1 distance $1/3$. Then by Item 1 of Proposition 12, we have that for every n, there is a $\mathsf{SUM} \circ \mathcal{C}_{\pm}[O\left(n^k\right)]$-circuit C such that $\|C\|_\infty \leq 1$ and

$$\langle (L_{\pm})_n, C\rangle \geq 1/3.$$

Suppose

$$C(x) := \sum_i |\alpha_i| \cdot C_i(x),$$

where $C_i \in \mathcal{C}_{\pm}[O\left(n^k\right)]$ and $\sum_i |\alpha_i| \leq O\left(n^k\right)$. Then

$$\begin{aligned}
1/3 &\leq \left\langle (L_{\pm})_n, \sum_i \alpha_i \cdot C_i \right\rangle \\
&= \sum_i \alpha_i \cdot \langle (L_{\pm})_n, C_i\rangle \\
&\leq \sum_i |\alpha_i| \cdot \langle (L_{\pm})_n, C_i\rangle \\
&\leq \max_i \langle (L_{\pm})_n, C_i\rangle \cdot \sum_i |\alpha_i| \\
&\leq \max_i \langle (L_{\pm})_n, C_i\rangle \cdot O\left(n^k\right),
\end{aligned}$$

which implies that there exists some i such that

$$\langle (L_{\pm})_n, C_i\rangle \geq \frac{1}{O(n^k)}.$$

This contradicts Item 6.

Item 7 ⇒ Item 8. This implication is obvious.

Item 8 ⇒ Item 6. By Item 2 of Proposition 12, we have that Item 8 implies that there is a language $L\colon \{0,1\}^* \to \{-1,1\}$ in E and $\delta = 1/\ell^b$, where $b \geq 1$ is a constant, such that for every $k' \geq 1$, on infinitely many input lengths there is no $\mathsf{SUM} \circ \mathcal{C}_{\pm}[\ell^{k'}]$-circuit C with $\|C\|_\infty \leq 1$ such that

$$\langle L_\ell, C\rangle \leq 1 - 2\delta. \tag{1}$$

Now consider the following language $L'\colon \{0,1\}^* \to \{-1,1\}$: on input x of length n, let ℓ be the largest integer such that $\ell \cdot \ell^b \log^2(\ell) \leq n$ and view the input as $x = (x_1, \ldots, x_t, y)$, where $t := \ell^b \log^2(\ell)$ and $x_i \in \{0,1\}^\ell$ for $i \in [t]$. Then let

$$L'(x) := \prod_{i \in [t]} L(x_i).$$

Note that for large enough n we have

$$n < 2\ell \cdot t < \ell^{b+2}.$$

We claim that L' is strongly average-case hard against $\mathcal{C}_{\pm}$-circuits. For the sake of contradiction, suppose there is $k \geq 1$ and an n-variate circuit $C' \in \mathcal{C}_{\pm}[n^k]$ such that, for all large enough n,

$$\langle L'_n, C' \rangle > \frac{1}{n^k}.$$

By an averaging argument, where we fix the y-part of the input to some value, there exists some $(\ell \cdot t)$-variate $\mathcal{C}_{\pm}$-circuit C' of size $n^k \leq \ell^{k(b+2)}$ such that

$$\left\langle L_{\ell}^{\oplus t}, C'' \right\rangle > \frac{1}{n^k}.$$

Note that for $\delta = 1/\ell^b$ and our choice of $t = \ell^b \log^2(\ell)$, we have

$$\frac{1}{n^k} > (1-2\delta)^t + \frac{1}{2\delta \cdot \ell^{k(b+2)} \cdot \ell^b}.$$

By Theorem 13, there is a $\mathsf{SUM} \circ \mathcal{C}_{\pm}$ C where $\|C\|_{\infty} \leq 1$, the top SUM has complexity $10 \cdot \ell \cdot \left(\ell^{k(b+2)} \cdot \ell^b\right)^2 \leq \ell^{O(kb)}$ and the bottom layer $\mathcal{C}_{\pm}$-circuits have size $\ell^{k(b+2)}$ such that

$$\langle L_{\ell}, C \rangle > 1 - 2\delta,$$

for all large enough ℓ. This contradicts Equation (1). ◀

References

1. Eric Allender, Vikraman Arvind, and Fengming Wang. Uniform derandomization from pathetic lower bounds. In *RANDOM-APPROX*, pages 380–393. Springer, 2010.
2. Josh Alman, Timothy M. Chan, and R. Ryan Williams. Polynomial representations of threshold functions and algorithmic applications. In *FOCS*, pages 467–476, 2016.
3. Benny Applebaum. Garbled circuits as randomized encodings of functions: a primer. In *Tutorials on the Foundations of Cryptography*, pages 1–44. Springer International Publishing, 2017.
4. Benny Applebaum, Yuval Ishai, and Eyal Kushilevitz. Cryptography in NC^0. *SIAM J. Comput.*, 36(4):845–888, 2006.
5. Sergei Artemenko and Ronen Shaltiel. Lower bounds on the query complexity of non-uniform and adaptive reductions showing hardness amplification. *Comput. Complex.*, 23(1):43–83, 2014.
6. Richard Beigel and Jun Tarui. On ACC. *Comput. Complex.*, 4:350–366, 1994.
7. Abhishek Bhrushundi, Kaave Hosseini, Shachar Lovett, and Sankeerth Rao. Torus polynomials: An algebraic approach to ACC lower bounds. In *ITCS*, pages 13:1–13:16, 2019.
8. Ashok K. Chandra, Merrick L. Furst, and Richard J. Lipton. Multi-party protocols. In *STOC*, pages 94–99, 1983.
9. Arkadev Chattopadhyay and Nikhil S. Mande. A short list of equalities induces large sign rank. In *FOCS*, pages 47–58, 2018.
10. Lijie Chen. Non-deterministic quasi-polynomial time is average-case hard for ACC circuits. In *FOCS*, pages 1281–1304, 2019.
11. Lijie Chen, Zhenjian Lu, Xin Lyu, and Igor Carboni Oliveira. Majority vs. approximate linear sum and average-case complexity below NC1. *Electron. Colloquium Comput. Complex.*, 28:40, 2021.
12. Lijie Chen and Xin Lyu. Inverse-exponential correlation bounds and extremely rigid matrices from a new derandomized XOR lemma. In *STOC*, 2021.
13. Lijie Chen, Xin Lyu, and Ryan Williams. Almost-everywhere circuit lower bounds from non-trivial derandomization. In *FOCS*, pages 1–12, 2020.

14 Lijie Chen and Hanlin Ren. Strong average-case lower bounds from non-trivial derandomization. In *STOC*, pages 1327–1334, 2020.

15 Lijie Chen and Ryan Williams. Stronger connections between circuit analysis and circuit lower bounds, via PCPs of proximity. In *CCC*, pages 19:1–19:43, 2019.

16 Bill Fefferman, Ronen Shaltiel, Christopher Umans, and Emanuele Viola. On beating the hybrid argument. *Theory Comput.*, 9:809–843, 2013.

17 Jürgen Forster. A linear lower bound on the unbounded error probabilistic communication complexity. In *CCC*, pages 100–106, 2001.

18 Mikael Goldmann, Johan Håstad, and Alexander A. Razborov. Majority gates vs. general weighted threshold gates. *Comput. Complex.*, 2:277–300, 1992.

19 Oded Goldreich, Noam Nisan, and Avi Wigderson. On Yao's XOR-lemma. In *Studies in Complexity and Cryptography*, pages 273–301. Springer, 2011.

20 Shafi Goldwasser, Dan Gutfreund, Alexander Healy, Tali Kaufman, and Guy N. Rothblum. Verifying and decoding in constant depth. In *STOC*, pages 440–449, 2007.

21 Frederic Green, Johannes Köbler, and Jacobo Torán. The power of the middle bit. In *CCC*, pages 111–117, 1992.

22 Aryeh Grinberg, Ronen Shaltiel, and Emanuele Viola. Indistinguishability by adaptive procedures with advice, and lower bounds on hardness amplification proofs. In *FOCS*, pages 956–966, 2018.

23 Dan Gutfreund and Guy N. Rothblum. The complexity of local list decoding. In *RANDOM-APPROX*, pages 455–468, 2008.

24 András Hajnal, Wolfgang Maass, Pavel Pudlák, Mario Szegedy, and György Turán. Threshold circuits of bounded depth. *J. Comput. Syst. Sci.*, 46(2):129–154, 1993.

25 Johan Håstad. On the size of weights for threshold gates. *SIAM J. Discret. Math.*, 7(3):484–492, 1994.

26 Johan Håstad and Mikael Goldmann. On the power of small-depth threshold circuits. *Comput. Complex.*, 1:113–129, 1991.

27 Xuangui Huang and Emanuele Viola. Average-case rigidity lower bounds. In *CSR*, 2021.

28 Russell Impagliazzo. Hard-core distributions for somewhat hard problems. In *FOCS*, pages 538–545, 1995.

29 Russell Impagliazzo and Sam McGuire. Comparing computational entropies below majority (or: When is the dense model theorem false?). In *ITCS*, pages 2:1–2:20, 2021.

30 Russell Impagliazzo and Avi Wigderson. $P = BPP$ if E requires exponential circuits: Derandomizing the XOR lemma. In *STOC*, pages 220–229, 1997.

31 Yuval Ishai and Eyal Kushilevitz. Perfect constant-round secure computation via perfect randomizing polynomials. In *ICALP*, pages 244–256, 2002.

32 Daniel M. Kane and Ryan Williams. Super-linear gate and super-quadratic wire lower bounds for depth-two and depth-three threshold circuits. In *STOC*, pages 633–643, 2016.

33 Adam R. Klivans. On the derandomization of constant depth circuits. In *RANDOM-APPROX*, pages 249–260, 2001.

34 Vaibhav Krishan. Upper bound for torus polynomials. In *CSR*, 2021.

35 Leonid A. Levin. One-way functions and pseudorandom generators. *Comb.*, 7(4):357–363, 1987.

36 Chi-Jen Lu, Shi-Chun Tsai, and Hsin-Lung Wu. Complexity of hard-core set proofs. *Comput. Complex.*, 20(1):145–171, 2011.

37 Cody D. Murray and R. Ryan Williams. Circuit lower bounds for nondeterministic quasi-polytime from a new easy witness lemma. *SIAM J. Comput.*, 49(5), 2020.

38 Noam Nisan and Mario Szegedy. On the degree of boolean functions as real polynomials. *Comput. Complex.*, 4:301–313, 1994.

39 Noam Nisan and Avi Wigderson. Hardness vs randomness. *J. Comput. Syst. Sci.*, 49(2):149–167, 1994.

40 Alexander A. Razborov. Lower bounds on the size of constant-depth networks over a complete basis with logical addition. *Mathematicheskie Zametki*, 41(4):598–607, 1987.

41 Alexander A. Razborov and Avi Wigderson. $n^{\Omega(\log n)}$ lower bounds on the size of depth-3 threshold circuits with AND gates at the bottom. *Inf. Process. Lett.*, 45(6):303–307, 1993.

42 Ronen Shaltiel and Emanuele Viola. Hardness amplification proofs require majority. *SIAM J. Comput.*, 39(7):3122–3154, 2010.

43 Madhu Sudan, Luca Trevisan, and Salil P. Vadhan. Pseudorandom generators without the XOR lemma. *J. Comput. Syst. Sci.*, 62(2):236–266, 2001.

44 Suguru Tamaki. A satisfiability algorithm for depth two circuits with a sub-quadratic number of symmetric and threshold gates. *Electron. Colloquium Comput. Complex.*, 23:100, 2016.

45 Emanuele Viola. The complexity of constructing pseudorandom generators from hard functions. *Comput. Complex.*, 13(3-4):147–188, 2005.

46 Emanuele Viola. Guest column: correlation bounds for polynomials over $\{0,1\}$. *SIGACT News*, 40(1):27–44, 2009.

47 Emanuele Viola. Constant-error pseudorandomness proofs from hardness require majority. *ACM Trans. Comput. Theory*, 11(4):19:1–19:11, 2019.

48 Emanuele Viola. New lower bounds for probabilistic degree and AC^0 with parity gates. *Electron. Colloquium Comput. Complex.*, 27:15, 2020.

49 Ryan Williams. Improving exhaustive search implies superpolynomial lower bounds. *SIAM J. Comput.*, 42(3):1218–1244, 2013.

50 Ryan Williams. Limits on representing boolean functions by linear combinations of simple functions: Thresholds, ReLUs, and low-degree polynomials. In *CCC*, pages 6:1–6:24, 2018.

51 Ryan Williams. New algorithms and lower bounds for circuits with linear threshold gates. *Theory of Computing*, 14(1):1–25, 2018.

Near-Optimal Two-Pass Streaming Algorithm for Sampling Random Walks over Directed Graphs

Lijie Chen
MIT, Cambridge, MA, USA

Gillat Kol
Princeton University, NJ, USA

Dmitry Paramonov
Princeton University, NJ, USA

Raghuvansh R. Saxena
Princeton University, NJ, USA

Zhao Song
Institute for Advanced Study, Princeton, NJ, US

Huacheng Yu
Princeton University, NJ, USA

Abstract

For a directed graph G with n vertices and a start vertex u_{start}, we wish to (approximately) sample an L-step random walk over G starting from u_{start} with minimum space using an algorithm that only makes few passes over the edges of the graph. This problem found many applications, for instance, in approximating the PageRank of a webpage. If only a single pass is allowed, the space complexity of this problem was shown to be $\widetilde{\Theta}(n \cdot L)$. Prior to our work, a better space complexity was only known with $\widetilde{O}(\sqrt{L})$ passes.

We essentially settle the space complexity of this random walk simulation problem for *two-pass* streaming algorithms, showing that it is $\widetilde{\Theta}(n \cdot \sqrt{L})$, by giving almost matching upper and lower bounds. Our lower bound argument extends to every constant number of passes p, and shows that any p-pass algorithm for this problem uses $\widetilde{\Omega}(n \cdot L^{1/p})$ space. In addition, we show a similar $\widetilde{\Theta}(n \cdot \sqrt{L})$ bound on the space complexity of any algorithm (with *any* number of passes) for the related problem of sampling an L-step random walk from *every* vertex in the graph.

2012 ACM Subject Classification Theory of computation → Streaming, sublinear and near linear time algorithms

Keywords and phrases streaming algorithms, random walk sampling

Digital Object Identifier 10.4230/LIPIcs.ICALP.2021.52

Category Track A: Algorithms, Complexity and Games

Related Version *Full Version*: `https://arxiv.org/abs/2102.11251` [11]

Funding *Lijie Chen*: Lijie Chen is supported by an IBM Fellowship.
Zhao Song: Zhao Song is supported in part by Schmidt Foundation, Simons Foundation, NSF, DARPA/SRC, Google and Amazon AWS.

Acknowledgements We would like to thank Rajesh Jayaram for discussions on ℓ_1 heavy hitters.

1 Introduction

1.1 Background and Motivation

Graph streaming algorithms. *Graph streaming algorithms* have been the focus of extensive study over the last two decades, mainly due to the important practical motivation in analyzing potentially *huge structured data* representing the relationships between a set of entities (*e.g.*, the link graph between webpages and the friendship graph in a social network). In the graph streaming setting, an algorithm gets access to a sequence of graph edges given in an arbitrary order and it can read them one-by-one in the order in which they appear in the sequence. The goal here is to design algorithms solving important graph problems that only make one or few passes through the edge sequence, while using as little memory as possible.

Much of the streaming literature was devoted to the study of *one-pass* algorithms and an $\Omega(n^2)$ space lower bound for such algorithms was shown for many fundamental graph problems. A partial list includes: maximum matching and minimum vertex cover [15, 19], s-t reachability and topological sorting [7, 15, 22], shortest path and diameter [15, 16], maximum and (global or s-t) minimum cut [39], maximal independent set [3, 13], and dominating set [4, 14].

Recently, the *multi-pass* streaming setting received quite a bit of attention. For some graph problems, allowing a few passes instead of a single pass can reduce the memory consumption of a streaming algorithm dramatically. In fact, even a single additional pass over the input can already greatly enhance the capability of the algorithms. For instance, minimum cut and s-t minimum cut in undirected graphs can be solved in two passes with only $\widetilde{O}(n)$ and $O(n^{5/3})$ space, respectively [33] (as mentioned above, any one-pass algorithm for these problems must use $\Omega(n^2)$ space). Additional multi-pass algorithms include an $O(1)$-pass algorithm for approximate matching [17, 19, 28, 29], an $O(\log \log n)$-pass algorithm for maximal independent set [3, 13, 18], and $O(\log n)$-pass algorithms for approximate dominating set [4, 8, 21] and weighted minimum cut [30].

Simulating random walks on graphs. Simulating random walks on graphs is a well-studied algorithmic problem with may applications in different areas of computer science, such as connectivity testing [32], clustering [1, 2, 10, 36], sampling [26], generating random spanning tree [35], and approximate counting [25]. Since most applications of random-walk simulation are concerned with huge networks that come from practice, it is of practical interest to design low-space graph streaming algorithms with few passes for this problem.

In an influential paper by Das Sarma, Gollapudi and Panigrahy [34], an $\widetilde{O}(\sqrt{L})$-pass and $\widetilde{O}(n)$ space algorithm for simulating L-step random walks on directed graphs was established. (Streaming algorithms with almost linear space complexity, like this one, are often referred to as *semi-streaming* algorithms). Using this algorithm together with some additional ideas, [34] obtained space-efficient algorithms for estimating *PageRank* on graph streams. Recall that the PageRank of a webpage corresponds to the probability that a person that randomly clicks on web links arrives at this particular page[1]. However, scanning the sequence of edges $\widetilde{O}(\sqrt{L})$ times may be time-inefficient in many realistic settings.

In the one-pass streaming setting, a folklore algorithm with $\widetilde{O}(n \cdot L)$ space complexity for simulating L-step random walks is known [34] (see Subsection 2.1 for a description of this algorithm), and it is proved to be optimal [27]. We mention that the work of [27]

[1] Given a web-graph $G = (V, E)$ representing the webpages and links between them, the PageRank of the vertices satisfy $\mathsf{PageRank}(u) = \sum_{(v,u)\in E} \mathsf{PageRank}(v)/d(v)$, simultaneously for all u, where $d(\cdot)$ denotes the out-degree, [6].

also considers random walks on *undirected graphs*, and shows that $\widetilde{\Theta}(n \cdot \sqrt{L})$ space is both necessary and sufficient for simulating L-step random walks on undirected graphs with n vertices in one pass.

Both of these known algorithms for general directed graphs have their advantages and disadvantage (either requiring many passes or more space). A natural question is whether one can interpolate between these two results and obtain an algorithm with pass complexity much smaller than $\sqrt{L}$, yet with a space complexity much smaller than $n \cdot L$. Prior to our work, it was not even known if an $o(\sqrt{L})$-pass streaming algorithm with $n \cdot L^{0.99}$ space is possible.

1.2 Our Results

We answer the above question in the affirmative by giving a two-pass streaming algorithm with $\widetilde{O}(n \cdot \sqrt{L})$ space for sampling a random walk of length L on a directed graph with n vertices. We complement this result by an almost matching $\widetilde{\Omega}(n \cdot \sqrt{L})$ lower bound on the space complexity of every two-pass streaming algorithm for this problem. In fact, our two-pass lower bound generalizes to an $\widetilde{\Omega}(n \cdot L^{1/p})$ lower bound on the space consumption of any p-pass algorithm, for a constant p.

1.2.1 Two-Pass Algorithm for Random Walk Sampling

For a directed graph $G = (V, E)$, a vertex $u_{\mathsf{start}} \in V$ and a non-negative integer L, we use $\mathsf{RW}_L^G(u_{\mathsf{start}})$ to denote the distribution of L-step random walks $(v_0, \ldots, v_L)$ in G starting from $v_0 = u_{\mathsf{start}}$ (see Subsection 3.2 for formal definitions). For a distribution $\mathcal{D}$ over a finite domain Ω, we say that a randomized algorithm *samples* from $\mathcal{D}$ if, over its internal randomness, it outputs an element $\omega \in \Omega$ distributed according to $\mathcal{D}$. We give a space-efficient streaming algorithm for (approximate) sampling from $\mathsf{RW}_L^G(u_{\mathsf{start}})$ with small error:

▶ **Theorem 1** (Two-pass algorithm). *There exists a streaming algorithm $\mathbb{A}_{\mathsf{two\text{-}pass}}$ that given an n-vertex* directed *graph $G = (V, E)$, a starting vertex $u_{\mathsf{start}} \in V$, a non-negative integer L indicating the number of steps to be taken, and an error parameter $\delta \in (0, 1/n)$, satisfies the following conditions:*

1. *$\mathbb{A}_{\mathsf{two\text{-}pass}}$ uses at most $\widetilde{O}(n \cdot \sqrt{L} \cdot \log \delta^{-1})$ space*[2] *and makes two passes over the input graph G.*
2. *$\mathbb{A}_{\mathsf{two\text{-}pass}}$ samples from some distribution $\mathcal{D}$ over V^{L+1} satisfying $\|\mathcal{D} - \mathsf{RW}_L^G(u_{\mathsf{start}})\|_{\mathrm{TV}} \leq \delta$.*

Our algorithm can also be generalized to the *turnstile* model, paying a $\operatorname{poly}\log n$ factor in the space usage. See Subsection 4.4.

Observe that our algorithm $\mathbb{A}_{\mathsf{two\text{-}pass}}$ allows for a considerable saving in space compared to the folklore single-pass algorithm ($\widetilde{O}(n \cdot \sqrt{L})$ vs $\widetilde{O}(n \cdot L)$) and considerable saving in the number of passes compared to [34] (2 vs $\widetilde{O}(\sqrt{L})$), at least if we allow some small error δ.

We mention that $\mathbb{A}_{\mathsf{two\text{-}pass}}$ can also be used to sample a random path from *every* vertex[3] of G with the same storage cost of $\widetilde{O}(n \cdot \sqrt{L} \cdot \log \delta^{-1})$ and two passes[4]. This is because $\mathbb{A}_{\mathsf{two\text{-}pass}}$ satisfies the useful property of *obliviousness to the starting vertex* u_{start}, meaning

[2] The $\widetilde{O}$ hides logarithmic factors in n. We may assume without loss of generality that $L \leq n^2$, as otherwise $n \cdot \sqrt{L} > n^2$ and that algorithm can store the entire input graph.

[3] Note, however, that the random walks from different vertices in the graph may be correlated.

[4] We count towards the space complexity only the space on the work tape used by the algorithm and do not count space on the output tape (otherwise an $\Omega(n \cdot L)$ lower bound is trivial).

that it scans the input graph before the start vertex is revealed. More formally, we say that an algorithm $\mathbb{A}$ is oblivious to the starting vertex if it first runs a *preprocessing* algorithm $\mathbb{P}$ and then a *sampling* algorithm $\mathbb{S}$; the algorithm $\mathbb{P}$ reads the input graph stream *without* knowing the starting vertex u_{start} (if $\mathbb{A}$ is a p-pass streaming algorithm, $\mathbb{P}$ makes p passes over the input graph stream), and outputs a string; $\mathbb{S}$ takes both the string outputted by $\mathbb{P}$ and a starting vertex u_{start} as an input, and outputs a walk on the input graph G.

1.2.2 Lower Bounds

We prove the following lower bound:

▶ **Theorem 2** (Multi-pass lower bound). *Fix a constant $\beta \in (0,1]$ and an integer $p \geq 1$. Let $n \geq 1$ be a sufficiently large integer and let $L = \lceil n^{\beta} \rceil$. Any randomized p-pass streaming algorithm that, given an n-vertex* directed *graph $G = (V,E)$ and a starting vertex $u_{\mathsf{start}} \in V$, samples from a distribution $\mathcal{D}$ such that $\|\mathcal{D} - \mathsf{RW}_L^G(u_{\mathsf{start}})\|_{\mathrm{TV}} \leq 1 - \frac{1}{\log^{10} n}$ requires $\widetilde{\Omega}(n \cdot L^{1/p})$ space.*

We remark that the theorem above shows that no low-space algorithms can achieve sampling error $1 - 1/\mathrm{poly}\log(n)$, which is quite strong as usual applications of simulating random walks would require at least a small constant sampling error. Plugging in $p = 2$ in Theorem 2, implies that our two-pass algorithm from Theorem 1 is essentially optimal. Also, with $p = 1$, the theorem reproduces the one-pass lower bound by [27]. In addition, Theorem 2 rules out the possibility of a semi-streaming algorithm with any constant number of passes.

Recall from Subsubsection 1.2.1, that our two-pass algorithm $\mathbb{A}_{\mathsf{two\text{-}pass}}$ utilizes $\widetilde{O}(n \cdot \sqrt{L})$ space and is oblivious to the starting vertex. Interestingly, we are able to show that *any* oblivious algorithm for random walk sampling (with any number of passes) requires $\widetilde{\Omega}(n \cdot \sqrt{L})$ space. Thus, any algorithm for random walk sampling with significantly less space than ours, has to be inherently different and have its storage depend on the starting vertex. Our lower bound for oblivious algorithms also implies that $\mathbb{A}_{\mathsf{two\text{-}pass}}$ gives an almost optimal algorithm for sampling a pass from every start vertex, even if any number of passes are allowed.

▶ **Theorem 3** (Lower bound for oblivious algorithms). *Let $n \geq 1$ be a sufficiently large integer and let L denote an integer satisfying that $L \in [\log^{40} n, n]$. Any randomized algorithm that is* oblivious to the start vertex *and given an n-vertex* directed *graph $G = (V,E)$ and a starting vertex $u_{\mathsf{start}} \in V$, samples from a distribution $\mathcal{D}$ such that $\|\mathcal{D} - \mathsf{RW}_L^G(u_{\mathsf{start}})\|_{\mathrm{TV}} \leq 1 - \frac{1}{\log^{10} n}$ requires $\widetilde{\Omega}(n \cdot \sqrt{L})$ space*[5].

1.3 Discussions and Open Problems

Better space complexity with more passes? Our results leave open a couple of interesting directions for future work. The most significant open question is to understand the streaming space complexity of sampling random walks with more than two passes. In particular, Theorem 2 implies that a three-pass streaming algorithm has space complexity at least $\widetilde{\Omega}(n \cdot L^{1/3})$. Can one get $\widetilde{O}(n \cdot L^{1/3})$ space with three passes, or at least $O(n \cdot L^{1/2-\varepsilon})$ space, for some constant $\varepsilon > 0$? Note that, as explained in Subsubsection 1.2.2, such an algorithm must utilize its knowledge of the starting vertex when it reads the graph stream.

[5] In fact, we show that Theorem 3 holds even if the preprocessing algorithm $\mathbb{P}$ and the sampling algorithm $\mathbb{S}$ are allowed to use an arbitrarily large amount of memory, as long as $\mathbb{P}$ passes a string of length at most (roughly) $n \cdot \sqrt{L}$ to $\mathbb{S}$.

Theorem 2 does not rule out semi-streaming $\widetilde{O}(n)$ space algorithms even when p is a moderately growing function of n and L. In [34], it is shown that such an $\widetilde{O}(n)$ space algorithm exists with $p = \widetilde{O}(\sqrt{L})$ passes. Does a semi-streaming algorithm with, say, $\operatorname{poly}\log(L)$ passes exist?

Undirected graphs? It would also be interesting to see what is the best two-pass streaming algorithm for simulating random walks on *undirected graphs*. Specifically, is it possible to combine our algorithm with the algorithm from [27] to obtain an improvement over the optimal $\widetilde{O}(n \cdot \sqrt{L})$ space complexity of a one-pass streaming algorithm for this problem?

Only outputting the end vertex? Finally, our lower bounds only apply to the case where the algorithms need to output an entire random path $(v_0, \ldots, v_L)$. If instead only the last vertex v_L in the random walk is required, can one design better two-pass algorithms or prove a non-trivial lower bound?

2 Techniques

2.1 The Two-Pass Algorithm

We next overview our two-pass algorithm from Theorem 1, that simulates random walks with only $\widetilde{O}(n \cdot \sqrt{L})$ space.

The folklore one-pass algorithm. Before discussing our algorithm, it would be instructive to review the folklore $\widetilde{O}(n \cdot L)$-space one-pass algorithm for simulating L-step random walks in a directed graph $G = (V, E)$ (for simplicity, we will always assume $L \leq n$ in the discussions). The algorithm is quite simple:

1. For every vertex $v \in V$, sample L of its outgoing neighbors *with replacement* and store them in a list L_v^{save} of length L (that is, for each $j \in [L]$, the j-th element of L_v^{save} is an *independent uniformly random* outgoing vertex of v). This can be done in a single pass over input graph stream using reservoir sampling [37].
2. Given a starting vertex $u_{\mathsf{start}} \in V$, our random walk starts from u_{start} and repeats the following for L steps: suppose we are currently at vertex v and it is the k-th time we visit this vertex, then we go from v to the k-th vertex in the list L_v^{save}.

It is not hard to see that the above algorithm works: whenever we visit a vertex $v \in V$, the next element in the list L_v^{save} will always be a uniformly random outgoing neighbor of v, *conditioned on* the walk we have produced so far; and we will never run out of the available neighbors of v as $|L_v^{\mathsf{save}}| = L$.

A naive attempt and the obstacles. Since we are aiming at only using $\widetilde{O}(n \cdot \sqrt{L})$ space, a naive attempt to improve the above algorithm is to just sample and store $\tau = O(\sqrt{L})$ outgoing neighbors instead of L neighbors, and simulate the walk starting from u_{start} in the same way. The issue here is that, during the simulation of an L-step walk, whenever one visits a vertex v more than τ times, one would run out of available vertices in the list L_v^{save}, and the algorithm can no longer produce a legit random walk. For a simple example, imagine we have a star-like graph where $n - 1$ vertices are connected to a center vertex via two-way edges. An L-step random walk starting at the center would require at least $\Omega(L)$ samples from the center's neighbors, and our naive algorithm completely breaks.

Our approach: heavy and light vertices. Observe, however, that in the above example of a star-like graph, we are only at risk of not storing enough random neighbors of the center node, as an L-step *random* walk would only visit the other non-center vertices a very small number of times. Thus, the algorithm may simply record all edges from the center with only $O(n)$ space. This observation inspires the following approach for a two-pass algorithm:

1. In the first pass, we identify all the vertices that are likely to be visited many times by a random walk (starting from *some* vertex). We call such vertices *heavy*, while all other vertices are called *light*.
2. In the second pass, we record *all* outgoing neighbors of all heavy vertices, as well as $O(\tau)$ random outgoing neighbors with replacement of each of the light vertices.

Observe that the obtained algorithm is indeed *oblivious to the starting vertex*: the two passes described above do not use the starting vertex. Still, given the set of outgoing neighbors stored by the second pass, we are able to sample a random walk from any start vertex.

First pass: how do we detect heavy vertices? The above approach requires that we detect, in a single pass, all vertices v that with a decent probability (say, $1/\text{poly}(n)$), are visited more than $O(\tau)$ times by an L-step random walk. To this end, we observe that if a random walk visits a vertex v more than τ times, this random walk must follow more than $\tau - 1$ self-circles around v in L steps. This, in turn, implies that a random walk that starts from v is likely to return to v in roughly $L/\tau = O(\sqrt{L})$ steps.

The above discussion suggests the following definition of heavy vertices: a vertex v is *heavy*, if a random walk starting from v is likely (say, with probability at least $1/3$) to revisit v in $O(\sqrt{L})$ steps. Indeed, this property is much easier to detect: we can run $O(\log n)$ independent copies of the folklore one-pass streaming algorithms to sample $O(\log n)$ $O(\sqrt{L})$-step random walks starting from v, and count how many of them return to v at some step.

Second pass: can we afford to store the neighbors? In Lemma 18, we show that for a light vertex v, an L-step random walk starting at any vertex visits v $O(\sqrt{L})$ times with high probability. Therefore, in the second pass, we can safely record only $O(\sqrt{L})$ outgoing neighbors for all light vertices. Still, we have to record all the outgoing neighbors for heavy vertices.

The crux of our analysis is a *structural result* about directed graphs, showing that the total outgoing degree of all heavy vertices is bounded by $O(n \cdot \sqrt{L})$, and therefore we can simply store all of their outgoing neighbors. This is proved in Lemma 10, which may also be of independent interest.

Intuition behind the structure lemma. Finally, we discuss the insights behind the above structure lemma for directed graphs. We will use $d_{\mathsf{out}}(v)$ to denote the number of outgoing neighbors of v. For concreteness, we now say a vertex v is heavy if a random walk starting from v revisits v in $\sqrt{L}$ steps with probability at least $1/3$.

Let $V_{\mathsf{heavy}} \subseteq V$ be the set of heavy vertices and let $v \in V_{\mathsf{heavy}}$. By a simple calculation, one can see that for at least a $1/6$ fraction of outgoing neighbors u of v, a random walk starting from u visits v in $\sqrt{L}$ steps with probability at least $1/6$. The key insight is to consider the number of pairs $(u, v) \in V^2$ such that a random walk starting from u visits v in $\sqrt{L}$ steps with probability at least $1/6$. We will use $\mathcal{S}$ to denote this set.

- By the previous discussions, we can see that for each heavy vertex v, it adds at least $1/6$ $d_{\mathsf{out}}(v)$ pairs to the set $\mathcal{S}$. Hence, we have

$$|\mathcal{S}| \geq \frac{1}{6} \cdot \sum_{v \in V_{\mathsf{heavy}}} d_{\mathsf{out}}(v). \tag{1}$$

- On the other hand, it is not hard to see that for each vertex v, there are at most $O(\sqrt{L})$ many pairs of the form $(v, u) \in \mathcal{S}$, since a $\sqrt{L}$-step walk can visit only $\sqrt{L}$ vertices. So we also have

$$|\mathcal{S}| \leq O(n\sqrt{L}). \tag{2}$$

Putting the above (Equation 1 and Equation 2) together, we get the desired bound

$$\sum_{v \in V_{\mathsf{heavy}}} d_{\mathsf{out}}(v) \leq O(n\sqrt{L}).$$

2.2 Lower Bound for p-Pass Algorithms

We now describe the ideas behind the proof of Theorem 2, our $\widetilde{\Omega}(n \cdot L^{1/p})$ space lower bound for p-pass randomized streaming algorithms for sampling random walks. We mention that many of the tools developed for proving space lower bounds are not directly applicable when one wishes to lower bound the space complexity of a *sampling* task and are more suitable for proving lower bounds on the space required to compute a function or a search problem[6].

From sampling to function computation. Our way around this is to first prove a reduction from streaming algorithms that sample a random walk from u_{start} to streaming algorithms that compute the $(p+1)$-neighborhood of the vertex u_{start}. This is done by considering a graph where a random walk returns to the vertex u_{start} every $p+2$ steps. If p is a constant, then a random walk of length L on such a graph can be seen as $L/(p+2) = O(L)$ copies of a random walk of length $p+2$. Observe that if the $(p+1)$-neighborhood of the vertex u_{start} has (almost) L vertices (and the probability of visiting each vertex is more or less uniform), then a random walk of length L is likely to visit all the vertices in the neighborhood and an algorithm that samples a random walk also outputs the entire neighborhood with high probability.

A lower bound for computing the $(p+1)$-neighborhood via pointer-chasing. Having reduced sampling a random walk to outputting the $(p+1)$-neighborhood, we now need to prove that a space efficient p-pass streaming algorithms cannot output the $(p+1)$-neighborhood of u_{start}, if this neighborhood has roughly L vertices. This is reminiscent of the "*pointer-chasing*" lower bounds found in the literature.

Pointer-chasing results are typically concerned with a graph with $p+1$ layers of vertices (p layers of edges) and show that given a vertex in the first layer, finding a vertex that is reachable from it in the last layer cannot be done with less than p passes, unless the memory is huge. Classical pointer-chasing lower bounds (*e.g.*, [31]), consider graphs where the out-degree of each vertex is 1, thus the start vertex reaches a unique vertex in the last layer. Unfortunately, this type of pointer-chasing instances are very *sparse* and a streaming algorithm can simply remember the entire graph in one pass using $\widetilde{O}(n)$ memory.

[6] One such tool that cannot be used directly for our purpose is the very useful *Yao's minimax principle* [38] that allows proving randomized communication lower bounds by proving the corresponding distributional (deterministic) communication lower bounds.

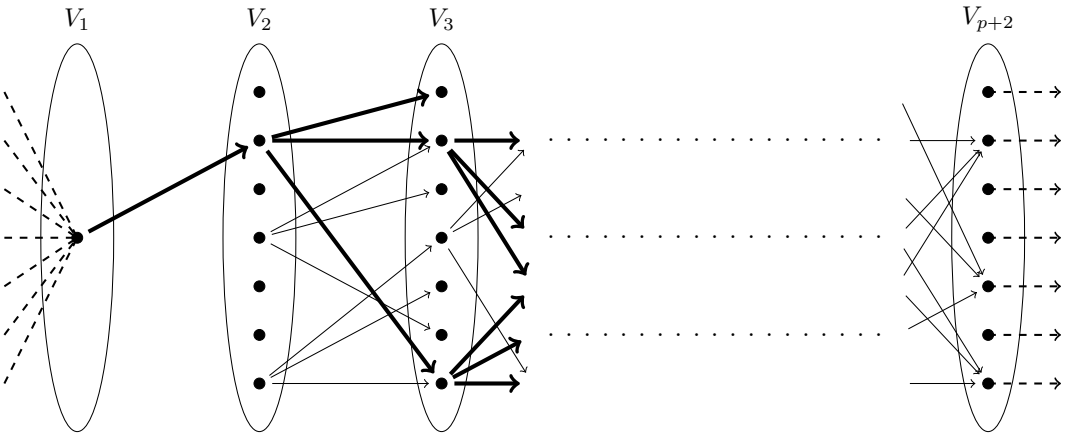


Figure 1 A depiction of our hard instance for p-pass streaming algorithms. Some edges omitted.

Since we wish to have roughly L vertices in a $(p+1)$-neighborhood of u_{start}, the out-degree of each vertex should be roughly $\Omega(L^{\frac{1}{p+1}})$ (assuming uniform degrees). Pointer-chasing lower bounds for this type of *dense* graphs were also proved (*e.g.*, [20] and [16]), showing that p-pass algorithms essentially need to store an entire layer of edges, which is $\Omega(n \cdot L^{\frac{1}{p+1}})$ in our case. However, this still does not give us the $\Omega(n \cdot L^{\frac{1}{p}})$ lower bound we aspire for (and which is tight, at least for two passes).

Towards a tight lower bound: combining dense and sparse. To get a better lower bound, we construct a hard instance that is a combination of the two above mentioned types of pointer-chasing instances, the dense and the sparse. Specifically, for a p-pass lower bound, we construct a layered graph with $p+2$ layers of vertices $V_1, \ldots, V_{p+2}$, where the first layer has only one vertex u_{start} and all the other layers are of equal size (see Figure 1). To ensure that vertex u_{start} is reached every $p+2$ steps, we connect all vertices in the last layer to u_{start}. Every vertex in layers $V_2, \ldots, V_{p+1}$ connects to a random set of roughly $L^{\frac{1}{p}}$ vertices in the next layer. Using Guruswami and Onak style arguments ([20]), it can be shown that when the edges are presented to the algorithm from right to left, finding a vertex in layer V_{p+2} that is reachable form a given vertex in V_2 with a $(p-1)$-pass algorithm requires $\Omega(n \cdot L^{\frac{1}{p}})$ space. We "squeeze out" an extra pass in the algorithm by connecting the start vertex u_{start} in V_1 to a single random vertex in V_2. Note that with this construction, it is indeed the case that a $(p+1)$-neighborhood of u_{start} consists of only roughly L vertices, but still, the out-degrees of vertices in $V_2, \ldots, V_{d+1}$ are roughly $L^{\frac{1}{p}}$ instead of only $L^{\frac{1}{p+1}}$.

2.3 Lower bounds for Oblivious Algorithms

Finally, we discuss the intuitions behind the proof of Theorem 3, showing that any algorithm that is oblivious to the starting vertex must use $\widetilde{\Omega}(n \cdot \sqrt{L})$ space space. Our proof is based on a reduction from a multi-output generalization of the well-studied INDEX problem for one-way communication protocols, denoted by $\mathsf{INDEX}_{m,\ell}$. In $\mathsf{INDEX}_{m,\ell}$, Alice gets ℓ strings $X_1, \ldots, X_\ell \in \{0,1\}^m$ and Bob gets an index $i \in [\ell]$. Alice sends a message to Bob and then Bob is required to output the string X_i. (Note that when $m = 1$ it becomes the original INDEX problem).

It is not hard to show that any one-way communication protocol solving $\mathsf{INDEX}_{m,\ell}$ with non-trivial probability (say, $1/\operatorname{poly}\log(m)$) requires Alice to send at least $\widetilde{\Omega}(m\ell)$ bits to Bob (see the full version of this paper [11] for details).

Our key observation here is that if there is a starting vertex oblivious algorithm $\mathbb{A} = (\mathbb{P}, \mathbb{S})$ with S space for approximate simulation of an $L = \widetilde{O}(m)$-step random walk on a graph with $n = O(\sqrt{m} \cdot \ell)$ vertices, then it implies a one-way communication protocol for $\mathsf{INDEX}_{m,\ell}$ with communication complexity S and a decent success probability. Recall the lower bound for $\mathsf{INDEX}_{m,\ell}$, we immediately have $S = \widetilde{\Omega}(m\ell) = \widetilde{\Omega}(n\sqrt{L})$.

In more detail, given an m-bit string X, we will build an $O(\sqrt{m})$-vertex graph $H(X)$ by encoding all bits of X as *existence/non-existence* of edges in H (this is possible since there are more than m potential edges in H). We also add some artificial edges to H to make sure it is strongly connected. Our construction will make sure that an $L = \widetilde{O}(m)$ steps random walk in H will reveal all edges in H with high probability, which in turn reveals all bits of X (see the proof of Theorem 3 in the full version of this paper for more details).

Now the reduction can be implemented as follows: given ℓ strings $X_1, \ldots, X_\ell \in \{0,1\}^m$, Alice constructs a graph $G = \bigsqcup_{i=1}^{\ell} H(X_i)$, as the joint union of ℓ graphs. Note that G has $n = O(\sqrt{m} \cdot \ell)$ vertices. Alice then runs the preprocessing algorithm $\mathbb{P}$ on G to obtain a string M, and sends it to Bob. Given an index $i \in [\ell]$, Bob simply runs $\mathbb{S}$ with M together with a suitable starting vertex inside the $H(X_i)$ component of G. By previous discussions, this reveals the string X_i with high probability and proves the correctness of this reduction. Hence, the space complexity of $\mathbb{A}$ must be $\widetilde{\Omega}(m\ell) = \widetilde{\Omega}(n\sqrt{L})$.

Organization of this paper

In Section 3 we introduce the necessary preliminaries for this paper. In Section 4 we present our nearly optimal two-pass streaming algorithm for simulating random walks and prove Theorem 1. See the full version of this paper [11] for the proof of Theorem 2 and Theorem 3.

3 Preliminaries

3.1 Notation

Let $n \in \mathbb{N}$. We use $[n]$ to denote the set $\{1, \ldots, n\}$. We often use sans-serif letters (*e.g.*, X) to denote random variables, and calligraphic font letters (*e.g.*, $\mathcal{X}$) to denote distributions. For two random variables X and Y, and for $Y \in \mathrm{supp}(\mathsf{Y})$, we use $(X|\mathsf{Y} = Y)$ to denote X conditioned on $\mathsf{Y} = Y$. For two lists a and b, we use $a \circ b$ to denote their concatenation.

For two distributions $\mathcal{D}_1$ and $\mathcal{D}_2$ on set $\mathcal{X}$ and $\mathcal{Y}$ respectively, we use $\mathcal{D}_1 \otimes \mathcal{D}_2$ to denote their product distribution over $\mathcal{X} \times \mathcal{Y}$, and $\|\mathcal{D}_1 - \mathcal{D}_2\|_{\mathrm{TV}}$ to denote the total variation distance between them.

3.2 Graphs

In this paper we will always consider directed graphs without multi-edges. A directed G is a pair (V, E), where V is the vertex set and $E \subseteq V \times V$ is the set of all edges.

For a vertex u in a graph $G = (V, E)$, we let $N^G_{\mathsf{out}}(u) \coloneqq \{v : (u, v) \in E\}$ and $N^G_{\mathsf{in}}(u) \coloneqq \{v : (v, u) \in E\}$. We also use $d^G_{\mathsf{out}}(u)$ and $d^G_{\mathsf{in}}(u)$ to denote its out and in degrees (*i.e.*, $|N^G_{\mathsf{out}}(u)|$ and $|N^G_{\mathsf{in}}(u)|$). For an edge $(u, v) \in E$, we say v is the *out-neighbor* of u and u is the *in-neighbor* of v.

Random walks on directed graphs. For a vertex u in a graph $G = (V, E)$ and an non-negative integer L, an L-step random walk $(v_0, v_1, \ldots, v_L)$ starting at u is generated as follows: set $v_0 = u$, for each $i \in [L]$, we draw v_i uniformly random from $N^G_{\mathsf{out}}(u)$. We say that $v_0 = u$ is the 0-th vertex on the walk, and v_i is the i-th vertex for each $i \in [L]$. We use $\mathsf{RW}^G_L(u)$ to denote the distribution of an L-step random walk starting from u in G.

We use $\mathsf{visit}^G_{[a,b]}(u,v)$ to denote the probability of a b-step random walk starting from u visits v between the a-th vertex and b-th vertex on the walk.

We often omit the superscript G when the graph G is clear from the context.

Starting vertex oblivious algorithms. Now we formally define a starting vertex oblivious streaming algorithm for simulating random walks.

▶ **Definition 4.** *We say a p-pass S-space streaming algorithm $\mathbb{A}$ for simulating random walks is starting vertex oblivious, if $\mathbb{A}$ can be decomposed into a preprocessing subroutine $\mathbb{P}$ and a sampling subroutine $\mathbb{S}$, such that:*

1. *(**Starting vertex oblivious preprocessing phase**) $\mathbb{P}$ makes p passes over the input graph stream, using at most S words of space. After that, $\mathbb{P}$ outputs at most S words, denoted as M.*
2. *(**Sampling phase**) $\mathbb{S}$ takes both the starting vertex u_{start} and M as input, and outputs a desired walk starting from u_{start}, using at most S words of space.*

3.3 Useful Concentration Bounds on Random Variables

The following standard concentration bounds will be useful for us.

▶ **Lemma 5** (Multiplicative Chernoff bound, [12])**.** *Suppose $X_1, \cdots, X_n$ are independent random variables taking values in $[0,1]$. Let X denote their sum and let $\mu = \mathbb{E}[X]$ denote the sum's expected value. Then,*

$$\Pr\left(X \geq (1+\delta)\mu\right) \leq \mathrm{e}^{-\frac{\delta^2\mu}{2+\delta}}, \qquad \forall 0 \leq \delta,$$

$$\Pr\left(X \leq (1-\delta)\mu\right) \leq \mathrm{e}^{-\frac{\delta^2\mu}{2}}, \qquad \forall 0 \leq \delta \leq 1.$$

In particular, we have that:

$$\Pr\left(X \geq (1+\delta)\mu\right) \leq \mathrm{e}^{-\frac{\delta\mu}{3}\cdot\min(\delta,1)}, \qquad \forall 0 \leq \delta,$$

$$\Pr\left(|X-\mu| \geq \delta\mu\right) \leq 2\cdot\mathrm{e}^{-\frac{\delta^2\mu}{3}}, \qquad \forall 0 \leq \delta \leq 1.$$

We also need the following Azuma-Hoeffding inequality.

▶ **Lemma 6** (Azuma-Hoeffding inequality, [5, 23])**.** *Let $Z_0, \ldots, Z_n$ be random variables satisfying (1) $\mathbb{E}[|Z_i|] < \infty$ for every $i \in \{0, \ldots, n\}$ and $\mathbb{E}[Z_i | Z_0, \ldots, Z_{i-1}] \leq Z_{i-1}$ for every $i \in [n]$ (i.e., $\{Z_i\}$ forms a supermartingale) and (2) for every $i \in [n]$, $|Z_i - Z_{i-1}| \leq 1$, then for all $\lambda > 0$, we have*

$$\Pr[Z_n - Z_0 \geq \lambda] \leq \exp(-\lambda^2/2n).$$

In particular, the following corollary will be useful for us.

▶ **Corollary 7** (Azuma-Hoeffding inequality for Boolean random variables, [5, 23])**.** *Let $X_1, \ldots, X_n$ be random variables satisfying $X_i \in \{0,1\}$ for each $i \in [n]$. Suppose that $\mathbb{E}[X_i | X_1, \ldots, X_{i-1}] \leq p_i$ for all i. Then for any $\lambda > 0$,*

$$\Pr\left[\sum_{i=1}^{n} X_i \geq \lambda + \sum_{i=1} p_i\right] \leq \exp(-\lambda^2/2n).$$

Proof. For $i \in \{0, \ldots, n\}$, let $Z_i = \sum_{j=1}^{i}(X_j - p_j)$. From the assumption one can see that all the Z_i form a supermartingale and $|Z_i - Z_{i-1}| \leq 1$, hence the corollary follows directly from Lemma 6. ◀

4 Two-Pass Streaming Algorithms for Simulating Directed Random Walk

In this section, we present our two-pass streaming algorithms for simulating random walks on directed graphs.

4.1 Heavy and Light Vertices

We first define the notion of heavy and light vertices.

▶ **Definition 8** (Heavy and light vertices). *Given a directed graph $G = (V, E)$ with n vertices and $\ell \in \mathbb{N}$.*

- *(**Heavy vertices**.) We say a vertex u is ℓ-heavy in G, if $\mathsf{visit}_{[1,\ell]}(u, u) \geq 1/3$ (i.e., if a random walk starting from u will revisit u in at most ℓ steps with probability at least $1/3$.)*
- *(**Light vertices**.) We say a vertex u is ℓ-light in G, if $\mathsf{visit}_{[1,\ell]}(u, u) \leq 2/3$ (i.e., if a random walk starting from u will revisit u in at most ℓ steps with probability at most $2/3$.)*

We also let $V^{\ell}_{\mathsf{heavy}}(G)$ and $V^{\ell}_{\mathsf{light}}(G)$ be the sets of ℓ-heavy and ℓ-light vertices in G. When G and ℓ are clear from the context, we simply refer to them as V_{heavy} and V_{light}.

▶ Remark 9. Note that if the revisiting probability is between $[1/3, 2/3]$, then the vertex is considered to be both heavy and light.

The following lemma is crucial for the analysis of our algorithm.

▶ **Lemma 10** (Upper bounds on the total out-degrees of heavy vertices). *Given a directed graph G with n vertices and $\ell \in \mathbb{N}$, it holds that*

$$\sum_{u \in V^{\ell}_{\mathsf{heavy}}(G)} d_{\mathsf{out}}(u) \leq O(n \cdot \ell).$$

Proof. We define a set $\mathcal{S}$ of pairs of vertices as follows:

$$\mathcal{S} \coloneqq \{(u, v) \in V^2 : \mathsf{visit}_{[0,\ell]}(u, v) \geq 1/6\}.$$

That is, a pair of vertices u and v belongs to $\mathcal{S}$ if and only if an ℓ-step random walk starting from u visits v with probability at least $1/6$.

For each fixed vertex u, we further define

$$\mathcal{S}_u \coloneqq \big\{v \in N_{\mathsf{out}}(u) \mid \mathsf{visit}_{[0,\ell]}(v, u) \geq 1/6\big\},$$

and

$$\mathcal{H}_u \coloneqq \big\{v \in V \mid \mathsf{visit}_{[0,\ell]}(u, v) \geq 1/6\big\}.$$

The following claim will be useful for the proof.

▷ Claim 11. The following two statements hold:

1. For every $u \in V$, it holds that $|\mathcal{H}_u| \leq O(\ell)$.
2. For every $u \in V_{\mathsf{heavy}}$, it holds that $|\mathcal{S}_u| \geq 1/6 \cdot d_{\mathsf{out}}(u)$.

Proof. Fixing $u \in V$, the first item follows from the simple fact that

$$\sum_{v \in V} \mathsf{visit}_{[0,\ell]}(u, v) \leq \ell + 1.$$

Now we move to the second item, and fix $u \in V_{\mathsf{heavy}}$. For the sake of contradiction, suppose that $|\mathcal{S}_u| < 1/6 \cdot d_{\mathsf{out}}(u)$. We have

$$\begin{aligned}\mathsf{visit}_{[1,\ell]}(u,u) &= \mathop{\mathbb{E}}_{v \in N_{\mathsf{out}}(u)}[\mathsf{visit}_{[0,\ell-1]}(v,u)] \\ &\leq \mathop{\mathbb{E}}_{v \in N_{\mathsf{out}}(u)}[\mathsf{visit}_{[0,\ell]}(v,u)] \\ &< \Pr_{v \in N_{\mathsf{out}}(u)}[v \in \mathcal{S}_u] \cdot 1 + \Pr_{v \in N_{\mathsf{out}}(u)}[v \notin \mathcal{S}_u] \cdot 1/6 < 1/6 + 1/6 < 1/3,\end{aligned}$$

a contradiction to the assumption that u is heavy. ◁

Finally, note that by definition of $\mathcal{H}_u$ and $\mathcal{S}_u$ we immediately have

$$|\mathcal{S}| = \sum_{u \in V} |\mathcal{H}_u| \geq \sum_{u \in V} |\mathcal{S}_u|.$$

By Claim 11, we have

$$\sum_{u \in V_{\mathsf{heavy}}} d_{\mathsf{out}}(u) \leq 6 \cdot \sum_{u \in V} |\mathcal{S}_u| \leq 6 \cdot \sum_{u \in V} |\mathcal{H}_u| \leq O\left(n \cdot \ell\right),$$

which completes the proof. ◀

4.2 A Simple One-Pass Algorithm for Simulating Random Walks

We first describe a simple one-pass algorithm for simulating random walks, which will be used as a sub-routine in our two-pass algorithm. Moreover, this one-pass algorithm is starting vertex oblivious, which will be crucial for us later.

Reservoir sampling in one pass. Before describing our one-pass subroutine, we need the following basic reservoir sampling algorithm.

▶ **Lemma 12** ([37]). *Given input access to a stream of n items such that each item can be described by $O(1)$ words, we can uniformly sample m of them without replacement using $O(m)$ words of space.*

Using m independent reservoir samplers each with capacity 1, one can also sample m items from the stream *with replacement* in a space-efficient way.

▶ **Corollary 13.** *Given input access to a stream of n items such that each item can be described by $O(1)$ words, we can uniformly sample m of them with replacement using $O(m)$ words of space.*

Description of the one-pass algorithm. Now we describe our one-pass algorithm for simulating random walks. Our algorithm $\mathbb{A}_{\mathsf{one\text{-}pass}}$ is starting vertex oblivious, and can be described by a preprocessing subroutine $\mathbb{P}_{\mathsf{one\text{-}pass}}$ and a sampling subroutine $\mathbb{S}_{\mathsf{one\text{-}pass}}$. Recall that as defined in Definition 4, $\mathbb{P}_{\mathsf{one\text{-}pass}}$ takes a single pass over the input graph streaming without knowing the starting vertex u_{start}, and $\mathbb{S}_{\mathsf{one\text{-}pass}}$ takes the output of $\mathbb{P}_{\mathsf{one\text{-}pass}}$ together with u_{start}, and outputs a desired sample fo the random walk.

Algorithm 1 Preprocessing phase of $\mathbb{A}_{\text{one-pass}}$: $\mathbb{P}_{\text{one-pass}}(G, \tau, V_{\text{full}})$.

Input: One pass streaming access to a directed graph $G = (V, E)$. A parameter $\tau \in \mathbb{N}$. A subset $V_{\text{full}} \subseteq V$, and we also let $V_{\text{samp}} = V \setminus V_{\text{full}}$.

1: For each vertex $v \in V_{\text{full}}$, we record all its out-neighbors in the list L_v^{save}. (That is, V_{full} stands for the set of vertices that we keep all its edges.)

2: For each vertex $v \in V_{\text{samp}}$, using Corollary 13, we sample τ of its out-neighbors uniformly at random with replacement in the list L_v^{save}. (That is, V_{samp} stands for the set of vertices that we sample some of its edges.)

3: For a big enough constant $c_2 > 1$, whenever the number of out-neighbors stored exceeds $c_2 \cdot \tau \cdot n$, the algorithm stops recording them. If this happens, we say the algorithm *operates incorrectly* and otherwise we say it *operates correctly*.

Output: A collection of lists $\vec{L}^{\text{save}} = \{L_v^{\text{save}}\}_{v \in V}$.

Algorithm 2 Sampling phase of $\mathbb{A}_{\text{one-pass}}$: $\mathbb{S}_{\text{one-pass}}(V, u_{\text{start}}, L, V_{\text{full}}, \vec{L}^{\text{save}} = \{L_v^{\text{save}}\}_{v \in V})$.

Input: A starting vertex u_{start}. The path length $L \in \mathbb{N}$. A subset $V_{\text{full}} \subseteq V$, and we also let $V_{\text{samp}} = V \setminus V_{\text{full}}$.

1: Let $v_0 = u_{\text{start}}$. For each $v \in V$, we set $k_v = 1$.
2: **for** $i \coloneqq 1 \to L$ **do**
3: **if** $v_{i-1} \in V_{\text{full}}$ **then**
4: v_i is set to be a uniformly random element from $L_{v_{i-1}}^{\text{save}}$
5: **else if** $k_{v_{i-1}} > |L_{v_{i-1}}^{\text{save}}|$ **then**
6: **return** failure
7: **else**
8: $v_i \leftarrow (L_{v_{i-1}}^{\text{save}})_{k_{v_{i-1}}}$.
9: $k_{v_{i-1}} \leftarrow k_{v_{i-1}} + 1$.
10: **end if**
11: **end for**

Output: The walk $(v_0, v_1, \ldots, v_L)$.

Analysis of the one-pass algorithm. Now we analyze the correctness of our one-pass algorithm. We first observe its space complexity can be easily bounded.

▶ **Observation 14** (Space complexity of $\mathbb{A}_{\text{one-pass}}$). *Given a directed graph $G = (V, E)$ with n vertices. For every $\tau \in \mathbb{N}$ and subset $V_{\text{full}} \subseteq V$, $\mathbb{P}_{\text{one-pass}}(G, \tau, V_{\text{full}})$ always takes at most $O(\tau \cdot n)$ words of space.*

Next we bound the statistical distance between its output distribution and the correct distribution of the random walk by the following lemma.

▶ **Lemma 15** (Correctness of $\mathbb{A}_{\text{one-pass}}$). *Given a directed graph $G = (V, E)$ with n vertices. For every integers $\tau, L \in \mathbb{N}$ and subset $V_{\text{full}} \subseteq V$ such that $\tau \cdot (n - |V_{\text{full}}|) + \sum_{v \in V_{\text{full}}} d_{\text{out}}(v) \leq c_2 \cdot \tau \cdot n$, let $\vec{\mathsf{L}}^{\text{save}}$ be random variable of the output of $\mathbb{P}_{\text{one-pass}}(G, \tau, V_{\text{full}})$. For every $u_{\text{start}} \in V$, the output distribution of $\mathbb{S}_{\text{one-pass}}(V, u_{\text{start}}, L, V_{\text{full}}, \vec{\mathsf{L}}^{\text{save}})$ has statistical distance β to $\mathsf{RW}_L^G(u_{\text{start}})$, where β is the probability that $\mathbb{S}_{\text{one-pass}}(V, u_{\text{start}}, L, V_{\text{full}}, \vec{\mathsf{L}}^{\text{save}})$ outputs failure.*

Proof. Conclude from $\tau \cdot (n - |V_{\mathsf{full}}|) + \sum_{v \in V_{\mathsf{full}}} d_{\mathsf{out}}(v) \leq c_2 \cdot \tau \cdot n$ that $\mathbb{P}_{\mathsf{one\text{-}pass}}(G, \tau, V_{\mathsf{full}})$ always operates correctly.

To bound the statistical distance between the distribution of $\mathbb{S}_{\mathsf{one\text{-}pass}}(V, u_{\mathsf{start}}, L, V_{\mathsf{full}}, \vec{\mathsf{L}}^{\mathsf{save}})$ and $\mathsf{RW}_L^G(u_{\mathsf{start}})$. We construct another random variable $(\vec{\mathsf{L}}^{\mathsf{save}})'$, in which for every vertex u, we sample another L out-neighbors of u uniformly at random with replacement, and add them to the end of the list L_v^{save} in $\vec{\mathsf{L}}^{\mathsf{save}}$.

Note that $\mathbb{S}_{\mathsf{one\text{-}pass}}(V, u_{\mathsf{start}}, L, V_{\mathsf{full}}, (\vec{\mathsf{L}}^{\mathsf{save}})')$ never outputs failure, and distributes exactly the same as $\mathsf{RW}_L^G(u_{\mathsf{start}})$. On the other hand, $\mathbb{S}_{\mathsf{one\text{-}pass}}(V, u_{\mathsf{start}}, L, V_{\mathsf{full}}, (\vec{\mathsf{L}}^{\mathsf{save}})')$ and $\mathbb{S}_{\mathsf{one\text{-}pass}}(V, u_{\mathsf{start}}, L, V_{\mathsf{full}}, \vec{\mathsf{L}}^{\mathsf{save}})$ are the same as long as $\mathbb{S}_{\mathsf{one\text{-}pass}}(V, u_{\mathsf{start}}, L, V_{\mathsf{full}}, \vec{\mathsf{L}}^{\mathsf{save}})$ does not output failure, which completes the proof. ◀

The following corollary follows immediately from the lemma above. (Note that this special case exactly corresponds to the folklore one-pass streaming algorithm for simulating random walks.)

▶ **Corollary 16.** *Given a directed graph $G = (V, E)$ with n vertices and an integer $L \in \mathbb{N}$. Let $\vec{\mathsf{L}}^{\mathsf{save}}$ be random variable of the output of $\mathbb{P}_{\mathsf{one\text{-}pass}}(G, L, \emptyset)$. For every $u_{\mathsf{start}} \in V$, the output distribution of $\mathbb{S}_{\mathsf{one\text{-}pass}}(V, u_{\mathsf{start}}, L, \emptyset, \vec{\mathsf{L}}^{\mathsf{save}})$ distributes identically as $\mathsf{RW}_L^G(u_{\mathsf{start}})$.*

4.3 Two-Pass Streaming Algorithm for Simulating Random Walks

Description of the two-pass algorithm. Now we are ready to describe our two pass algorithm $\mathbb{A}_{\mathsf{two\text{-}pass}}$, which is also starting vertex oblivious, and can be described by the following two sub-routines $\mathbb{P}_{\mathsf{two\text{-}pass}}$ and $\mathbb{S}_{\mathsf{two\text{-}pass}}$.

Algorithm 3 Preprocessing phase of $\mathbb{A}_{\mathsf{two\text{-}pass}}$: $\mathbb{P}_{\mathsf{two\text{-}pass}}(G, L, \delta)$.

Input: A directed graph $G = (V, E)$ with n vertices. An integer $L \in \mathbb{N}$. A failure parameter $\delta \in (0, 1/n)$. We also let $\ell = \sqrt{L}$, and $\gamma = c_1 \cdot \log \delta^{-1}$ where $c_1 \geq 1$ is a sufficiently large constant to be specified later.

1: **First pass: estimation of heavy and light vertices.**

1. Run γ independent instances of $\mathbb{P}_{\mathsf{one\text{-}pass}}(G, \ell, \emptyset)$ and let $(L^{\mathsf{save}})^{(1)}, \ldots, (L^{\mathsf{save}})^{(\gamma)}$ be the corresponding collections of lists.
2. For each vertex $u \in V$, by running $\mathbb{S}_{\mathsf{one\text{-}pass}}(V, u, \ell, \emptyset, (L^{\mathsf{save}})^{(j)})$ for each $j \in [\gamma]$, we take γ independent samples from RW_ℓ^G. Let w_u be the fraction of these random walks that revisit u in ℓ steps.
3. Let $\widetilde{V}_{\mathsf{heavy}}$ be the set of vertices with $w_u \geq 0.5$, and $\widetilde{V}_{\mathsf{light}}$ be the set of vertices with $w_u < 0.5$.

2: **Second Pass: heavy-light edge recording**

1. Let $V_{\mathsf{full}} = \widetilde{V}_{\mathsf{heavy}}$.
2. Run $\mathbb{P}_{\mathsf{one\text{-}pass}}(G, \gamma \cdot \ell, V_{\mathsf{full}})$ to obtain a collection of lists $\vec{L}^{\mathsf{save}}$.

Output: The set V_{full} and the collection of lists $\vec{L}^{\mathsf{save}}$.

Algorithm 4 Sampling phase of $\mathbb{A}_{\mathsf{two\text{-}pass}}$: $\mathbb{S}_{\mathsf{two\text{-}pass}}(V, u_{\mathsf{start}}, L, V_{\mathsf{full}}, \vec{L}^{\mathsf{save}} = \{L_v^{\mathsf{save}}\}_{v \in V})$.

Input: A starting vertex u_{start}. The path length $L \in \mathbb{N}$. A subset $V_{\mathsf{full}} \subseteq V$, and a collection of lists $\vec{L}^{\mathsf{save}}$.

Output: Simulate $\mathbb{S}_{\mathsf{one\text{-}pass}}(V, u_{\mathsf{start}}, L, V_{\mathsf{full}}, \vec{L}^{\mathsf{save}})$ and return its output.

Analysis of the algorithm. We first show that with high probability, $\widetilde{V}_{\mathsf{light}}$ and $\widetilde{V}_{\mathsf{heavy}}$ are subsets of V_{light} and V_{heavy} respectively.

▶ **Lemma 17.** *Given a directed graph $G = (V, E)$ with n vertices, $L \in \mathbb{N}$ and $\delta \in (0, 1/n)$, letting $\ell = \sqrt{L}$, with probability at least $1-\delta/2$ over the internal randomness of $\mathbb{P}_{\mathsf{two\text{-}pass}}(G, L, \delta)$, it holds that $\widetilde{V}_{\mathsf{light}} \subseteq V_{\mathsf{light}}$ and $\widetilde{V}_{\mathsf{heavy}} \subseteq V_{\mathsf{heavy}}$.*

Proof. Setting c_1 in Algorithm 3 to be a large enough constant and applying Corollary 16 and the Chernoff bound, with probability at least $1 - n \cdot \delta^3 \geq 1 - \delta/2$, $|w_u - \mathsf{visit}_{[1,\ell]}(u, u)| \leq 0.1$ for every $u \in V$. The lemma then follows from the definition of heavy and light vertices. ◀

Next, we show that with high probability, a random walk does not visit a light vertex too many times.

▶ **Lemma 18.** *Given a directed graph $G = (V, E)$ with n vertices, $L \in \mathbb{N}$ and $\delta \in (0, 1/n)$, letting $\ell = \sqrt{L}$ and $\gamma = c_1 \cdot \log \delta^{-1}$, where $c_1 > 1$ is the sufficiently large constant, for every vertex $u_{\mathsf{start}} \in V$ and vertex $v \in V^{\ell}_{\mathsf{light}}(G)$, an L-step random walk starting from u_{start} visits v more than $\gamma \cdot \ell$ times with probability at most $\delta/2n$.*

Proof. Suppose we have an infinite random walk W starting from u_{start} in G. Letting $\tau = \gamma\ell$, the goal here is to bound the probability that during the first L steps, W visits v more than τ times. We denote this as the bad event $\mathcal{E}_{\mathsf{bad}}$.

Let Z_i be the random variable representing the step at which W visits v for the i-th time (if W visits v less than i times in total, we let $\mathsf{Z}_i = \infty$). $\mathcal{E}_{\mathsf{bad}}$ is equivalent to that $\mathsf{Z}_{\tau+1} \leq L$.

$\mathsf{Z}_{\tau+1} \leq L$ further implies that for at least $(\tau - \ell)$ $i \in [\tau]$, $\mathsf{Z}_{i+1} - \mathsf{Z}_i \leq \ell$ and $\mathsf{Z}_i < \infty$. In the following we denote this event as $\mathcal{E}_1$ and bounds its probability instead.

For each $i \in [\tau]$, let Y_i be the random variable which takes value 1 if both $Z_i < \infty$ and $\mathsf{Z}_{i+1} - \mathsf{Z}_i \leq \ell$ hold, and 0 otherwise. Letting $\mathsf{Y}_{<i} = (\mathsf{Y}_1, \ldots, \mathsf{Y}_{i-1})$, the following claim is crucial for us.

▷ Claim 19. For every $i \in [\tau]$ and every possible assignments $Y_{<i} \in \{0, 1\}^{i-1}$, we have

$$\mathbb{E}[\mathsf{Y}_i | \mathsf{Y}_{<i} = Y_{<i}] \leq 2/3.$$

Proof. By the Markov property of the random walk, and noting that Y_i is always 0 when $\mathsf{Z}_i = \infty$, we have.

$$\begin{aligned}\mathbb{E}[\mathsf{Y}_i | \mathsf{Y}_{<i} = Y_{<i}] &= \sum_{j=0}^{\infty} \Pr[\mathsf{Z}_i = j | \mathsf{Y}_{<i} = Y_{<i}] \cdot \mathbb{E}[\mathsf{Y}_i | \mathsf{Y}_{<i} = Y_{<i}, \mathsf{Z}_i = j] \\ &= \sum_{j=0}^{\infty} \Pr[\mathsf{Z}_i = j | \mathsf{Y}_{<i} = Y_{<i}] \cdot \mathbb{E}[\mathsf{Y}_i | \mathsf{Z}_i = j].\end{aligned}$$

To further bound the quantity above, recall that the event $\mathsf{Z}_i = j$ means that the random walk W starting from u_{start} visits the light vertex v for the i-th time at W's j-th step, and we have

$$\mathbb{E}[\mathsf{Y}_i | \mathsf{Z}_i = j] = \Pr[\mathsf{Y}_i = 1 | \mathsf{Z}_i = j] = \Pr[\mathsf{Z}_{i+1} \leq j + \ell | \mathsf{Z}_i = j].$$

By the Markov property of the random walk W, $\Pr[\mathsf{Z}_{i+1} \leq j + \ell | \mathsf{Z}_i = j]$ equals the probability that a random walk starting from v revisits v in at most ℓ steps. By the definition of light vertices, we can bound that by $2/3$, which completes the proof. ◁

Then by the Azuma-Hoeffding inequality (Corollary 7),

$$\begin{aligned}\Pr_{\mathsf{W}}[\mathcal{E}_{\mathsf{bad}}] &\leq \Pr_{\mathsf{W}}[\mathcal{E}_1] \\ &= \Pr_{\mathsf{W}}\left[\sum_{i=1}^{\tau} \mathsf{Y}_i \geq (\tau - \ell)\right] \\ &\leq \exp(-\Omega(\tau - \ell - 2/3 \cdot \tau)) \leq \delta/2n,\end{aligned}$$

the last inequality follows from the fact that $\Omega(\tau - \ell - 2/3 \cdot \tau) = \Omega(\gamma)$, $\gamma = c_1 \cdot \log \delta^{-1}$ for a sufficiently large constant c_1, and $\delta \leq 1/n$. ◀

The correctness of the algorithm is finally completed by the following theorem.

▶ **Theorem 20** (Formal version of Theorem 1). *Given a directed graph $G = (V, E)$ with n vertices, $L \in \mathbb{N}$ and $\delta \in (0, 1/n)$. Let $\vec{\mathsf{L}}^{\mathsf{save}}$ and $\mathsf{V}_{\mathsf{full}}$ be the two random variables of the output of $\mathbb{P}_{\mathsf{two\text{-}pass}}(G, L, \delta)$. For every $u_{\mathsf{start}} \in V$, the following hold:*

- *The output distribution of $\mathbb{S}_{\mathsf{two\text{-}pass}}(V, u_{\mathsf{start}}, L, \mathsf{V}_{\mathsf{full}}, \vec{\mathsf{L}}^{\mathsf{save}})$ has statistical distance at most δ from $\mathsf{RW}_L^G(u_{\mathsf{start}})$.*
- *Both of $\mathbb{P}_{\mathsf{two\text{-}pass}}(G, L, \delta)$ and $\mathbb{S}_{\mathsf{two\text{-}pass}}(V, u_{\mathsf{start}}, L, \mathsf{V}_{\mathsf{full}}, \vec{\mathsf{L}}^{\mathsf{save}})$ use at most $O(n \cdot \sqrt{L} \cdot \log \delta^{-1})$ words of space.*

Proof. Note that we can safely assume $L \leq n^2$, since otherwise one can always use $O(n^2)$ words to store all the edges in the graph. In this case, we have that $L \leq n \cdot \sqrt{L}$ and the space for restoring the L-step output walk can be ignored.

Let $\widetilde{\mathsf{V}}_{\mathsf{heavy}} = \mathsf{V}_{\mathsf{full}}$ and $\widetilde{\mathsf{V}}_{\mathsf{light}} = V \setminus \widetilde{\mathsf{V}}_{\mathsf{heavy}}$. Let $\mathcal{E}_{\mathsf{good}}$ be the event that $\widetilde{\mathsf{V}}_{\mathsf{light}} \subseteq V_{\mathsf{light}}$ and $\widetilde{\mathsf{V}}_{\mathsf{heavy}} \subseteq V_{\mathsf{heavy}}$. By Lemma 17, we have that $\Pr[\mathcal{E}_{\mathsf{good}}] \geq 1 - \delta/2$.

Now we condition on the event $\mathcal{E}_{\mathsf{good}}$. In this case, it follows from Lemma 10 that $\mathbb{P}_{\mathsf{one\text{-}pass}}(G, \gamma \cdot \ell, \widetilde{\mathsf{V}}_{\mathsf{heavy}})$ operates correctly (by setting the constant c_2 in Algorithm 1 to be sufficiently large).

By Lemma 18 and a union bound, the probability of $\mathbb{S}_{\mathsf{two\text{-}pass}}(V, u_{\mathsf{start}}, L, \widetilde{\mathsf{V}}_{\mathsf{heavy}}, \vec{\mathsf{L}}^{\mathsf{save}})$ outputs failure is at most $\delta/2$. By Lemma 15, it follows that the statistical distance between the output distribution of $\mathbb{S}_{\mathsf{two\text{-}pass}}(V, u_{\mathsf{start}}, L, \widetilde{\mathsf{V}}_{\mathsf{heavy}}, \vec{\mathsf{L}}^{\mathsf{save}})$ and $\mathsf{RW}_L^G(u_{\mathsf{start}})$ is at most $\delta/2$.

The theorem follows by combing the above with the fact that $\Pr[\mathcal{E}_{\mathsf{good}}] \geq 1 - \delta/2$. ◀

4.4 Two-pass Streaming in the Turnstile Model

Similar to the algorithm in [27], our algorithms can also be easily adapted to work for the *turnstile graph streaming model*, where both insertions and deletions of edges are allowed. Note that our two-pass algorithm $\mathbb{A}_{\mathsf{two\text{-}pass}}$ only accesses the input graph stream via the one-pass preprocessing subroutine $\mathbb{P}_{\mathsf{one\text{-}pass}}$. Hence, it suffices to implement $\mathbb{P}_{\mathsf{one\text{-}pass}}$ in the turnstile model as well. There are two distinct tasks in $\mathbb{P}_{\mathsf{one\text{-}pass}}$: (1) for light vertices, we need to sample their outgoing neighbors with replacement and (2) for heavy vertices, we need to record all their outgoing neighbors.

Uniformly sampling via ℓ_1 sampler. For light vertices, uniformly sampling some out-neighbors from each vertex without replacement can be implemented via the following ℓ_1 sampler in the turnstile model.

▶ **Lemma 21** (ℓ_1 sampler in the turnstile model [24]). *Let $n \in \mathbb{N}$, failure probability $\delta \in (0, 1/2)$ and $f \in \mathbb{R}^n$ be a vector defined by a streaming of updates to its coordinates of the form $f_i \leftarrow f_i + \Delta$, where $\Delta \in \{-1, 1\}$. There is a randomized algorithm which reads the stream, and with probability at most δ it outputs FAIL, otherwise it outputs an index $i \in [n]$ such that:*

$$\Pr(i = j) = \frac{|f_j|}{\|f\|_1} + O(n^{-c}), \qquad \forall j \in [n]$$

where $c \geq 1$ is some arbitrarily large constant.

The space complexity of this algorithm is bounded by $O(\log^2(n) \cdot \log(1/\delta))$ bits in the random oracle model, and $O(\log^2(n) \cdot (\log\log n)^2 \cdot \log(1/\delta))$ bits otherwise.

▶ Remark 22. To get error in the statistical distance also to be at most δ, one can simply set n to be larger than $1/\delta$. And in that case the space complexity can be bounded by $O(\log^4(n/\delta))$.

Recording all outgoing neighbors via ℓ_1 heavy hitter. For heavy vertices, recording all their outgoing neighbors can be implemented using the following ℓ_1 heavy hitter in the turnstile model. (Recall that we assumed our graphs is a simple graph without multiple edges.)

▶ **Lemma 23** (ℓ_1 heavy hitter in the turnstile model [9]). *Let $n, k \in \mathbb{N}$, $\delta \in (0, 0.1)$ and $f \in \mathbb{R}^n$ be a vector defined by a streaming of updates to its coordinates of the form $f_i \leftarrow f_i + \Delta$, where $\Delta \in \{-1, 1\}$. There is an algorithm which reads the stream and returns a subset $L \subset [n]$ such that $i \in L$ for every $i \in [n]$ such that $|f_i| \geq \|f\|_1/k$ and $i \notin L$ for every $i \in [n]$ such that $|f_i| \leq \|f\|_1/2k$. The failure probability is at most δ, and the space complexity is at most $O(k \cdot \log(n) \cdot \log(n/\delta))$.*

Algorithm in the turnstile model. Modifying $\mathbb{P}_{\text{one-pass}}$ with Lemma 21 and Lemma 23, we can generalize our two-pass streaming algorithm to work in two-pass turnstile model.[7]

▶ Remark 24 (Two-pass algorithm in the turnstile model). There exists a streaming algorithm $\mathbb{A}_{\text{turnstile}}$ that given an n-vertex *directed* graph $G = (V, E)$ via a stream of both edge insertions and edge deletions, a starting vertex $u_{\text{start}} \in V$, a non-negative integer L indicating the number of steps to be taken, and an error parameter $\delta \in (0, 1/n)$, satisfies the following conditions:

1. $\mathbb{A}_{\text{turnstile}}$ uses at most $\widetilde{O}(n \cdot \sqrt{L} \cdot \log \delta^{-1})$ space and makes two passes over the input graph G.
2. $\mathbb{A}_{\text{turnstile}}$ samples from some distribution $\mathcal{D}$ over V^{L+1} satisfying $\|\mathcal{D} - \mathsf{RW}_L^G(u_{\text{start}})\|_{\text{TV}} \leq \delta$.

References

1 Reid Andersen, Fan Chung, and Kevin Lang. Using pagerank to locally partition a graph. *Internet Mathematics*, 4(1):35–64, 2007.

2 Reid Andersen and Yuval Peres. Finding sparse cuts locally using evolving sets. In *Proceedings of the forty-first annual ACM symposium on Theory of computing*, pages 235–244, 2009.

[7] In more details, for each light vertex u, we run τ independent copies of the ℓ_1 sampler to obtain τ samples from its outgoing neighbors with replacement. We also let $k = c_2 \cdot \tau \cdot n$ and use the ℓ_1 heavy hitter to record all outgoing neighbors for all heavy vertices in $\widetilde{O}(n \cdot \sqrt{L} \cdot \log(1/\delta))$ space.

3 Sepehr Assadi, Yu Chen, and Sanjeev Khanna. Sublinear algorithms for $(\Delta + 1)$ vertex coloring. In *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 767–786. SIAM, 2019.

4 Sepehr Assadi, Sanjeev Khanna, and Yang Li. Tight bounds for single-pass streaming complexity of the set cover problem. In *48th Annual ACM SIGACT Symposium on Theory of Computing (STOC)*, pages 698–711. Association for Computing Machinery, 2016.

5 Kazuoki Azuma. Weighted sums of certain dependent random variables. *Tohoku Mathematical Journal, Second Series*, 19(3):357–367, 1967.

6 Sergey Brin and Lawrence Page. The anatomy of a large-scale hypertextual web search engine. *Computer networks and ISDN systems*, 30(1-7):107–117, 1998.

7 Amit Chakrabarti, Prantar Ghosh, Andrew McGregor, and Sofya Vorotnikova. Vertex ordering problems in directed graph streams. In *Proceedings of the Fourteenth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1786–1802. SIAM, 2020.

8 Amit Chakrabarti and Anthony Wirth. Incidence geometries and the pass complexity of semi-streaming set cover. In *Proceedings of the twenty-seventh annual ACM-SIAM symposium on Discrete algorithms (SODA)*, pages 1365–1373. SIAM, 2016.

9 Moses Charikar, Kevin Chen, and Martin Farach-Colton. Finding frequent items in data streams. In *International Colloquium on Automata, Languages, and Programming (ICALP)*, pages 693–703. Springer, 2002.

10 Moses Charikar, Liadan O'Callaghan, and Rina Panigrahy. Better streaming algorithms for clustering problems. In *Proceedings of the thirty-fifth annual ACM symposium on Theory of computing*, pages 30–39, 2003.

11 Lijie Chen, Gillat Kol, Dmitry Paramonov, Raghuvansh Saxena, Zhao Song, and Huacheng Yu. Near-optimal two-pass streaming algorithm for sampling random walks over directed graphs. *CoRR*, abs/2102.11251, 2021. `arXiv:2102.11251`.

12 Herman Chernoff. A measure of asymptotic efficiency for tests of a hypothesis based on the sum of observations. *The Annals of Mathematical Statistics*, pages 493–507, 1952.

13 Graham Cormode, Jacques Dark, and Christian Konrad. Independent sets in vertex-arrival streams. In *46th International Colloquium on Automata, Languages, and Programming (ICALP)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2019.

14 Yuval Emek and Adi Rosén. Semi-streaming set cover. In *International Colloquium on Automata, Languages, and Programming (ICALP)*, pages 453–464. Springer, 2014.

15 Joan Feigenbaum, Sampath Kannan, Andrew McGregor, Siddharth Suri, and Jian Zhang. On graph problems in a semi-streaming model. In *International Colloquium on Automata, Languages, and Programming (ICALP)*, pages 531–543. Springer, 2004.

16 Joan Feigenbaum, Sampath Kannan, Andrew McGregor, Siddharth Suri, and Jian Zhang. Graph distances in the data-stream model. *SIAM Journal on Computing*, 38(5):1709–1727, 2009.

17 Buddhima Gamlath, Sagar Kale, Slobodan Mitrovic, and Ola Svensson. Weighted matchings via unweighted augmentations. In *Proceedings of the 2019 ACM Symposium on Principles of Distributed Computing (PODC)*, pages 491–500, 2019.

18 Mohsen Ghaffari, Themis Gouleakis, Christian Konrad, Slobodan Mitrović, and Ronitt Rubinfeld. Improved massively parallel computation algorithms for mis, matching, and vertex cover. In *Proceedings of the 2018 ACM Symposium on Principles of Distributed Computing (PODC)*, pages 129–138, 2018.

19 Ashish Goel, Michael Kapralov, and Sanjeev Khanna. On the communication and streaming complexity of maximum bipartite matching. In *Proceedings of the twenty-third annual ACM-SIAM symposium on Discrete Algorithms (SODA)*, pages 468–485. SIAM, 2012.

20 Venkatesan Guruswami and Krzysztof Onak. Superlinear lower bounds for multipass graph processing. *Algorithmica*, 76(3):654–683, 2016.

21 Sariel Har-Peled, Piotr Indyk, Sepideh Mahabadi, and Ali Vakilian. Towards tight bounds for the streaming set cover problem. In *Proceedings of the 35th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems (PODS)*, pages 371–383, 2016.

22 Monika Rauch Henzinger, Prabhakar Raghavan, and Sridhar Rajagopalan. Computing on data streams. *External memory algorithms*, 50:107–118, 1998.

23 Wassily Hoeffding. Probability inequalities for sums of bounded random variables. In *The Collected Works of Wassily Hoeffding*, pages 409–426. Springer, 1994.

24 Rajesh Jayaram and David P. Woodruff. Perfect lp sampling in a data stream. In Mikkel Thorup, editor, *59th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2018, Paris, France, October 7-9, 2018*, pages 544–555. IEEE Computer Society, 2018.

25 Mark Jerrum and Alistair Sinclair. Approximating the permanent. *SIAM journal on computing*, 18(6):1149–1178, 1989.

26 Mark R Jerrum, Leslie G Valiant, and Vijay V Vazirani. Random generation of combinatorial structures from a uniform distribution. *Theoretical computer science*, 43:169–188, 1986.

27 Ce Jin. Simulating random walks on graphs in the streaming model. In Avrim Blum, editor, *10th Innovations in Theoretical Computer Science Conference, ITCS 2019, January 10-12, 2019, San Diego, California, USA*, volume 124 of *LIPIcs*, pages 46:1–46:15. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019. doi:10.4230/LIPIcs.ITCS.2019.46.

28 Michael Kapralov. Better bounds for matchings in the streaming model. In *Proceedings of the twenty-fourth annual ACM-SIAM symposium on Discrete algorithms (SODA)*, pages 1679–1697. SIAM, 2013.

29 Andrew McGregor. Finding graph matchings in data streams. In *Approximation, Randomization and Combinatorial Optimization. Algorithms and Techniques*, pages 170–181. Springer, 2005.

30 Sagnik Mukhopadhyay and Danupon Nanongkai. Weighted min-cut: sequential, cut-query, and streaming algorithms. In *Proceedings of the 52nd Annual ACM SIGACT Symposium on Theory of Computing (STOC)*, pages 496–509, 2020.

31 Noam Nisan and Avi Wigderson. Rounds in communication complexity revisited. In *Proceedings of the 23rd Annual ACM Symposium on Theory of Computing, May 5-8, 1991, New Orleans, Louisiana, USA*, pages 419–429. ACM, 1991.

32 Omer Reingold. Undirected connectivity in log-space. *Journal of the ACM (JACM)*, 55(4):1–24, 2008.

33 Aviad Rubinstein, Tselil Schramm, and Seth Matthew Weinberg. Computing exact minimum cuts without knowing the graph. In *9th Innovations in Theoretical Computer Science (ITCS)*, page 39. Schloss Dagstuhl-Leibniz-Zentrum fur Informatik GmbH, Dagstuhl Publishing, 2018.

34 Atish Das Sarma, Sreenivas Gollapudi, and Rina Panigrahy. Estimating pagerank on graph streams. *J. ACM*, 58(3):13:1–13:19, 2011. doi:10.1145/1970392.1970397.

35 Aaron Schild. An almost-linear time algorithm for uniform random spanning tree generation. In *Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing (STOC)*, pages 214–227, 2018.

36 Daniel A Spielman and Shang-Hua Teng. A local clustering algorithm for massive graphs and its application to nearly linear time graph partitioning. *SIAM Journal on computing*, 42(1):1–26, 2013.

37 Jeffrey Scott Vitter. Random sampling with a reservoir. *ACM Trans. Math. Softw.*, 11(1):37–57, 1985. doi:10.1145/3147.3165.

38 Andrew Chi-Chin Yao. Probabilistic computations: Toward a unified measure of complexity. In *18th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 222–227. IEEE Computer Society, 1977.

39 Mariano Zelke. Intractability of min-and max-cut in streaming graphs. *Information Processing Letters*, 111(3):145–150, 2011.

Sublinear Time Hypergraph Sparsification via Cut and Edge Sampling Queries

Yu Chen
Department of Computer and Information Science,
University of Pennsylvania, Philadelphia, PA, USA

Sanjeev Khanna
Department of Computer and Information Science,
University of Pennsylvania, Philadelphia, PA, USA

Ansh Nagda
University of Washington, Seattle, WA, USA

Abstract

The problem of sparsifying a graph or a hypergraph while approximately preserving its cut structure has been extensively studied and has many applications. In a seminal work, Benczúr and Karger (1996) showed that given any n-vertex undirected weighted graph G and a parameter $\varepsilon \in (0,1)$, there is a near-linear time algorithm that outputs a weighted subgraph G' of G of size $\tilde{O}(n/\varepsilon^2)$ such that the weight of every cut in G is preserved to within a $(1\pm\varepsilon)$-factor in G'. The graph G' is referred to as a $(1\pm\varepsilon)$-*approximate cut sparsifier* of G. Subsequent recent work has obtained a similar result for the more general problem of hypergraph cut sparsifiers. However, all known sparsification algorithms require $\Omega(n+m)$ time where n denotes the number of vertices and m denotes the number of hyperedges in the hypergraph. Since m can be exponentially large in n, a natural question is if it is possible to create a hypergraph cut sparsifier in time polynomial in n, *independent of the number of edges*. We resolve this question in the affirmative, giving the first sublinear time algorithm for this problem, given appropriate query access to the hypergraph.

Specifically, we design an algorithm that constructs a $(1\pm\varepsilon)$-approximate cut sparsifier of a hypergraph $H(V,E)$ in polynomial time in n, independent of the number of hyperedges, when given access to the hypergraph using the following two queries:

1. given any cut $(S,\bar{S})$, return the size $|\delta_E(S)|$ (*cut value queries*); and
2. given any cut $(S,\bar{S})$, return a uniformly at random edge crossing the cut (*cut edge sample queries*).

Our algorithm outputs a sparsifier with $\tilde{O}(n/\varepsilon^2)$ edges, which is essentially optimal. We then extend our results to show that cut value and cut edge sample queries can also be used to construct hypergraph *spectral sparsifiers* in poly(n) time, independent of the number of hyperedges.

We complement the algorithmic results above by showing that any algorithm that has access to only one of the above two types of queries can not give a hypergraph cut sparsifier in time that is polynomial in n. Finally, we show that our algorithmic results also hold if we replace the cut edge sample queries with a *pair neighbor sample query* that for any pair of vertices, returns a random edge incident on them. In contrast, we show that having access only to cut value queries and queries that return a random edge incident on a given single vertex, is not sufficient.

2012 ACM Subject Classification Theory of computation → Sketching and sampling; Theory of computation → Streaming, sublinear and near linear time algorithms; Theory of computation → Sparsification and spanners

Keywords and phrases hypergraphs, graph sparsification, cut queries

Digital Object Identifier 10.4230/LIPIcs.ICALP.2021.53

Category Track A: Algorithms, Complexity and Games

Related Version *Full Version*: `https://arxiv.org/pdf/2106.10386.pdf`

Funding This research was partially supported by NSF awards CCF-1763514, CCF-1617851, CCF-1934876, and CCF-2008305.

Acknowledgements We thanks the anonymous referees for their valuable comments on an earlier version of this paper.

1 Introduction

In many applications, the underlying graphs are too large to fit in the main memory, and one typically builds a compressed representation that preserves relevant properties of the graph. Cuts in graphs are a fundamental object of study, and play a central role in the study of graph algorithms. Consequently, the problem of *sparsifying* a graph while approximately preserving its cut structure has been extensively studied (see, for instance, [17, 6, 18, 25, 1, 2, 13, 5, 3, 21, 15, 4, 16], and references therein). A cut-preserving sparsifier not only reduces the space requirement for any computation, but it can also reduce the time complexity of solving many fundamental cut, flow, and matching problems as one can now run the algorithms on the sparsifier which may contain far fewer edges. In a seminal work, Benczúr and Karger [6] showed that given any n-vertex undirected weighted graph G and a parameter $\varepsilon \in (0,1)$, there is a near-linear time algorithm that outputs a weighted subgraph G' of G of size $\tilde{O}(n/\varepsilon^2)$ such that the weight of every cut in G is preserved to within a multiplicative $(1\pm\varepsilon)$-factor in G'. The graph G' is referred to as the $(1\pm\varepsilon)$-*approximate cut sparsifier* of G.

In this work, we consider the problem of cut sparsification for hypergraphs. A hypergraph $H(V,E)$ consists of a vertex set V and a set E of hyperedges where each edge $e \in E$ is a subset of vertices. The rank of a hypergraph is the size of the largest edge in the hypergraph, that is, $\max_{e\in E}|e|$. Hypergraphs are a natural generalization of graphs and many applications require estimating cuts in hypergraphs (see, for instance, [8, 9, 14, 26]). Note that unlike graphs, an n-vertex hypergraph may contain exponentially many (in n) hyperedges. It is thus natural to ask if cut-preserving sparsifiers in the spirit of graph sparsifiers can also be created for hypergraphs as this would allow algorithmic applications to work with hypergraphs whose size is polynomially bounded in n.

Kogan and Krauthgamer [19] initiated a study of this basic question and showed that given any weighted hypergraph H, there is an $O(mn^2)$ time algorithm to find a $(1\pm\varepsilon)$-approximate cut sparsifier of H of size $\tilde{O}(\frac{nr}{\varepsilon^2})$ where r denotes the rank of the hypergraph. Similar to the case of graphs, the *size* of a hypergraph sparsifier refers to the number of edges in the sparsifier. Since r can be as large as n, in general, this gives a hypergraph cut sparsifier of size $\tilde{O}(n^2/\varepsilon^2)$, which is a factor of n larger than the Benczúr-Karger bound for graphs. Chekuri and Xu [10] designed a more efficient algorithm for building a hypergraph sparsifier. They gave a near-linear time algorithm in the total representation size (sum of the sizes of all hyperedges) to construct a hypergraph sparsifier of size $\tilde{O}(nr^2/\varepsilon^2)$ in hypergraphs of rank r, thus speeding up the run-time obtained in the work of Kogan and Krauthgamer [19] by at least a factor of n, but at the expense of an increased sparsifier size. Until recently, it was an open question if the Benczúr-Karger bound is also achievable on hypergraphs, that is, do there exist hypergraph sparsifiers with $\tilde{O}(n/\varepsilon^2)$ edges. In a very recent work [11], we were able to resolve this question in the affirmative by giving a $\tilde{O}(mn + n^{10}/\varepsilon^7)$ time algorithm for creating hypergraph sparsifiers of size $\tilde{O}(n/\varepsilon^2)$.

All known results for creating poly(n) size hypergraph sparsifiers have at least one thing in common – the running time of these algorithms has at least a linear dependence on both n and m. All known algorithms are essentially based on sampling edges in proportion to their importance, and they primarily differ in how the importance of an edge is defined and computed. A linear dependence on n is unavoidable since the hypergraph size itself is $\Omega(n)$.

However, since the number of hyperedges can be exponential in n, even a linear dependence on m means that the running time of a sparsification algorithm can be exponentially large in n in the worst-case. This motivates the following natural question: is there an algorithm for building a hypergraph sparsifier that runs in time that is polynomial in n? In other words, is there a sublinear time algorithm for creating hypergraph sparsifiers?

In order to tackle this question, we need to first define our model for accessing the input hypergraph $H = (V, E)$. The most basic requirement is to have the ability to efficiently evaluate the size or weight of any cut in a given hypergraph. We assume here access to a *cut value oracle*, denoted as O_{value}, which takes as input a cut $C = (S, \bar{S})$, returns the size of the cut $|\delta_H(S)|$. This is akin to the standard assumption in submodular function minimization, namely, the algorithm has an oracle access to the value of the submodular function on any set S since the cut function is a submodular function. However, as it turns out, it is easy to show that the access to a cut value oracle is provably not sufficient to construct a sparsifier, regardless of the time allowed as this oracle can not differentiate between hypergraphs where all edges have size 2 from hypergraphs where all edges have size 3[1]. So we also need a mechanism for accessing edges of the underlying graph. We thus introduce a second oracle, referred to as the *cut edge oracle*, denoted as O_{edge}, which takes as input a cut $C = (S, \bar{S})$, returns a random edge crossing the cut. Given access to both these oracles, we are indeed able to solve the problem of hypergraph sparsification in polynomial time in n.

▶ **Theorem 1.** *Suppose we are given an unweighted hypergraph $H = (V, E)$ that can be accessed using the oracles O_{value} and O_{edge}. Then for any $0 < \varepsilon < 1$, a $(1 \pm \varepsilon)$-approximate sparsifier with $\tilde{O}(n/\varepsilon^2)$ hyperedges can be constructed in $O(n^{10}/\varepsilon^7)$ time, independent of the number of hyperedges.*

At a high-level, graph and hypergraph sparsification algorithms work by estimating the importance of each edge in preserving cut sizes, and then sampling edges with probability proportional to their importance and assigning them an appropriately scaled weight. The main technical challenge in proving the above theorem is that the cut value oracle on the original graph cannot be used to estimate cut sizes in the vertex-induced subgraphs of the original graph – a step that is implicit in determining importance of edges in preserving the cut structure. This issue does not arise in normal graphs where each edge contains 2 vertices, and the cut value oracle on the original graph indeed suffices to recover cut values in any induced subgraph. But once we consider hypergraphs with even edges of size 3, it is easy to show that the cut value oracle on the original graph can not distinguish between induced subgraphs that have minimum cut value 0 and induced subgraphs where the minimum cut value is polynomially large. We refer the reader to Section 3 for a more detailed discussion of this. We get around this issue by introducing for any subset of vertices X, a weaker notion of *pseudo cut size* for approximating cut sizes in the subgraph induced by X. The new cut size function remains submodular, and we show that it suffices to approximate the importance of each edge to within a factor n of its true importance. We then use the O_{edge} oracle to sample edges in accordance with their approximate importance. The resulting sparsifier H' has poly(n) edges which we further sparsify to $\tilde{O}(n/\varepsilon^2)$ edges in poly(n) time by applying the result of [11] to H'.

[1] For instance, the cut value oracle can not distinguish between a copy of K_4 and the hypergraph that contains all possible hyperedges of size 3 on 4 vertices. Note that this does not rule out the possibility of efficiently constructing a data structure/sketch that can be used to answer cut queries. Our focus in this paper, however, is on constructing sparsifiers, namely, sparse subgraphs of the original graph that preserve all cuts.

We complement the algorithmic result above by showing that just like the oracle O_{value} alone is not sufficient to achieve the result above, the oracle O_{edge} alone is also not sufficient to create a poly(n) size hypergraph sparsifier in poly(n) time.

▶ **Theorem 2.** *There is no polynomial time randomized algorithm that can use O_{edge} queries alone to construct a $(1 \pm \varepsilon)$-approximate sparsifier of an underlying hypergraph H with probability better than $o(1)$.*

One may wonder if the oracle O_{edge} can be replaced with another access oracle that is used in sublinear algorithms for standard graphs, namely, ability to access the i_{th} neighbor of a vertex v for any integer i that is at most the degree of v. It is easy to see that this is essentially same as the ability to access a random edge incident on a vertex v. We can generalize this idea to the setting of hypergraphs as follows. A neighbor query oracle in a hypergraph takes as input a set $S \subseteq V$, and returns a random edge that contains all vertices in S if there is such an edge, and returns NIL if there is no edge. We say that a neighbor query is a *single vertex neighbor query* if $|S| = 1$, and it is a *vertex pair neighbor query* if $|S| = 2$. We denote the oracles that answer a single vertex neighbor query and a vertex pair neighbor query as O^1_{nbr} and O^2_{nbr} respectively. We next show that the oracle O_{edge} can be replaced with the oracle O^2_{nbr}, to obtain an alternate poly(n) time implementation of the result in Theorem 1.

▶ **Theorem 3.** *Given an unweighted hypergraph $H = (V, E)$, suppose the algorithm can access the hypergraph using O_{value} and O^2_{nbr}, then for any $0 < \varepsilon < 1$, a $(1 \pm \varepsilon)$-approximate sparsifier with $\tilde{O}(n/\varepsilon^2)$ hyperedges can be constructed in $O(n^1 0/\varepsilon^7)$ time in n, independent of the number of hyperedges.*

In contrast to the result above, we show any algorithm that has access only to oracles O_{value} and O^1_{nbr}, requires exponentially many queries in the worst-case to construct a poly(n) size sparsifier.

▶ **Theorem 4.** *There is no polynomial time randomized algorithm that can use O_{value} and O^1_{nbr} queries alone to construct a $(1 \pm \varepsilon)$-approximate sparsifier of an underlying hypergraph H with probability better than $o(1)$.*

Hypergraph Spectral Sparsification. We also consider the problem of hypergraph spectral sparsification, a notion that strengthens cut sparsification. A $(1 \pm \varepsilon)$-approximate *spectral sparsifier* of a graph $G(V, E)$ is a weighted graph $G'(V, E')$ such that for every vector $x \in \mathbb{R}^n$, we have

$$|x^T L_{G'}\ x - x^T L_G\ x| \leq \varepsilon(x^T L_G\ x),$$

where L_G and $L_{G'}$ denote the Laplacian matrices of G and G', respectively. To see that the notion of spectral sparsifier only strengthens the notion of a cut sparsifier, observe that the cut sparsification requirement for any cut $(S, \bar{S})$ is captured by the definition above when we choose x to be the 0/1-indicator vector of the set S. Batson, Spielman, and Srivastava [5] gave a polynomial-time algorithm that for every graph G, gives a weighted graph G' with $O(n/\varepsilon^2)$ edges such that G' is a $(1 \pm \varepsilon)$-approximate spectral sparsifier of G. Subsequently, Lee and Sun [21] gave an $O(m/\varepsilon^{O(1)})$ time algorithm to construct a spectral graph sparsifier with $O(n/\varepsilon^2)$ edges.

The notion of spectral sparsification can be extended to hypergraphs [22, 27] as follows. The *Laplacian* L_H of a hypergraph H is a function $\mathbb{R}^n \to \mathbb{R}^n$, such that for any n-dimensional vector x, we have

$$x^T L_H(x) = \sum_{e \in E} w(e) \max_{u,v \in e} (x(u) - x(v))^2.$$

Given a weighted hypergraph H with n vertices, a $(1 \pm \varepsilon)$-spectral sparsifier H' is a subgraph of H such that for any n-dimensional vector x, we have

$$(1 - \varepsilon) x^T L_{H'}(x) \leq x^T L_H(x) \leq (1 + \varepsilon) x^T L_{H'}(x).$$

Soma and Yoshida [23] give a polynomial-time algorithm that outputs a weighted spectral sparsifier with $\tilde{O}(n^3)$ hyperedges. The algorithm of [23] is also based on sampling edges based on a suitable notion of importance, and in their work, the importance of a hyperedge e is measured by $\min_{u,v \in e} |E(\{u, v\})|$ where $E(\{u, v\})$ is the set of edges that contains both u and v. If we assume access to the underlying hypergraph using O^2_{nbr} queries, then we can sample a random hyperedge in $E(\{u, v\})$ for any pair of vertices u and v, which makes it in turn straightforward to simulate the algorithm of [23].

▶ **Theorem 5.** *Given an unweighted hypergraph $H = (V, E)$, suppose the algorithm can access the hypergraph using O_{value} and O^2_{nbr} queries. Then for any $0 < \varepsilon < 1$, a $(1 \pm \varepsilon)$-spectral sparsifier with $\tilde{O}(n^3/\varepsilon^2)$ hyperedges can be constructed in polynomial time in n, independent of the number of hyperedges.*

The more interesting case is when we can only access the hypergraph using O_{value} and O_{edge} queries. It is now provably impossible to get a handle on $|E(\{u, v\})|$ using only polynomially many queries, and thus there is no direct way to simulate the algorithm in [23]. Recently, Bansal, Svensson, and Trevisan [4] designed another hypergraph spectral sparsification algorithm that in polynomial-time algorithm creates a weighted spectral sparsifier with $\tilde{O}(nr^3)$ hyperedges; here r denotes the maximum arity of any hyperedge. Unlike the algorithm of [23], their measure of importance of a hyperedge is derived from an auxiliary standard graph created by converting every hyperedge into a clique. The importance of a hyperedge is then given by the maximum effective resistance among all the edges in the clique associated with that hyperedge. We show that we can simulate this process using only $\text{poly}(n)$ many O_{value} and O_{edge} queries.

▶ **Theorem 6.** *Given an unweighted hypergraph $H = (V, E)$, suppose the algorithm can access the hypergraph using only O_{value} and O_{edge} queries. Then for any $0 < \varepsilon < 1$, a $(1 \pm \varepsilon)$-spectral sparsifier with $\tilde{O}(n^3/\varepsilon^2)$ hyperedges can be constructed in polynomial time in n, independent of the number of hyperedges.*

The idea is to associate the strength of a hyperedge with the resistance of the edges inside the clique associated with this hyperedge. We first show that in any standard graph the effective resistance of an edge f is at most n/k_f where k_f denote the strength of the edge f. We then show that the strength of a hyperedge e is at most the strength of any edge in the clique associated with the hyperedge e. With this pair of relationships, we are able to simulate the algorithm in [4] by the algorithm in Theorem 1, except that we will sample somewhat larger number of hyperedges to meet the sampling probability requirement for the sparsification algorithm in [4].

Note that unlike hypergraph cut sparsification which is now well-understood [11], the problem of determining the optimal size of a hypergraph spectral sparsifier is still open. However, note that any further improvements on the size of hypergraph spectral sparsifiers can be used in a black-box manner with our sparsification algorithms – we can simply apply the improved sparsification algorithm to the sparsifier generated by our algorithm.

Finally, we note that since any spectral sparsifier is also a cut sparsifier, the lower bounds in Theorem 2 and Theorem 4 also hold for spectral sparsifiers.

Extension to weighted hypergraphs. All our algorithmic results stated thus far are for sparsifying unweighted hypergraphs. We note that all algorithms can be extended in a straightforward manner to sparsifying weighted hypergraphs assuming natural weighted versions of the access oracles used in proving these results. Specifically, it suffices to assume that the oracle O_{value} returns the *weight* of a cut, and the oracles O_{edge} and O^2_{nbr} return an edge with probability *proportional to its weight*.

Organization. We set up our notation and state some useful background results in Section 2. In Section 3, we give a poly(n) time algorithm for creating a $\tilde{O}(n/\varepsilon^2)$ size sparsifier using the cut value and cut edge sample oracles, proving Theorem 1. In Section 4, we give a poly(n) time algorithm for creating a $\tilde{O}(n/\varepsilon^2)$ size sparsifier using the cut value and the vertex pair neighbor oracles, proving Theorem 3. Then in Section 5, we present poly(n) time algorithms for hypergraph spectral sparsification proving Theorems 5 and 6. We show in Section 6 that any weakening of the oracles assumed in Theorems 1 and 3 necessarily requires worst-case exponential time for creating a poly(n) size hypergraph cut sparsifier, proving Theorems 2 and 4. In Section 7, we briefly describe how the results of Theorems 1 and 3 can be extended to weighted hypergraphs. Finally, we conclude in Section 8 with some directions for future work.

2 Preliminaries

2.1 Notation

Given an integer n and a probability p, let $B(n,p)$ be the Bernoulli distribution with n trials where each trial succeeds with probability p. Suppose a set has n elements, and given a probability p. Then the following process will sample each element in the set with probability p: we first sample a number $N \sim B(n,p)$, then randomly sample N elements in the set.

Given any weight function $w : S \to \mathbb{R}_{\geq 0}$, we extend it to also be a function on subsets of S so that $w(S') = \sum_{e \in S'} w(e)$ for $S' \subseteq S$.

Given a graph $H = (V, E)$ and a subset of vertices $V' \subseteq V$, we define $G[V']$ to be the weighted subgraph/subhypergraph of G induced by the vertices in V'. We identify an edge/hyperedge e with the set of vertices that are contained in e. A hyperedge e has *rank* k *if* $|e| = k$. *The* rank of a hypergarph is the maximum rank of its hyperedges.

Given a weighted hypergraph $H = (V, E, w)$ and a cut $C = (S, \bar{S})$, we say an edge e crosses the cut if $e \cap S \neq \emptyset$ and $e \cap \bar{S} \neq \emptyset$. We denote by $\delta_H(S)$ the set of the edges crossing the cut C in H. By definition, $|\delta_H(S)|$ is the number of edges crossing C and $w(\delta_H(S))$ is the weight of C. For any $\varepsilon > 0$, a $(1 \pm \varepsilon)$-*approximate cut sparsifier* of H is a hypergraph $H' = (V, E', w')$ with $E' \subseteq E$ such that

$$\forall S \subseteq V, \quad |w'(\delta_{H'}(S)) - w(\delta_H(S))| \leq \varepsilon w(\delta_H(S)).$$

Given a weighted normal graph G with n vertices, the *Laplacian* $L_G \in \mathbb{R}^{n\times n}$ is defined as follows: for any u, $L_G(u,u)$ is the total weight edges incident on u, and for any $u \neq v$, $L_G(u,v)$ is minus the weight of edges between u and v. For any n-dimensional vector $x \in \mathbb{R}^n$, $x^T L_G x = \sum_{(u,v)\in e} w(e)(x(u)-x(v))^2$. A $(1\pm\varepsilon)$-spectral sparsifier G' is a subgraph of G such that for any vector x,

$$(1-\varepsilon)x^T L_{G'} x \leq x^T L_G x \leq (1+\varepsilon)x^T L_{G'} x.$$

The notion of spectral sparsification can be extended to hypergraphs as follows. The *Laplacian* L_H of a hypergraph H is a function $\mathbb{R}^n \to \mathbb{R}^n$, such that for any n-dimensional vector x, we have

$$x^T L_H(x) = \sum_{e\in E} w(e) \max_{u,v\in e}(x(u)-x(v))^2.$$

Given a weighted hypergraph H with n vertices, a $(1\pm\varepsilon)$-spectral sparsifier H' is a subgraph of H such that for any n-dimensional vector x, we have

$$(1-\varepsilon)x^T L_{H'}(x) \leq x^T L_H(x) \leq (1+\varepsilon)x^T L_{H'}(x).$$

2.2 Hypergraph Cut Sparsification

In this section, we review some important concepts and results of hypergraph cut sparsification.

Given a weighted hypergraph $H=(V,E,w)$, a *k-strong component* of H is a maximal induced subgraph of G that has minimum cut at least k. For any edge e, the *strength* of e, denoted by k_e, in H is the maximum value of k such that e is *fully* contained in a k-strong component of H. Alternatively, the strength of an edge $e \in E$ is the largest minimum cut size among all induced subgraphs $H[X]$ that contain e, where X ranges over all subsets of V. The sum of w_e/k_e is at most $n-1$.

▶ **Lemma 7** ([19]). *Given a weighted hypergraph $H=(V,E,w)$, we have $\sum_{e\in E} w_e/k_e \leq n-1$.*

Benczúr and Karger [6, 7] showed that when we are dealing with a normal graph where each edge contains exactly two vertices, if we sample each edge e independently with probability $p_e = O(\log n/\varepsilon^2 k_e)$, and give it weight $1/p_e$ if sampled, then the resulting graph will be a $(1\pm\varepsilon)$-approximate cut sparsifier of G with high probability.

Kogan and Krauthgamer [19] generalized this approach to hypergraphs. They showed that given a hypergraph $H=(V,E)$ with rank r, for each hyperedge e, if we sample e with probability $p_e = O((\log n + r)/\varepsilon^2 k_e)$, and has weight $1/p_e$ if get sampled, the resulting graph will be a cut sparsifier of H with high probability.

▶ **Theorem 8** ([19]). *Let H be a hypergraph with rank r, and let $\varepsilon > 0$ be an error parameter. Consider the hypergraph H' obtained by sampling each hyperedge e in H independently with probability $p_e = \min\{1, \frac{3((d+2)\log n + r)}{k_e \varepsilon^2}\}$, giving it weight $1/p_e$ if included. Then with probability at least $1 - O(n^{-d})$*

1. *the hypergraph H' has $O(\frac{n}{\varepsilon^2}(r+\log n))$ edges, and*
2. *H' is a $(1\pm\varepsilon)$-approximate cut sparsifier of H.*

In fact, if for each edge e, the sampling proability p_e is at least $\frac{3((d+2)\log n + r)}{k_e\varepsilon^2}$, then the resulting graph is still a $(1\pm\varepsilon)$-approximate cut sparsifier. This is because in the proof of Theorem 8, the authors showed that each cut has a very small probability that the cut size

in H' is not within a factor of $(1 \pm \varepsilon)$ of the cut size in H, and the probability that there is no such cut is also very small by taking a union bound over all possible cuts. To bound the probability that a cut has a similar size in H' and H, the authors use the Chernoff bound. However, we can also use the following concentration bound to prove the same result.

▶ **Lemma 9** (Theorem 2.2 in [12]). *Let $\{x_1, \ldots, x_k\}$ be a set of random variables, such that for $1 \leq i \leq k$, each x_i independently takes value $1/p_i$ with probability p_i and 0 otherwise, for some $p_i \in [0, 1]$. Then for all $N \geq k$ and $\varepsilon \in (0, 1]$,*

$$\Pr\left(\left|\sum_{i\in[k]} x_i - k\right| \geq \varepsilon N\right) \leq 2e^{-0.38\varepsilon^2 \cdot \min_i p_i \cdot N}$$

So if we replace the probability of sampling an edge e with $q_e \geq p_e$, the concentration bound in Lemma 9 still holds. In other words, if we sample according to q_e then the probability that each cut in the sampled graph H' has size close to the cut size in H is at least as large as the probability when edges are sampled according to p_e.

▶ **Lemma 10.** *Let H be a hypergraph with rank r, and let $\varepsilon > 0$ be an error parameter. Consider the hypergraph H' obtained by sampling each hyperedge e in H independently with probability $p_e \geq \min\{1, \frac{3((d+2)\log n + r)}{k_e \varepsilon^2}\}$, giving it weight $1/p_e$ if included. Then with probability at least $1 - O(n^{-d})$, H' is a $(1 \pm \varepsilon)$-approximate cut sparsifier of H.*

Recently, [11] showed that for every n-vertex hypergraph, there is a $(1 \pm \varepsilon)$-approximate cut sparsifier with $\tilde{O}(n)$ edges. Moreover, this sparsifier can be constructed in polynomial time in the number of vertices and the number of hyperedges.

▶ **Theorem 11** ([11]). *Given a weighted hypergraph H, for any $0 < \varepsilon < 1$, there exists an randomized algorithm that constructs a $(1 \pm \varepsilon)$-approximate cut sparsifier of H with $O(\frac{n \log n}{\varepsilon^2})$ hyperedges in $O(mn + n^{10}/\varepsilon^7)$ time with high probability.*

2.3 Hypergraph Spectral Sparsification

In this section, we review some important concepts and results on spectral sparsification in both normal graphs and hypergraphs.

Given a weighted normal graph G, the *effective resistance* r_e of an edge $e = (u, v)$ is defined to be the electrical effective resistance between u and v if we view G as a electrical network on n nodes in which each edge e corresponds to a resistor with conductance $w(e)$.

Spielman and Srivastava [24] showed that given an unweighted graph G, if we sample each edge e independently with probability $p_e = O(r_e \log n/\varepsilon^2)$ and give it a weight of w_e/p_e if sampled, then the resulting graph is a $(1 \pm \varepsilon)$-spectral sparsifier.

In the case of hypergraphs, Soma and Yoshida [23] showed that if we sample each hyperedge e with probability proportional to $n \log n/(\varepsilon^2 \min_{u,v\in e} |E(\{u, v\})|)$ where $E(\{u, v\})$ is the set of hyperedges that contains both u and v, then the resulting graph is a $(1 \pm \varepsilon)$-spectral sparsifier.

▶ **Theorem 12** ([23]). *Given an unweighted hypergraph H, if we sample each edge e with probability $p_e = \min\{1, \frac{Cn \log n}{\varepsilon^2 \min_{u,v\in e} |E(\{u,v\})|}\}$ where C is a universal constant, and give weight $1/p_e$ if sampled, then the resulting hypergraph H' is a $(1 \pm \varepsilon)$-spectral sparsifier of H with high probability. The sparsifier has $\tilde{O}(n^3/\varepsilon^2)$ hyperedges. The running time of this algorithm is $\tilde{O}(mn^2 + m + n^3/\varepsilon^2)$.*

Bansal et al. [4] take a different approach to hypergraph spectral sparsification. For any hypergraph H, define the auxiliary graph G_H as a normal graph that is obtained by, for each hyperedge e, transforming e into a clique F_e over the vertices in e. For any hyperedge e, we now define $r_e = \max_{f \in F_e} r_f$. Bansal et al. showed that if we sample each hyperedge e with probability proportional to $r^4 r_e \log n/\varepsilon^2$ where r is the maximum size of any hyperedge in H, then the resulting graph is a $(1 \pm \varepsilon)$-spectral sparsifier.

▶ **Theorem 13** ([4]). *Given an unweighted hypergraph H, if we sample each edge e with probability $p_e = \min\{1, \frac{Cr^4 r_e \log n}{\varepsilon^2}\}$ where C is a universal constant, and assign it weight $1/p_e$ if sampled, then the resulting hypergraph H' is a $(1 \pm \varepsilon)$-spectral sparsifier of H with high probability. The sparsifier has $\tilde{O}(nr^3/\varepsilon^2)$ hyperedges.*

Similar to the case of Theorem 8, Theorem 12 and Theorem 13 both work when we sample each edge with a probability $q_e \geq p_e$ instead of p_e and give weight $1/q_e$ if sampled.

2.4 Minimizing a Submodular Function Using Value Queries

A set function $f : 2^\Omega \to \mathbb{R}$ is a *submodular function* if for every $S, T \subseteq \Omega$, $f(S) + f(T) \geq f(S \cup T) + f(S \cap T)$. Given a graph/hypergraph, for any vertex set S, the cut function $f(S)$, defined as the weight of edges crossing cut $(S, \bar{S})$ is easily shown to be submodular. There is an algorithm that for any submodular function f finds a set S that minimizes the value of function f using $\tilde{O}(n^3)$ value queries and in $\tilde{O}(n^4)$ time.

▶ **Theorem 14** ([20]). *There is an algorithm for submodular function minimization with $O(n^3 \log^2 n)$ value queries and $O(n^4 \log^{O(1)} n)$ time where n is the size of the ground set.*

3 Sublinear Time Cut Sparsification with Cut Size and Cut Edge Sampling Queries

We now present an algorithm that, given access to a hypergraph H through cut size queries (oracle O_{value}) and queries to sample a random edge crossing a cut (oracle O_{edge}), outputs a $(1 \pm \varepsilon)$-approximate sparsifier with $\tilde{O}(n/\varepsilon^2)$ hyperedges in poly(n) time. At a high-level, our algorithm will first create a poly(n) size sparsifier H_1 by indirectly implementing the algorithm underlying Theorem 8. We then use the algorithm in Theorem 11 to construct a sparsifier H_2 of H_1 which has $\tilde{O}(n/\varepsilon^2)$ hyperedges. By the definition of cut sparsifier, H_2 is also a cut sparsifier of H. We can thus focus on the construction of the sparsifier H_1.

The primary challenge in simulating the algorithm of Theorem 8 is to sample edges according to their strength with a small number of queries. Consider the following recursive algorithm. We start with the graph H, and then at each step, we find the minimum cut of the connected graph, and sample $\Theta((r + \log n)/\varepsilon^2)$ edges from the cut. We then recursively execute this algorithm on each side of the cut. Algorithm 1 gives an implementation of this idea.

It is easy to see that this algorithm samples each edge independently, and that the sampling probability is at least that of Kogan-Krauthgamer in Theorem 8. The challenge is that unlike cut queries in the normal graph, it is hard to compute the cut size in an induced subgraph of a hypergraph using only cut queries on the original graph, which is crucial as the algorithm proceeds recursively.

We first note that this task is straightforward to do in graphs where each edge has exactly two vertices. For any two disjoint subsets of vertices S, T, the number of edges in $S \times T$ is $\frac{1}{2}(|\delta(S)| + |\delta(T)| - |\delta(S \cup T)|)$.

Algorithm 1 Sampling edges with probability proportional to their strength.

Input : A subset of vertices $V' \subseteq V$.

Let $(S, \bar{S})$ be a minimum cut of the induced graph $G[V']$;
Let c be the number of edges crossing $(S, \bar{S})$ in $G[V']$;
Sample an integer $N \sim B(c, \frac{10(\log n + r)}{\epsilon^2 c})$;
Sample N edges from $\delta_{G[V']}(S)$ uniformly at random, and assign each of them a weight of $\frac{\epsilon^2 c}{10(\log n + r)}$;
Delete all edges in $\delta_{G[V']}(S)$ and recurse on each of the newly created connected components;

However, this is far from true in the hypergraph setting. The problem is that there may be some hyperedges that intersect with each of S, T, and $V \setminus (S \cup T)$. These edges are inside all of $\delta(S)$, $\delta(T)$ and $\delta(S \cup T)$. We have

$$\begin{aligned}
&|\delta(S)| + |\delta(T)| - |\delta(S \cup T)| \\
=&2\left|\{e | e \cap S \neq \emptyset, e \cap T \neq \emptyset, e \cap (\overline{S \cup T}) = \emptyset\}\right| + \left|\{e | e \cap S \neq \emptyset, e \cap T \neq \emptyset, e \cap (\overline{S \cup T}) \neq \emptyset\}\right| \\
=&|\{e | e \cap S \neq \emptyset, e \cap T \neq \emptyset\}| + \left|\{e | e \cap S \neq \emptyset, e \cap T \neq \emptyset, e \cap (\overline{S \cup T}) = \emptyset\}\right|
\end{aligned}$$

▶ **Example 15.** Consider a hypergraph H that consists of three equal size sets of vertices A, B, C, such that each hyperedge has a non-empty intersection with each of A, B, and C. Then there are no hyperedges in $H[A \cup B]$. But the quantity $\frac{1}{2}(|\delta(A)| + |\delta(B)| - |\delta(A \cup B)|)$ is half the total number of hyperedges which could be exponentially large in n.

▶ **Example 16.** Consider the following pair of hypergraphs on 4 vertices, say $\{v_1, v_2, v_3, v_4\}$: the graph H_1 is a (rank 2) clique on 4 vertices while the graph H_2 contains every possible edge of size 3 on these 4 vertices. It is easy to verify that the answer to every cut query is the same on the graphs H_1 and H_2. Now consider the subgraph of these graphs induced by the vertices $X = \{v_1, v_2\}$. In case of H_1, the minimum cut in the induced subgraph is 1 while in H_2, the minimum cut in the graph induced by X is 0. We can amplify this gap to 0 versus $\Omega(n)$ by taking $n/4$ copies of H_1 in one case, and $n/4$ copies of H_2 in the other case, and defining X to be union of arbitrarily chosen pairs of vertices from each copy. This means that O_{value} queries can not be used to estimate cut size in induced subgraphs to any multiplicative factor or to better than a polynomial additive error.

To get around the challenge highlighted by examples above, we next introduce notions of *pseudo cut size* over a subset of vertices and *pseudo strength* of hyperedges, such that the pseudo cut sizes are easy to compute by cut queries and pseudo strength of any hyperedge is at most a factor n larger than the strength of the hyperedge. We develop these ideas in detail in the next subsection.

3.1 Pseudo Cuts and Pseudo Strengths

Given a set of vertices X, we define $\Delta_X(S)$, the *pseudo cut* size of a set $S \subset X$ as $\frac{1}{2}(|\delta(S)| + |\delta(X \setminus S)| - |\delta(X)|)$, and define the *pseudo min cut* over X as a cut $(S, X \setminus S)$ that minimizes $\Delta_X(S)$. Note that $\Delta_X(S)$ is at most the number of edges that intersect both S and $X \setminus S$. The following lemma shows that $\Delta_X(S)$ is a submodular function, so we can compute the pseudo min cut over any vertex set in $\text{poly}(n)$ time by Theorem 14.

▶ **Lemma 17.** *For any vertex set $X \subseteq V$, $\Delta_X(S)$ is a submodular function.*

Proof. Let $f_1(S)$ be the number of edges that intersect both S and $X \setminus S$, and let $f_2(S)$ be the number of edges that intersect both S and $X \setminus S$ but are fully contained in X. By definition, we have $\Delta_X(S) = \frac{1}{2}(f_1(S) + f_2(S))$, so to prove that $\Delta_X(S)$ is a submodular function, it is sufficient to prove that both f_1 and f_2 are submodular.

Since f_2 is the cut function in the induced graph $H[X]$, it is submodular. In fact, f_1 is also the cut function of the hypergraph whose vertex set is X and edge set is $\{e \cap X | e \in E\}$. So f_1 is also a submodular function. ◀

For any edge e, we define the *pseudo strength* k'_e as the largest pseudo min-cut size among all sets X that contain e, where X ranges over all subsets of V. It is easy to see that for any edge e, k'_e is at least k_e since for any set of vertices X, the minimum cut size of $H[X]$ is at most the pseudo min-cut size of set X. More interestingly, although Example 15 showed that the pseudo min-cut size of a set X may be arbitrarily larger than the minimum cut size of $H[X]$, the lemma below shows that the pseudo strength of an edge is at most a factor n larger than its strength.

▶ **Lemma 18.** *For any edge e, $k'_e \leq nk_e$.*

Proof. Let X be any set of vertices that contains the edge e and has pseudo min-cut size k'_e in H. To prove the lemma, it is sufficient to prove that the pseudo min-cut size of X in H is at most nk_e.

Let $Y = V$ and $E_c = \emptyset$. Consider the following iterative process: we find the minimum cut $(S, Y \setminus S)$ in $H[Y]$. If either S or $Y \setminus S$ fully contains the set X, we add all edges crossing the cut into E_c, and set Y to be S or $Y \setminus S$ (whichever fully contains X), and repeat. Otherwise, we stop the process.

After the process terminates, suppose $(S, Y \setminus S)$ is the minimum cut in $H[Y]$. Since the process terminated, $(S, Y \setminus S)$ must partition X. Let $S' = S \cap X$, and consider the pseudo cut $(S', X \setminus S')$. We prove that the number of edges in H that intersect both S' and $X \setminus S'$ (which is an upper bound on $\Delta_X(S')$) is at most nk_e.

First, note that no edge e' such that $e' \not\subseteq Y$ and $e' \notin E_c$ can intersect with the set Y; hence any such edge e' also does not intersect with S' or $X \setminus S'$. Therefore every edge that intersects with both S' and $X \setminus S'$ either belongs to E_c or is completely contained in Y. During the iterative process, the set Y always fully contains e, so by the definition of strength, the minimum cut size of $H[Y]$ is at most k_e. This implies during each step, at most k_e edges are added into E_c. On the other hand, the process repeats at most $n - 2$ times, since each time the size of Y is reduced by at least 1. So $|E_c| \leq (n-2)k_e$. Finally, any edge that is fully contained in Y and intersects with both S' and $X \setminus S'$ crosses the cut $(S, Y \setminus S)$ in $H[Y]$, and the number of such edges is at most the minimum cut size of $H[Y]$, which is at most k_e. So in total, there are at most $|E_c| + k_e \leq (n-1)k_e$ edges that intersect both S' and $X \setminus S'$. ◀

3.2 Sampling the Edges

We are now ready to present an algorithm that uses the cut size queries and cut edge sample queries to sample each edge with probability inversely proportional to its strength. Specifically, we will ensure that each edge e gets sampled with probability at least n^2/k'_e which is at least n/k_e by Lemma 18. The algorithm is similar to Algorithm 1, but uses pseudo cuts and pseudo strengths instead. To sample the edges, we call Algorithm 2 on set V.

Algorithm 2 Sampling edges with probability proportional to their pseudo strength.

Input : A subset of vertices $V' \subseteq V$
1 Find the pseudo min-cut $(S, V' \setminus S)$ within the set V';
2 Let c be $|\delta(S)|$, the cut size of $(S, V \setminus S)$;
3 Sample an integer $N \sim B(c, \min\{1, \frac{10n^3}{\epsilon^2 c}\})$;
4 Keep sampling edges in cut $(S, \bar{S})$ until we get N different hyperedges. ;
5 Recurse on both S and $V' \setminus S$;

We now prove that each edge gets sampled with probability at least as large as the sampling probability in Theorem 8. Fix an edge e, let S_1 be the last input set that fully contains e. For any $i \geq 1$, if S_i is not V, we define S_{i+1} to be the input set in the recursion that generates a recursive call of the algorithm on the set S_i. In other word, $(S_i, S_{i+1} \setminus S_i)$ is the pseudo min cut within set S_{i+1}. Let $(S_0, S_1 \setminus S_0)$ be the pseudo min cut within S_1, by definition, $e \cap S_0 \neq \emptyset$ and $e \cap S_1 \setminus S_0 \neq \emptyset$. When the algorithm works on set S_1, e gets sampled with probability $\min\{1, \frac{10n^3}{\varepsilon^2 |\delta(S_0)|}\}$. If e gets sampled with probability 1, then it is clearly as large as the probability in Theorem 8. Otherwise we need to prove that $n^3/|\delta(S_0)| = \Omega((\log n + r)/k_e)$. Since $n = \Omega(\log n + r)$, by Lemma 18, it is sufficient to prove that $|\delta(S_0)| \leq nk'_e$.

▶ **Lemma 19.** $|\delta(S_0)| \leq nk'_e$.

Proof. We partition the edges crossing the cut $(S_0, \overline{S_0})$ into sets $E_1, E_2, \ldots$ such that for any $i \geq 0$, E_i is the set of edges that are fully contained in S_{i+1} but not in S_i. Note that $|\delta(S_0)| = \sum_i |E_i|$. Since the algorithm has at most n levels of recursion, to prove the lemma, it is sufficient to prove $|E_i| \leq k'_e$ for all $i \geq 0$.

For any edge $e' \in E_i$, $e' \cap S_i \neq \emptyset$ since e' crosses the cut $(S_0, \overline{S_0})$ and $S_0 \subseteq S_i$. We also have $e' \cap S_{i+1} \setminus S_i \neq \emptyset$ and $e' \cap \overline{S_{i+1}} = \emptyset$ since e' is fully contained in S_{i+1} but not S_i. So $|E_i| \leq \Delta_{S_{i+1}}(S_i)$. On the other hand, by definition of pseudo strength, $k'_e \geq \Delta_{S_{i+1}}(S_i)$ since e is fully contained in S_{i+1}. Therefore, $|E_i| \leq k'_e$. ◀

By Lemma 19, we proved that each edge e is sampled with probability at least the required probability in Theorem 8. Next, we need to assign weights to each sampled edge.

We do this after we finish sampling. For each edge e that gets sampled, we need to know the probability that it gets sampled. Since we sample edges from each cut independently, we only need to know the probability that e gets sampled during each recursive call, and that probability depends only on the size of the cut and whether e crosses the cut. So we can compute the probability that e gets sampled during Algorithm 2.

To complete the proof of Theorem 1, we need to show that the running time of the whole process is polynomial in n.

Proof of Theorem 1. During each call to Algorithm 2, we need $\tilde{O}(n^3)$ queries to cut size query oracle and $\tilde{O}(n^4)$ time to figure out the pseudo min-cut within the set V' by Theorem 14 and Lemma 17. At line 4, we call cut edge sample query $10n^3/\varepsilon^2$ times in expectation. Total number of recursive calls to Algorithm 2 is $O(n)$, since each time, the input set gets partitioned into two sets, and there are n sets in the end. Thus the running time of Algorithm 2 is $\tilde{O}(n^5 + n^4/\varepsilon^2)$.

We sample the edges in $O(n)$ cuts, so when assigning the weights, we only need to query the size of these $O(n)$ cuts and calculate the probability of each sampled edge, which can also be done in $O(n)$ time for each edge. So the running time of assigning the weights is $\tilde{O}(n^5/\varepsilon^2)$.

After sampling the edges and assigning weights, we get a $(1\pm\varepsilon)$-approximate cut sparsifier H_1 of H with polynomial size in n. Then we run the algorithm in Theorem 11 to find a $(1+\varepsilon)$-approximate cut sparsifier H_2 of H_1 with $\tilde{O}(n/\varepsilon^2)$ number of edges in polynomial time in n. By definition of cut sparsifier, H_2 is a $(1\pm\varepsilon)^2$-approximate cut sparsifier of H. Since H_1 contains $\tilde{O}(n^4/\varepsilon^2)$ edges, by Theorem 11, the running time is $O(n^{10}/\varepsilon^7)$.

So the total running time is $O(n^{10}/\varepsilon^7)$. ◀

4 Sublinear Time Cut Sparsification with Cut Size and Pair Neighbor Queries

In this section, we show that cut edge queries can be simulated by a $\text{poly}(n)$ number of cut size queries (oracle O_{value}) and pair neighbor queries (oracle O^2_{nbr}), establishing that cut size query oracle and pair neighbor query oracle are also sufficient to compute a $(1\pm\varepsilon)$-approximate cut sparsifier in $\text{poly}(n)$ time.

Given a pair of vertices u and v, let $E(\{u,v\})$ be the set of edges that contain both u and v. We first show how to approximate $|E(\{u,v\})|$ to within a factor of $(1\pm\varepsilon)$ with probability $1-\xi$, for some small ξ. Note that we can compute $2\Delta_{\{u,v\}}(\{u\}) = |E(\{u,v\})| + |E\cap\{u,v\}|$, where $|E\cap\{u,v\}|$ is the number of copies of the edge $\{u,v\}$. We now describe an algorithm to approximate $|E(\{u,v\})|$:

Algorithm 3 Approximating $|E(\{u,v\})|$.

Input : A pair of vertices $u,v\in V$, ε, ξ.
Output : An approximation $\hat{E}(\{u,v\})$ of $|E(\{u,v\})|$.

1 Define $k = 12\log(2/\xi)/(\varepsilon^2)$.
2 Call the oracle O^2_{nbr} k times on (u,v), and let $\hat{\alpha}$ be the fraction of returned edges that were $\{u,v\}$.
3 Return $\hat{E}(\{u,v\}) := 2\Delta_{\{u,v\}}(\{u\})\cdot\frac{1}{1+\hat{\alpha}}$.

Note that this algorithm makes $k = O(\frac{\log(1/\xi)}{\varepsilon^2})$ queries.

▶ **Lemma 20.** *With probability at least $1-\xi$, $\hat{E}(\{u,v\})$ is an approximation of $|E(\{u,v\})|$ to within a factor of $(1\pm\varepsilon)$.*

Proof. Let $\alpha := \frac{|E\cap\{u,v\}|}{|E(\{u,v\})|}$ be the fraction of hyperedges that are $\{u,v\}$. The algorithm runs a Monte Carlo simulation to approximate α by the ratio $\hat{\alpha}$. In order to prove concentration of $\hat{\alpha}$ around α, let k' be the total number of $\{u,v\}$ edges returned, and observe that k' is the sum of k independent Bernoulli random variables each having probability equal to α. By Chernoff bound, $\Pr[|k'-\alpha k| > \varepsilon k/2] \le 2\exp(-\alpha k\varepsilon^2/12\alpha) \le 2\exp(-k\varepsilon^2/12) = 2\exp(\log(\xi/2)) = \xi$. Therefore with probability at least $1-\xi$, $|\hat{\alpha}-\alpha|\le\varepsilon/2$. This implies that $\frac{1}{1+\hat{\alpha}} \in \frac{1}{1+\alpha\pm\varepsilon/2} \subseteq \frac{1}{(1\pm\varepsilon/2)(1+\alpha)}$. Finally, we use that $\frac{1}{(1\mp\varepsilon/2)} \subseteq (1\pm\varepsilon)$ to conclude that $\frac{1}{1+\hat{\alpha}} \in \frac{1\pm\varepsilon}{1+\alpha}$, so

$$\frac{2\Delta_{\{u,v\}}(\{u\})}{1+\hat{\alpha}} \in (1\pm\varepsilon)\frac{2\Delta_{\{u,v\}}(\{u\})}{1+\alpha} = (1\pm\varepsilon)|E(\{u,v\})|.$$ ◀

We now describe an algorithm to sample a random edge from $\delta(S)$, simulating a response to O_{edge}. We first approximate the size of $E(\{u,v\})$ for each pair of vertices $u\in S$ and $v\in\bar{S}$. Then we sample a pair of u,v with probability proportional to $|E(\{u,v\})|$, sample an edge in $E(\{u,v\})$, and then decide whether we keep it or not with probability proportional to its size. If we decide not to pick the edge, we repeat the whole process again.

Algorithm 4 Sampling an edge in $\delta(S)$.

Input : A subset $S \subseteq V$
Output : An edge $e \in \delta(S)$

1 For each pair of vertices u, v such that $u \in S$ and $v \in \bar{S}$, call Algorithm 3 with $\xi = 1/n^{20}$, and let $\hat{E}(\{u,v\})$ be the output;
2 Sample a pair of vertices $(u,v) \in S \times \bar{S}$ with probability proportional to $\hat{E}(\{u,v\})$;
3 Use the oracle O^2_{nbr} to sample an edge e in $E(\{u,v\})$;
4 With probability $\frac{1}{|e\cap S|\cdot|e\cap\bar{S}|}$, return e. Otherwise go to Step 2.

▶ **Lemma 21.** *With probability at least $1 - 1/n^{-10}$, Algorithm 4 samples each edge in $\delta(S)$ gets with probability $\frac{1\pm\varepsilon}{|\delta(S)|}$. The expected running time is $\tilde{O}(n^2/\varepsilon^2)$.*

Proof. We first condition on the $|S| \cdot |\bar{S}| \le n^2$ events that for each pair u, v with $u \in S$ and $v \in \bar{S}$, the estimate $\hat{E}(\{u,v\})$ was indeed in $(1 \pm \varepsilon) \cdot |E(\{u,v\})|$, which happens with probability at least $1 - n^2\xi > 1 - 1/n^{-10}$. Now fix an edge $e \in \delta(S)$. The probability that it was sampled at a particular iteration of Algorithm 4 is

$$\sum_{u\in S\cap e, v\in\bar{S}\cap e} \frac{\hat{E}(\{u,v\})}{\sum_{(u',v')\in S\times\bar{S}}\hat{E}(\{u',v'\})} \cdot \frac{1}{|E(\{u,v\})|} \cdot \frac{1}{|e\cap S|\cdot|e\cap\bar{S}|}$$

$$\in \frac{1}{\sum_{(u',v')\in S\times\bar{S}}\hat{E}(\{u',v'\})} \sum_{u\in S\cap e, v\in\bar{S}\cap e} \frac{(1\pm\varepsilon)}{|e\cap S|\cdot|e\cap\bar{S}|} = \frac{(1\pm\varepsilon)}{\sum_{(u',v')\in S\times\bar{S}}\hat{E}(\{u',v'\})}$$

That is, the probability of sampling each edge at any given iteration of Algorithm 4 is within $(1 \pm \varepsilon)$ of every other edge. Therefore the probability of sampling each edge is within a factor of $(1 \pm \varepsilon)$ of every other edge.

Step 1 calls Algorithm 3 $O(n^2)$ times, so the running time is $\tilde{O}(n^2/\varepsilon^2)$. At step 4, the probability that we keep the edge and finish the algorithm is at least $1/n^2$, so the expected number of iterations through step 2 to 4 is at most n^2. So the total running time on step 2 to 4 is $\tilde{O}(n^2)$ in expectation. ◀

Proof of Theorem 3. We run Algorithm 2, but each time it calls O_{edge}, we instead run Algorithm 4 twice. With high probability, each time we simulate O_{edge} by Algorithm 4, the probability of any edge in the cut being sampled is within a $(1 \pm \varepsilon)$ factor of the uniform distribution. Denote by q'_e be the probability that an edge e is sampled by this algorithm, and let q_e be the probability that the edge e is sampled in Algorithm 2. We have $q'_e \in 2(1 \pm \varepsilon)q_e$, which is larger than p_e the probability of sampling an edge in Theorem 8. Also we cannot directly compute q'_e, but we can approximate it to within a factor of $(1 \pm \varepsilon)$, which only adds another $(1 \pm \varepsilon)$ factor to the approximation achieved by the cut sparsifier.

Since the number of calls to O_{edge} oracle in Algorithm 2 is $\tilde{O}(n^4/\varepsilon^2)$. So we need $\tilde{O}(n^6/\varepsilon^2) = o(n^10/\varepsilon^7)$ queries to simulate these calls. So the running time of the algorithm is still $O(n^10/\varepsilon^7)$. ◀

5 Sublinear Time Hypergraph Spectral Sparsification

In this section, we consider the problem of hypergraph spectral sparsification in the access models considered in the previous sections. We will focus on unweighted hypergraphs here, and show that these results can be extended to the weighted case in Section 7. We first consider the setting when we can access the underlying hypergraph using O_{value} and O^2_{nbr}

queries. It is easy to simulate the approach in Theorem 12 since O^2_{nbr} allows us to sample from $E(\{u,v\})$ for any u and v. For any pair of vertices u and v, we sample $Cn\log n/\varepsilon^2$ edges in $E(\{u,v\})$. For any hyperedge e, e gets sampled with probablity q_e which is at least the required value p_e. Next we need to assign weights to the sampled edges. For any pair of vertices u and v, we use Algorithm 3 to approximate $|E(\{u,v\})|$. Then for any e, we approximate q_e by the approximation of $|E(\{u,v\})|$ for all pair u and v in e, and then assign the weight of e as $1/q_e$.

Proof of Theorem 5. For each pair of vertices u and v, we sample $\tilde{O}(n)$ edges in $E(\{u,v\})$, and also use Algorithm 3 to approximate $|E(\{u,v\})|$. The total number of edges in the sparsifier and the total number of queries we perform is $\tilde{O}(n^3)$. For every sampled edge e, we need to calculate q_e, which costs at most $O(n^2)$ time as we need to combine the probabilities of sampling e through any pair of vertices inside e. So the total time complexity of the algorithm is $\tilde{O}(n^5)$. ◀

We next consider the case when we are given access to hypergraph via O_{value} and O_{edge} queries only. We observe that we cannot simulate the algorithm in Theorem 12. Consider the following hypergraph H: there is a pair of vertices u and v, such that H contains all possible hyperedges that do not contain both u and v. H also contains an edge $\{u,v\}$. Since $\{u,v\}$ is the only edge that contains both u and v, we need to sample this edge with probability 1 in the algorithm of Theorem 12. However, any cut in the hypergraph has exponential (in n) size. So we need an exponential number of queries to sample the edge $\{u,v\}$.

Given the obstacle above, we will instead simulate Theorem 13. Our approach is to show that the task of implementing Theorem 13 can be accomplished by our algorithm in Section 3, except that we will sample poly(n) times more hyperedges. By doing so, we will guarantee that the probability that each hyperedge e gets sampled is larger than the p_e in Theorem 13. We now develop the ideas needed to establish this coupling between Theorem 13 and our algorithm in Section 3.

We first observe a relationship between the effective resistance and strength of an edge in a normal graph.

▶ **Lemma 22.** *For any edge f in a normal (weighted) graph G, we have $r_f \le \frac{n}{k_f}$.*

Proof. Suppose the edge $f=(u,v)$, and let c denote the min-cut size between u and v. Since for any vertex-induced subgraph of G that contains both u and v, the min-cut size is at most c, it follows that $k_f \le c$. So it is sufficient to prove that $r_f \le \frac{n}{c}$.

Since the min-cut size between u and v is c, the max-flow size between u and v is also c, which means we have a set of c edge-disjoint paths from u to v. As each path can have length at most n, this set of edge-disjoint paths can be interpreted as c parallel resistors each with resistance at most n. By Rayleigh's monotonicity law, it then follows that $r_f \le \frac{n}{c}$. ◀

We now briefly review the approach underlying Theorem 13 (see Section 2.3). Recall that for any hyperedge e in a hypergraph H, F_e is the clique associated with e in the auxiliary graph G_H, and $r_e = \max_{f\in F_e} r_f$. Define $\kappa_e = \min_{f\in F_e} k_f$. Then by Lemma 22, we have $r_e \le \frac{n}{\kappa_e}$. The following lemma shows a relationship between κ_e and k_e.

▶ **Lemma 23.** *For any hyperedge e in a hypergraph H, $\kappa_e \ge k_e$.*

Proof. For any subset of vertices X, consider the corresponding vertex-induced subgraphs of H and G_H, which we denote by $H[X]$ and $G_H[X]$, respectively. For any cut defined by a partition of X, and for any hyperedge e crossing this cut in $H[X]$, at least one edge in F_e

must cross this cut in $G_H[X]$. So the min-cut size of $H[X]$ is at most the min-cut size of $G_H[X]$. Let X_e be the set of vertices such that $e \subseteq X_e$ and the min-cut size of $H[X_e]$ equals k_e. The min-cut size of $G_H[X_e]$ is at least k_e. For any edge $f \in F_e$, f is contained in X_e and so $k_f \geq k_e$, which means $\kappa_e = \min_{f \in F_e} k_f \geq k_e$. ◀

By Lemma 22 and Lemma 23, we have $\frac{n}{k_e} \geq r_e$ for any hyperedge e. So by Theorem 13, if we sample each hyperedge with probability q_e which is at least $\min\{1, \frac{Cn^5 \log n}{\varepsilon^2 k_e}\}$, and assign it weight $1/q_e$ if sampled, then the resulting hypergraph is a $(1 \pm \varepsilon)$-spectral sparsifier. So we can use the same process as described in Section 3, except that we oversample hyperedges by a factor of n^4. The resulting hypergraph will then be a $(1 \pm \varepsilon)$-spectral hypergraph sparsifier.

Proof of Theorem 6. We run Algorithm 2, except that in line 3, we set $N \sim B(c, \min\{1, \frac{Cn^7}{\varepsilon^2 c}\})$ where C is the constant in Theorem 13. By the same argument as in Section 3, we sample each edge with probability at least $\min\{1, \frac{Cn^5 \log n}{\varepsilon^2 k_e}\}$, which means the resulting graph is a $(1 \pm \varepsilon)$-spectral sparsifier if we assign the weight of each edge also using the same process in Section 3. The number of hyperedges sampled is $\tilde{O}(n^8/\varepsilon^2)$. We then run the algorithm in Theorem 12 on our sparsifier and get a $(1 \pm 2\varepsilon)$-spectral sparsifier with $\tilde{O}(n^3/\varepsilon^2)$ hyperedges. The time taken by running Algorithm 2 and assigning weights is $\tilde{O}(n^9/\varepsilon)$ since there are n^4 times more hyperedges being sampled. The running time of the algorithm in Theorem 12 is $\tilde{O}(n^{10})$ since there are $\tilde{O}(n^8)$ hyperedges sampled by Algorithm 2. So the overall running time is $\tilde{O}(n^{10})$. ◀

6 Lower Bounds

In this section we show that any natural relaxation of the assumptions underlying Theorem 1 and 3 rules out poly(n) time sparsification algorithms, proving Theorem 2 and 4.

6.1 Queries O_{value} and O^1_{nbr} Together are not Sufficient

In this section, we prove that if any randomized algorithm can only access the underlying hypergraph via O_{value} and O^1_{nbr}, it is not possible to find with probability better than $o(1)$ a $(1 \pm \varepsilon)$-approximate cut sparsifier with only poly(n) queries, proving Theorem 4. We start by showing a weaker result, as stated in the lemma below, which shows that the failure probability of a poly(n) time algorithm must be at least $1/2 - o(1)$, and then show how to amplify the failure probability to $1 - o(1)$.

▶ **Lemma 24.** *There is no polynomial time algorithm that can use O_{value} and O^1_{nbr} queries alone to construct a $(1 \pm \varepsilon)$-approximate sparsifier of an underlying hypergraph H with probability at least $1/2 + \xi$ for any constant $\xi > 0$.*

Proof. Suppose the runtime of the algorithm is bounded by some polynomial $f(n)$. We will construct two graphs $H_1 = (V \cup V', E_1)$ and $H_2 = (V \cup V', E_2)$ with $|V| = |V'| = n$ and the algorithm is shown with probability $1/2$ the graph H_1 and with probability $1/2$ the graph H_2. We will then show that (a) any algorithm that can only access the underlying graph using O_{value} and O^1_{nbr} cannot distinguish between these two graphs with probability at least $1/2 + \xi$ for any constant $\xi > 0$, and (b) there exists a non-empty cut such that H_1 and H_2 do not have any common edges crossing the cut. Together, these properties immediately imply the lemma .

Let $u, v \in V$ and $u', v' \in V'$ be two arbitrary pairs of vertices. Let $E = 2^V \cup 2^{V'}$ be the union of the complete hypergraphs on V and V'. We define E_1 as E along with all possible edges of size two among $\{u, v, u', v'\}$. We define E_2 as E along with all possible edges of

size 3 among $\{u, v, u', v'\}$. It is easy to verify that for any cut, the number of edges in E_1 crossing the cut equals the number of edges in E_2 crossing the cut. So any cut size query O_{value} has the same answer in H_1 and H_2, and hence can not distinguish between these two graphs, no matter the number of queries allowed.

The algorithm can additionally make at most $f(n)$ calls to O^1_{nbr}. But since each vertex $w \in V \cup V'$ has at least 2^n edges incident on it, the probability that a uniformly random edge incident on w is not in E is at most $3/2^n$. Using a union bound over all $f(n)$ queries along with the fact that $3f(n)/2^n \leq \xi$ for sufficiently large n, we get that for both hypergraphs, with probability at least $1 - 3f(n)/2^n \geq 1 - \xi$, all sampled edges are in E.

Thus conditioned on the event that all of the sampled edges are in E, the algorithm cannot distinguish between H_1 and H_2. On the other hand, there are no common edges crossing the cut (V, V') in H_1 and H_2, so to output a proper $(1 \pm \varepsilon)$-approximate cut sparsifier, the algorithm must distinguish between H_1 and H_2. Hence the probability that algorithm succeeds is at most $1/2 + \xi$. ◀

To amplify the failure probability to $1 - o(1)$, we can independently generate $\log n$ instances from the distribution above with each instance containing $n/\log n$ vertices. We now let our underlying graph be a union of these $\log n$ instances. Any algorithm that outputs a $(1 \pm \varepsilon)$-approximate sparsifier, must successfully identify for each of the $\log n$ instances whether it is an instance of H_1 or H_2. Thus the probability of success is at most $(1/2 + o(1))^{\log n} = o(1)$. This completes the proof of Theorem 4.

6.2 O_{edge} Queries Alone are not Sufficient

In this section, we prove that if the algorithm can access the hypergraph through only O_{edge} queries, it is not possible to find a proper $(1 \pm \varepsilon)$-approximate cut sparsifier with $\text{poly}(n)$ queries with success probability better than $o(1)$, proving Theorem 2. As above, we start by showing a weaker result, which shows that the failure probability of a $\text{poly}(n)$ time algorithm must be at least $1/2 - o(1)$, and then show how to amplify the failure probability to $1 - o(1)$.

We first define two distributions of hypergraphs $\mathcal{H}_1$ and $\mathcal{H}_2$ such that for any sequence of the queries the algorithm asks to O_{edge}, the distribution of the answers are almost identical regardless of whether the graph was chosen from $\mathcal{H}_1$ or $\mathcal{H}_2$.

A graph in each of the distributions $\mathcal{H}_1$ and $\mathcal{H}_2$ is generated as follows. There are $n + 1$ vertices $v_0, v_1, \ldots, v_n$ and the generated graph will have $2^n - n - 1$ edges. If the graph is generated by $\mathcal{H}_1$, then we randomly choose $2^{n/2}$ subsets of $\{v_1, \ldots, v_n\}$ with size at least 2. If the graph is generated by $\mathcal{H}_2$, then we randomly choose $2^{n/4}$ subsets of $\{v_1, \ldots, v_n\}$ with size at least 2. Then for any subset S of $\{v_1, \ldots, v_n\}$ of size at least 2, if S is chosen in the previous step, then the edge $S \cup \{v_0\}$ is in the graph, otherwise the edge S is in the graph.

The algorithm is presented with probability $1/2$ a graph H generated by $\mathcal{H}_1$, and with probability $1/2$ a graph H generated by $\mathcal{H}_2$, that is, the algorithm sees a graph H generated by the distribution $1/2\mathcal{H}_1 + 1/2\mathcal{H}_2$. Since the cut sizes of $(\{v_0\}, \overline{\{v_0\}})$ in the graph generated by $\mathcal{H}_1$ and $\mathcal{H}_2$ are $2^{n/2}$ and $2^{n/4}$ respectively, any algorithm that outputs a $(1 \pm \varepsilon)$-approximate cut sparsifier with $\varepsilon < 1$ must be able to distinguish between the graphs generated by $\mathcal{H}_1$ and $\mathcal{H}_2$. However, the following lemma shows that unless the algorithm makes exponential number of queries, it cannot distinguish between the graphs generated by $\mathcal{H}_1$ and $\mathcal{H}_2$. The proof of this lemma is deferred to the full version of this paper.

▶ **Lemma 25.** *Any algorithm that only makes k O_{edge} queries where $k = \text{Poly}(n)$ cannot determine with probability better than $\frac{1}{2} + \frac{k^2}{2^{n/4}}$ if the underlying graph H is generated from $\mathcal{H}_1$ or $\mathcal{H}_2$.*

Thus any algorithm that makes only $\text{poly}(n)$ O_{edge} queries, fails with probability at least $1/2 - o(1)$. To amplify the failure probability to $1 - o(1)$, we can as before independently generate $\log n$ instances from the distribution above with each instance containing $n/\log n$ vertices. We now let our underlying graph be a union of these $\log n$ instances. Any algorithm that outputs a $(1 \pm \varepsilon)$-approximate sparsifier, must successfully identify for each of the $\log n$ instances whether it was generated from the first distribution or the second. Thus the probability of success is at most $(1/2 + o(1))^{\log n} = o(1)$. This completes the proof of Theorem 2.

7 Weighted Hypergraphs

In this section, we describe how to extend the results of Theorem 1, Theorem 3, Theorem 5, and Theorem 6 to weighted hypergraphs. Given a weighted hypergraph $H = (V, E, w)$, we consider access to this graph by weighted generalizations of oracles O_{value}, O_{edge}, and O^2_{nbr}: the oracle O_{value} now returns the weight of a cut rather than its size, and instead of sampling uniformly, the oracles O_{edge} and O^2_{nbr} sample edges with probability proportional to their weight.

We first note that all the Theorems we use to derive these results (Theorem 8, Theorem 12, and Theorem 13) can be modified to work when the input hypergraph is weighted.

▶ **Lemma 26** (Weighted version of Theorem 8). *Let $H = (V, E, w)$ be a weighted hypergraph with rank r, and let $\varepsilon > 0$ be an error parameter. Consider the hypergraph H' obtained by sampling each hyperedge e in H independently with probability $p_e \geq \min\{1, w(e) \cdot \frac{3((d+2)\log n + r)}{k_e \varepsilon^2}\}$, giving it weight $w(e)/p_e$ if included. Then with probability at least $1 - O(n^{-d})$, H' is a $(1 \pm \varepsilon)$-approximate cut sparsifier of H, and has $O(\frac{n}{\varepsilon^2}(r + \log n))$ hyperedges.*

▶ **Lemma 27** (Weighted version of Theorem 12). *Let $H = (V, E, w)$ be a weighted hypergraph, and let $\varepsilon > 0$ be an error parameter. Consider the hypergraph H' obtained by sampling each hyperedge e in H independently with probability $p_e \geq \min\{1, w(e) \cdot \frac{Cn \log n}{\varepsilon^2 \min_{u,v \in e} w(E(\{u,v\}))}\}$, giving it weight $w(e)/p_e$ if included. Then with high probability, H' is a $(1 \pm \varepsilon)$-approximate spectral sparsifier of H, and has $\tilde{O}(n^3/\varepsilon^2)$ hyperedges.*

▶ **Lemma 28** (Weighted version of Theorem 13). *Let $H = (V, E, w)$ be a weighted hypergraph with rank r, and let $\varepsilon > 0$ be an error parameter. Consider the hypergraph H' obtained by sampling each hyperedge e in H independently with probability $p_e \geq \min\{1, w(e) \cdot \frac{Cr^4 r_e \log n}{\varepsilon^2}\}$, giving it weight $w(e)/p_e$ if included. Then with high probability, H' is a $(1 \pm \varepsilon)$-approximate spectral sparsifier of H, and has $\tilde{O}(nr^3/\varepsilon^2)$ hyperedges.*

Most of our arguments and definitions for the unweighted case of Theorem 1 and Theorem 6 translate directly to the weighted case once we replace every mention of the cardinality of an edge set by the weight of that edge set. In particular, if we generalize the definition of pseudo cut size to be $\Delta_X(S) = \frac{1}{2}(w(\delta(S)) + w(\delta(X \setminus S)) - w(\delta(X)))$, then the proofs for Lemma 17, Lemma 18, and Lemma 19 are completely analogous. The only difference in the analysis of Algorithm 2 is that the probability that a sample from the cut $(S_0, \bar{S_0})$ returns an edge e in the cut is now $w(e)/w(\delta(S))$, implying that the probability that e is sampled by Algorithm 2 is at least $\min\{1, \frac{10w(e)n^3}{\epsilon^2 w(\delta(S_0))}\}$. Since this is at least the requisite sampling probability in Lemma 26, the hypergraph H_1 is a sparsifier of H with high probability. In the case of Theorem 6, we still have that for every hyperedge e, $\frac{n}{k_e} \geq r_e$, so Algorithm 2 can be applied to sample each hyperedge e with the desired probability of at least $\min\{1, w(e) \cdot \frac{Cr^4 r_e \log n}{\varepsilon^2}\}$.

Similarly for Theorem 3 and Theorem 5, the proof of correctness of Algorithm 3 is almost completely analogous (although this time, the algorithm outputs an estimate of the total weight of hyperedges containing both u and v). The proof of correctness of Algorithm 4 is modified to assert that the probability of sampling an edge $e \in E(\{u, v\})$ is $w(e)/w(E(\{u, v\}))$, implying that the probability that the algorithm samples e in each iteration is proportional to $(1 \pm \varepsilon) \cdot w(e)$.

Note that the running time of our algorithms are independent of the number of edges in the unweighted setting. Similarly, in the weighted setting, our running times have no dependence on the weights of the edges, and the running time and the size of sparsifier are the same as in the unweighted cases.

8 Concluding Remarks

We presented the first sublinear time algorithms for creating a hypergraph sparsifier. Given access to a hypergraph through cut size and suitable edge sampling queries, our algorithm outputs a $(1 \pm \varepsilon)$-approximate sparsifier with $\tilde{O}(n/\varepsilon^2)$ hyperedges in polynomial time in n, independent of the number of hyperedges. We also showed that for any natural weakening of our query access assumptions, there is no poly(n) time algorithm for building a hypergraph sparsifier of poly(n) size. An intriguing question is if an information-theoretic cut sparsifier can be constructed using cut value queries alone. Cut value queries alone can not distinguish between hypergraphs which only contain edges of rank 2 from hypergraphs which only contain edges of rank 3, making it impossible for them to output a proper sparsifier. But this does not rule out the possibility that a suitable data structure can be created using these queries alone that can recover the value of any cut to within a $(1 \pm \varepsilon)$-approximation.

References

1. Kook Jin Ahn and Sudipto Guha. Graph sparsification in the semi-streaming model. In Susanne Albers, Alberto Marchetti-Spaccamela, Yossi Matias, Sotiris E. Nikoletseas, and Wolfgang Thomas, editors, *Automata, Languages and Programming, 36th Internatilonal Colloquium, ICALP 2009, Rhodes, Greece, July 5-12, 2009, Proceedings, Part II*, volume 5556 of *Lecture Notes in Computer Science*, pages 328–338. Springer, 2009.
2. Kook Jin Ahn, Sudipto Guha, and Andrew McGregor. Graph sketches: sparsification, spanners, and subgraphs. In *Proceedings of the 31st ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems, PODS 2012, Scottsdale, AZ, USA, May 20-24, 2012*, pages 5–14, 2012.
3. Kook Jin Ahn, Sudipto Guha, and Andrew McGregor. Spectral sparsification in dynamic graph streams. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques - 16th International Workshop, APPROX 2013, and 17th International Workshop, RANDOM 2013, Berkeley, CA, USA, August 21-23, 2013. Proceedings*, pages 1–10, 2013.
4. Nikhil Bansal, Ola Svensson, and Luca Trevisan. New notions and constructions of sparsification for graphs and hypergraphs. In *60th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2019, Baltimore, Maryland, USA, November 9-12, 2019*, pages 910–928, 2019.
5. Joshua D. Batson, Daniel A. Spielman, and Nikhil Srivastava. Twice-ramanujan sparsifiers. *SIAM J. Comput.*, 41(6):1704–1721, 2012.
6. András A. Benczúr and David R. Karger. Approximating s-t minimum cuts in $\tilde{O}(n^2)$ time. In *Proceedings of the Twenty-Eighth Annual ACM Symposium on the Theory of Computing, Philadelphia, Pennsylvania, USA, May 22-24, 1996*, pages 47–55, 1996.
7. András A. Benczúr and David R. Karger. Randomized approximation schemes for cuts and flows in capacitated graphs. *SIAM J. Comput.*, 44(2):290–319, 2015.

8 Ümit V. Çatalyürek and Cevdet Aykanat. Hypergraph-partitioning-based decomposition for parallel sparse-matrix vector multiplication. *IEEE Trans. Parallel Distrib. Syst.*, 10(7):673–693, 1999.

9 Ümit V. Çatalyürek, Erik G. Boman, Karen D. Devine, Doruk Bozdag, Robert T. Heaphy, and Lee Ann Riesen. A repartitioning hypergraph model for dynamic load balancing. *J. Parallel Distributed Comput.*, 69(8):711–724, 2009.

10 Chandra Chekuri and Chao Xu. Minimum cuts and sparsification in hypergraphs. *SIAM J. Comput.*, 47(6):2118–2156, 2018.

11 Yu Chen, Sanjeev Khanna, and Ansh Nagda. Near-linear size hypergraph cut sparsifiers. In *61st IEEE Annual Symposium on Foundations of Computer Science, FOCS*. IEEE Computer Society, 2020.

12 Wai Shing Fung, Ramesh Hariharan, Nicholas J. A. Harvey, and Debmalya Panigrahi. A general framework for graph sparsification. *SIAM J. Comput.*, 48(4):1196–1223, 2019.

13 Ashish Goel, Michael Kapralov, and Ian Post. Single pass sparsification in the streaming model with edge deletions. *CoRR*, abs/1203.4900, 2012. `arXiv:1203.4900`.

14 Yuchi Huang, Qingshan Liu, and Dimitris N. Metaxas. Video object segmentation by hypergraph cut. In *2009 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR 2009), 20-25 June 2009, Miami, Florida, USA*, pages 1738–1745, 2009.

15 Michael Kapralov, Yin Tat Lee, Cameron Musco, Christopher Musco, and Aaron Sidford. Single pass spectral sparsification in dynamic streams. *SIAM J. Comput.*, 46(1):456–477, 2017.

16 Michael Kapralov, Aida Mousavifar, Cameron Musco, Christopher Musco, Navid Nouri, Aaron Sidford, and Jakab Tardos. Fast and space efficient spectral sparsification in dynamic streams. In Shuchi Chawla, editor, *Proceedings of the 2020 ACM-SIAM Symposium on Discrete Algorithms, SODA 2020, Salt Lake City, UT, USA, January 5-8, 2020*, pages 1814–1833. SIAM, 2020.

17 David R. Karger. Global min-cuts in RNC, and other ramifications of a simple min-cut algorithm. In *Proceedings of the Fourth Annual ACM/SIGACT-SIAM Symposium on Discrete Algorithms, 25-27 January 1993, Austin, Texas, USA.*, pages 21–30, 1993.

18 David R. Karger. Random sampling in cut, flow, and network design problems. *Math. Oper. Res.*, 24(2):383–413, 1999.

19 Dmitry Kogan and Robert Krauthgamer. Sketching cuts in graphs and hypergraphs. In *Proceedings of the 2015 Conference on Innovations in Theoretical Computer Science, ITCS 2015, Rehovot, Israel, January 11-13, 2015*, pages 367–376, 2015.

20 Yin Tat Lee, Aaron Sidford, and Sam Chiu-wai Wong. A faster cutting plane method and its implications for combinatorial and convex optimization. In *IEEE 56th Annual Symposium on Foundations of Computer Science, FOCS 2015, Berkeley, CA, USA, 17-20 October, 2015*, pages 1049–1065, 2015.

21 Yin Tat Lee and He Sun. An sdp-based algorithm for linear-sized spectral sparsification. In Hamed Hatami, Pierre McKenzie, and Valerie King, editors, *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2017, Montreal, QC, Canada, June 19-23, 2017*, pages 678–687. ACM, 2017.

22 Anand Louis. Hypergraph markov operators, eigenvalues and approximation algorithms. In *Proceedings of the Forty-Seventh Annual ACM on Symposium on Theory of Computing, STOC 2015, Portland, OR, USA, June 14-17, 2015*, pages 713–722, 2015.

23 Tasuku Soma and Yuichi Yoshida. Spectral sparsification of hypergraphs. In Timothy M. Chan, editor, *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2019, San Diego, California, USA, January 6-9, 2019*, pages 2570–2581. SIAM, 2019.

24 Daniel A. Spielman and Nikhil Srivastava. Graph sparsification by effective resistances. *SIAM J. Comput.*, 40(6):1913–1926, 2011.

25 Daniel A. Spielman and Shang-Hua Teng. Nearly-linear time algorithms for graph partitioning, graph sparsification, and solving linear systems. In László Babai, editor, *Proceedings of the 36th Annual ACM Symposium on Theory of Computing, Chicago, IL, USA, June 13-16, 2004*, pages 81–90. ACM, 2004.

26 Yutaro Yamaguchi, Anna Ogawa, Akiko Takeda, and Satoru Iwata. Cyber security analysis of power networks by hypergraph cut algorithms. *IEEE Trans. Smart Grid*, 6(5):2189–2199, 2015.

27 Yuichi Yoshida. Cheeger inequalities for submodular transformations. In *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2019, San Diego, California, USA, January 6-9, 2019*, pages 2582–2601, 2019.

Streaming and Small Space Approximation Algorithms for Edit Distance and Longest Common Subsequence*

Kuan Cheng
Peking University, Beijing, China

Alireza Farhadi
University of Maryland, College Park, MD, USA

MohammadTaghi Hajiaghayi
University of Maryland, College Park, MD, USA

Zhengzhong Jin
Johns Hopkins University, Baltimore, MD, USA

Xin Li
Johns Hopkins University, Baltimore, MD, USA

Aviad Rubinstein
Stanford University, CA, USA

Saeed Seddighin
Toyota Technological Institute, Chicago, IL, USA

Yu Zheng
Johns Hopkins University, Baltimore, MD, USA

Abstract

The *edit distance* (ED) and *longest common subsequence* (LCS) are two fundamental problems which quantify how similar two strings are to one another. In this paper, we first consider these problems in the asymmetric streaming model introduced by Andoni, Krauthgamer and Onak [11] (FOCS'10) and Saks and Seshadhri [64] (SODA'13). In this model we have random access to one string and streaming access the other one. Our main contribution is a constant factor approximation algorithm for ED with memory $\tilde{O}(n^{\delta})$ for any constant $\delta > 0$. In addition to this, we present an upper bound of $\tilde{O}_{\epsilon}(\sqrt{n})$ on the memory needed to approximate ED or LCS within a factor $1 \pm \epsilon$. All our algorithms are deterministic and run in polynomial time in a single pass.

We further study small-space approximation algorithms for ED, LCS, and *longest increasing sequence* (LIS) in the non-streaming setting. Here, we design algorithms that achieve $1 \pm \epsilon$ approximation for all three problems, where $\epsilon > 0$ can be any constant and even slightly sub-constant. Our algorithms only use poly-logarithmic space while maintaining a polynomial running time. This significantly improves previous results in terms of space complexity, where all known results need to use space at least $\Omega(\sqrt{n})$. Our algorithms make novel use of triangle inequality and carefully designed recursions to save space, which can be of independent interest.

2012 ACM Subject Classification Theory of computation → Design and analysis of algorithms

Keywords and phrases Edit Distance, Longest Common Subsequence, Longest Increasing Subsequence, Space Efficient Algorithm, Approximation Algorithm

Digital Object Identifier 10.4230/LIPIcs.ICALP.2021.54

Category Track A: Algorithms, Complexity and Games

Related Version The full proofs of our results can be found in [36] and [31].
Full Version: https://arxiv.org/abs/2002.11342
Full Version: https://arxiv.org/abs/2002.08498

Funding *Kuan Cheng*: supported by a start-up funding of Peking University.
Alireza Farhadi: supported in part by Facebook Fellowship.
MohammadTaghi Hajiaghayi: supported in part by NSF BIGDATA grant IIS-1546108, and NSF SPX grant CCF-1822738.

* This paper is obtained by a recent merge of [36] and [31]. As a result of this joint effort, we not only improve the running times of all our algorithms to polynomial time. (Our streaming algorithms had exponential running times before.) But also extensively study algorithms for edit distance and LCS with small space in depth by achieving several novel results.

Zhengzhong Jin: supported by NSF Award CCF-1617713 and NSF CAREER Award CCF-1845349.
Xin Li: supported by NSF Award CCF-1617713 and NSF CAREER Award CCF-1845349.
Aviad Rubinstein: supported in part by a David and Lucile Packard Fellowship.
Saeed Seddighin: supported by an Adobe Research Award and a Google Research Gift.
Yu Zheng: supported by NSF Award CCF-1617713 and NSF CAREER Award CCF-1845349.

1 Introduction

We consider *edit distance* (ED) and *longest common subsequence* (LCS) which are classic problems measuring the similarity between two strings. Edit distance is defined on two strings s and $\bar{s}$ and seeks the smallest number of character insertions, character deletions, and character substitutions to transform s into $\bar{s}$. While in edit distance the goal is to make a transformation, longest common subsequence asks for the largest string that appears as a subsequence in both s and $\bar{s}$.

Edit distance and longest common subsequence have applications in various contexts, such as computational biology, text processing, compiler optimization, data analysis, image analysis, among others. As a result, both problems have been subject to a plethora of studies since 1950 (e.g. see [19, 20, 2, 14, 13, 11, 47, 17, 32, 34, 46, 57, 33, 42, 54, 23, 6, 26, 63, 24, 62, 48, 40, 38, 59, 3, 25, 45, 8, 10, 9, 7, 15, 39, 29, 53, 64, 43, 4, 21, 44, 26]).

Both problems are often used to measure the similarity of long strings. For example, a human genome consists of almost three billion base pairs that are modeled as a string for similarity testing. Classic algorithms for the problems require quadratic runtime as well as linear memory to find a solution. These bounds might be impractical for some real-world applications. Therefore, recent work on ED and LCS focus on obtaining fast algorithms [61, 44, 20, 13, 11, 8, 9, 60, 22, 52, 26] as well as solutions with small memory [45, 20, 28, 41].

As can be seen from the aforementioned references, while there have been a lot of previous works on obtaining fast algorithms, the equally important question of achieving algorithms using small space has not been studied in depth. In this paper our focus is to design algorithms with significantly improved space complexity for these string measures, in several different models.

Our first model is related to the *streaming* setting studied in most previous works on space complexity for ED and LCS. This is an increasingly popular framework to model memory constraints. In this setting, the input arrives as a data stream while only sublinear memory is available to the algorithm. The goal is to design an algorithm that solves/approximates the solution by taking a few passes over the data. While several works have studied ED and LCS in the streaming model (see Section 1.1 for a detailed discussion), positive results are known only for the low-distance regime [56, 66, 18, 28, 27]. In addition to this, strong lower bounds are given for the streaming variant of LCS [56, 66].

Inspired by the work of [11] (FOCS'10), Saks and Seshadhri [64] (SODA'13) studied the problem of approximating $n -$ LCS (which is half of the the edit distance between two strings of length n when only insertions and deletions are allowed) in the asymmetric model. In this model we have random access to one of the strings (say $\bar{s}$) and streaming access to the other string (say s). They showed that $(1+\epsilon)$ approximation of $n -$ LCS can be found with a memory of $\tilde{O}_\epsilon(\sqrt{n})$. Another work by Saha [63] gives an algorithm for ED in this setting, using a memory of $O(\frac{\sqrt{n}}{\epsilon})$, and achieving an ϵn *additive* approximation.

In this work, we study ED and LCS in the asymmetric streaming model. We present a single-pass deterministic constant factor approximation algorithm for ED that uses only $\tilde{O}(n^\delta)$ memory for any constant $\delta > 0$. In addition to this, we show that with the memory of $\tilde{O}_\epsilon(\sqrt{n})$ one can approximate both ED and LCS within a factor of $1 \pm \epsilon$. All our algorithms

are deterministic and run in a single-pass and in polynomial time. Moreover, the memory bound of our algorithm for LCS is tight due to a lower bound given in [37]. It is also worth mentioning that the lower bound of $\Omega(\log^2 n/\epsilon)$ is known for computing $1+\epsilon$ approximation of $n-$ LCS due to the result of [58].

LIS and distance to monotonicity (DTM) are special cases of LCS and ED that are also studied in the streaming model [41, 64]. In these two problems, one of the strings is a length n string over the alphabet $[n]$ and the second string is the sorted permutation $\langle 1, 2, \ldots, n\rangle$. Therefore, any asymmetric streaming algorithm for LCS and ED also implies a classical streaming algorithm for LIS and DTM, since we can assume that we have random access to the sorted permutation $\langle 1, 2, \ldots, n\rangle$. As a result, our algorithms for ED and LCS can be seen as a generalization of [41, 64] on streaming LIS and distance to monotonicity.

More generally, we also study the memory requirements of ED, LCS, and LIS in the non-streaming model, where we have random access to all input strings. Even in this setting, classic algorithms that compute ED and LCS exactly using dynamic programming need linear space. A recent work [50] slightly improves the space complexity to $O(\frac{n\log^{1.5} n}{2^{\sqrt{\log n}}})$ while preserving a polynomial running time, however achieving strongly sublinear space (i.e., space $n^{1-\alpha}$ for some constant $\alpha > 0$) with a polynomial running time for ED and LCS still seems out of reach. Therefore, in this paper we turn to the relaxed goal of approximating ED and LCS using significantly smaller space while preserving a polynomial running time.

The only previous positive results in this setting we are aware of, are the previously mentioned works on streaming algorithms and the work of Saha [63]. Again, for ED the streaming results only work in the low-distance regime, except the work of Saks and Seshadhri [64] which gives a $(1+\epsilon)$ approximation of $n-$ LCS using $\tilde{O}_\epsilon(\sqrt{n})$ space. Additionally, [64] also gives a randomized algorithm that achieves an ϵn additive approximation of LCS in this model, using space $O(k\log^2 n/\epsilon)$ where k is the maximum number of times any symbol appears in y. [63] gives a small space algorithm for ED, although the result only works for constant alphabet size and only provides an additive approximation instead of multiplicative approximation. Further the space needed is at least $\Omega(n^{2/3})$. For LIS the situation is slightly better. In particular, the work of Gopalan, Jayram, Krauthgamer and Kumar [41] provides a deterministic streaming algorithm that approximates LIS to within a $1-\epsilon$ factor, using time $O(n\log n)$ and space $O(\sqrt{n/\epsilon}\log n)$; while a very recent work by Kiyomi, Ono, Otachi, Schweitzer and Tarui [51] obtains a deterministic algorithm that computes LIS *exactly* using $O(n^{1.5}\log n)$ time and $O(\sqrt{n}\log n)$ space. [63] in addition provides an algorithm for LIS using space $O(\frac{\log n}{\epsilon})$ that achieves an ϵn additive approximation. Thus here we seek to obtain an $1-\epsilon$ approximation of LIS using space much smaller than $\sqrt{n}$.

More broadly, the questions studied in this paper are closely related to the general question of *non-deterministic small space computation* vs. *deterministic small space computation*. Specifically, the decision versions of all three problems (ED, LCS, and LIS) can be easily shown to be in the class NL (i.e., non-deterministic log-space), and the question of whether NL = L (i.e., if non-deterministic log-space computation is equivalent to deterministic log-space computation) is a major open question in complexity theory. Note that if NL = L, this would trivially imply polynomial time algorithms for *exactly* computing ED, LCS, and LIS in logspace. However, although we know that NL $\subseteq$ P and NL $\subseteq$ SPACE$(\log^2 n)$ (by Savitch's theorem [65]), it is not known if every problem in NL can be solved simultaneously in polynomial time and polylog space, i.e., if NL $\subseteq$ SC where SC is Steve's class. In fact, it is not known if an NL-complete language (e.g., directed s-t connectivity) can be solved simultaneously in polynomial time and strongly sub linear space (i.e., space $n^{1-\alpha}$ for some fixed constant $\alpha > 0$). Thus, studying special problems such as ED, LCS, and LIS, and the relaxed version of approximation is a reasonable first step towards major open problems.

In the non-streaming setting, for all three problems ED, LCS, and LIS, we give efficient *deterministic* approximation algorithms that can achieve $1 \pm \epsilon$ approximation, using even polylog(n) space while maintaining a polynomial running time. By relaxing the space complexity to n^δ for any constant $\delta > 0$, we obtain algorithms whose running time is comparable to the standard dynamic programming approach. This is in sharp contrast to the time complexity of ED, LCS, where we only know how to beat the standard dynamic programming significantly by using *randomized* algorithms and with much worse approximation guarantees.

Last but not least, it turns out that we can also use our techniques developed for the non-streaming model to significantly reduce the running time of our algorithm in the asymmetric streaming model, resulting in a polynomial time algorithm.

1.1 Related work

Quadratic time solutions for ED and LCS have been known for many decades [55]. Recently, it has been shown that a truly subquadratic time solution for either ED or LCS refutes *Strong Exponential Time Hypothesis* (SETH), a conjecture widely believed in the community (see [14, 2, 25]). Therefore, much attention is given to approximation algorithms for the two problems. For edit distance, a series of works [54], [16], [17], and [13] improve the approximation factor culminating in the seminal work of Andoni, Krauthgamer, and Onak [11] that finally obtains a polylogarithmic approximation factor in near-linear time. More recently constant factor approximation algorithms with truly subquadratic runtimes are obtained for edit distance (a question which was open for a few decades): first a quantum algorithm [20], then a classic solution [26], and finally near linear time solutions are given [22, 52, 12]. LCS has also received tremendous attention in recent years [44, 60, 61, 1, 4, 30]. Only trivial solutions were known for LCS until very recently: a 2 approximate solution when the alphabet is 0/1 and an $O(\sqrt{n})$ approximate solution for general alphabets in linear time. Both these bounds are recently improved by [44] and Rubinstein and Song [60] (see also a recent approximation algorithms given by [61]).

Streaming algorithms for edit distance have been limited to the case that the distance between the two strings is *bounded* by a parameter k which is substantially smaller than n. In particular, [28] gives a randomized one-pass algorithm in a variant of the streaming model where one can scan the two strings s and $\bar{s}$ simultaneously in the coordinated fashion, using space $O(k^6)$ and time $O(n + k^6)$. This was later improved to space $O(k)$ and time $O(n + k^2)$ in [27, 18]. [18] further gives a randomized one-pass algorithm in the standard streaming model using space $O(k^8 \operatorname{polylog} n)$ and time $\tilde{O}(k^2 n)$. It is improved to $O(k^3 \operatorname{polylog} n)$ space recently in [49]. Note that in all these algorithms the space can be as large as n.

Table 1 The results for asymmetric streaming model of this paper along with previous work.

problem	approximation factor	memory	reference
ED	$O(2^{1/\delta})$	$\tilde{O}(n^\delta/\delta)$	Theorem 1
ED	$1+\epsilon$	$\tilde{O}_\epsilon(\sqrt{n})$	Theorem 2
LCS	$1-\epsilon$	$\tilde{O}_\epsilon(\sqrt{n})$	Theorem 2
LIS	$1-\epsilon$	$\tilde{O}_\epsilon(\sqrt{n})$	[41] (SODA'07)
n − LCS	$1+\epsilon$	$\tilde{O}_\epsilon(\sqrt{n})$	[64] (SODA'13)
DTM	$1+\epsilon$	$O_\epsilon(\log^2 n)$	[64, 58] (SODA'13, SODA'14)
DTM	$1+\epsilon$	$\tilde{O}_\epsilon(\sqrt{n})$	[41] (SODA'07)
DTM	2	$O(\log^2 n)$	[35] (SODA'08)
DTM	4	$O(\log^2 n)$	[41] (SODA'07)

1.2 Our Results

We summarize our main theorems in this section and in Table 1. In the next subsection we demonstrate our main techniques and general ideas in details. Formal proofs are deferred to the appendix. First we start with the following results for the asymmetric streaming model.

▶ **Theorem 1.** *Given online string s and offline string $\bar{s}$ both in Σ^n, for any constant $\delta > 0$, there exists a deterministic algorithm that, making one pass through s, outputs a $O(2^{1/\delta})$ approximation of $\mathsf{ED}(s, \bar{s})$ using $\tilde{O}(n^\delta/\delta)$ space and $\tilde{O}_\delta(n^4)$ time.*

▶ **Theorem 2.** *Given online string s and offline string $\bar{s}$ both in Σ^n, for any constant $\epsilon \in (0, 1)$, there is a deterministic algorithm that, making one pass through s, outputs a $(1-\epsilon)$-approximation of $\mathsf{LCS}(s, \bar{s})$ using $\tilde{O}(\frac{\sqrt{n}}{\epsilon})$ space, or $(1+\epsilon)$-approximation of $\mathsf{ED}(s, \bar{s})$ using $\tilde{O}(\sqrt{\frac{n}{\epsilon}})$ space, in polynomial time.*

Theorem 1 is surprising in the sense that by allowing random access to one string, we can design an efficient deterministic one pass streaming algorithm achieving a constant approximation of ED with space n^δ for any constant $\delta > 0$. Previously no such positive results with sublinear space are known in the full streaming model even for randomized algorithms. The proof of this theorem makes novel uses of triangle inequality, and combines the techniques we develop for the non-streaming model. Theorem 2 further shows that we can in fact achieve an $(1+\epsilon)$-approximation of ED, albeit using a larger space $(\tilde{O}(\sqrt{\frac{n}{\epsilon}}))$. The result also extends to give the first $(1-\epsilon)$-approximation of LCS with sublinear space in the asymmetric streaming model, and the space matches the lower bound implied from the lower bound for LIS in [37].

Next we bring our results for the (non-streaming) small space model.

▶ **Theorem 3.** *Given any strings x and y each of length n, there are deterministic algorithms that approximate $\mathsf{ED}(x, y)$ with the following parameters:*

1. *For any constants $\delta \in (0, \frac{1}{2})$ and $\epsilon \in (0, 1)$, an algorithm that outputs a $1+\epsilon$ approximation of $\mathsf{ED}(x, y)$ using $\tilde{O}_{\epsilon,\delta}(n^\delta)$ space and $\tilde{O}_{\epsilon,\delta}(n^2)$ time.*
2. *An algorithm that outputs a $1 + O(\frac{1}{\log\log n})$ approximation of $\mathsf{ED}(x, y)$ using $O(\frac{\log^4 n}{\log\log n})$ space and $n^{7+o(1)}$ time.*

Note that our first algorithm for ED uses roughly the same running time as the standard dynamic programming, but much smaller space. Indeed, we can use space n^δ for any constant $\delta > 0$. This also significantly improves the previous result of [64], which needs to use space $\Omega(\sqrt{n}\log n)$ and only provides a $2+\epsilon$ approximation for standard ED. With a larger (but still polynomial) running time, we can achieve space complexity $O(\frac{\log^4 n}{\log\log n})$.

▶ **Theorem 4.** *Given any strings x and y each of length n, there are deterministic algorithms that approximate $\mathsf{LCS}(x, y)$ with the following parameters:*

1. *For any constants $\delta \in (0, \frac{1}{2})$ such that $\frac{1}{\delta}$ is an integer, and $\epsilon \in (0, 1)$,an algorithm that outputs a $1-\epsilon$ approximation of $\mathsf{LCS}(x, y)$ using $\tilde{O}_{\epsilon,\delta}(n^\delta)$ space and $\tilde{O}_{\epsilon,\delta}(n^{3-\delta})$ time.*
2. *An algorithm that outputs a $1 - O(\frac{1}{\log\log n})$ approximation of $\mathsf{LCS}(x, y)$, using $O(\frac{\log^4 n}{\log\log n})$ space and $n^{6+o(1)}$ time.*

To the best of our knowledge, Theorem 4 (together with Theorem 2) is the first $1-\epsilon$ approximation of LCS using truly sub-linear space, and in fact we can achieve space n^δ for any constant $\delta > 0$ with only a slightly larger running time than the standard dynamic programming approach. We can achieve space $O(\frac{\log^4 n}{\log\log n})$ with an even larger (but still polynomial) running time. For LIS, we have a similar theorem.

▶ **Theorem 5.** *Given any string x of length n, there are deterministic algorithms that approximate $\mathsf{LIS}(x)$ with the following parameters:*

1. *For any constants $\delta \in (0, \frac{1}{2})$ such that $\frac{1}{\delta}$ is an integer, and $\epsilon \in (0,1)$, an algorithm that computes a $1-\epsilon$ approximation of $\mathsf{LIS}(x)$ using $\tilde{O}_{\epsilon,\delta}(n^{\delta})$ space and $\tilde{O}_{\epsilon,\delta}(n^{2-2\delta})$ time.*
2. *An algorithm that outputs a $1 - O(\frac{1}{\log\log n})$ approximation of $\mathsf{LIS}(x)$ using $O(\frac{\log^4 n}{\log\log n})$ space and $n^{5+o(1)}$ time.*

2 Preliminaries

Let Σ be an alphabet. For a string $s \in \Sigma^n$, we use $s[i]$ to denote the i^{th} character in s. We use $s[i,j]$ to denote the substring of s from the i^{th} character to the j^{th} character. We also use $s[i,j)$ to denote the substring of s from the i^{th} character to $(j-1)^{\text{th}}$ character ($s[i,i)$ is an empty string). We denote the concatenation of two strings x and y by $x \circ y$. We use two special symbols ∞ and $-\infty$ with $\infty > i$ and $-\infty < i$ for any character $i \in \Sigma$.

Given two strings s and $\bar{s}$ in Σ^n, the longest common subsequence (LCS) of s and $\bar{s}$ is a string t with the maximum length such that t is a subsequence of both s and $\bar{s}$. In other words, t can be obtained from both s and $\bar{s}$ by removing some of the characters. We use $\mathsf{LCS}(s,\bar{s})$ to denote the length of the LCS of two strings s and $\bar{s}$. The edit distance (ED) between two strings s and $\bar{s}$, denoted by $\mathsf{ED}(s,\bar{s})$, is the minimum number of character insertions, deletions, and substitutions needed to transform one string to the other string. The longest increasing subsequence (LIS) problem is defined as follows. We assume there is a given total order on the alphabet set Σ. We say the string $x \in \Sigma^t$ is an *increasing subsequence* of $s \in \Sigma^n$ if there exists indices $1 \le i_1 < i_2 < \cdots < i_t \le n$ such that $x = s_{i_1} s_{i_2} \cdots s_{i_t}$ and $s_{i_1} < s_{i_2} < \cdots < s_{i_t}$. We denote the length of the longest increasing subsequence of string s by $\mathsf{LIS}(s)$.

Asymmetric streaming model. Throughout this paper, we assume that the input of the algorithm consists of two strings $\bar{s}$ and s. We assume for simplicity and without loss of generality that the two strings have equal length n. We call the string $\bar{s}$ the *offline* string and assume that the algorithm has random access to the characters of $\bar{s}$ via making a query. The other string s arrives as a stream of characters. We call s the *online* string.

3 Constant Approximation for Streaming Edit distance

Our main results in the asymmetric streaming setting is an algorithm that given any constant $\delta > 0$ finds a constant approximation of the edit distance using $\tilde{O}(n^{\delta})$ memory. As we discussed in the previous section, instead of directly solving the edit distance, we aim to find a substring of s such that its edit distance is smallest to $\bar{s}$. We can this problem *Closest Substring*. It is defined as follows. It takes two inputs: an offline string $\bar{s}$ and an online string s. The goal is to find two indices l, r and $\mathsf{ED}(\bar{s}[l,r],s)$ such that $\mathsf{ED}(\bar{s}[l,r],s) \le \mathsf{ED}(\bar{s}[i,j],s)$ for every $1 \le i \le j \le n$.

We first show that how solving the closest substring problem can give us a good approximation of the edit distance. Let $\bar{s}[l,r]$ be the substring of $\bar{s}$ with the minimum edit distance to s. We know by the definition of edit distance that it satisfies the *triangle inequality*[1].

[1] $\mathsf{ED}(s_1,s_3) \le \mathsf{ED}(s_1,s_2) + \mathsf{ED}(s_2,s_3)$ for any strings s_1, s_2, s_3.

Therefore, we have

$$\mathsf{ED}(\bar{s}, s) \leq \mathsf{ED}(\bar{s}, \bar{s}[l, r]) + \mathsf{ED}(\bar{s}[l, r], s)\,. \tag{1}$$

We also have,

$$\begin{aligned}
&\mathsf{ED}(\bar{s}, \bar{s}[l, r]) + \mathsf{ED}(\bar{s}[l, r], s) \\
&\quad \leq \mathsf{ED}(\bar{s}, s) + \mathsf{ED}(\bar{s}[l, r], s) + \mathsf{ED}(\bar{s}[l, r], s) && \text{By the triangle inequality.} \\
&\quad \leq 3\mathsf{ED}(\bar{s}, s) && \text{Since } \bar{s}[l, r] \text{ has the minimum ED to } s.
\end{aligned} \tag{2}$$

It follows from (1) and (2) that $\mathsf{ED}(\bar{s}, \bar{s}[l, r]) + \mathsf{ED}(\bar{s}[l, r], s)$ is a 3-approximation of the edit distance between $\bar{s}$ and s. Therefore, if we design a streaming algorithm that finds $\bar{s}[l, r]$ and its edit distance from s, we can then estimate the edit distance of s and $\bar{s}$ by computing $\mathsf{ED}(\bar{s}, \bar{s}[l, r]) + \mathsf{ED}(\bar{s}[l, r], s)$. In the following theorem which directly implies from Savitch's theorem [65], we show that $\mathsf{ED}(\bar{s}, \bar{s}[l, r]) + \mathsf{ED}(\bar{s}[l, r], s)$ can be computed using a poly-logarithmic memory. In specific, we show that the edit distance between any two substrings of the offline string can be computed using a very small memory of $O(\log^2 n)$. The proof is deferred to the full version [36]

▶ **Theorem 6.** *Suppose that we have random access to two given strings s and $\bar{s}$ of length n. Then $\mathsf{LCS}(s, \bar{s})$ and $\mathsf{ED}(s, \bar{s})$ can be computed using $O(\log^2 n)$ memory.*

Therefore, by finding the substring that has the minimum edit distance to s, we can get a good approximation of the edit distance. Nonetheless, we do not know any streaming algorithm with the memory of $O(n^\delta)$ for finding closest substring, and our algorithm only finds an approximate solution for this problem. In other words, it finds a substring of $\bar{s}$ such that its approximate edit distance to s is close to the minimum. In the rest of the section, we show that how we can approximately solve the closest substring problem with the memory of $\tilde{O}(n^\delta)$. Given an online string, we divide the online string into $n^{1-\delta}$ windows of size n^δ. Our algorithm (formally as Algorithm 1), then recursively finds substrings of $\bar{s}$ that have the minimum edit distance from each of these windows. Note that for each window we can store the result of solving the closest substring problem in $O(\log n)$ (We can store only three numbers which are the start and the end of the interval and the approximate edit distance to the online string). Therefore, by the end of all recursive calls our algorithm needs to store $O(n^\delta)$ values.

In order to find the solution of the closest substring problem using these partial solutions, our algorithm considers all different substrings $\bar{s}[l, r]$ of $\bar{s}$ and all different mappings between the windows of the s and the substrings of $\bar{s}[l, r]$. Then, for any mapping it estimates the edit distance between a window of s and its mapped substring of $\bar{s}[l, r]$ using the solution of the closest substring problem that we have found in the recursive call.

In order to analyze our algorithm, we first show that finding any approximation of the closest substring problem, can yield us an approximation for the edit distance. We first define an approximate version of the closest substring problem as follows.

▶ **Definition 7.** *Given an offline string $\bar{s}$ and online string s, we say that the substring $\bar{s}[l, r]$ along with its approximate edit distance d is an α-approximation for the closest substring problem if for any substring $\bar{s}[l^*, r^*]$ we have*

$$\mathit{ED}(\bar{s}[l, r], s) \leq d \leq \alpha \cdot \mathit{ED}(\bar{s}[l^*, r^*], s)\,. \tag{3}$$

Algorithm 1 Algorithm APPROXIMATE-CLOSEST-SUBSTR for approximating ED.

Data: An offline string $\bar{s}$ of length n, a stream of characters of the online string s, and a parameter $\delta > 0$.

if $|s| \leq n^{\delta}$ **then**
  Store all characters of s in the memory.
  Find a substring of $\bar{s}$ that has the minimum edit distance to s. Let $\bar{s}[l, r]$ be this substring and d be its edit distance.
  return l, r and d.
else
  $\xi \leftarrow n^{\delta}$.
  Divide s into ξ windows $s_1^*, s_2^*, \ldots, s_{\xi}^*$ of size $|s|/\xi$.
  for $i \in [\xi]$ **do**
    Recursively find the closest substring of $\bar{s}$ from s_i^* . Let l_i, r_i be the start and the end of this substring respectively, and d_i be the approximate edit distance of this substring to s_i^*.
  $min\_dist \leftarrow \infty$.
  for $1 \leq p_0 \leq p_1 \leq \ldots \leq p_{\xi} \leq n + 1$ **do**
    $dist = \sum_{i=1}^{\xi} d_i + \mathsf{ED}\big(\bar{s}[p_{i-1}, p_i), \bar{s}[l_i, r_i]\big)$.
    if $dist < min\_dist$ **then**
      $min\_dist \leftarrow dist$.
      $l \leftarrow p_0$.
      $r \leftarrow p_{\xi} - 1$.
  return l, r and min_dist.

In the following claim we show that we can use any α-approximation of the closest substring problem to get a $O(\alpha)$-approximation for the edit distance.

▷ Claim 8. Let $\bar{s}[l, r]$ be an α approximation of the closest substring problem and let d be its approximate edit distance to s. Then for any substring $\bar{s}[l^*, r^*]$, $d + \mathsf{ED}\big(\bar{s}[l, r], \bar{s}[l^*, r^*]\big)$ is a $(2\alpha + 1)$-approximation for the edit distance between $\bar{s}[l^*, r^*]$ and s.

Proof. First we show that $\big(d + \mathsf{ED}\big(\bar{s}[l, r], \bar{s}[l^*, r^*]\big)\big)$ is not less than the edit distance between $\bar{s}[l^*, r^*]$ and s.

$$\begin{aligned} d + \mathsf{ED}\big(\bar{s}[l, r], \bar{s}[l^*, r^*]\big) &\geq \mathsf{ED}(\bar{s}[l, r], s) + \mathsf{ED}\big(\bar{s}[l, r], \bar{s}[l^*, r^*]\big) && \text{By (3).} \\ &\geq \mathsf{ED}\big(\bar{s}[l^*, r^*], s\big)\,. && \text{By the triangle inequality.} \end{aligned}$$

We now show that the value of $\big(d + \mathsf{ED}\big(\bar{s}[l, r], \bar{s}[l^*, r^*]\big)\big)$ is at most $(2\alpha + 1) \cdot \mathsf{ED}\big(s, \bar{s}[l^*, r^*]\big)$. Thus, it gives us a $(2\alpha + 1)$-approximation of the edit distance. We have

$$\begin{aligned} d + \mathsf{ED}\big(\bar{s}[l, r], \bar{s}[l^*, r^*]\big) &\leq d + \mathsf{ED}\big(s, \bar{s}[l, r]\big) + \mathsf{ED}\big(s, \bar{s}[l^*, r^*]\big) && \text{By the triangle inequality.} \\ &\leq d + \alpha \cdot \mathsf{ED}\big(s, \bar{s}[l^*, r^*]\big) + \mathsf{ED}\big(s, \bar{s}[l^*, r^*]\big) && \text{By (3).} \\ &= d + (\alpha + 1) \cdot \mathsf{ED}\big(s, \bar{s}[l^*, r^*]\big) \\ &\leq \alpha \cdot \mathsf{ED}\big(s, \bar{s}[l^*, r^*]\big) + (\alpha + 1) \cdot \mathsf{ED}\big(s, \bar{s}[l^*, r^*]\big) && \text{By (3).} \\ &= (2\alpha + 1) \cdot \mathsf{ED}\big(s, \bar{s}[l^*, r^*]\big)\,, \end{aligned}$$

which completes the proof of the claim. ◁

Based on our discussion above, we design an algorithm that finds a constant approximation of the edit distance using $\tilde{O}(n^\delta)$ memory for any $\delta > 0$. The algorithm first divides the online string into n^δ windows with the equal length. Therefore, the length of each window is $n^{1-\delta}$. It then finds an approximate solution of the closest substring problem for each window recursively. By Claim 8, we can use the approximate solution of the closest substring problem for each window, to find its edit distance from every other substring of the offline string. The algorithm uses these approximate solutions to approximate the edit distance between the entire online string and any substring of the offline string.

Note that by each recursive call the length of the online string will get smaller by a multiplicative factor of $n^{-\delta}$. Therefore, when the depth of the recursive calls becomes $1/\delta$, the length of the remaining online string is bounded by $O(n^\delta)$ and we can store all of this remaining online string in the memory and find the exact solution of the closest substring problem. Thus, the depth of the recursion is bounded by $O(1/\delta)$. In the following theorem we show that the approximation ratio of our algorithm is $O(2^{1/\delta})$.

▶ **Theorem 9.** *Given an offline string $\bar{s}$, an online string s and any constant $\delta > 0$, let n be the length of the offline string and n^γ be the length of the online string where $\gamma > 0$. Then, Algorithm 1 finds a $O\left(2^{\gamma/\delta}\right)$ approximation for the closest substring problem.*

The full proof of Theorem 9 is deferred to the full version [36]. Now we are ready to prove a version of Theorem 1 without the time complexity bound. We introduce how to achieve polynomial runtime in Section 5.2.

Proof of Theorem 1 without the time complexity bound. By Theorem 9, Algorithm 1 finds a $O(2^{1/\delta})$ approximation of the closest substring problem. Recall that by Theorem 6, we can find the edit distance of any two substrings of $\bar{s}$ using a very small memory. Therefore by Claim 8, we can find a $O(2^{1/\delta})$ approximation of the edit distance between s and $\bar{s}$.

Now we show that the memory of Algorithm 1 is at most $\tilde{O}(n^\delta/\delta)$. While the length of the online string is larger than n^δ, Algorithm 1 divides the online string into n^δ windows and recursively solves the closest substring problem for each window. Therefore, by each recursive call the length of the online string will decrease by a multiplicative factor of $n^{-\delta}$. Thus, the maximum depth of the recursive calls is bounded by $O(1/\delta)$. At each call the algorithm acquires a memory of $\tilde{O}(n^\delta)$ which is the memory needed for storing the result of the recursive calls and iterating over all possible mappings. Therefore, the memory of the algorithm is bounded by $\tilde{O}(n^\delta/\delta)$. ◀

4 $1+\epsilon$-Approximation of Streaming ED

A previous work by Saks and Seshadri [64] studied the approximation algorithm for deletion distance in the asymmetric streaming model. Here, the deletion distance (DD) is defined as the minimum number of insertions and deletions required to transform s to $\bar{s}$. Thus, assuming $|s| = |\bar{s}| = n$, $\mathsf{DD}(s, \bar{s}) = 2(n - \mathsf{LCS}(s, \bar{s}))$. [64] gives an algorithm that outputs a $(1+\epsilon)$ approximation of $\mathsf{DD}(s, \bar{s})$ for any constant $\epsilon > 0$ with $\tilde{O}_\epsilon(\sqrt{n})$ space.

▶ **Theorem 10** ([64])**.** *Given an offline string $\bar{s}$, an online string s, both with length n, and any constant $\epsilon \in (0, 1]$, there is a deterministic algorithm that outputs a $1+\epsilon$ approximation of $DD(\bar{s}, s)$ with $O(\sqrt{n \log n/\epsilon})$ space in polynomial time.*

Although deletion distance is not equivalent to edit distance we study in this paper, there is a simple transformation that reduces edit distance to deletion distance. The transformation can be found in previous works (see example 6.9 of [67] for example). More specifically,

for any given string s, we can obtain a new string s' by prepending a special symbol \$ (\$ is not the in the alphabet set) to each character of s. Thus, say $s = s_1 s_2 \cdots s_n$, we let $s' = \$s_1\$s_2 \cdots \$s_n \in (\Sigma \cup \{\$\})^{2n}$. We can obtain a string $\bar{s}'$ from $\bar{s}$ with the same transformation. The following lemma shows that the above transformation reduces computing edit distance to computing deletion distance.

▶ **Lemma 11.** *$DD(s', \bar{s}') = 2ED(s, \bar{s})$.*

Proof. We first show that $2\mathsf{ED}(s, \bar{s}) \geq \mathsf{DD}(s', \bar{s}')$. Assume we can transform s to $\bar{s}$ with d edit operations (insertion, deletion, and substitution), we show that it is possible to transform s' to $\bar{s}'$ with $2d$ insdel operations (insertion and deletion). To see this, if we delete one symbol from s, we delete the corresponding symbol in s' together with the \$ prepended to it. If we insert one symbol to s, we insert the corresponding symbol to s' together with a \$ prepended to it. If we substitute one symbol with another in s, we can first delete the corresponding symbol in s' and insert the new symbol at the same position.

We now show that $2\mathsf{ED}(s, \bar{s}) \leq \mathsf{DD}(s', \bar{s}')$. We consider an LCS between s' and $\bar{s}'$, and consider each pair of adjacent matches of \$ in this LCS. Without loss of generality, we can assume that in at least one side (say s'), this pair looks like $\$a\$$ where a is a symbol (we can also append a \$ at the end if needed), because otherwise if both sides have at least two symbols between the \$'s, then there is another pair of \$'s in the middle that can be matched and added to the LCS. Suppose now in $\bar{s}'$ there are t non-\$ symbols between the two \$'s. We have two cases: 1. If in the LCS, a is matched to one of the t symbols in $\bar{s}$. Then the number of insdel operations between s' and $\bar{s}'$ we need for this part is $2t - 2$, while for the corresponding part of s and $\bar{s}$, we only need $t - 1$ deletions. 2. If in the LCS a is not matched to any of the t symbols in $\bar{s}$. Then the number of insdel operations between s' and $\bar{s}'$ we need for this part is $2t$, while for s and $\bar{s}$ we need $t - 1$ deletions and one substitution (we can substitute a for any of the t symbols in $\bar{s}$), so that's t operations. Thus, if we can transform s' to $\bar{s}'$ with d insdel operations, we can transform s to $\bar{s}$ with $\frac{d}{2}$ edit operations. ◀

Note that the reduction can be implemented in a streaming manner, thus the following theorem is a direct result of the reduction and Theorem 10.

▶ **Theorem 12.** *Given an offline string $\bar{s}$, an online string s, both with length n, and any constant $\epsilon > 0$, there is a deterministic algorithm that outputs a $1 + \epsilon$ approximation of $ED(\bar{s}, s)$ with $\tilde{O}(\sqrt{\frac{n}{\epsilon}})$ space in polynomial time.*

5 Space Efficient Algorithms for ED in the Non-Streaming Model

5.1 Space Efficient algorithms for ED

In this section, we present our algorithm for approximating ED with small space. The pseudocode for our algorithm is given in Algorithm 2. Our algorithm is based on recursion. In each level of recursion, we use an idea from [45] to approximate the edit distance between certain pairs of substrings. We start by giving a brief description of the algorithm in [45].

Let x and y be two input strings such that $|x| = n$ and $|y| = m$. We assume a value Δ is given to us. If Δ is a $(1+\epsilon)$-approximation of $\mathsf{ED}(x, y)$, then the algorithm will output a good approximation of $\mathsf{ED}(x, y)$. Otherwise, the algorithm will output a value at least $\mathsf{ED}(x, y)$. Thus, we can try every $\Delta = \lceil (1 + \epsilon)^i \rceil \leq n$ for some integer i and take the minimum. This only increases the running time by a $\log_{1+\epsilon} n$ factor.

Given such a Δ, we first divide x into b blocks each of length $\frac{n}{b}$. [45] showed that, for each block of x that is not matched to a too large or too small interval in y, there is a way to choose $O(\frac{b}{\epsilon} \log_{1+\epsilon} n) = O(\frac{b}{\epsilon^2} \log n)$ candidate intervals such that one of them is close enough to the

Algorithm 2 Algorithm SpaceEfficientApproxED.

Data: Two strings x and y, parameters $b \leq \sqrt{n}$ and $\epsilon \in (0,1)$

```
if |x| ≤ b then
    compute ED(x, y) exactly.
    return ED(x, y).
ed ← ∞.
set n = |x| and m = |y|.
divide x into b block each of length at most ⌈n/b⌉ such that x = x^1 ∘ x^2 ∘ ··· ∘ x^b.
for all Δ = 0 or ⌈(1+ε)^j⌉ for some integer j and Δ ≤ max{|x|, |y|} do
    for i = 1 to b do
        for all (a, b) ∈ CandidateSet(n, m, (l_i, r_i), ε, Δ) do
            M(i, (a, b)) ← SpaceEfficientApproxED(x^i, y[a, b], b, ε).
    e ← min{e, DPEditDistance(n, m, b, ε, Δ, M)}.
return e.
```

optimal alignment. We compute the edit distance between each block and all of its candidate intervals, which gives $O(\frac{b^2}{\epsilon^2}\log n)$ values. After this, a simple dynamic programming gives $(1+O(\epsilon))$-approximation of the edit distance if Δ is a $(1+\epsilon)$-approximation of $\mathsf{ED}(x,y)$. The dynamic programming algorithm takes $O(\frac{b^3\log n}{\epsilon^3})$ time.

Since each block has length $\frac{n}{b}$, computing the edit distance of each block with one of its candidate intervals in y takes at most $O(\frac{n}{b}\log n)$ space (we assume each symbol can be stored with space $O(\log n)$). We can run this algorithm sequentially and reuse the space for each computation. Storing the edit distance of each pair takes $O(\frac{b^2}{\epsilon^2}\log^2 n)$ bits of space. Thus, if we take $b = n^{1/3}$, the above algorithm uses a total of $\tilde{O}_\epsilon(n^{2/3})$ space.

We now run the above algorithm recursively to further reduce the space required. Algorithm 2 takes four inputs: two strings $x, y \in \Sigma^n$, two parameters b, ϵ such that $b \leq \sqrt{n}$ and $\epsilon \in (0,1)$. The goal is to output a good approximation of $\mathsf{ED}(x,y)$ with small space (related to parameters b and ϵ). Similarly, we first divide x into b blocks. We try every Δ that is equal to $\lceil(1+\epsilon)^i\rceil$ for some integer i, and for each Δ there is a set of candidate intervals for each block of x. Then, for each block of x, we use a sub algorithm called *CandidateSet* to return all its $O(\frac{b}{\epsilon^2}\log n)$ candidate intervals. Instead of computing the edit distance between each block of x and its candidate intervals exactly, we recursively call our space efficient approximation algorithm with this pair as input, while keeping b and ϵ unchanged. After getting the results from the recursive calls, we combine these results with a dynamic programming (the sub algorithm *DPEditDistance*). We argue that if the recursive call outputs a $(1+\gamma)$-approximation of the actual edit distance, the output of the dynamic programming increases by at most a $(1+\gamma)$ factor. Thus if Δ is a $(1+\epsilon)$-approximation of $\mathsf{ED}(x,y)$, the output of the dynamic programming is guaranteed to be a $(1+O(\epsilon))(1+\gamma)$-approximation. The recursion stops whenever one of the input strings has length smaller than b, where we compute the edit distance exactly with $O(b\log n)$ space.

Notice that at each level of the recursion, the first input string is divided into b blocks if it has length larger than b. Thus the length of first input string at the i-th level of recursion is at most $\frac{n}{b^{i-1}}$. The depth of recursion is at most $d = \log_b n$.

At the d-th level, Algorithm 2 computes the edit distance exactly. Using this as a base case, we can show that the output of the i-th level of recursion is a $(1+O(\epsilon))^{d-i}$ approximation of the edit distance by induction on i from d to 1. Thus, the output in the first level is guaranteed to be a $(1+O(\epsilon))^d$-approximation. Since $d \leq \log_b n$, we get a $(1+O(\epsilon))^d = 1+O(\epsilon d) = 1+O(\epsilon\log_b n)$ approximation of $\mathsf{ED}(x,y)$.

To analyze the time and space complexity, we study the recursion tree in our algorithm. Notice that for each block x^i and each choice of Δ, we consider $O(\frac{b}{\epsilon^2}\log n)$ candidate intervals. Since there are b blocks and $O(\log_{1+\epsilon} n)$ choices of Δ, we need to solve $O(\frac{b^2}{\epsilon^3}\log^3 n)$ subproblems by recursion. Thus, the degree of the recursion tree is $O(\frac{b^2}{\epsilon^3}\log^3 n)$.

The dynamic programming at each level can be divided into b steps. At the j-th step, the inputs are the edit distances between block x^j and each of its candidate intervals. The information we need to maintain is an approximation of edit distances between the first $j-1$ blocks of x and the substrings $y[1,l]$ of y, where l is chosen from the set of starting points of the candidate intervals of x^j. There are $O(\frac{b}{\epsilon})$ choices for l and we query the approximated edit distance between x^j and each of its candidate intervals by recursively applying our algorithm. Thus, we only need to maintain $O(\frac{b}{\epsilon})$ values at any time for the dynamic programming.

At the i-th level of recursion, we either invoke one more level of recursion and maintain $O(\frac{b}{\epsilon})$ values where each value takes $O(\log n)$ space, or do an exact computation of edit distance when one of the input strings has length at most b, which takes $O(b\log n)$ space. Hence, the space used at each level is bounded by $O(\frac{b}{\epsilon}\log n)$. There are at most $d=\log_b n$ levels. The aggregated space used by our recursive algorithm is still $O(\frac{b\log^2 n}{\epsilon\log b})$.

We compute the running time by counting the number of nodes in the recursion tree. Notice that the number of nodes at level i is at most $(O(\frac{b^2}{\epsilon^3}\log^3 n))^{i-1}$. For each leaf node, we do exact computation with time $O(\frac{b^2}{\epsilon})$, and the number of leaf nodes is bounded by $(O(\frac{b^2}{\epsilon^3}\log^3 n))^{d-1}$. For each inner node, we run the dynamic programming $O(\log_{1+\epsilon} n)$ times (the number of choices of Δ) which takes $O(\frac{b^3\log^2 n}{\epsilon^4})$ time, and the number of inner nodes is bounded by $(d-1)(O(\frac{b^2}{\epsilon^3}\log^3 n))^{d-2}$.

If we take $b=\log n$ and $\epsilon=\frac{1}{\log n}$, we get a $(1+O(\frac{1}{\log\log n}))$-approximation using $O(\frac{\log^4 n}{\log\log n})$ space and $n^{7+o(1)}$ time. If we take $b=n^\delta$ for $\delta\in(0,\frac{1}{2})$ and ϵ a small constant, we get a $(1+O(\epsilon))$-approximation using $\tilde{O}_{\epsilon,\delta}(n^\delta)$ space and $\tilde{O}_{\epsilon,\delta}(n^2)$ time.

5.2 Improving the Runtime of Streaming ED

We can now use our techniques for small space approximation of ED to reduce the runtime of our algorithm for ED in the asymmetric streaming model proposed in Section 3. Recall that the most time consuming part of our streaming algorithm is finding the optimal window-compatible solution given a series of windows. More specifically, this can be formulated as follows. Suppose we divide the online string s into $b=n^\delta$ windows $\{s^i\}$, and in the recursion our algorithm already finds b windows $\{\bar{s}[l_i,r_i]\}$ in the offline string $\bar{s}$, that are approximately the closest to $\{s^i\}$. We now need to use the windows $\{\bar{s}[l_i,r_i]\}$ to find a substring in $\bar{s}$ that is approximately the closest to s. Previously, this is done by a brute force approach: we try all possible $1\le p_0\le p_1\le\ldots\le p_b\le n+1$ to find the set of p_i's that minimizes $\sum_{i=1}^b \mathsf{ED}(\bar{s}[p_{i-1},p_i),\bar{s}[l_i,r_i])$. Let the optimal set be $\{p_i^*\}$ and record the substring $\bar{s}[l,r]$ of $\bar{s}$ where $l=p_0^*$ and $r=p_b^*-1$. The exponential running time comes from two parts: First, the step of trying all possible $1\le p_0\le p_1\le\ldots\le p_b\le n+1$ needs to examine $\binom{n}{n^\delta+1}$ such choices. Second, when computing $\mathsf{ED}(\bar{s}[p_{i-1},p_i),\bar{s}[l_i,r_i])$, the $O(\log^2 n)$ algorithm from Savitch's theorem uses quasi-polynomial time.

Our main observation now is that, the step of trying all possible $1\le p_0\le p_1\le\ldots\le p_b\le n+1$ to find the set of p_i's that minimizes $\sum_{i=1}^b \mathsf{ED}(\bar{s}[p_{i-1},p_i),\bar{s}[l_i,r_i])$, is equivalent to finding the substring of $\bar{s}$ that minimizes the edit distance to the concatenation of $\bar{s}[l_i,r_i]$ from $i=1$ to b. Thus, instead of trying all possible p_i's, we can try all substrings of y (there

are only $O(n^2)$ such substrings) and compute the edit distance between each substring and the concatenation of the $\bar{s}[l_i, r_i]$'s. Furthermore, instead of an exact computation which either uses $\Omega(n)$ space or $2^{\Omega(\log^2 n)}$ time, we can use our $(1+\epsilon)$-approximation for ED with $\tilde{O}(n^\delta)$ space and $\tilde{O}(n^2)$ time. The approximation factor is now increased to $O((2+\epsilon)^{\frac{1}{\delta}})$, which is still $O(2^{\frac{1}{\delta}})$ if we take ϵ to be small enough. But the running time decreases to $\tilde{O}(n^4)$, and the space complexity remains $\tilde{O}(n^\delta/\delta)$.

6 $1-\epsilon$-Approximation of Streaming LCS

In this section, we design a streaming algorithm for finding a $(1-\epsilon)$ approximation of the LCS using $\tilde{O}(\sqrt{n}/\epsilon)$ memory. We first define the LCSPosition function as below. Given the online string s and the offline string $\bar{s}$, it takes a position p in $\bar{s}$ and a non-negative integer k as inputs. The output is the smallest position q such that $\mathsf{LCS}(\bar{s}[p,q], s[l,r]) \geq k$. If no such q exists, the output is ∞.

For a position p in $\bar{s}$, a substring $s[l,r]$ of s, and a non-negative integer k, we use $\text{LCSPosition}_{l,r}(p,k)$ to denote the result of the mentioned function.[2] It is easy to verify that the LCS of two strings $\bar{s}$ and s is equal to the largest k such that $\text{LCSPosition}_{1,n}(1,k) < \infty$. Therefore, instead of solving the LCS problem, we can solve the $\text{LCSPosition}_{1,n}$ problem and report the largest k such that $\text{LCSPosition}_{1,n}(1,k) < \infty$.

Algorithm 3 Algorithm for approximating the LCS in streaming model.

Data: An offline string $\bar{s}$ of length n, an online string s, and an $\epsilon^* > 0$.
1: Divide s into $\sqrt{n}$ windows $s^*_1, s^*_2, \ldots, s^*_{\sqrt{n}}$ of size $\sqrt{n}$.
2: $D \leftarrow$ an array of size $\lfloor \log_{1+\epsilon^*} n \rfloor$ initially containing ∞ in all cells.
3: **for** $i \in \lceil \sqrt{n} \rceil$ **do**
4: $\quad T \leftarrow$ an array of size $\lfloor \log_{1+\epsilon^*} n \rfloor$ initially containing ∞ in all cells.
5: $\quad$ **for** $0 \leq k \leq \lfloor \log_{1+\epsilon^*} n \rfloor$ **do**
6: $\quad\quad T[k] \leftarrow \text{LCSPosition}_{(i-1)\sqrt{n}+1, i\sqrt{n}}\left(1, \lfloor (1+\epsilon^*)^k \rfloor\right)$.
7: $\quad\quad$ **for** $0 \leq k_1 \leq k$ **do**
8: $\quad\quad\quad$ **if** $D[k_1] < \infty$ **then**
9: $\quad\quad\quad\quad$ Find $\text{LCSPosition}_{(i-1)\sqrt{n}+1, i\sqrt{n}}\left(D[k_1]+1, \lfloor (1+\epsilon^*)^k \rfloor - \lfloor (1+\epsilon^*)^{k_1} \rfloor\right)$ using any offline algorithm. Let q be this result.
10: $\quad\quad\quad\quad T[k] \leftarrow \min\{T[k], q\}$.
11: $\quad D \leftarrow T$.
12: **return** The largest value $\lfloor (1+\epsilon^*)^k \rfloor$ such that $D[k] < \infty$.
13: $\quad$ **return** 0 if no such k exists.

We now describe our algoirthm. The pseudocode is given in algorithm 3. It first divides the online string into $\sqrt{n}$ windows of equal sizes. We assume w.l.o.g., that length of the strings is divisible by $\sqrt{n}$. Otherwise we can always pad offline and online strings with different characters that are not in Σ such that their new length get divisible by $\sqrt{n}$. The algorithm divides s into $\sqrt{n}$ windows $s^*_1, s^*_2, \cdots, s^*_{\sqrt{n}}$ each with the size of $\sqrt{n}$ where s^*_i is the substring $s[(i-1)\sqrt{n}+1, i\sqrt{n}]$. Given an $\epsilon^* > 0$, the algorithm keeps an array D of the size $\lfloor \log_{1+\epsilon^*} n \rfloor$ where $D[k]$ is an estimation of $\text{LCSPosition}(1, \lfloor (1+\epsilon^*)^k \rfloor)$ in the subsequence of the online string that has arrived so far in the stream. Specifically, after arrival of the window

[2] We also define $\text{LCSPosition}_{l,r}(p,0)$ to be $p-1$.

s_i^* in the stream, the algorithm keeps an estimation of $\text{LCSPosition}_{1,i\sqrt{n}}(1, \lfloor(1+\epsilon^*)^k\rfloor)$ in $D[k]$. Please see the full version [36] for more details on how we update the array D upon arrival of a new window and why we can achieve the approximation guarantee. We have the following Theorem. The full proof is deferred to [36].

▶ **Theorem 13.** *There exists a single-pass deterministic streaming algorithm that finds a $(1-\epsilon)$ approximation of the LCS between $\bar{s}$ and s using $\tilde{O}(\sqrt{n}/\epsilon)$ memory and polynomial time.*

7 Space Efficient Algorithms for LIS in the Non-Streaming Model

We now consider approximating the LIS of a string $x \in \Sigma^n$ where the alphabet Σ has a total order. We assume each symbol in Σ can be stored with $O(\log n)$ space. Let ∞ and $-\infty$ be two special symbols such that for any symbol $\sigma \in \Sigma$, $-\infty < \sigma < \infty$. We denote the length of the longest increasing subsequence of x by $\mathsf{LIS}(x)$.

Again our algorithm is a recursive one, and in each recursion we use an approach similar to the deterministic streaming algorithm from [41] that gives a $1-\epsilon$ approximation of $\mathsf{LIS}(x)$ with $O(\sqrt{n/\epsilon}\log n)$ space. Before describing their approach, we first give a brief introduction to a classic algorithm for LIS, known as *PatienceSorting*. The algorithm initializes a list P with n elements such that $P[i] = \infty$ for all $i \in [n]$, and then scans the input sequence x from left to right. When reading a new symbol x_i, we find the smallest index l such that $P[l] \geq x_i$ and set $P[l] = x_i$. After processing the string x, for each i such that $P[i] < \infty$, we know $\sigma = P[i]$ is the smallest possible character such there is an increasing subsequence in x of length i ending with σ. Finally the algorithm returns the largest index l such that $P[l] < \infty$. This computes LIS exactly in $O(n \log n)$ time and $O(l \log n)$ space (see [5] for more details).

In the streaming algorithm from [41], we maintain a set S and a list Q, such that, $Q[i]$ is stored only for $i \in S$ and $S \subseteq [n]$ is a set of size $O(\sqrt{n})$. We use S and Q as an approximation to the list P in *PatienceSorting* in the sense that for each $s \in S$, there is an increasing subsequence in x of length s ending with $Q[s]$. More specifically, we can generate a list P' from S and Q such that $P'[i] = Q[j]$ for the smallest $j \geq i$ that lies in S. For i larger than the maximum element in S, we set $P'[i] = \infty$. Each time we read a new element, we update Q and S accordingly, which is equivalent to doing *PatienceSorting* on the list P'. When S gets larger than $2\sqrt{n}$, we do a cleanup to S by only keeping $\sqrt{n}/\epsilon$ evenly picked values from 1 to $\max S$ and storing $Q[s]$ for $s \in S$. This loses at most $\frac{\epsilon}{\sqrt{n}}\mathsf{LIS}(x)$ in the length of the longest increasing subsequence detected. Since we only do $O(\sqrt{n})$ cleanups, we are guaranteed an increasing subsequence of length at least $(1-\epsilon)\mathsf{LIS}(x)$.

We now modify the above algorithm into another form. This time we first divide x evenly into many small blocks. Meanwhile, we also maintain a set S and a list Q. We now process x from left to right, and update S and Q each time we have processed one block of x. If the number of blocks in x is small, we can get the same approximation as in [41] with S and Q having smaller size. For example, we can divide x into $n^{1/3}$ blocks each of size $n^{2/3}$, and we update S and Q once after processing each block. If we do exact computation within each block, we only need to maintain the set S and the list Q of size $O(\frac{n^{1/3}}{\epsilon})$. We can still get a $(1-\epsilon)$ approximation, because we do $n^{1/3}$ cleanups and for each cleanup, we lose about $\frac{\epsilon}{n^{1/3}}\mathsf{LIS}(x)$ in the length of the longest increasing subsequence detected.

This almost already gives us an $\tilde{O}_\epsilon(n^{1/3})$ space algorithm, except the exact computation within each block needs $O(n^{2/3}\log n)$ space. A natural idea to reduce the space complexity is to replace the exact computation with an approximation. When each block x^i has size $n^{2/3}$, running the approximation algorithm from [41] takes $O(\frac{n^{1/3}}{\epsilon}\log n)$ space and thus we

can hope to reduce the total space required to $O(\frac{n^{1/3}}{\epsilon}\log n)$. However, a problem here is that the approximation algorithm on each block x^i only gives us an approximation of $\mathsf{LIS}(x^i)$. This alone does not give us enough information on how to update S and Q. Also, for a longest increasing subsequence of x, say τ, the subsequence of τ that lies in the block x^i may be much shorter than $\mathsf{LIS}(x^i)$. This subsequence of τ may be ignored if we run the approximation algorithm instead of using exact computation.

We now give some intuition of our approach to fix these issues. Let us consider a longest increasing subsequence τ of x such that τ can be divided into many parts, where the i-th part τ^i lies in x^i. We denote the length of τ^i by d_i. Let the first symbol of τ^i be α_i and the last symbol be β_i if τ^i is not empty. When we process the block x^i, we want to make sure that our algorithm can detect an increasing subsequence of length very close to d_i in x^i, where the first symbol is at least α_i and the last symbol is at most β_i. We can achieve this by running a bounded version of the approximation algorithm which only considers increasing subsequences no longer than d_i. Since we do not know α_i or d_i in advance, we can guess α_i by trying every symbol in $Q[s]$ where one of them is close enough to α_i. For d_i, we can try $O(\log_{1+\epsilon} n)$ different values of l such that one of them is close enough to d_i. In this way, we are guaranteed to detect a good approximation of τ_i.

Based on the above intuition, we now introduce our space-efficient algorithm for LIS called **ApproxLIS**. The pseudocode is given in Algorithm 4. It takes three inputs, a string $x \in \Sigma^*$, two parameters b and ϵ. We also introduce a slightly modified version of **ApproxLIS** called **ApproxLISBound**. It takes an additional input l, which is an integer at most n. We want to guarantee that if the string x has an increasing subsequence of length l ending with $\alpha \in \Sigma$, then **ApproxLISBound**(x, b, ϵ, l) can detect an increasing subsequence of length close to l ending with some symbol no larger than α. The idea is to run **ApproxLIS** but only consider increasing subsequence of length at most l. **ApproxLISBound** has the same space and time complexity as **ApproxLIS**.

We now describe **ApproxLIS**. When the input string x has length at most b^2, we compute an $(1-\epsilon)$-approximation of LIS using the algorithm in [41] with $O(\frac{b}{\epsilon}\log n)$ space. Otherwise, we divide the input string into b blocks each of length $\frac{n}{b}$. Similar to the streaming algorithm in [41], we maintain two sets S and Q of size $O(\frac{b}{\epsilon})$ as an approximation of the list P when running *PatienceSorting*. We will show that it is enough to use $O(\frac{b}{\epsilon}\log n)$ bits for S and Q, because we only update them b times and we lose about $O(\frac{\epsilon}{b})\,\mathsf{LIS}(x)$ after each update. Initially, S contains only one element 0 and $Q[0] = -\infty$. We update S and Q after processing each block of x as follows.

For simplicity, we denote S and Q after processing the t-th block by S_t and Q_t. To see how S and Q are updated, we take the t-th update as an example. Given S_{t-1} and Q_{t-1}, we first determine the length of the LIS in $x^1 \circ \cdots \circ x^t$ that can be detected based on S_{t-1} and Q_{t-1}. We denote this length by k_t. Notice that, for each $s \in S_{t-1}$, we know there is an increasing subsequence in $x^1 \circ \cdots \circ x^{t-1}$ of length s ending with $Q_{t-1}[s] \in \Sigma$. This gives us $|S_{t-1}|$ increasing subsequences. The idea is to find the best extension of these increasing subsequences in the block x^t and see which one gives us the longest increasing subsequence of $x^1 \circ \cdots \circ x^t$. Since each block is of size $\frac{n}{b}$, we cannot always afford to do exact computation. Thus we compute an approximation of the LIS by recursively calling **ApproxLIS** itself. For each $s \in S_{t-1}$, we run **ApproxLIS**(z^s, b, ϵ) where z^s is the subsequence of x^t with only symbols larger than $Q_{t-1}[s]$. Finally, we let $k_t = \max_{s\in S_{t-1}}(s + \textbf{ApproxLIS}(z^s, b, \epsilon))$. Given k_t, we then set S_t to be the $\frac{b}{\epsilon}$-th evenly picked integers from 0 to k_t.

The next step is to compute Q_t. We first set $Q_t[s] = \infty$ for all $s \in S_t$ except $s = 0$, and we set $Q_t[0] = -\infty$. Then, for each $s \in S_{t-1}$ and $l = 1, 1+\epsilon, (1+\epsilon)^2, \ldots, k_t - s$, we run **ApproxLISBound**(z^s, b, ϵ, l). For each $s' \in S_t$ such that $s \le s' \le s + l$, we update $Q_t[s']$

Algorithm 4 Algorithm **ApproxLIS** (**ApproxLISBound**) for approximating LIS.

Data: A string x , parameters b and ϵ. And an additional parameter l for **ApproxLISBound**

if $|x| \leq b^2$ **then**
  compute an $(1-\epsilon)$-approximation of $\mathsf{LIS}(x)$ with the streaming algorithm from [41] using $O(\frac{b}{\epsilon}\log n)$ space. (For **ApproxLISBound**, we only consider LIS of length at most l.)
  return
divide x evenly into b blocks such that $x = x^1 \circ x^2 \circ \cdots \circ x^b$. ▷ $|x^i| \leq \lceil n/b \rceil$
initialize $S = \{0\}$ and $Q[0] = -\infty$.
for $i = 1$ **to** b **do**
  $k = 0$.
  for all $s \in S$ **do**
    let z be the subsequence of x^i by only considering the elements larger than $Q[s]$.
    $d =$ **ApproxLIS**(z, b, ϵ).
    $k = \max\{k, s+d\}$. (For **ApproxLISBound**, we let $k = \min\{l, \max\{k, s+d\}\}$) .
  if $k \leq b/\epsilon$ **then**
    let $S' = \{0, 1, 2, \ldots, k\}$.
  else
    let $S' = \{0, \frac{\epsilon}{b}k, 2\frac{\epsilon}{b}k, \ldots, k\}$. ▷ *evenly pick* $b/\epsilon + 1$ *integers from* 0 *to* k *(including* 0 *and* k*).*
  $Q'[s] = \infty$ for all $s' \in S'$ except $Q'[0] = -\infty$.
  for all $s \in S$ **do**
    let z be the subsequence of x^i by only considering the elements larger than $Q[s]$.
    for all $l = 1, 1+\epsilon, (1+\epsilon)^2, \ldots, k-s$ **do**
      $\tilde{S}, \tilde{Q} \leftarrow$ **ApproxLISBound**(z, b, ϵ, l) .
      for each $s' \in S'$ such that $s \leq s' \leq s+l$, let $\tilde{s}$ be the smallest element in $\tilde{S}$ that is larger than $s'-s$ and set $Q'[s'] = \min\{\tilde{Q}[\tilde{s}], Q'[s']\}$.
  $S \leftarrow S'$, $Q \leftarrow Q'$.
return $\max S$. (for **ApproxLISBound**, we return the set S and Q)

if **ApproxLISBound**(z^s, b, ϵ, l) detects an increasing subsequence of length at least $s'-s$ ending with a symbol smaller than the old $Q_t[s']$. The intuition is that, with the bound l, we may be able to find a smaller symbol in Σ such that there is an increasing subsequence of length l ending with it. This information can be easily ignored if l is a lot smaller than the actual length of LIS in x^t. To see why this is important, let τ be a longest increasing subsequence of x, and let τ^t be the part of τ that lies in the block x^t. The length of τ^t may be much smaller than the length of LIS in x^t. When the bound l is close to $|\tau^t|$, we will be able to detect a good approximation of τ^t by running **ApproxLISBound**(z^s, b, ϵ, l) on z^s for each $s \in S_{t-1}$. Since we do not know the length of τ^t, we will guess it by trying $O_\epsilon(\log n)$ values of l and always record the optimal $Q_t[s]$ for $s \in S_t$.

Continue doing this, we get S_b and Q_b. **ApproxLIS** outputs the largest element in S_b.

ApproxLIS is recursive. We denote the depth of recursion by d, and it can be seen that d is at most $\log_b n - 1$. To see the correctness of our algorithm, given fixed b, ϵ, we show the output at the r-th recursive level is a $(1 - O((d-r)\epsilon))$-approximation. Thus, the final output will be a $(1 - O(\epsilon \log_b n))$-approximation to $\mathsf{LIS}(x)$.

The proof is by induction on r from d to 1. For the base case of $r = d$, the statement follows from [41]. Now consider the computation at the r-th level. For convenience, we denote the input string by x, which has length at most $\frac{n}{b^{r-1}}$. Let τ be an increasing subsequence of x where τ^i lies in x^i. For our analysis, let P'_t be the list generated by S_t and Q_t in the following way: for every i let $P'_t[i] = Q_t[j]$ for the smallest $j \geq i$ that lies in S_t. If no such j exists, set $P'_t[i] = \infty$. Correspondingly, P_t is the list P after running *PatienceSorting* with input $x^1 \circ x^2 \circ \cdots \circ x^t$.

Let $h_t = \sum_{j=1}^{t} |\tau^j|$ and $k_t = \max S_t$, our main observation is the following inequality:

$$P'_t[(1 - 3(d - (r+1)\epsilon - \epsilon)h_t - 2t\frac{\epsilon}{b}k_t] \leq P_t[h_t] \tag{4}$$

Note that $h_b = |\tau| = \mathsf{LIS}(x)$. We have $P_b[h_b] < \infty$ by the correctness of *PatienceSorting*. If inequality 4 holds, then by $k_t \leq h_t$, there must exist an element in S_b larger than $(1 - 3(d-r)\epsilon)\,\mathsf{LIS}(x)$ which gives the correctness of the computation at the r-th level.

We prove inequality 4 by induction on t. The intuition is that, at the t-th update, if inequality 4 holds for $t-1$, we know that there must exist an $s \in S_{t-1}$ that is close to h_{t-1} and $Q_{t-1}[s] \leq P_{t-1}[h_{t-1}] = \beta_{t-1} < \alpha_t$. By trying $l = 1, 1+\epsilon, (1+\epsilon), \ldots, k_t - s$, one l is close enough to $|\tau^t|$. Thus we are guaranteed to detect a good approximation of τ_t in x^t and the inequality also holds for t.

At each recursive level, we need to maintain the sets S and Q with space $O(\frac{b}{\epsilon}\log n)$. Thus the total space is $O(d\frac{b}{\epsilon}\log n) = O(\frac{b\log^2 n}{\epsilon \log b})$. The analysis of time complexity is similar to the case of edit distance, where we analyze the recursion tree and bound the number of nodes.

If we take $b = \log n$ and $\epsilon = \frac{1}{\log n}$, we get a $(1 - O(\frac{1}{\log\log n}))$-approximation algorithm using $O(\frac{\log^4 n}{\log\log n})$ space and $O(n^{5+o(1)})$ time. Let $\delta \in (0, \frac{1}{2})$ be a constant such that $\frac{1}{\delta}$ is an integer. If we take $b = n^\delta$ and ϵ a small constant, we get a $(1 + O(\epsilon))$-approximation algorithm using $\tilde{O}_{\epsilon,\delta}(n^\delta)$ space and $\tilde{O}_{\epsilon,\delta}(n^{2-2\delta})$ time.

8 Space Efficient ALgorithms for LCS in the Non-Streaming Model

Our algorithm for LCS is based on a reduction to LIS. Given input strings x and y, for each $i \in [n]$ let $b^i \in [m]^*$ be the sequence consisting of all distinct indices j in $[m]$ such that $x_i = y_j$, arranged in descending order. Note that b^i may be empty. Let $z = b^1 \circ b^2 \circ \cdots \circ b^n$, which has length $O(mn)$ since each sequence b^i is of length at most m. We claim that $\mathsf{LIS}(z) = \mathsf{LCS}(x, y)$. This is because for every increasing subsequence of z, say $t = t_1 t_2 \cdots t_d$, the corresponding subsequence $y_{t_1} y_{t_2} \cdots y_{t_d}$ of y also appears in x. Conversely, for every common subsequence of x and y, we can find an increasing subsequence in z with the same length. We call this procedure ReduceLCStoLIS. Note that in our algorithms, z need not be stored, since we can compute each element in z as necessary in logspace by querying x and y. Thus our reduction is a logspace reduction.

Based on the reduction, we can use similar techniques as we used for LIS. Specifically, let $z = \textbf{ReduceLCStoLIS}(x, y)$. We call our space efficient algorithm for LCS **ApproxLCS**. If one input string is shorter than the parameter b, we know $\mathsf{LCS}(x, y) \leq b$ and we can compute $\mathsf{LIS}(z)$ using *PatienceSorting* with $O(b \log n)$ space. Otherwise, the goal is to compute an approximation of $\mathsf{LIS}(z)$. One difference here is, instead of dividing z evenly into b blocks, we divide z according to x. That is, we first divide x evenly into b blocks, and then divide z into b blocks such that $z^i = \text{ReduceLCStoLIS}(x^i, y)$. This gives us a slight improvement on running time over the naive approach of running our LIS algorithm on z. Note that $\mathsf{LIS}(z^i)$ is at most $\frac{n}{b}$ since the length of x^i is $\frac{n}{b}$. We compute an approximation of $\mathsf{LIS}(z^i)$ by recursively calling **ApproxLCS** with inputs x, y, b, ϵ. The input size to the next recursive level is decreased by a factor of b. The remaining analysis is similar to the case of LIS.

References

1 Amir Abboud and Arturs Backurs. Towards hardness of approximation for polynomial time problems. In *ITCS*, 2017.

2 Amir Abboud, Arturs Backurs, and Virginia Vassilevska Williams. Tight hardness results for LCS and other sequence similarity measures. In *FOCS*, 2015.

3 Amir Abboud, Thomas Dueholm Hansen, Virginia Vassilevska Williams, and Ryan Williams. Simulating branching programs with edit distance and friends or: a polylog shaved is a lower bound made. In *STOC*, 2016.

4 Amir Abboud and Aviad Rubinstein. Fast and deterministic constant factor approximation algorithms for LCS imply new circuit lower bounds. In *ITCS*, 2018.

5 David Aldous and Persi Diaconis. Longest increasing subsequences: from patience sorting to the baik-deift-johansson theorem. *Bulletin of the American Mathematical Society*, 36(4):413–432, 1999.

6 Carlos ER Alves, Edson N Cáceres, and Siang Wun Song. A coarse-grained parallel algorithm for the all-substrings longest common subsequence problem. *Algorithmica*, 45(3):301–335, 2006.

7 A. Andoni, M. Deza, A. Gupta, P. Indyk, and S. Raskhodnikova. Lower bounds for embedding edit distance into normed spaces. In *SODA*, 2003.

8 Alexandr Andoni, Assaf Goldberger, Andrew McGregor, and Ely Porat. Homomorphic fingerprints under misalignments: Sketching edit and shift distances. In *STOC*, 2013.

9 Alexandr Andoni and Robert Krauthgamer. The computational hardness of estimating edit distance [extended abstract]. In *FOCS*, 2007.

10 Alexandr Andoni and Robert Krauthgamer. The smoothed complexity of edit distance. In *ICALP*, 2008.

11 Alexandr Andoni, Robert Krauthgamer, and Krzysztof Onak. Polylogarithmic approximation for edit distance and the asymmetric query complexity. In *FOCS*, 2010.

12 Alexandr Andoni and Negev Shekel Nosatzki. Edit distance in near-linear time: it's a constant factor. In *Proceedings of the 61st Annual Symposium on Foundations of Computer Science (FOCS)*, 2020.

13 Alexandr Andoni and Krzysztof Onak. Approximating edit distance in near-linear time. In *STOC*, 2009.

14 Arturs Backurs and Piotr Indyk. Edit distance cannot be computed in strongly subquadratic time (unless SETH is false). In *STOC*, 2015.

15 Nikhil Bansal, Moshe Lewenstein, Bin Ma, and Kaizhong Zhang. On the longest common rigid subsequence problem. *Algorithmica*, 56(2):270–280, February 2010. doi:10.1007/s00453-008-9175-1.

16 Ziv Bar-Yossef, TS Jayram, Robert Krauthgamer, and Ravi Kumar. Approximating edit distance efficiently. In *FOCS*, 2004.

17 Tuğkan Batu, Funda Ergun, and Cenk Sahinalp. Oblivious string embeddings and edit distance approximations. In *SODA*, 2006.

18 Djamal Belazzougui and Qin Zhang. Edit distance: Sketching, streaming, and document exchange. In *FOCS*, 2016.

19 Richard Bellman. Dynamic programming (dp), 1957.

20 Mahdi Boroujeni, Soheil Ehsani, Mohammad Ghodsi, MohammadTaghi HajiAghayi, and Saeed Seddighin. Approximating edit distance in truly subquadratic time: Quantum and MapReduce. In *SODA*, 2018.

21 Mahdi Boroujeni and Saeed Seddighin. Improved MPC algorithms for edit distance and Ulam distance. In *SPAA*, 2019.

22 Joshua Brakensiek and Aviad Rubinstein. Constant-factor approximation of near-linear edit distance in near-linear time. In *Proceedings of the 52nd Annual ACM SIGACT Symposium on Theory of Computing*, pages 685–698, 2020.

23 Karl Bringman and Marvin Künnemann. Multivariate fine-grained complexity of longest common subsequence. In *SODA*, 2018.

24 Karl Bringmann, Fabrizio Grandoni, Barna Saha, and Virginia Vassilevska Williams. Truly sub-cubic algorithms for language edit distance and RNA-folding via fast bounded-difference min-plus product. In *FOCS*, 2016.

25 Karl Bringmann and Marvin Kunnemann. Quadratic conditional lower bounds for string problems and dynamic time warping. In *FOCS*, 2015.

26 Diptarka Chakraborty, Debarati Das, Elazar Goldenberg, Michal Koucky, and Michael Saks. Approximating edit distance within constant factor in truly sub-quadratic time. In *FOCS*, 2018.

27 Diptarka Chakraborty, Elazar Goldenberg, and Michal Koucký. Streaming algorithms for computing edit distance without exploiting suffix trees. *arXiv preprint*, 2016. `arXiv:1607.03718`.

28 Diptarka Chakraborty, Elazar Goldenberg, and Michal Koucký. Streaming algorithms for embedding and computing edit distance in the low distance regime. In *STOC*, 2016.

29 Moses Charikar, Ofir Geri, Michael P Kim, and William Kuszmaul. On estimating edit distance: Alignment, dimension reduction, and embeddings. In *ICALP*, 2018.

30 Lijie Chen, Shafi Goldwasser, Kaifeng Lyu, Guy N Rothblum, and Aviad Rubinstein. Fine-grained complexity meets IP=PSPACE. In *SODA*, 2019.

31 Kuan Cheng, Zhengzhong Jin, Xin Li, and Yu Zheng. Space efficient deterministic approximation of string measures. *CoRR*, abs/2002.08498, 2020. `arXiv:2002.08498`.

32 Maxime Crochemore, Costas S Iliopoulos, Yoan J Pinzon, and James F Reid. A fast and practical bit-vector algorithm for the longest common subsequence problem. *Information Processing Letters*, 80(6):279–285, 2001.

33 Maxime Crochemore, Gad M Landau, and Michal Ziv-Ukelson. A subquadratic sequence alignment algorithm for unrestricted scoring matrices. *SIAM Journal on Computing*, 32(6):1654–1673, 2003.

34 J Boutet de Monvel. Extensive simulations for longest common subsequences. *The European Physical Journal B-Condensed Matter and Complex Systems*, 7(2):293–308, 1999.

35 Funda Ergün and Hossein Jowhari. On distance to monotonicity and longest increasing subsequence of a data stream. In *SODA*, 2008.

36 Alireza Farhadi, MohammadTaghi Hajiaghayi, Aviad Rubinstein, and Saeed Seddighin. Streaming with oracle: New streaming algorithms for edit distance and LCS. *CoRR*, abs/2002.11342, 2020. `arXiv:2002.11342`.

37 Anna Gál and Parikshit Gopalan. Lower bounds on streaming algorithms for approximating the length of the longest increasing subsequence. In *FOCS*, 2007.

38 Minos Garofalakis and Amit Kumar. Correlating XML data streams using tree-edit distance embeddings. In *PODS*, 2003.

39 Omer Gold and Micha Sharir. Dynamic time warping and geometric edit distance: Breaking the quadratic barrier. In *ICALP*, 2017.

40 Elazar Goldenberg, Robert Krauthgamer, and Barna Saha. Sublinear algorithms for gap edit distance. In *FOCS*, 2019.

41 Parikshit Gopalan, T. S. Jayram, Robert Krauthgamer, and Ravi Kumar. Estimating the sortedness of a data stream. In *SODA*, 2007.

42 Dan Gusfield. *Algorithms on strings, trees and sequences: computer science and computational biology*. Cambridge University Press, 1997.

43 Bernhard Haeupler, Aviad Rubinstein, and Amirbehshad Shahrasbi. Near-linear time insertion-deletion codes and $(1+\epsilon)$-approximating edit distance via indexing. In *STOC*, 2019.

44 MohammadTaghi Hajiaghayi, Masoud Seddighin, Saeed Seddighin, and Xiaorui Sun. Approximating lcs in linear time: Beating the $\sqrt{n}$ barrier. In *SODA*, 2019.

45 MohammadTaghi Hajiaghayi, Saeed Seddighin, and Xiaorui Sun. Massively parallel approximation algorithms for edit distance and longest common subsequence. In *SODA*, 2019.

46 James W Hunt and Thomas G Szymanski. A fast algorithm for computing longest common subsequences. *Communications of the ACM*, 20(5):350–353, 1977.

47 Piotr Indyk. Algorithmic applications of low-distortion geometric embeddings. In *FOCS*, 2001.

48 Rajesh Jayaram and Barna Saha. Approximating language edit distance beyond fast matrix multiplication: Ultralinear grammars are where parsing becomes hard! In *ICALP*, 2017.

49 Ce Jin, Jelani Nelson, and Kewen Wu. An improved sketching algorithm for edit distance. In *38th International Symposium on Theoretical Aspects of Computer Science (STACS 2021)*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2021.

50 Masashi Kiyomi, Takashi Horiyama, and Yota Otachi. Longest common subsequence in sublinear space. *arXiv preprint*, 2020. `arXiv:2009.08588`.

51 Masashi Kiyomi, Hirotaka Ono, Yota Otachi, Pascal Schweitzer, and Jun Tarui. Space-efficient algorithms for longest increasing subsequence. *Theory of Computing Systems*, pages 1–20, 2018.

52 Michal Koucký and Michael Saks. Constant factor approximations to edit distance on far input pairs in nearly linear time. In *Proceedings of the 52nd Annual ACM SIGACT Symposium on Theory of Computing*, pages 699–712, 2020.

53 William Kuszmaul. Dynamic time warping in strongly subquadratic time: Algorithms for the low-distance regime and approximate evaluation. *arXiv preprint*, 2019. `arXiv:1904.09690`.

54 Gad M Landau, Eugene W Myers, and Jeanette P Schmidt. Incremental string comparison. *SIAM Journal on Computing*, 27(2):557–582, 1998.

55 Charles Eric Leiserson, Ronald L Rivest, Thomas H Cormen, and Clifford Stein. *Introduction to algorithms*, volume 6. MIT press Cambridge, MA, 2001.

56 David Liben-Nowell, Erik Vee, and An Zhu. Finding longest increasing and common subsequences in streaming data. In *COCOON*, 2005.

57 William J Masek and Michael S Paterson. A faster algorithm computing string edit distances. *Journal of Computer and System Sciences*, 20(1):18–31, 1980.

58 Timothy Naumovitz and Michael Saks. A polylogarithmic space deterministic streaming algorithm for approximating distance to monotonicity. *SODA*, 2014.

59 Rafail Ostrovsky and Yuval Rabani. Low distortion embeddings for edit distance. In *STOC*, 2005.

60 Aviad Rubinstein and Zhao Song. Reducing approximate longest common subsequence to approximate edit distance. *arXiv preprint*, 2019. `arXiv:1904.05451`.

61 Aviad Runbinstein, Saeed Seddighin, Zhao Song, and Xiaorui Sun. Approximation algorithms for LCS and LIS with truly improved running times. In *FOCS*, 2019.

62 Barna Saha. Language edit distance and maximum likelihood parsing of stochastic grammars: Faster algorithms and connection to fundamental graph problems. In *FOCS*, 2015.

63 Barna Saha. Fast & space-efficient approximations of language edit distance and RNA folding: An amnesic dynamic programming approach. In *FOCS*, 2017.

64 Michael Saks and C Seshadhri. Space efficient streaming algorithms for the distance to monotonicity and asymmetric edit distance. In *SODA*, 2013.

65 Walter J Savitch. Relationships between nondeterministic and deterministic tape complexities. *Journal of computer and system sciences*, 4(2):177–192, 1970.

66 Xiaoming Sun and David P. Woodruff. The communication and streaming complexity of computing the longest common and increasing subsequences. In *SODA*, 2007.

67 Alexander Tiskin. Semi-local string comparison: Algorithmic techniques and applications. *Mathematics in Computer Science*, 1(4):571–603, 2008.

Quantum Query Complexity with Matrix-Vector Products

Andrew M. Childs
Joint Center for Quantum Information and Computer Science, Department of Computer Science, and Institute for Advanced Computer Studies, University of Maryland, College Park, MD, USA

Shih-Han Hung
Joint Center for Quantum Information and Computer Science, Department of Computer Science, and Institute for Advanced Computer Studies, University of Maryland, College Park, MD, USA

Tongyang Li
Joint Center for Quantum Information and Computer Science, Department of Computer Science, and Institute for Advanced Computer Studies, University of Maryland, College Park, MD, USA
Center for Theoretical Physics, MIT, Cambridge, MA, USA

Abstract

We study quantum algorithms that learn properties of a matrix using queries that return its action on an input vector. We show that for various problems, including computing the trace, determinant, or rank of a matrix or solving a linear system that it specifies, quantum computers do not provide an asymptotic speedup over classical computation. On the other hand, we show that for some problems, such as computing the parities of rows or columns or deciding if there are two identical rows or columns, quantum computers provide exponential speedup. We demonstrate this by showing equivalence between models that provide matrix-vector products, vector-matrix products, and vector-matrix-vector products, whereas the power of these models can vary significantly for classical computation.

2012 ACM Subject Classification Theory of computation → Quantum query complexity; Theory of computation → Design and analysis of algorithms

Keywords and phrases Quantum algorithms, quantum query complexity, matrix-vector products

Digital Object Identifier 10.4230/LIPIcs.ICALP.2021.55

Category Track A: Algorithms, Complexity and Games

Related Version *Full Version*: `https://arxiv.org/abs/2102.11349`

Funding AMC and SHH acknowledge support from the Army Research Office (grant W911NF-20-1-0015); the Department of Energy, Office of Science, Office of Advanced Scientific Computing Research, Quantum Algorithms Teams and Accelerated Research in Quantum Computing programs; and the National Science Foundation (grant CCF-1813814).
Tongyang Li: TL acknowledges support from ARO contract W911NF-17-1-0433, NSF grant PHY-1818914, and a Samsung Advanced Institute of Technology Global Research Partnership.

Acknowledgements We thank Robin Kothari for bringing our attention to work on classical algorithms in the matrix-vector and vector-matrix-vector query models, and for providing feedback on an initial version of this paper. We thank Max Simchowitz and Blake Woodworth for a discussion that clarified aspects of their paper [8], and Jialin Zhang for clarifications of her paper [21]. We also thank Ashley Montanaro for pointing out connections to his paper [18], and Joran van Apeldoorn and Sander Gribling for a discussion of their paper [4].

1 Introduction

Algorithms for linear algebra problems – for example, solving linear systems and determining basic properties of matrices such as rank, trace, determinant, eigenvalues, and eigenvectors – constitute a fundamental research area in applied mathematics and theoretical computer science. Such tasks have widespread applications in scientific computation, statistics, operations research, and many other related areas. Algorithmic linear algebra also provides a fundamental toolbox that can inspire the design of algorithms in general.

There are several possible models of access to a matrix, and linear-algebraic algorithms can depend significantly on how the input is represented (as discussed further below). One natural model is the *matrix-vector product* (Mv) oracle. For a matrix $M \in \mathbb{F}^{n\times m}$ in a given field $\mathbb{F}$, the Mv oracle takes $x \in \mathbb{F}^m$ as input and outputs $Mx \in \mathbb{F}^n$. Matrix-vector products arise, for example, as the elementary step of the power method (and the related Lanczos method) for computing the largest eigenvector of a matrix. Matrix-vector products also commonly appear in streaming algorithms, especially in the technique of sketching (see the survey [22] for more information).

Recent work has studied the classical complexity of various basic problems in the Mv model. Specifically, Sun, Woodruff, Yang, and Zhang [21] studied the complexities of various linear algebra, statistics, and graph problems using matrix-vector products, and Braverman, Hazan, Simchowitz, and Woodworth [8] proved tight bounds on maximum eigenvalue computation and linear regression in this model. Rashtchian, Woodruff, and Zhu [19] considered a generalization to the vector-matrix-vector product (vMv) oracle, which returns $x^\top My$ for given input vectors $x \in \mathbb{F}^n, y \in \mathbb{F}^m$, and studied the complexity of various linear algebra, statistics, and graph problems in this setting. Table 1 includes a partial summary of these results.

Quantum computers can solve certain problems much faster than classical computers, so it is natural to study quantum query complexity with matrix-vector products. Lee, Santha, and Zhang recently studied the quantum query complexity of graph problems with cut queries [17], which are closely related to matrix-vector queries. For a weighted graph $G = (V, w)$ where $|V| = n$ and w assigns a nonnegative integer weight to each edge, the input of a cut query is a subset $S \subseteq V$ and the output is $|w(S, V \setminus S)|$, the total weight of the edges between S and $V \setminus S$. This can be viewed as a version of the vMv model over $\mathbb{Z}$, with the extra assumptions that $x \in \{0,1\}^n, y \in \{0,1\}^m$ are both boolean and M is a symmetric matrix with nonnegative integer entries. Reference [17] gives quantum algorithms for determining all connected components of G with $O(\log^6 n)$ quantum cut queries, and for outputting a spanning forest of G with $O(\log^8 n)$ quantum cut queries. Both problems require $\Omega(n/\log n)$ classical cut queries, so the quantum algorithms provide exponential speedups.

In other recent work on structured queries for graph problems, Montanaro and Shao studied the problem of learning an unknown graph with "parity queries" [18]: for an unknown graph with adjacency matrix A, the parity oracle takes as input a string x that encodes a subset of the vertices, and returns $x^\top Ax \bmod 2$. This query model is the vMv model over $\mathbb{F}_2$ with the extra restriction that the left and right vectors are identical.

Van Apeldoorn and Gribling studied Simon's problem for linear functions over a prime field $\mathbb{F}_p$ [4]. In this problem, the oracle encodes a linear function $f\colon \mathbb{F}_p \to \mathbb{F}_p$, and the task is to determine if the function is one-to-one, or if there is a one-dimensional subspace $H \subset \mathbb{F}_p$ such that for every $x, x' \in \mathbb{F}_p^n$, $f(x) = f(x')$ if and only if $x - x' \in H$. Such a function can be represented by a square matrix over $\mathbb{F}_p$, and the problem is equivalent to determining whether that matrix is full rank or has nullity 1 using matrix-vector product queries.

Other past work has developed linear algebraic quantum algorithms using different input models. Quantum algorithms for high-dimensional linear algebra have been studied extensively since Harrow, Hassidim, and Lloyd introduced a method for generating a quantum state proportional to the solution of a large, sparse system of linear equations [14]. This algorithm assumes a quantum oracle that determines the locations and values of the nonzero entries of a matrix in any given row or column, and the ability to generate a quantum state that encodes the right-hand side of the linear system. Subsequent work has led to improved and generalized algorithms under similar assumptions. However, it is challenging to find practical applications that achieve speedup over classical computation [2,11]. Recent work by Apers and de Wolf [5] gives polynomial quantum speedup for producing an explicit classical description of the solution of a Laplacian linear system, assuming adjacency-list access to the underlying graph of the Laplacian. Note also that for various problems including determinant estimation, rank testing, linear regression, etc., there is a large separation between the classical query complexities under Mv and entrywise queries ($\tilde{\Theta}(n)$ [21] and $\Theta(n^2)$, respectively). These results show how the model of access to a matrix can significantly impact the complexity of solving linear-algebraic problems. A better understanding of the quantum matrix-vector oracle could therefore provide a useful tool for the design of future quantum algorithms.

Contributions. We conduct a systematic study of quantum query complexity with a matrix-vector oracle for a matrix $M \in \mathbb{F}_q^{m\times n}$, where $\mathbb{F}_q$ is a given finite field. Using this model, we provide results on the quantum query complexities of linear algebra and statistics problems.

First, we prove that various linear algebra problems, including

- computing the trace $\mathrm{tr}(M)$ of $M \in \mathbb{F}_q^{n\times n}$;
- computing the determinant $\det(M)$ of $M \in \mathbb{F}_q^{n\times n}$;
- solving the linear system $Ax = b$ for $A \in \mathbb{F}_q^{n\times n}$; and
- testing whether $\mathrm{rank}(M) = n$ or $\mathrm{rank}(M) \le n/2$ for a matrix $M \in \mathbb{F}_q^{m\times n}$;

require $\Omega(n)$ quantum queries to the Mv oracle. Since $O(n)$ queries suffice to determine the entire matrix, even classically, these results show that no quantum speedup is possible. (As a side effect, we improve the $\Omega(n/\log n)$ classical lower bound for trace computation [21] to $\Omega(n)$.)

Our quantum lower bound for trace computation applies results of Copeland and Pommersheim [12] by viewing the problem as a special case of *coset identification*. Our lower bounds for other linear algebra problems are all proved by the polynomial method [1,6]. We show how to symmetrize the success probability to a univariate polynomial, and then give a lower bound on the polynomial degree using an observation of Koiran, Nesme, and Portier [16].

On the other hand, we determine the matrix-vector quantum query complexity of several statistics problems, including

- computing the row and column parities of $M \in \mathbb{F}_2^{m\times n}$;
- deciding if there exist two identical columns in $M \in \mathbb{F}_2^{m\times n}$; and
- deciding if there exist two identical rows in $M \in \mathbb{F}_2^{m\times n}$.

Specifically, we prove that their quantum query complexities with an Mv oracle are $O(1)$, $O(\log n)$, and $O(\log m)$, respectively. Compared to the classical bounds using either the Mv oracle [21] or the vMv oracle [19], our quantum algorithms achieve *exponential* quantum speedups.

Technically, these results build upon our observation that the quantum query complexities in the Mv model under left or right multiplication are *identical* (Theorem 5). In particular, one right Mv query can be simulated using one left Mv query, and vice versa. In contrast,

classically there is a significant difference between matrix-vector (Mv) and vector-matrix (vM) queries – for example, computing the parity of rows over $\mathbb{F}_2$ only takes $O(1)$ Mv queries, but computing the parity of columns over $\mathbb{F}_2$ requires $\Theta(n)$ Mv queries. In contrast, for both problems a quantum computer can achieve the smaller query complexity by switching to the easier side.

Table 1 Comparison of classical and quantum query complexities with matrix-vector (Mv) and vector-matrix-vector (vMv) product oracles for an $m \times n$ matrix. For trace and linear regression, $m = n$. Known query complexities over $\mathbb{R}$ and $\mathbb{F}_q$ are included for completeness; results over different fields are incomparable in general.

Problem	Classical Mv	Classical vMv	Quantum (this paper)
Trace	$O(n), \Omega(n/\log n)$ for matrix with entries in $\{0, 1, \ldots, n^3\}$ & queries with entries in $\{0, 1, \ldots, n^C\}$, $C \in \mathbb{N}$ [21]; $\Theta(n)$ over $\mathbb{F}_q$ (Theorem 16)	$O(n), \Omega(n/\log n)$ for matrix with entries in $\{0, 1, \ldots, n^3\}$ & queries with entries in $\{0, 1, \ldots, n^C\}$, $C \in \mathbb{N}$ [19]; $\Theta(n)$ over $\mathbb{F}_q$ (Theorem 16)	$\Theta(n)$ over $\mathbb{F}_q$ (Theorem 16)
Linear regression	$\Theta(n)$ over $\mathbb{R}$ [8]; $\Theta(n)$ over $\mathbb{F}_q$ (Theorem 24)	$\Theta(n^2)$ over $\mathbb{F}_q$ (Corollary 25)	$\Theta(n)$ over $\mathbb{F}_q$ (Theorem 24)
Rank testing	$k+1$ to distinguish rank $\leq k$ from $k' > k$ over $\mathbb{R}$ [21]; $\Theta(n)$ over $\mathbb{F}_q$ (Theorem 27)	$\Omega(k^2)$ to distinguish rank k from $k+1$ over $\mathbb{F}_q$ [19]; $\Omega(n^{2-O(\epsilon)})$ for non-adaptive $(1 \pm \epsilon)$-approximation over $\mathbb{R}$ [19]	$\Theta(\min\{m, n\})$ to distinguish rank $\min\{m, n\}$ from $\leq \frac{1}{2}\min\{m, n\}$ over $\mathbb{F}_q$ (Theorem 27)
Two identical columns	$O(n/m)$, $m = \Omega(\log(n/\epsilon))$ over $\mathbb{F}_2$ [21]	$O(n \log n), \Omega(n)$ over $\mathbb{F}_2$ [19]	$O(\log n)$ over $\mathbb{F}_2$ (Corollary 8)
Two identical rows	$O(\log m)$ over $\mathbb{F}_2$ [21]	$O(n \log n), \Omega(n)$ over $\mathbb{F}_2$ [19]	$O(\log m)$ over $\mathbb{F}_2$ (Corollary 8)
Majority of columns	$\Omega(n/\log n)$ for binary matrices over $\mathbb{R}$ [21]	$\Theta(n^2)$ over $\mathbb{F}_2$ [19]	$O(1)$ for binary matrices over $\mathbb{R}$ (Corollary 10)
Majority of rows	$O(1)$ for binary matrices over $\mathbb{R}$ [21]	$\Theta(n^2)$ over $\mathbb{F}_2$ [19]	$O(1)$ for binary matrices over $\mathbb{R}$ (Corollary 10)
Parity of columns	$\Theta(n)$ over $\mathbb{F}_2$ [21]	$\Theta(n)$ over $\mathbb{F}_2$ (Lemma 7)	$O(1)$ over $\mathbb{F}_2$ (Corollary 6)
Parity of rows	$O(1)$ over $\mathbb{F}_2$ [21]	$\Theta(m)$ over $\mathbb{F}_2$ (Lemma 7)	$O(1)$ over $\mathbb{F}_2$ (Corollary 6)

Our results are summarized in Table 1, including some implications of our results for classical query complexity and a few additional results over $\mathbb{R}$. Note that there can be large gaps between the classical query complexities with Mv and vMv queries, but they are the same in the quantum setting due to an equivalence between quantum Mv and vMv queries (Theorem 11), which follows along similar lines to the equivalence between Mv and vM queries. The Mv–vMv equivalence is closely related to a similar equivalence shown in the work of Lee, Santha, and Zhang [17], as we discuss further in Section 3.2.

Open questions. Our paper leaves several natural open questions for future investigation:

- For linear algebra problems such as those we studied, can we also prove quantum query lower bounds for matrices over the real field $\mathbb{R}$? Our proofs rely on the polynomial method, and it is unclear how to adapt them to a setting with continuous input.
- Can we prove a quantum lower bound for the task of minimizing a quadratic form $f(x) = \frac{1}{2}x^\top Ax + b^\top x$, where $A \in \mathbb{R}^{n\times n}$ and $b \in \mathbb{R}^n$? Note that f is minimized at $x = -A^{-1}b$, and we can determine the vector b and implement Mv queries to the matrix A using fast quantum gradient computation [15], so this is closely related to the previous open question. Quadratic form minimization is a special case of optimizing a convex function $f\colon \mathbb{R}^n \to \mathbb{R}$ by quantum evaluation queries, where previous works [3, 10, 13] left a quadratic gap between the best known quantum upper and lower bounds of $\tilde{O}(n)$ and $\Omega(\sqrt{n})$, respectively.
- For the finite field case, can we identify other problems with quantum speedup over the classical matrix-vector oracle, or find advantage compared to other quantum oracles such as entrywise queries?

Organization. We review necessary background in Section 2. We prove the equivalence of quantum matrix-vector and vector-matrix-vector product oracles in Section 3. In Section 4, we prove tight quantum query complexity lower bounds on various linear algebra problems, including trace, determinant, linear systems, and rank.

2 Preliminaries

2.1 The quantum query model

Given a set X and an abelian group G, let $f\colon X \to G$ be a function. Access to f is provided by a black-box unitary operation $U_f\colon |x, y\rangle \mapsto |x, y + f(x)\rangle$ for all $x \in X$ and $y \in G$. We call an application of U_f a (standard) query.

For a finite abelian group G, the Fourier transform over G is

$$F_G := \frac{1}{|G|^{1/2}} \sum_{x\in G} \sum_{y\in \hat{G}} \chi_y(x)|y\rangle\langle x|, \tag{1}$$

where $\hat{G}$ is a complete set of characters of G, and $\chi_y\colon G \to \mathbb{C}$ denotes the y^{th} character of G. Since $\hat{G} \cong G$, we label elements of $\hat{G}$ using elements of G. Note that χ_y is a group homomorphism, i.e., $\chi_y(x + z) = \chi_y(x)\chi_y(z)$. In addition, the characters satisfy the orthogonality condition

$$\frac{1}{|G|} \sum_{z\in G} \chi_y(z)^* \chi_w(z) = \delta_{yw}. \tag{2}$$

A phase query is defined as a standard query conjugated by the Fourier transform acting on the output register. In other words, for $x \in X$ and $y \in G$, a phase query acts as

$$\begin{aligned} |x, y\rangle &\xmapsto{\mathbb{1}\otimes F_G^\dagger} \frac{1}{|G|^{1/2}} \sum_{z\in G} \chi_y(z)^* |x, z\rangle \\ &\xmapsto{U_f} \frac{1}{|G|^{1/2}} \sum_{z\in G} \chi_y(z)^* |x, z + f(x)\rangle \\ &\xmapsto{\mathbb{1}\otimes F_G} \frac{1}{|G|} \sum_{z\in G} \chi_y(z)^* \chi_w(z + f(x))|x, w\rangle = \chi_y(f(x))|x, y\rangle. \end{aligned} \tag{3}$$

The equality in (3) follows from the orthogonality condition in (2). Since one can simulate a phase query using a single standard query and vice versa, the query complexities of any problem are equal with these two models.

Over a finite field $\mathbb{F}_q$ for prime power $q = p^r$, the Fourier transform over $\mathbb{F}_q$ is the unitary transformation $|x\rangle \mapsto q^{-1/2} \sum_{y \in \mathbb{F}_q} e(xy)|y\rangle$, where the exponential function $e\colon \mathbb{F}_q \to \mathbb{C}$ is defined as $e(z) := e^{2\pi i \mathrm{Tr}_{\mathbb{F}_q/\mathbb{F}_p}(z)/p}$ and the trace function $\mathrm{Tr}_{\mathbb{F}_q/\mathbb{F}_p}\colon \mathbb{F}_q \to \mathbb{F}_p$ is defined as $\mathrm{Tr}_{\mathbb{F}_q/\mathbb{F}_p}(z) := z + z^p + z^{p^2} + \cdots + z^{p^{r-1}}$.

Over the field of real numbers, the quantum Fourier transform is

$$F_{\mathbb{R}} := \int_{\mathbb{R}} \mathrm{d}y \int_{\mathbb{R}} \mathrm{d}x \, e^{2\pi i y x} |y\rangle\langle x|. \tag{4}$$

The basis states $\{|x\rangle : x \in \mathbb{R}\}$ are normalized to the Dirac delta function, i.e., for $x, x' \in \mathbb{R}$, $\langle x'|x\rangle = \delta(x - x')$. Here the Dirac delta function δ satisfies $\int_{\mathbb{R}} \mathrm{d}x' \, \delta(x - x') f(x') = f(x)$ for any function f. Furthermore, we have $\int_{\mathbb{R}} \mathrm{d}y \, e^{2\pi i y(x - x')} = \delta(x - x')$. By direct calculation using these facts, $F_{\mathbb{R}}^\dagger F_{\mathbb{R}} = \int_{\mathbb{R}} \mathrm{d}x \, |x\rangle\langle x| = \mathbb{1}$.

While we can formally consider a model of query complexity over $\mathbb{R}$ with arbitrary precision, its practical instantiation requires discrete approximation. We can achieve precision ϵ by approximating real numbers with $s = O(\log(1/\epsilon))$ bits, and can then replace the continuous Fourier transform with the discrete Fourier transform over $\mathbb{Z}_{2^s}$. It is straightforward to show that a discretized phase query over $\mathbb{Z}_{2^s}$ can be implemented by Fourier transforming a standard query that maps discretized inputs to discretized function values.

2.2 The coset identification problem

Copeland and Pommersheim studied a kind of quantum query problem that they call the *coset identification problem* [12]. They define this problem in a generalized query model where the black box does not necessarily perform a standard or phase query, although their definition includes those cases. In the coset identification problem, we fix a finite group G and a subgroup $H \leq G$. The algorithm is given access to a unitary transformation $\pi(g)$, where π is a representation of G on vector space V. When π is given, the vector space V is called the representation space (or simply, the representation) of G [20, Chapter 1]. The goal is to determine which coset of H the unknown element $g \in G$ belongs to.

▶ **Definition 1** (Coset identification problem [12]). *A coset identification problem for a finite group G and subgroup $H \leq G$ is a 3-tuple (π, V, F) such that*

- *π is a unitary representation of G in the complex vector space V, and*
- *F is a function constant on left cosets of $H \leq G$ and distinct on distinct cosets, i.e., $F(g) = F(g')$ if and only if $g' = gh$ for some $h \in H$.*

Given a black box that performs the unitary transformation $\pi(g)$, the goal is to compute $F(g)$.

Copeland and Pommersheim show that the optimal success probability of a t-query algorithm for a coset identification problem can be calculated by taking, over all irreps Y of H, the maximum of the fraction of the induced representation $Y^\uparrow$ of G shared with $V^{\otimes t}$. Furthermore, the optimal algorithm can be non-adaptive. For a representation V, let $I(V)$ denote the set of irreducible characters of G appearing in V.

▶ **Theorem 2** (Optimal success probability of coset identification [12, Corollary 5.7]). *The optimal success probability of any t-query quantum algorithm $\mathcal{A}$ for the coset identification problem (π, V, F) for finite group G and subgroup $H \leq G$, under uniformly random inputs in G, is*

$$\Pr[\mathcal{A}^{\pi(g)} = F(g)] = \max_Y \frac{\dim Y^\uparrow_{V^{\otimes t}}}{\dim Y^\uparrow}, \tag{5}$$

where the probability is maximized over all irreducible representations Y of H, $Y^{\uparrow}$ is the induced representation of G, and A_B is the maximal subrepresentation of A such that $I(A_B) \subseteq I(B)$ for representations A, B.

The *oracle discrimination problem* is the special case of the coset identification problem where H is the trivial group, i.e., the function F is injective. In this case, $Y^{\uparrow} = \mathrm{span}\{|g\rangle : g \in G\}$.

▶ **Corollary 3** (Optimal success probability of oracle discrimination [12, Theorem 4.2])**.** *The optimal success probability of the oracle discrimination problem is*

$$\frac{1}{|G|} \sum_{i \in I(V^{\otimes t})} d_i^2, \tag{6}$$

where $I(V^{\otimes t})$ is the irrep content of $(\pi^{\otimes t}, V^{\otimes t})$ and d_i is the dimension of irrep $i \in I(V^{\otimes t})$.

We consider the complexity of standard queries in the matrix-vector model. In this model, oracle access to a matrix $M \in \mathbb{F}^{m \times n}$ for field $\mathbb{F}$ and positive integers m, n is the unitary operation $U(M)\colon |x, y\rangle \mapsto |x, y + Mx\rangle$. The map U is a representation of the additive group of matrices since it is a group homomorphism satisfying $U(M)U(N) = U(M + N)$ for all matrices M, N of the same dimensions. A phase query is also a unitary representation since it is a standard query conjugated by a fixed unitary matrix (the quantum Fourier transform).

2.3 The polynomial method

We will use the polynomial method to obtain quantum lower bounds. Here we state a version for non-boolean functions as used in [1].

▶ **Lemma 4.** *Let $\mathcal{A}$ be a t-query quantum algorithm given access to the input $x \in [m]^n$ for $m, n \in \mathbb{Z}$ through oracle $U_x\colon |i, j\rangle \mapsto |i, j + x_i\rangle$ for $i \in [n]$ and $j \in [m]$. The acceptance probability of $\mathcal{A}$ on input x is a degree-$(2t)$ polynomial in $x_1, \ldots, x_n$.*

3 Equivalence of matrix-vector and vector-matrix-vector products

In this section, we show that the matrix-vector and vector-matrix-vector models are equivalent, i.e., for any problem, the quantum query complexities in these models differ by at most a constant factor. Furthermore, we show that in the matrix-vector model, left matrix-vector products and right matrix-vector products are equivalent. This is in stark contrast to the classical case where these query complexities can differ significantly, as mentioned in Section 1 and discussed further below.

3.1 Left and right matrix-vector queries

We first show that left matrix-vector products and right matrix-vector products are equivalent.

▶ **Theorem 5.** *Quantum query complexities in the left and right matrix-vector models over a finite field are identical. In particular, one right* Mv *query can be simulated using one left* Mv *query, and vice versa.*

Proof. For input matrix $M \in \mathbb{F}_q^{n \times m}$, a matrix-vector (Mv) query applies the unitary transformation

$$U^{\mathsf{Mv}}(M)\colon |x, y\rangle \mapsto |x, y + Mx\rangle \tag{7}$$

for every $x \in \mathbb{F}_q^m$ and $y \in \mathbb{F}_q^n$. Conjugating by a quantum Fourier transform on the output register yields a phase query

$$\begin{aligned}
|x,y\rangle &\xmapsto{\mathbb{1}\otimes F_{\mathbb{F}_q^n}^\dagger} q^{-1/2}\sum_z e(-y^\top z)|x,z\rangle \\
&\xmapsto{U^{\mathsf{Mv}}(M)} q^{-1/2}\sum_z e(-y^\top z)|x,z+Mx\rangle \\
&\xmapsto{\mathbb{1}\otimes F_{\mathbb{F}_q^n}} q^{-1}\sum_{z,w} e(-y^\top z + w^\top(z+Mx))|x,w\rangle \\
&= \sum_w \delta[y=w] e(-y^\top z + w^\top(z+Mx))|x,w\rangle \\
&= e(y^\top Mx)|x,y\rangle. \qquad (8)
\end{aligned}$$

We denote this unitary transformation by $U^{\widetilde{\mathsf{Mv}}}(M)$.

Conjugating a phase query by a swap gate, we have

$$\begin{aligned}
|x,y\rangle &\xmapsto{\mathsf{SWAP}} |y,x\rangle \\
&\xmapsto{U^{\widetilde{\mathsf{Mv}}}(M)} e(x^\top My)|y,x\rangle \\
&\xmapsto{\mathsf{SWAP}} e(x^\top My)|x,y\rangle \\
&= e(y^\top M^\top x)|x,y\rangle. \qquad (9)
\end{aligned}$$

This yields $U^{\widetilde{\mathsf{Mv}}}(M^\top)$, which in turn gives $U^{\mathsf{Mv}}(M^\top)$ upon conjugation by an inverse quantum Fourier transform on the output register. Thus one can simulate the oracle $U^{\mathsf{Mv}}(M^\top)$ using one query to $U^{\mathsf{Mv}}(M)$, showing equivalence of the two models. ◀

In contrast to Theorem 5, Sun, Woodruff, Yang, and Zhang show that for the task of computing the row parities of an $m \times n$ matrix M over $\mathbb{F}_2$, the left query complexity is $\Omega(m)$, whereas the right query complexity is 1 [21]. Thus we have shown that computing column parities over $\mathbb{F}_2$ in the Mv model has quantum query complexity 1, significantly less than the classical query complexity of $\Omega(n)$.

▶ **Corollary 6.** *The query complexity of computing the row parities and the column parities of an $m \times n$ matrix over $\mathbb{F}_2$ is 1.*

Note that it is easy to understand the randomized query complexities of these problems in the vMv model.

▶ **Lemma 7.** *The randomized query complexities of computing the row parities and the column parities of an $m \times n$ matrix over $\mathbb{F}_2$ are $\Theta(m)$ and $\Theta(n)$, respectively.*

Proof. Each query reveals one bit of information, while the row parities convey m bits, giving a lower bound of $\Omega(m)$. An algorithm querying $(e_1, 1^n), \ldots, (e_m, 1^n)$ learns the row parities with probability 1, giving an upper bound of m. The query complexity of column parities follows immediately from the symmetry of the vMv oracle. ◀

The randomized query complexities of determining if there exist identical columns or identical rows are $\Theta(n/m)$ and $\Theta(\log m)$, respectively [21]. Theorem 5 implies that for identical columns, there is an exponential quantum speedup.

▶ **Corollary 8.** *The query complexities of deciding if there exist two identical columns and rows in a $m \times n$ matrix over $\mathbb{F}_2$ are $O(\log n)$ and $O(\log m)$, respectively.*

Proof. By Theorem 5, it suffices to give an algorithm for determining if there are two identical rows. To make the proof self-contained, we describe the algorithm of Sun, Woodruff, Yang, and Zhang [21, Section 4.2]. The algorithm makes q random queries $v_1, \ldots, v_q$, the entries of which are sampled uniformly. The algorithm outputs 1 if and only if there exist two entries i, j such that $(Mv_k)_i = (Mv_k)_j$ for $k \in [q]$.

To analyze the performance, for any two identical rows $m_i^\top, m_j^\top$, $\Pr_v[m_i^\top v = m_j^\top v] = 1$. For $m_i \neq m_j$, $\Pr_v[m_i^\top v = m_j^\top v] \leq 1/2$. Therefore for a matrix that has two identical rows, the algorithm outputs 1 with probability 1. On the other hand, for a matrix that has no identical rows, the algorithm outputs 1 with probability

$$\begin{aligned}\Pr_{v_1,\ldots,v_q}[\exists i,j \in [m],\ \forall \ell \in [q], m_i^\top v_\ell = m_j^\top v_\ell] &\leq \sum_{i,j\in[m],i\neq j} \Pr_{v_1,\ldots,v_q}[\forall \ell \in [q], m_i^\top v_\ell = m_j^\top v_\ell] \\ &\leq \binom{m}{2} 2^{-q}. \end{aligned} \tag{10}$$

Taking $q = 2\log m$, the probability is no more than $\frac{1}{2} - \frac{1}{2m}$. ◀

The equivalence of left and right queries also holds over the reals.

▶ **Theorem 9.** *Quantum query complexities in the left and the right matrix-vector models over $\mathbb{R}$ are identical. In particular, one right* Mv *query can be simulated using one left* Mv *query, and vice versa.*

Proof. The same idea as in the proof of Theorem 5 applies. First, a phase query can be simulated by conjugating a standard query by the quantum Fourier transform. This yields $U^{\widetilde{\mathsf{Mv}}}(M)$. Conjugating a phase query by a swap gate gives $U^{\widetilde{\mathsf{Mv}}}(M^\top)$ with the same calculation as in (9). This in turn yields $U^{\mathsf{Mv}}(M^\top)$ upon conjugating $U^{\widetilde{\mathsf{Mv}}}(M^\top)$ by an inverse quantum Fourier transform. ◀

Note that with finite precision, a phase query can be simulated using the quantum Fourier transform over an integer modulus (see Section 2.1 for details).

As an example, we determine the query complexity of the majority of rows or columns: given a binary matrix $M \in \{0,1\}^{m\times n}$, compute the majority of each row or column over $\mathbb{R}$, i.e., for each row or column, determine if there are more 1s than 0s.

▶ **Corollary 10.** *In the matrix-vector model, the query complexities of computing the majorities of rows and columns of an $m \times n$ matrix over $\mathbb{R}$ are 1.*

Proof. By Theorem 9, it suffices to show the query complexity of the majority of rows is 1. With a single query $(1, 1, \ldots, 1)^\top$, the majority of each row is determined. ◀

This result is not significantly affected by considering computation with finite precision. The number of 1s in each row and each column is an integer in $[0, k]$ for $k = \max\{m, n\}$. Thus a truncation with $O(\log k)$ bits suffices to perform the computation with no error.

3.2 The vector-matrix-vector model

We now relate the power of the matrix-vector and vector-matrix-vector query models. In the vector-matrix-vector model, the algorithm is given access to M via $U^{\mathsf{vMv}}\colon |x, y, a\rangle \mapsto |x, y, a + y^\top Mx\rangle$. We can simulate one vMv query using two Mv queries and an ancilla space storing a matrix-vector product:

$$\begin{aligned}|x,y,a\rangle &\xmapsto{U^{\mathsf{Mv}}(M)} |x,y,a\rangle|Mx\rangle \\ &\mapsto |x,y,a+y^\top Mx\rangle|Mx\rangle \\ &\xmapsto{U^{\mathsf{Mv}}(M)^\dagger} |x,y,a+y^\top Mx\rangle|0\rangle. \end{aligned} \tag{11}$$

On the other hand, an Mv phase query (defined previously in (8)) can be simulated using a vMv phase query by setting $a = 1$:

$$|x,y,1\rangle \mapsto e(y^\top Mx)|x,y,1\rangle. \tag{12}$$

Such a vMv phase query can be constructed using one application of U^{vMv}:

$$\begin{aligned}|x,y,a\rangle &\xmapsto{\mathbb{1}\otimes\mathbb{1}\otimes F_{\mathbb{F}_q}^\dagger} \sum_b e(-ab)|x,y,b\rangle \\ &\xmapsto{U^{\mathsf{vMv}}(M)} \sum_b e(-ab)|x,y,b+y^\top Mx\rangle \\ &\xmapsto{\mathbb{1}\otimes\mathbb{1}\otimes F_{\mathbb{F}_q}} \sum_{bc} e(-ab+c(b+y^\top Mx))|x,y,c\rangle \\ &= e(ay^\top Mx)|x,y,a\rangle. \end{aligned} \tag{13}$$

Thus we have shown the following.

▶ **Theorem 11.** *Quantum query complexities in the matrix-vector and vector-matrix-vector models differ by at most a constant factor. In particular, one* vMv *query can be simulated using two* Mv *queries, and one* Mv *query can be simulated using one* vMv *query.*

This is again in stark contrast to the classical case, where the Mv model can be much more powerful than the vMv model. For example, for distinguishing a full-rank matrix from a rank-$(n-1)$ matrix, the randomized query complexity in the vMv model is $\Omega(n^2)$ [19], while the randomized query complexity in the Mv model is $O(n)$ [21].

Note that Lee, Santha, and Zhang [17] previously studied the equivalence between quantum Mv and vMv oracles. They focus on the special case where the matrix M is the adjacency matrix of a graph with nonnegative integer weights and the inputs $x \in \{0,1\}^n, y \in \{0,1\}^m$ are boolean. In that setting, they prove equivalence between the vMv oracle and the additive oracle $a\colon 2^{[n]} \to \mathbb{Z}$ that returns $a(S) = \sum_{(u,v)\in S^{(2)}} w(u,v)$ for $S \subseteq [n]$, where $S^{(2)}$ denotes the set of cardinality-2 subsets of S. They also study relationships with other oracles that encode specific information about graphs (cuts, disjoint cuts, etc.; see Section 4 of [17]). In contrast, our Theorem 5, Theorem 9, and Theorem 11 work for inputs and matrices in fields, and do not apply to other graph oracles. While these results are, strictly speaking, incomparable, they are closely related, both following from a generalization of the Bernstein-Vazirani algorithm [7].

4 Linear algebra over finite fields

We now consider the quantum query complexity of particular linear algebra problems in the matrix-vector query model. Specifically, we consider learning the trace (Section 4.1), computing the null space and determinant (Section 4.2), solving linear systems (Section 4.3), and estimating the rank (Section 4.4).

4.1 Trace

In this section, we show that the quantum query complexity of computing the trace of an $n \times n$ matrix over $\mathbb{F}_q$ is $\Theta(n)$. Since there is a trivial algorithm that computes the trace by learning the entire matrix using n queries, we focus on the lower bound.

Learning the trace can be regarded as a coset identification problem (defined in Section 2.2) in the group $G = \mathbb{F}_q^{n\times n}$ with subgroup $H = \{M \in \mathbb{F}_q^{n\times n} : \mathrm{tr}M = 0\} \cong \mathbb{F}_q^{n^2-1}$. The irreducible characters χ_Z of H are indexed by $Z \in \mathbb{Z}_m^{n\times n}$ with $Z_{nn} = 0$, and satisfy $\chi_Z(M) = e(\langle Z, M\rangle)$ where $\langle Z, M\rangle \coloneqq \sum_{i,j=1}^n Z_{ij}M_{ij}$.

4.1.1 Learning the trace over $\mathbb{F}_2$

First we consider the case $q = 2$. Then the irreducible characters χ_Z of H for $Z \in \mathbb{Z}_m^{n\times n}$ (with $Z_{nn} = 0$) satisfy

$$\chi_Z(M) = (-1)^{\langle Z,M\rangle}. \tag{14}$$

For irredicible character Z, the induced representation can be decomposed into two irreducible characters of G:

$$\chi_{Z,0}(M) = (-1)^{\langle Z,M\rangle}; \qquad \chi_{Z,1}(M) = (-1)^{\langle Z,M\rangle+\mathrm{tr}M}. \tag{15}$$

It is easy to check that for $M \in G$, $\chi_{Z,0}(M + E^{(nn)}) = \chi_{Z,0}(M)$ and $\chi_{Z,1}(M + E^{(nn)}) = -\chi_{Z,1}(M)$, where $E^{(ij)}$ is an $n \times n$ matrix whose entries are zero except that $(E^{(ij)})_{ij} = 1$. We emphasize that in (15), $M \in G$ (rather than in H since we are now looking at the representations of the entire group), and $Z_{nn} = 0$.

On the other hand, recall that the phase query oracle is $U(M)\colon |x, y\rangle \mapsto (-1)^{y^\top Mx}|x, y\rangle$, which is a unitary representation of M with character $\xi(M) \coloneqq \mathrm{tr}(U(M)) = \sum_{x,y\in\mathbb{F}_2^n}(-1)^{y^\top Mx}$. To determine the optimal success probability, we calculate the irrep content of $U^{\otimes t}$. The character of $U^{\otimes t}$ is ξ^t, satisfying

$$\begin{aligned}\mathrm{tr}(U^{\otimes t}(M)) &= \mathrm{tr}(U(M))^t = (\xi(M))^t \\ &= \left(\sum_{x,y\in\mathbb{F}_2^n}(-1)^{y^\top Mx}\right)^t = \sum_{x_1,\ldots,x_t,y_1,\ldots,y_t\in\mathbb{F}_2^n}(-1)^{\sum_i y_iMx_i}.\end{aligned} \tag{16}$$

Thus it has non-zero Fourier coefficient at W if and only if $W \in R_t$, where R_t is the set of matrices of rank no more than t.

We now check containment of the irreps (15) in $U^{\otimes t}$. We find

$$m_{Z,0}^{(t)} = \langle \xi^t, \chi_{Z,0}\rangle > 0 \iff Z \in R_t, \qquad m_{Z,1}^{(t)} = \langle \xi^t, \chi_{Z,0}\rangle > 0 \iff Z + \mathbb{1}_n \in R_t. \tag{17}$$

By Theorem 2, to succeed with probability better than $1/2$, we must choose a Z such that both $m_{Z,0}^{(t)} > 0$ and $m_{Z,1}^{(t)} > 0$. However, now we show this is impossible with $t < n/2$.

▶ **Lemma 12.** *The set* $\{Z : m_{Z,0}^{(t)} > 0 \wedge m_{Z,1}^{(t)} > 0\}$ *is empty for* $t < n/2$.

Proof. We show that the set is non-empty only if $t \geq n/2$. Suppose there exists Z such that $m_{Z,0} > 0$ and $m_{Z,1} > 0$. By (17), $Z \in R_t$ and $Z + \mathbb{1}_n \in R_t$. Since the ranks of Z and $Z + \mathbb{1}_n$ are no more than t, we conclude that the rank of $\mathbb{1}_n = Z + Z + \mathbb{1}_n$ is no more than $2t$. Therefore $t \geq n/2$. ◀

This implies an $n/2$ lower bound, formally stated as follows.

▶ **Lemma 13.** *For $t < n/2$, any t-query quantum algorithm computing the trace of an $n \times n$ matrix over $\mathbb{F}_2$ succeeds with probability at most $1/2$.*

Proof. By Theorem 2 and Lemma 12, the optimal success probability for a uniformly random matrix in $\mathbb{F}_2^{n\times n}$ is

$$\frac{1}{2}\max_Z \sum_{b=0}^{1} \delta[m_{Z,b} > 0] \leq \frac{1}{2} \tag{18}$$

for $t < n/2$. ◀

On the upper bound side, we present an $\lceil n/2 \rceil$-query quantum algorithm, showing that the above lower bound is achievable.

▶ **Lemma 14.** *In the matrix-vector query model, there exists an $\lceil n/2 \rceil$-query quantum algorithm that computes the trace of an $n \times n$ matrix over $\mathbb{F}_2$ with probability 1.*

Proof. First we pad the matrix with one extra zero row and one extra zero column if n is odd, and denote the padded matrix by M'. Let $\ell = \lceil n/2 \rceil$. It is clear that one query to $M' \in \mathbb{F}_2^{2\ell\times 2\ell}$ can be simulated using one query to M. By Theorem 2, it suffices to find an irreducible character such that both $m_{Z,0} > 0$ and $m_{Z,1} > 0$. Now consider

$$Z = \begin{bmatrix} \mathbb{1}_\ell & 0 \\ 0 & 0 \end{bmatrix} = \sum_{i=1}^{\ell} e_i e_i^\top, \qquad Z + \mathbb{1}_{2\ell} = \begin{bmatrix} 0 & 0 \\ 0 & \mathbb{1}_\ell \end{bmatrix} = \sum_{i=\ell+1}^{2\ell} e_i e_i^\top. \tag{19}$$

The algorithm first prepares the state

$$|\psi_0\rangle = \frac{1}{\sqrt{2}}|e_1,\ldots,e_\ell\rangle|e_1,\ldots,e_\ell\rangle + \frac{1}{\sqrt{2}}|e_{\ell+1},\ldots,e_{2\ell}\rangle|e_{\ell+1},\ldots,e_{2\ell}\rangle. \tag{20}$$

Making ℓ phase queries in parallel, we have

$$\begin{aligned} |\psi_M\rangle &= \widetilde{U^{\mathsf{Mv}}}(M')|\psi_0\rangle \\ &= \frac{1}{\sqrt{2}}(-1)^{\sum_{i=1}^{\ell} M'_{ii}}|e_1,\ldots,e_\ell\rangle|e_1,\ldots,e_\ell\rangle \\ &\quad + \frac{1}{\sqrt{2}}(-1)^{\sum_{i=\ell+1}^{2\ell} M'_{ii}}|e_{\ell+1},\ldots,e_{2\ell}\rangle|e_{\ell+1},\ldots,e_{2\ell}\rangle. \end{aligned} \tag{21}$$

Measuring in the basis $\{|\psi_0\rangle\langle\psi_0|, |\psi_1\rangle\langle\psi_1|\}$, where

$$|\psi_1\rangle = \frac{1}{\sqrt{2}}|e_1,\ldots,e_\ell\rangle|e_1,\ldots,e_\ell\rangle - \frac{1}{\sqrt{2}}|e_{\ell+1},\ldots,e_{2\ell}\rangle|e_{\ell+1},\ldots,e_{2\ell}\rangle, \tag{22}$$

the algorithm outputs the trace with probability 1. ◀

The results of this section are summarized in the following theorem.

▶ **Theorem 15.** *In the matrix-vector query model, no quantum algorithm can compute the trace of an $n \times n$ matrix over $\mathbb{F}_2$ with probability better than 1/2 using fewer than $n/2$ queries, and there exists a quantum algorithm that succeeds with probability 1 using $\lceil n/2 \rceil$ queries.*

4.1.2 Learning the trace over $\mathbb{F}_q$

Now we prove a linear lower bound for the task of learning the trace over $\mathbb{F}_q$. The proof idea is the same as in the case $q = 2$, generalized to any finite field.

▶ **Theorem 16.** *In the matrix-vector query model over $\mathbb{F}_q$, computing the trace of an $n \times n$ matrix with probability more than $1/q$ requires at least $n/2$ queries.*

Proof. The induced representation of Z (defined in the second paragraph of Section 4.1) can be decomposed into q 1-dimensional irreps whose characters are

$$\chi_{Z,s}(M) = e(\langle Z, M\rangle + s \cdot \mathrm{tr}M) = e(\langle Z + s\mathbb{1}_n, M\rangle) \tag{23}$$

for $s \in \mathbb{F}_q$. Again, recall that a phase query oracle $U(M)\colon |x, y\rangle \mapsto e(y^\top Mx)|x, y\rangle$ is a unitary representation of M. The character of U is the trace $\xi(M) \coloneqq \mathrm{tr}(U(M)) = \sum_{x,y\in\mathbb{F}_q^n} e(y^\top Mx)$. The optimal success probability is determined by the irrep content of $U^{\otimes t}$, and the character of $U^{\otimes t}$ is ξ^t, satisfying

$$\mathrm{tr}(U^{\otimes t}(M)) = \xi^t(M) = \sum_{x_1,\ldots,x_t,y_1,\ldots,y_t\in\mathbb{F}_q^n} e\left(\sum_{i=1}^t y_i^\top Mx_i\right). \tag{24}$$

Thus for every $s \in \mathbb{Z}_m$,

$$m_{Z,s}^{(t)} = \langle \xi^t, \chi_{Z,s}\rangle > 0 \iff Z + s \cdot \mathbb{1}_n \in R_t, \tag{25}$$

where R_t is the set of matrices of rank no more than t. Since $\mathbb{1}_n \notin R_{n-1}$, we conclude for $t < n/2$ the success probability is at most $1/q$, as claimed. ◀

4.2 Null space

In this section, we show a linear lower bound on the matrix-vector quantum query complexity of computing the rank of a matrix $M \in \mathbb{F}_q^{m\times n}$ for $m \geq n$. This is without loss of generality since for $m < n$, by Theorem 5, we can simulate oracle access to $M^\top$ using one query to M.

The rank problem is an instance of the hidden subgroup problem (HSP) over $\mathbb{F}_q^m$ since two vectors map to the same value if and only if their difference is in the null space. However, the lower bound for the abelian HSP [16] does not directly apply to this problem since the instance is more structured – specifically, the subgroup hiding function is a linear transformation.

We recall some standard facts from linear algebra over finite fields. For $\ell \geq m$, let $\binom{\ell}{m}_q \coloneqq \frac{\prod_{i=0}^{m-1}(q^\ell - q^i)}{\prod_{i=0}^{m-1}(q^m - q^i)}$ denote a Gaussian binomial coefficient.

▶ **Lemma 17.** *The number of m-dimensional subspaces of an ℓ-dimensional space over $\mathbb{F}_q$ is $\binom{\ell}{m}_q$.*

▶ **Lemma 18.** *For integers $k \leq m \leq \ell$ and any k-dimensional space V over $\mathbb{F}_q$, the number of m-dimensional subspaces of an ℓ-dimensional space containing V is $\binom{\ell-k}{m-k}_q$.*

For proofs of these facts, see for example [9, Lemma 9.3.2].

Computing the rank. Now we consider the problem of computing the rank of a matrix $M \in \mathbb{F}_q^{m\times n}$ for $m \geq n$. A matrix M has rank r if and only if its null space is $(n - r)$-dimensional.

By Lemma 4, the success probability of a t-query algorithm is a degree-$2t$ polynomial in δ_{xy}. This polynomial P can be written as

$$P(\delta) = \sum_{S \subseteq \mathbb{F}_q^n \times \mathbb{F}_q^m} c_S \prod_{(x,y)\in S} \delta_{xy}, \tag{26}$$

with $c_S = 0$ for $|S| > \deg(P)$. For an input M, the assignments to these variables are $\delta_{xy} = \delta[Mx = y]$; we will sometimes write $\delta_{xy} = \delta_{xy}(M)$ to emphasize that δ is a function of M.

Now symmetrize by averaging over all matrices with nullity d, giving

$$\begin{aligned} Q(d) &\coloneqq \mathop{\mathbb{E}}_{M\sim Y_d}[P(\delta(M))] \\ &= \sum_{S \subseteq \mathbb{F}_q^n \times \mathbb{F}_q^m} c_S \mathop{\mathbb{E}}_{M\sim Y_d}\left[\prod_{(x,y)\in S} \delta_{xy}(M)\right] \\ &= \sum_{S \subseteq \mathbb{F}_q^n \times \mathbb{F}_q^m} c_S \Pr_{M\sim Y_d}[Mx = y \ \forall (x,y) \in S], \end{aligned} \tag{27}$$

where Y_d is the set of matrices of nullity d. Here M is drawn uniformly from Y_d. Since $0 \le P(\delta(M)) \le 1$, we have $0 \le Q(d) \le 1$. The following lemma states that we can approximate $Q(d)$ with a low-degree polynomial. Van Apeldoorn and Gribling previously showed the same statement in their proof of a lower bound for Simon's problem for linear functions [4, Lemma 3]. That problem can be viewed as a special case of our problem with $m = n$. We observe that essentially the same proof establishes this lemma for $m \ge n$.

▶ **Lemma 19.** *There exists a polynomial R of degree at most $2t$ such that for each $d \in [n]$, $R(q^d) = Q(d)$.*

We emphasize that we do not bound the degree of $Q(d)$ because we do not know how to represent it as a polynomial in d. Instead, the lower bound is established by showing (i) a lower bound on the degree of the polynomial R and (ii) that the degree of R is no more than $2t$.

Next, recall a lemma by Koiran, Nesme, and Portier [16, Lemma 5].

▶ **Lemma 20.** *Let $c > 0$ and $\xi > 1$ be constants and let f be a real polynomial with the following properties:*
1. *for any integer $0 \le i \le n$, $|f(\xi^i)| \le 1$;*
2. *for some real number $1 \le x_0 \le \xi$, $|f'(x_0)| \ge c$.*

Then $\deg f = \Omega(n)$.

Lemma 19 and Lemma 20 imply an $\Omega(\min\{m,n\})$ lower bound for distinguishing a matrix is full-rank or has nullity 1. The case $m = n$ was previously shown by van Apeldoorn and Gribling [4, Theorem 1]. We briefly explain the main ideas for completeness. By Lemma 19, for $d \in \{0, 1, \ldots, n-1\}$, $R(q^d) = Q(d)$ and $\deg(R) \le 2t$. For distinguishing a full-rank matrix (i.e., $d = 0$) from a rank $n-1$ matrix (i.e, $d = 1$), we set $R(1) \ge 1 - \epsilon$ and $R(q) \le \epsilon$. There exists $x_0 \in [1, q]$ such that $R'(x_0) \ge \frac{|R(q)-R(1)|}{q-1} \ge \frac{1-2\epsilon}{q-1}$. By Lemma 20, $t = \Omega(n)$ for $m \ge n$. For $m < n$, an $\Omega(m)$ lower bound follows from Theorem 5. Overall, this gives the following.

▶ **Theorem 21.** *The bounded-error matrix-vector quantum query complexity of deciding if an $m \times n$ matrix over $\mathbb{F}_q$ is full-rank is $\Omega(\min\{m,n\})$. In particular, $\Omega(\min\{m,n\})$ queries are needed to decide whether the matrix is full-rank or has nullity 1.*

There is a trivial algorithm that learns an entire $m \times n$ matrix using $\min\{m, n\}$ queries. Thus the query complexity of computing the rank is $\Theta(\min\{m, n\})$.

▶ **Corollary 22.** *The bounded-error query matrix-vector quantum complexity of computing the rank of an $m \times n$ matrix over $\mathbb{F}_q$ is $\Theta(\min\{m, n\})$.*

With the same argument, the quantum query complexity of computing the determinant of an $n \times n$ matrix over $\mathbb{F}_q$ is $\Theta(n)$. Moreover, the classical query complexity is $\Theta(n^2)$, implied by the $\Omega(n^2)$ lower bound for rank testing by Rashtchian, Woodruff, and Zhu [19, Theorem 3.3].

▶ **Corollary 23** (Determinant). *The bounded-error classical and quantum query complexities of computing the determinant of an $n \times n$ matrix over $\mathbb{F}_q$ through matrix-vector products are $\Theta(n^2)$ and $\Theta(n)$, respectively.*

4.3 Solving linear systems

In this section, we consider the quantum query complexity of solving the linear system $Ax = b$ for $A \in \mathbb{F}_q^{n \times n}$ is $\Theta(n)$. Since there is an n-query algorithm learning the entire matrix using n matrix-vector queries, we focus on the lower bound.

Our proof is based on a randomized reduction from deciding whether a submatrix is full rank. For a square matrix A, let A^{ij} be the submatrix obtained by deleting the i^{th} row and the j^{th} column, and let A_{ij} denote the (i, j) element of A. The elements of A^{-1} can be computed as

$$(A^{-1})_{ij} = \frac{\det A^{ij}}{\det A}. \tag{28}$$

Given an invertible A, one can use a linear system solver to decide whether $(A^{-1})_{11}$ is non-zero, and thus decide if the minor A^{11} is full-rank.

In our reduction, to decide whether $M \in \mathbb{F}_q^{n \times n}$ is full-rank given access to matrix-vector products, we pad M with one extra random row and one extra random column, giving a matrix $A \in \mathbb{F}_q^{(n+1)\times(n+1)}$. We show that with sufficiently high probability, the padded matrix is full-rank. Thus, invoking a linear system solver with $b = e_1$, we learn whether $\det M = 0$. Thus the linear regression lower bound follows from Theorem 21.

▶ **Theorem 24.** *The bounded-error matrix-vector quantum query complexity of solving an $n \times n$ linear system is $\Omega(n)$.*

Proof. Assume toward contradiction that $\mathcal{A}$ is a t-query quantum algorithm for determining whether $(A^{-1})_{11}$ is non-zero for any invertible $A \in \mathbb{F}_q^{(n+1)\times(n+1)}$, succeeding with probability $p \geq 1/3$ with $t = o(n)$. We present a t-query algorithm for determining whether an $n \times n$ matrix is full-rank with probability $p(1 - 1/q)^2 \geq 1/12$.

Given access to $M \in \mathbb{F}_q^{n \times n}$, the algorithm first samples two random vectors $u, v \in \mathbb{F}_q^n$ and a random element $a \in \mathbb{F}_q$ to give the padded matrix

$$A = \begin{bmatrix} a & u^\top \\ v & M \end{bmatrix}. \tag{29}$$

The matrix-vector product $A(x_0, x^\top)^\top$ for $x_0 \in \mathbb{F}_q, x \in \mathbb{F}_q^n$ can be computed using one Mv query to Mx since

$$A \begin{bmatrix} x_0 \\ x \end{bmatrix} = \begin{bmatrix} a_0 + u^\top x \\ x_0 v + Mx \end{bmatrix}. \tag{30}$$

We show that with probability at least $(1-1/q)^2$, the matrix A is invertible (i.e., $\det A \neq 0$) given that $\operatorname{rank}(M) \geq n-1$. If M is invertible, the submatrix $B = (v, M)$ is full-rank. If $\operatorname{rank}(M) = n-1$, then without loss of generality, we consider the case that the first $n-1$ rows of M are linearly independent, and the last row is a linear combination of the first $n-1$ rows, since other cases can be handled accordingly by rearranging the rows. We let

$$M = \begin{bmatrix} M' \\ w^\top \end{bmatrix}. \tag{31}$$

for an $(n-1) \times n$ matrix M' and an $n \times 1$ vector w. Since $w^\top$ is a linear combination of the first $n-1$ rows, we write $w^\top = c^\top M'$ for an $(n-1) \times 1$ vector c. Since M' is full-rank, the vector c satisfying $w^\top = c^\top M'$ is unique. Now write the vector

$$v = \begin{bmatrix} z \\ b \end{bmatrix} \tag{32}$$

for an $(n-1) \times 1$ matrix z and $b \in \mathbb{F}_q$. The matrix B is not full rank if and only if the last row is a linear combination of the first $n-1$ rows, i.e., $c^\top z = b$, since the first $n-1$ rows of B are linearly independent. Since v is a random vector with each element chosen independently, we have

$$\Pr[B \text{ is not full-rank}] = \Pr_{z,b}[c^\top z = b] = 1/q. \tag{33}$$

Thus with probability at least $1 - 1/q$ the matrix B is full-rank.

Conditioned on B being full-rank, the matrix A is not full-rank if and only if the vector $(a, u^\top)$ is in the vector space spanned by the rows of B. The number of vectors in the vector space is $q^{(n-1)}$. Thus

$$\Pr_{a,u,v}[A \text{ is not full-rank} \mid B \text{ is full-rank}] = 1/q. \tag{34}$$

Therefore with probability at least $1 - 1/q$, A is invertible. Conditioned on successfully simulating Mv queries of an invertible A, the algorithm $\mathcal{A}$ determines whether $(A^{-1})_{11}$ is nonzero with probability p. Thus the algorithm succeeds with probability at least $p(1-1/q)^2 \geq 1/12$ using $t = o(n)$ queries to M. By Theorem 21 we have a contradiction. ◀

The same proof idea shows that a lower bound for rank testing implies a lower bound for linear regression in the vMv model. Rashtchian, Woodruff, and Zhu show that the query complexity of distinguishing rank-n matrices from rank-$(n-1)$ matrices over $\mathbb{F}_q$ is $\Omega(n^2)$ [19, Theorem 3.3].

▶ **Corollary 25.** *The bounded-error classical* vMv *query complexity of solving an* $n \times n$ *linear system over* $\mathbb{F}_q$ *is* $\Omega(n^2)$.

Proof. By the same idea as in the proof of Theorem 24, it suffices to show that one vMv query to the $(n+1) \times (n+1)$ matrix A in (29) can be simulated with one vMv query to the $n \times n$ matrix M. For any query x, y, we let $x = (x_0, x_1^\top)^\top$ and $y = (y_0, y_1^\top)^\top$ for $n \times 1$ matrices x_1, y_1. The product $y^\top A x$ can be computed using one vMv query to M since $y^\top A x = a y_0 x_0 + y_0 u^\top x_1 + y_1^\top v x_0 + y_1^\top M x_1$. Since no $o(n^2)$-query classical algorithm can distinguish rank-n matrices from rank-$(n-1)$ matrices [19, Theorem 3.3], the bounded-error query complexity of solving linear systems is $\Omega(n^2)$. ◀

4.4 Rank testing

In this section, we show a linear lower bound on distinguishing whether an $m \times n$ matrix M has $\mathrm{rank}(M) = n$ or $\mathrm{rank}(M) \leq n/2$, where $m \geq n$. First we show the following lemma using ideas from [16].

▶ **Lemma 26.** *Let $\xi \geq 2$ and let n be an even integer. Then any polynomial f satisfying*
1. $0 \leq f(\xi^i) \leq 1$ *for* $i \in \{0, 1, \ldots, n-1\}$ *and*
2. $f(1) \leq 1/3$ *and* $f(\xi^i) \geq 2/3$ *for* $i \in \{n/2, n/2+1, \ldots, n-1\}$

has $\deg(f) = \Omega(n)$.

Proof. Let $d = \deg(f)$. Toward contradiction, we assume $d = o(n)$. For intervals $S_i := [\xi^i, \xi^{i+1})$, since $\deg(f'), \deg(f'') = o(n)$, there exists an index $a \in \{9n/10, \ldots, n-3, n-2\}$ such that none of the roots of f' and f'' has its real part in S_a. This implies that f' is monotonically increasing or decreasing in S_a, i.e., f is concave or convex. In each case, $f(\frac{\xi^a+\xi^{a+1}}{2}) \in [0, 1]$. If f is convex in S_a,

$$\left|f'\Big(\frac{\xi^a+\xi^{a+1}}{2}\Big)\right| \leq \frac{1}{\xi^{a+1} - \frac{\xi^{a+1}+\xi^a}{2}} = \frac{2}{\xi^{a+1}-\xi^a} \leq \frac{2}{\xi^a} \leq 2\xi^{-9n/10}. \tag{35}$$

If f is concave in S_a, reflecting about the x-axis gives the same bound.

By the second constraint, there exists $x_0 \in [1, \xi^{n/2}]$ such that

$$|f'(x_0)| \geq \frac{|f(\xi^{n/2}) - f(1)|}{\xi^{n/2}-1} \geq \xi^{-n/2}/3. \tag{36}$$

Therefore

$$\left|\frac{f'(\frac{\xi^a+\xi^{a+1}}{2})}{f'(x_0)}\right| \leq 6\xi^{-2n/5} \leq \xi^{3-2n/5}. \tag{37}$$

On the other hand, since $\deg(f') = d-1$, denoting the roots $a_1, \ldots, a_{d-1} \in \mathbb{C}$, we write

$$f'(x) = \lambda \prod_{i=1}^{d-1}(x - a_i). \tag{38}$$

Thus

$$\left|\frac{f'(\frac{\xi^a+\xi^{a+1}}{2})}{f'(x_0)}\right| = \prod_{i=1}^{d-1}\left|\frac{\frac{\xi^a+\xi^{a+1}}{2} - a_i}{x_0 - a_i}\right| = \prod_{i=1}^{d-1}|g(a_i)|, \tag{39}$$

where

$$g(x) = \frac{x - \frac{\xi^a+\xi^{a+1}}{2}}{x - x_0}. \tag{40}$$

Our goal is to show that for each i, $|g(a_i)| \geq \frac{1}{2\xi}$. Recall that for each i, $\Re(a_i) \notin S_a$. Also for real $x \notin S_a$, $x \geq x_0$, we have $|g(x)| \geq \frac{\xi-1}{2\xi} \geq \frac{1}{2\xi}$. For real roots, $|g(a_i)| \geq \frac{1}{2\xi}$. Now we consider the case where $a_i = \alpha + \beta i$ for $\beta \neq 0$, giving

$$|g(\alpha+\beta i)|^2 = \frac{(\alpha - \frac{\xi^a+\xi^{a+1}}{2})^2 + \beta^2}{(\alpha - x_0)^2 + \beta^2}. \tag{41}$$

If $(\alpha - \frac{\xi^a+\xi^{a+1}}{2})^2 \geq (\alpha - x_0)^2$, then $|g(\alpha+\beta i)| \geq 1$. Otherwise,

$$|g(\alpha+\beta i)| \geq \left| \frac{\alpha - \frac{\xi^a+\xi^{a+1}}{2}}{\alpha - x_0} \right| \geq \frac{1}{2\xi}. \tag{42}$$

We have shown that $|g(a_i)| \geq \frac{1}{2\xi}$ for every root a_i. Now we have

$$\left| \frac{f'(\frac{\xi^a+\xi^{a+1}}{2})}{f'(x_0)} \right| = \prod_{i=1}^{d-1} |g(a_i)| \geq (2\xi)^{-d+1} \geq \xi^{2-2d}. \tag{43}$$

Thus by (37), we have $\xi^{3-2n/5} \geq \xi^{2-2d}$ and conclude $d \geq n/5 - 1/2 = \Omega(n)$ – a contradiction. ◀

Lemma 19 and Lemma 26 imply the following theorem.

▶ **Theorem 27.** *The bounded-error matrix-vector quantum query complexity of determining whether a matrix* $M \in \mathbb{F}_q^{m\times n}$ *has* $\operatorname{rank}(M) = n$ *or* $\operatorname{rank}(M) \leq n/2$ *is* $\Omega(n)$.

References

1 Scott Aaronson. Quantum lower bound for the collision problem. In *Proceedings of the 34th Annual ACM Symposium on Theory of Computing*, pages 635–642, 2002. arXiv:quant-ph/0111102.

2 Scott Aaronson. Read the fine print. *Nature Physics*, 11(4):291, 2015.

3 Joran van Apeldoorn, András Gilyén, Sander Gribling, and Ronald de Wolf. Convex optimization using quantum oracles. *Quantum*, 4:220, 2020. arXiv:1809.00643.

4 Joran van Apeldoorn and Sander Gribling. Simon's problem for linear functions, 2018. arXiv:1810.12030.

5 Simon Apers and Ronald de Wolf. Quantum speedup for graph sparsification, cut approximation and laplacian solving. In *Proceedings of the 61st IEEE Annual Symposium on Foundations of Computer Science*, pages 637–648. IEEE, 2020. arXiv:1911.07306.

6 Robert Beals, Harry Buhrman, Richard Cleve, Michele Mosca, and Ronald de Wolf. Quantum lower bounds by polynomials. *Journal of the ACM*, 48(4):778–797, 2001. arXiv:quant-ph/9802049.

7 Ethan Bernstein and Umesh Vazirani. Quantum complexity theory. *SIAM Journal on Computing*, 26(5):1411–1473, 1997.

8 Mark Braverman, Elad Hazan, Max Simchowitz, and Blake Woodworth. The gradient complexity of linear regression. In *Conference on Learning Theory*, pages 627–647, 2020. arXiv:1911.02212.

9 Andries E. Brouwer, Arjeh M. Cohen, and Arnold Neumaier. *Distance-Regular Graphs*. Springer-Verlag, 1989.

10 Shouvanik Chakrabarti, Andrew M. Childs, Tongyang Li, and Xiaodi Wu. Quantum algorithms and lower bounds for convex optimization. *Quantum*, 4:221, 2020. arXiv:1809.01731.

11 Andrew M. Childs. Equation solving by simulation. *Nature Physics*, 5:861, 2009. doi:10.1038/nphys1473.

12 Daniel Copeland and Jamie Pommersheim. Quantum query complexity of symmetric oracle problems. *Quantum*, 5:403, 2021. arXiv:1812.09428.

13 Ankit Garg, Robin Kothari, Praneeth Netrapalli, and Suhail Sherif. No quantum speedup over gradient descent for non-smooth convex optimization. In *12th Innovations in Theoretical Computer Science Conference (to appear)*, 2021. arXiv:2010.01801.

14 Aram W. Harrow, Avinatan Hassidim, and Seth Lloyd. Quantum algorithm for linear systems of equations. *Physical Review Letters*, 103(15):150502, 2009. arXiv:0811.3171.

15 Stephen P. Jordan. Fast quantum algorithm for numerical gradient estimation. *Physical Review Letters*, 95(5):050501, 2005. arXiv:quant-ph/0405146.

16 Pascal Koiran, Vincent Nesme, and Natacha Portier. The quantum query complexity of the abelian hidden subgroup problem. *Theoretical Computer Science*, 380(1-2):115–126, 2007.

17 Troy Lee, Miklos Santha, and Shengyu Zhang. Quantum algorithms for graph problems with cut queries. In *Proceedings of the 32nd ACM-SIAM Symposium on Discrete Algorithms*, pages 939–958. SIAM, 2021. arXiv:2007.08285.

18 Ashley Montanaro and Changpeng Shao. Quantum algorithms for learning graphs and beyond, 2020. arXiv:2011.08611.

19 Cyrus Rashtchian, David P. Woodruff, and Hanlin Zhu. Vector-matrix-vector queries for solving linear algebra, statistics, and graph problems. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques (APPROX/RANDOM 2020)*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2020. arXiv:2006.14015.

20 Jean-Pierre Serre. *Linear Representations of Finite Groups*, volume 42 of *Graduate Texts in Mathematics*. Springer, 1977.

21 Xiaoming Sun, David P. Woodruff, Guang Yang, and Jialin Zhang. Querying a matrix through matrix-vector products. In *46th International Colloquium on Automata, Languages, and Programming*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2019. arXiv:1906.05736.

22 David P. Woodruff. Sketching as a tool for numerical linear algebra. *Foundations and Trends in Theoretical Computer Science*, 10(1–2):1–157, 2014. arXiv:1411.4357.

Truthful Allocation in Graphs and Hypergraphs

George Christodoulou
University of Liverpool, UK

Elias Koutsoupias
University of Oxford, UK

Annamária Kovács
Goethe University, Frankfurt am Main, Germany

Abstract

We study truthful mechanisms for allocation problems in graphs, both for the minimization (i.e., scheduling) and maximization (i.e., auctions) setting. The minimization problem is a special case of the well-studied unrelated machines scheduling problem, in which every given task can be executed only by two pre-specified machines in the case of graphs or a given subset of machines in the case of hypergraphs. This corresponds to a multigraph whose nodes are the machines and its hyperedges are the tasks. This class of problems belongs to multidimensional mechanism design, for which there are no known general mechanisms other than the VCG and its generalization to affine minimizers. We propose a new class of mechanisms that are truthful and have significantly better performance than affine minimizers in many settings. Specifically, we provide upper and lower bounds for truthful mechanisms for general multigraphs, as well as special classes of graphs such as stars, trees, planar graphs, k-degenerate graphs, and graphs of a given treewidth. We also consider the objective of minimizing or maximizing the L^p-norm of the values of the players, a generalization of the makespan minimization that corresponds to $p = \infty$, and extend the results to any $p > 0$.

2012 ACM Subject Classification Theory of computation → Algorithmic mechanism design

Keywords and phrases Algorithmic Game Theory, Scheduling Unrelated Machines, Mechanism Design

Digital Object Identifier 10.4230/LIPIcs.ICALP.2021.56

Category Track A: Algorithms, Complexity and Games

Related Version *Full Version*: `https://arxiv.org/abs/2106.03724`

Funding *Elias Koutsoupias*: This work was partially supported by ERC Advanced Grant 321171 (ALGAME).

1 Introduction

This work belongs to the area of mechanism design, one of the most researched branches of Game Theory and Microeconomics with numerous applications in environments where a protocol of conduct of selfish participants is required. The goal is to design an algorithm, called mechanism, which is robust under selfish behavior and that produces a social outcome with a certain guaranteed quality. The mechanism solicits the preferences of the participants over the outcomes, in forms of bids, and then selects one of the outcomes. The challenge stems from the fact that the real preferences of the participants are private, and the participants care only about maximizing their private utilities and hence they will lie if a false report is profitable. A *truthful* mechanism provides incentives such that a truthful bid is the best action for each participant.

Despite the importance of the problem the only general positive result for multi-dimensional domains is the celebrated Vickrey-Clarke-Groves (VCG) mechanism [43, 15, 26] and its affine extensions, known as affine maximizers.

In their seminal paper on algorithmic mechanism design, Nisan and Ronen [39] proposed the scheduling problem on unrelated machines as a central problem to understand the algorithmic aspects of mechanism design. The objective is to incentivize n machines to execute m tasks, so that the maximum completion time of the machines, i.e. the makespan, is minimized. Scheduling, a problem that has been extensively studied from the classical algorithmic perspective, proved to be the perfect ground to study the limitations that truthfulness imposes on algorithm design.

Nisan and Ronen applied the VCG mechanism, the most successful generic machinery in mechanism design, which truthfully implements the outcome that maximizes the social welfare. In the case of scheduling, the allocation of the VCG is the greedy allocation in which each task is assigned to the machine with minimum processing time. This mechanism is truthful, but has a poor approximation ratio of n for the makespan. They conjectured that this is the best guarantee that can be achieved by any deterministic (polynomial-time or not) truthful mechanism and this conjecture, known as the Nisan-Ronen conjecture, is widely perceived as the holy grail in algorithmic mechanism design.

An interesting special case of the scheduling problem, which is well-understood, is the single-dimensional mechanism design in which the values of each player are linear expressions of a single parameter. The principal representative is the problem of scheduling *related* machines, where the cost of each machine can be expressed via a single parameter, its *speed*. This was first studied by Archer and Tardos [1], who showed that in contrast to the unrelated machines version, an algorithm that minimizes the makespan can be truthfully implemented – albeit in exponential time. It was subsequently shown that truthfulness has essentially no impact on the computational complexity of the problem. Specifically, a randomized truthful-in-expectation[1] PTAS was given in [18] and a deterministic PTAS was given in [14]; a PTAS is the best possible algorithm even for the pure algorithmic problem (unless $P = NP$).

1.1 Summary of Results

In this work, we show how to combine these two main positive results of VCG and single-dimensional mechanisms into a single mechanism, which we call the *Hybrid Mechanism*. This new mechanism applies to domains in which some players are multidimensional and some players are single-dimensional. A typical example is to schedule m tasks, such that task i can only be executed by player 0 and player i. In this case, player 0 is multidimensional and the other m players are single-dimensional. We call this the *star balancing* problem. This is a multidimensional mechanism design problem for which the VCG mechanism, as well as every other known mechanism, performs very poorly. However, as we show in Section 3.1, the Hybrid Mechanism has approximation ratio 2, optimal among all truthful mechanisms. We generalize the star balancing problem in three directions: *graphs/multigraphs*, *hyperstars* and also to objectives other than makespan minimization. Due to space limitations, omitted proofs are presented in the full version.

(Multi)Graphs. A generalization of the star balancing problem to graphs and multigraphs is the Unrelated Graph Balancing problem (Section 3). This is a special case of unrelated machines scheduling in which there is a (multi)graph whose nodes represent the machines

[1] This is one of the two main definitions of truthfulness for randomized mechanisms, where truth-telling maximizes the expected utility of each player.

and whose edges represent tasks that can be executed only by the incident nodes. For general graphs, all machines are multiparameter, but we can still apply the Hybrid Mechanism, if we first decompose the graph into stars and then apply the Hybrid Mechanism to each one of them. The combined mechanism, which we call the *Star-Cover Mechanism*, has surprisingly good approximation ratio for certain classes of graphs – ratio 4 for trees, 8 for planar graphs, and $2k+2$ for k-degenerate graphs (Corollary 15). These results use as ingredient the analysis of star graphs, in which the Hybrid Mechanism has approximation ratio 2.

Hyperstars. In the hyperstar version, there are k multidimensional players/machines and every task can be executed by any one of these k players or by a task-specific single-dimensional player. Specifically, there are k different *root players* (players $1, 2, \ldots, k$ with bids $(r_{ij})_{k\times m}$) and each of them are allowed to process all tasks. In addition, for each task there is one *leaf player*, which can process only this single task (players $k+1, k+2, \ldots, k+m$ with bids $(\ell_1, \ell_2, \ldots \ell_m)$). Note that the root players without the leaves form a classic input for unrelated scheduling mechanisms with k players and m-tasks. We can now state the Hybrid Mechanism for this case.

▶ **Definition 1** (Hybrid Mechanism). *The Hybrid Mechanism minimizes*

$$\min_T \left\{ \left(\min_{x^T} \sum_{i=1}^{k} \lambda_i r_i \cdot x_i^T \right) + g_T(\ell) \right\},$$

where the λ_i can be arbitrary non-negative real numbers and $(g_T)_{T\subseteq M}$ can be any functions that guarantee that the leaf players are truthful. The output of the mechanism is the subset of tasks T that are allocated to the multidimensional root players together with their allocation matrix x^T. The remaining tasks, $M \setminus T$, are allocated to the leaf players.

VCG fairs poorly, yielding approximation ratio m in this domain, but the Hybrid Mechanism has approximation ratio $k+1$, as stated in the next theorem. Due to space limitations, we provide details and proofs in the full version of the paper.

▶ **Theorem 2.** *For the hyperstar scheduling problem, the Hybrid Mechanism with $g_T(\ell) = \max_{j\notin T} \ell_j$, and with $\lambda_i = 1$, for every i, is $(k+1)$-approximate.*

In Section 4 we provide general definitions as well as necessary and sufficient conditions for truthfulness of the Hybrid Mechanism.

Mechanisms for L^p-norm optimization. In Section 5, we consider the much more general objective of minimizing or maximizing the L^p-norm of the values of the players, for $p > 0$. The scheduling problem is the special case of minimizing the L^∞-norm. We show that the Hybrid Mechanism performs very well for this much more general problem, and in some cases it has the optimal approximation ratio among all truthful mechanisms. This illustrates the applicability and usefulness of the Hybrid Mechanism in applications with various domains and objectives. We emphasize that for all these cases, even for stars, all known mechanisms such as the VCG and affine maximizers have very poor performance.

Relation to the Nisan-Ronen conjecture. Our results on (multi)graphs show that this domain may provide an easier way to attack the Nisan-Ronen conjecture. In a recent work [12], we showed a $\Omega(\sqrt{n})$ lower bound for multistars with edge multiplicity only 2, when the root player has submodular or supermodular valuations. In contrast, our results in this

work show that for additive valuations, the Star-Cover Mechanism has approximation ratio 4 on the very same multigraphs. However, the Hybrid and the Star-Cover Mechanisms have high approximation for multistars with high edge-multiplicity or for simple clique graphs. It is natural to ask whether there are other, better mechanisms for these cases. Recently we have proved a $\Omega(\sqrt{n})$ lower bound for the former case, which is the first super-constant lower bound for the Nisan-Ronen problem [11], and we conjecture that the latter case admits similarly a high, perhaps even linear, lower bound.

We remark that all previous lower bound proofs use inherently either (multi)graphs [13, 29, 11] or, recently, hypergraphs with hyperedges of small size [24, 20]. Our work provides new methodological tools to study these objects, that can help to identify certain (hyper)graph structures as good candidates for high lower bounds and to avoid those where low upper bounds exist. For example, the 2.755 lower bound construction of [24] uses a hyperstar with $k = 2$, for which the Hybrid Mechanism achieves an upper bound of 3 (Thm 2).

All our lower bounds are information theoretic and hold independently of the computational time of the mechanisms. Conversely, all upper bounds are polynomial time algorithms when the star decomposition is given. We leave it open whether computing an optimal star decomposition of a graph is in P, although it follows from our results that it can be approximated with an additive term of 1 in polynomial time (actually in linear time).

1.2 Related Work

The Nisan-Ronen conjecture [39] has become one of the central problems in Algorithmic Game Theory, and despite intensive efforts it remains open. The original paper showed that no truthful deterministic mechanism can achieve an approximation ratio better than 2 for two machines, which was later improved to 2.41 [13] for three machines, and finally to 2.618 [29] which was the best known bound for over a decade. Recent progress improved this bound to 2.755 [24], to 3 [20] and finally to the first non-constant lower bound of $1 + \sqrt{n-1}$ [11]. The best known upper bound is n [39].

The purely algorithmic problem of makespan minimization on unrelated machines is one of the most important scheduling problems. The seminal paper of Lenstra, Shmoys and Tardos [32], gave a 2-approximation algorithm, and also showed that it is NP-hard to approximate within a factor of 3/2. Closing this gap has remained open for 30 years, and is considered one of the most important open questions in scheduling.

In this work we consider the design of truthful mechanisms for the *Unrelated Graph Balancing* problem, a special but quite rich case of the unrelated machines problem, which was previously studied by Verschae and Wiese [42], for which each task can only be assigned to two machines. This can be formulated as a graph problem, where given an undirected (multi)-graph $G = (V, E)$, each vertex corresponds to a machine, and each edge corresponds to a task. The goal is to allocate each edge to one of its nodes, in a way that minimizes the maximum (weighted) in-degree.

The special case of this problem where each direction of an edge corresponds to the same processing time $t(e)$ is known as Graph Balancing, and was introduced by Ebenlendr, Krcál, and Sgall [21] who showed an 1.75-approximate algorithm and also demonstrated that the problem retains the hardness of the unrelated machines problem, by showing that it is NP-hard to approximate within a factor better than 3/2.

Graph Balancing. As was already mentioned, for the pure graph balancing problem, the best approximation ratio for classical polynomial time algorithms is 1.75 by [21]. Wang and Sitters [44] showed a different LP-based algorithm with a higher ratio of $11/6 \approx 1.83$, while Huang and Ott [27] designed a purely combinatorial approximation algorithm but with also a higher guarantee of 1.857.

Jansen and Rohwedder [28] studied the so-called *configuration LP* which was introduced by Bansal and Sviridenko [6]. They showed that it has an integrality gap of at most 1.749 breaking the 1.75 barrier of the integrality gaps of the previous LP formulations. This leaves open the possibility of using this LP to produce an approximation algorithm with a ratio better than 1.75.

Verschae and Wiese [42] studied the *unrelated* version of graph balancing (whose strategic variant we consider in this paper) and showed that the integrality gap of the configuration LP is equal to 2, which is much higher comparing to graph balancing. They also showed a 2-approximation algorithm for the problem of maximizing the minimum load, which is the best possible unless P=NP.

The problem has been studied for various special graph classes. For the case of simple graphs (also known as Graph Orientation), Asahiro et al [2] showed that the problem is in P for the case of trees, while Asahiro, Miyano and Ono [3] showed that it becomes strongly NP-hard for planar and bipartite graphs. Finally, Lee, Leung and Pinedo [31] concluded the case of trees in the case of multiple edges, showing an FPTAS which is the best possible, given that the problem in multi-graphs is immediately NP-hard even for the simple case of two vertices (due to reduction from Subset Sum).

Truthful Scheduling. The lack of progress in the original unrelated machine problem led to the study of special cases where progress has been made. Ashlagi et al.[4], resolved a restricted version of the Nisan-Ronen conjecture, for the special but natural class of *anonymous* mechanisms. Lavi and Swamy [30] studied a restricted input domain which however retains the multi-dimensional flavour of the setting. They considered inputs with only two possible values "low" and "high", that are publicly known to the designer. For this case they showed an elegant deterministic mechanism with an approximation factor of 2. They also showed that even for this setting achieving the optimal makespan is not possible under truthfulness, and provided a lower bound of 11/10. Yu [45] extended the results for a range of values, and Auletta et al. [5] studied multi-dimensional domains where the private information of the machines is a single bit.

Randomization has led to mildly improved guarantees. There are two extensions of truthfulness for randomized mechanisms; *universal truthfulness* if the mechanism is described as a probability distribution over deterministic truthful mechanisms, and *truthfulness-in-expectation*, if in expectation no player can benefit by lying. The former notion was first considered in [39] for two machines, it was later extended to n machines by Mu'alem and Schapira [38] and finally Lu and Yu [35] showed a $0.837n$-approximate mechanism, which is currently the best known. Lu and Yu [36] showed a truthful-in-expectation mechanism with an approximation guarantee of $(m+5)/2$. Mu'alem and Schapira [38], showed a lower bound of $2-1/m$, for both notions of randomization. Christodoulou, Koutsoupias and Kovács [10] extended this lower bound for fractional mechanisms, where each task can be split to multiple machines, and they also showed a fractional mechanism with a guarantee of $(m+1)/2$. The special case of two machines [34, 36] is still unresolved; currently, the best upper bound is 1.587 due to Chen, Du, and Zuluaga [9].

The case of *related* machines is well understood. It falls into the so-called *single-dimensional* mechanism design in which the valuations of a player are linear expressions of a single parameter. In this case, the cost of each machine is expressed via a single parameter, its *(inverse) speed* multiplied by the workload allocated to the machine, instead of an m-valued vector, as it is the case for the unrelated machines and the Graph Balancing setting. Archer and Tardos [1] showed that, in contrast to the unrelated machines version, the optimal

makespan can be achieved by an (exponential-time) truthful algorithm, while [14] gave a deterministic truthful PTAS which is the best possible even for the pure algorithmic problem (unless P=NP).

Truthful implementation of other objectives was considered by Mu'alem and Schapira [38] for multi-dimensional problems and by Epstein, Levin and van Stee [22] for single-dimensional ones. Leucci, Mamageishvili and Penna [33] demonstrated high lower bounds for other min-max objectives on some combinatorial optimization problems on graphs, showing essentially that VCG is the best mechanism for these problems. Minooei and Swamy [37] considered a multi-dimensional vertex cover problem, and approached in by decomposition into single parameter problems.

The Bayesian setting, where the players costs are drawn from a probability distribution has also been studied. Daskalakis and Weinberg [17] showed a mechanism that is at most a factor of 2 from the *optimal truthful mechanism*, but not with respect to the optimal makespan. Chawla et al. [8] provided bounds of prior-independent mechanisms (where the input distribution is unknown to the mechanism), while Giannakopoulos and Kyropoulou [25] showed that the VCG mechanism achieves a factor of $O(\log n/\log\log n)$ under some distributional and symmetry assumptions.

Recently Christodoulou, Koutsoupias, and Kovács [12] showed a lower bound of $\sqrt{n-1}$ for all deterministic truthful mechanisms, when the cost of processing a subset of tasks is given by a submodular (or supermodular) set function, instead of an additive function which is assumed in the standard scheduling setting.

2 Preliminaries

Scheduling. In the classical *unrelated machines scheduling* there is a set N of n machines and a set M of m tasks that need to be scheduled on the machines. The input is given by a nonnegative matrix $t = (t_{ij})_{n\times m}$: machine i needs time $t_{ij} \in \mathbb{R}_{\geq 0}$ to process task j, and her *costs* are additive, i.e., the processing time for machine i for a set of tasks $X_i \subset M$ is $t_i(X_i) := \sum_{j\in X_i} t_{ij}$. The objective is to minimize the makespan (min-max objective). An allocation to all machines $X = (X_1, X_2, \ldots, X_n)$, (which is a partition of M) can also be denoted by the characteristic matrix $x = (x_{ij})$ where $x_{ij} = 1$ if $j \in X_i$, and $x_{ij} = 0$ otherwise.

The current work essentially considers a special case of unrelated scheduling, in which every task can be processed by two designated machines. The tasks can thus be modelled by the edges of a graph, and the associated problem is also known as *Unrelated Graph Balancing*. More formally, in the Unrelated Graph Balancing problem, there is a given undirected graph $G = (V, E)$; the vertices correspond to a set of machines $N = V$ and the edges to a set of tasks $M = E$. For each edge $e \in E$ only its two incident vertices can process the job e, and they have in general different processing times $t_i(e)$, and $t_{i'}(e)$. The goal is to assign a direction to each edge $e = (i, i')$ (allocate the corresponding task) of the graph, to one of the incident vertices (machines). The *completion time* of each vertex i is then the total processing time of the jobs X_i assigned to it $t_i(X_i) = \sum_{e\in X_i} t_i(e)$. The objective is to find an allocation that minimizes the *makespan*, i.e. the maximum completion time over all vertices.

Mechanism design setting. We assume that each machine $i \in N$ is controlled by a selfish agent that is reluctant to process the tasks and the cost function t_i is private information (also called the *type* of agent i). A *mechanism* asks the agents to report *(bid)* their types t_i, and based on the collected bids it allocates the jobs, and gives payments to the agents. A player may report a false cost function $b_i \neq t_i$, if this serves her interests.

Formally, a mechanism (X, P) consists of two parts:

An allocation algorithm: The allocation algorithm X allocates the tasks to the machines depending on the players' bids $b = (b_1, \ldots, b_n)$. We denote by $X_i(b)$ the subset of tasks assigned to machine i in the bid profile b.

A payment scheme: The payment scheme $P = (P_1, \ldots, P_n)$ determines the payments also depending on the bid values b. The functions $P_1, \ldots, P_n$ stand for the payments that the mechanism hands to each agent.

The *utility* u_i of a player i is the payment that she gets minus the *actual* time that she needs to process the set of tasks assigned to her, $u_i(b) = P_i(b) - t_i(X_i(b))$. We are interested in *truthful* mechanisms. A mechanism is truthful, if for every player, reporting his true type is a *dominant strategy*. Formally,

$$u_i(t_i, b_{-i}) \geq u_i(t'_i, b_{-i}), \qquad \forall i \in N, \;\; t_i, t'_i \in \mathbb{R}^m_{\geq 0}, \;\; b_{-i} \in \mathbb{R}^{(n-1)\times m}_{\geq 0},$$

where b_{-i} denotes the reported bidvectors of all players disregarding i.

We are looking for *truthful* mechanisms with *low approximation ratio* of the allocation algorithm for the makespan irrespective of the running time to compute X and P. In other words, our lower bounds are information-theoretic and do not take into account computational issues.

A useful characterization of truthful mechanisms in terms of the following monotonicity condition, helps us to get rid of the payments and focus on the properties of the allocation algorithm.

▶ **Definition 3** (Weak Monotonicity). *An allocation algorithm X is called* weakly monotone (WMON) *if it satisfies the following property: for every two inputs $t = (t_i, t_{-i})$ and $t' = (t'_i, t_{-i})$, the associated allocations X and X' satisfy $t_i(X_i) - t_i(X'_i) \leq t'_i(X_i) - t'_i(X'_i)$.*

It is well known that the allocation function of every truthful mechanism is WMON [7], and also that this is a sufficient condition for truthfulness in convex domains [41].

The following lemma was essentially shown in [39] and has been a useful tool to show lower bounds for truthful mechanisms for several variants (see for example [13, 38]).

▶ **Lemma 4.** *Let t be a bid vector, and let $S = X_i(t)$ be the subset assigned to player i by a weakly monotone allocation X. For any bid vector $t' = (t'_i, t_{-i})$ such that only the bid of machine i has changed and in such a way that for every task in S it has decreased (i.e., $t'_{ij} < t_{ij}, j \in S$) and for every other task it has increased (i.e., $t'_{ij} > t_{ij}, j \in M \setminus S$). Then the mechanism does not change the allocation to machine i, i.e., $X_i(t') = X_i(t) = S$.*

In general, when the values of a machine change, the allocation of the other machines may change, this issue being the pivotal difficulty of truthful unrelated scheduling. Allocation algorithms that "promise" not to change the allocation of other machines as long as changing (only) t_i does not affect the set X_i, are less problematic. These allocation rules are called *local* in [39], where it is shown that local truthful mechanisms cannot have a better than n approximation.

▶ **Definition 5** (Local mechanisms). *A mechanism is* local *if for every $i \in N$, for every t_{-i}, and t_i, t'_i for which $X_i(t_i, t_{-i}) = X_i(t'_i, t_{-i})$ also holds that $X_j(t_i, t_{-i}) = X_j(t'_i, t_{-i}) \quad (\forall j \in N)$.*

There are several special classes of mechanisms that satisfy this property, perhaps the most prominent one is the class of *affine minimizers* (see, e.g., [13]).

3 Graph Balancing

In this section we focus on the (Unrelated) Graph Balancing problem, which is a special case of makespan minimization of scheduling unrelated machines. The Graph Balancing is a multi-parameter mechanism design problem that retains most of the difficulty of the Nisan-Ronen conjecture, yet has certain features that make it more amenable.

One of the difficulties in dealing with truthful mechanisms is that while truthfulness is a local property (i.e., independent truthfulness conditions, one per player), the allocation algorithm is a global function (that involves all players). Local algorithms attempt to reconcile this tension by insisting that the allocation is also "local", but they take this notion too far. The results of this work show that locality in mechanisms is very restrictive in some domains, where the Hybrid Mechanism outperforms every local mechanism.

The Graph Balancing problem is more amenable than the general scheduling problem because it exhibits another kind of locality, *domain locality*: when a machine does not get a task, we know which machines gets it. Yet, this locality is not very restrictive and the problem retains most of its original difficulty.

In this section, we take advantage of domain locality to obtain an optimal mechanism for stars. It turns out that this mechanism, the Hybrid Mechanism, is a special case of a more general mechanism. But since the Hybrid Mechanism does not apply to general graphs, we also propose the Star-Cover mechanism for general graphs: decompose the graph into stars and apply the Hybrid Mechanism independently to each star. In this way, we obtain a 4-approximation algorithm for trees and similar positive results for other types of graphs.

Makespan minimization is the special case, when $p = \infty$, of minimizing the L^p-norm of the values of the players. Other special cases of the L^p-norm optimization is the case $p = 1$, which corresponds to welfare maximization, and the case $p = 0$, which is related to Nash Social Welfare [16]. We deal with this more general problem in another section (Section 5). Most of the results and proofs of this section generalize to any $p \geq 1$. We provide almost all the proofs in this section, because we believe that the Graph Balancing problem is an important problem in its own right and because the treatment is simpler and more intuitive, and we omit most of the (more general) results of Section 5 that deals with the L^p-norm minimization, due to space limitations.

3.1 Stars and the Hybrid Mechanism

In this subsection, we focus on star graphs, where there are $n = m + 1$ players and m tasks. Player 0 is the root of the star, and has processing times given by a vector $r = (r_1, r_2, \dots r_m)$. We also refer to this player as the *root player* or *r-player*. For given bids r of the root player, and task set $T \subseteq M$ we use the short notation $r(T) = \sum_{j \in T} r_j$.

There are also m *leaf-players*, one for each leaf of the star with processing times $\ell = (\ell_1, \dots, \ell_m)$ respectively. Each task j can only be assigned to two players; either to the root, with processing time r_j, or to the leaf with processing time ℓ_j.

As usual, we denote by r_{-i} the vector of bids of the root player except for the bid for task i, and similarly ℓ_{-i} denotes the bids of all leaf-players, except for player i. The vector of all input bids is given by $t = (r, \ell)$.

As we show later in the Lower Bound section (Section 3.3), all previously known mechanisms for the Unrelated Graph Balancing problem, e.g. affine minimizers and task independent mechanisms, have approximation ratio at least $\sqrt{n-1}$ for graphs, even for stars.

In contrast, we now show that the Hybrid Mechanism has constant approximation ratio for stars.

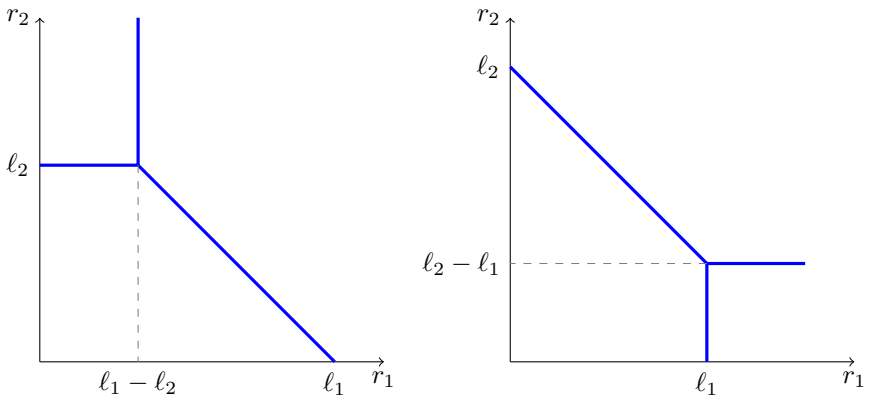


Figure 1 An instance of the Hybrid Mechanism, for the star of $m = 2$ leaves. It shows the partition of bid-space of the root player induced by the allocation of the Hybrid Mechanism when $\ell_1 \geq \ell_2$ (left) and when $\ell_2 \geq \ell_1$ (right). In the left case, the root gets both tasks in the area near $(0, 0)$, it gets only task 1 when $r_1 \leq \ell_1 - \ell_2$ and $r_2 \geq \ell_2$, and it gets neither task otherwise. Note that, in contrast to VCG, for every set of fixed values for the leaves, only three allocations are possible.

▶ **Definition 6** (Hybrid Mechanism for Graph Balancing). *Consider an instance of the Unrelated Graph Balancing problem on a star of n nodes and set of tasks M. Let*

$$S \in \arg\min_{T \subseteq M} \{r(T) + \max_{i \notin T} \ell_i\}. \tag{1}$$

The mechanism assigns a set of tasks S to the root and the remaining tasks to leaves. Ties are broken in a deterministic way (e.g., lexicographically).

Figure 1 shows the partition of the space of the root player induced by the Hybrid Mechanism for a star of two leaves.

The argmin expression that defines the Hybrid Mechanism and a corresponding expression that defines the VCG mechanism are similar: in the definition of VCG, instead of $\max_{i \notin T} \ell_i$, we have $\sum_{i \notin T} \ell_i$. It is a happy coincidence that replacing the operator sum with max preserves the truthfulness of the mechanism, a fact that rarely holds.

▶ **Lemma 7.** *The Hybrid Mechanism for Graph Balancing on stars is truthful and has approximation ratio 2.*

Proof. The root player has no incentive to lie since $-\max_{i \notin T} \ell_i$ can be interpreted as its payments. The reason that leaf players have no incentive to lie comes essentially from the fact that the expression in (1) is monotone in ℓ_i (see Section 4, for a more rigorous and extensive treatment of the truthfulness of the general Hybrid Mechanism).

Let $S^* = \arg\min_{T \subseteq M} \max\{r(T), \max_{i \notin T} \ell_i\}$ be the subset assigned to the root in the optimal allocation, OPT be the optimal makespan, and ALG be the makespan achieved by the Hybrid Mechanism. Then we have

$$ALG \leq \min_{T \subseteq M} \{r(T) + \max_{i \notin T} \ell_i\} \leq r(S^*) + \max_{i \notin S^*} \ell_i \leq 2 \max\{r(S^*), \max_{i \notin S^*} \ell_i\} = 2OPT. \quad \blacktriangleleft$$

3.2 Upper bound for general graphs and multigraphs

We now turn our attention to positive (upper bound) results for general graphs and multigraphs. We will need a few definitions first.

▶ **Definition 8** (Star decomposition). *A* star decomposition *of a (multi)graph $G(V, E)$ is a partition $T = \{T_1, \ldots, T_k\}$ of its edges into stars (see Figure 2 for an example). Let $V(T_i)$ denote the vertex set of the star spanned by T_i. The star contention number of a star decomposition is the maximum number of stars that include a node either as a root or as a leaf: $c(T) = \max_{v \in V} |\{i : v \in V(T_i), i = 1, \ldots, k\}|$. The* star contention number of a (multi)graph *is the minimum star contention number among all its star decompositions.*

In an optimal star decomposition of a graph (but not multigraph), we can assume that every node is the root of at most one star, otherwise we can merge stars with common root without changing the star contention number.

A related notion to star decomposition that has been studied extensively is the notion of edge orientation of a multigraph (or of load balancing when we consider multigraphs).

▶ **Definition 9** (Edge orientation number). *Define the orientation number of a given orientation of the edges of a multigraph G, as its maximum in-degree. The* edge orientation number $o(G)$ *of a multigraph G is the minimum orientation number among all its possible orientations.*

Indeed the two notions are closely related: every star decomposition corresponds to a graph orientation by orienting the edges in all stars from roots to leaves, and vice versa a graph orientation gives rise to a star decomposition in which every node with its outgoing edges defines a star. Given that in an optimal star decomposition of a graph, each node is the root of at most one star, we get that for *every graph G*:

$$o(G) \leq c(G) \leq o(G) + 1.$$

This relation for *multigraphs* is similar only that in the right hand side we add the maximum edge multiplicity w instead of 1, i.e., $o(G) \leq c(G) \leq o(G) + w$.

The following definition utilizes the Hybrid Mechanism on stars to obtain a general mechanism for arbitrary graphs (and multigraphs).

▶ **Definition 10** (Star-Cover Mechanism). *Let $G = (V, E)$ be a multigraph and let $T = \{T_1, \ldots, T_k\}$ be a fixed star decomposition. The Star-Cover mechanism runs the Hybrid Mechanism on every star of T independently. That is, if $S_{i,h}$ is the subset of tasks allocated to a player i by the Hybrid Mechanism when applied to a star T_h, the set of tasks allocated to player i is $S_i = \cup_{h=1}^{k} S_{i,h}$.*

We can now state and prove the general positive theorem of this section.

▶ **Theorem 11.** *The Star-Cover mechanism for a given multigraph G that uses the Hybrid Mechanism on every star of a fixed star decomposition $T = \{T_1, \ldots, T_k\}$ is truthful and has an approximation ratio at most $2c(T)$.*

Proof. Fix some player i and let $S_{i,h}$ be the subset of tasks allocated to player i by the Star Mechanism when applied to a star T_h, $h = 1, \ldots, k$. Truthfulness is an immediate consequence of the following two observations. First, since the fixed star decomposition is independent of player i's processing times, player i cannot affect it by lying. Second, $S_{i,h}$ is independent of player i's processing times $t_i(e)$ for all edges $e \notin T_h$, therefore player i cannot alter the assignment on T_h by changing its values outside T_h.

To see the approximation guarantee, let OPT, $OPT(T_h)$ be the optimal makespan on G and T_h respectively, and let ALG and $ALG(T_h)$ be the makespan achieved by the Star-Cover mechanism on G and T_h.

$$ALG \leq \max_{h=1,\ldots,k} c(T) \cdot ALG(T_h) \leq \max_{h=1,\ldots,k} c(T) \cdot 2OPT(T_h) \leq 2c(T) \cdot OPT. \qquad \blacktriangleleft$$

Due to the connection between star decompositions and edge orientations in graphs, we get

▶ **Corollary 12.** *The approximation ratio for graphs with edge orientation number $o(G)$ is at most $2o(G) + 2$.*

In the sequel, we consider particular bounds for certain classes of graphs. It is known that the edge orientation number of a given graph can be computed in polynomial time [2]. In fact, by an application of the max-flow-min-cut theorem it can be shown that $o(G) \leq \gamma$ iff for every subgraph H of G it holds that $|E(H)| \leq \gamma|V(H)|$. Since this equivalent condition[2] holds for planar graphs with $\gamma = 3$, we immediately obtain:

▶ **Theorem 13.** *For every planar graph, there exists a truthful mechanism with approximation ratio 8.*

A natural class of graphs fulfilling this property (with $\gamma = k$) is k-degenerate graphs. A graph $G(V, E)$ is called *k-degenerate* [23] (or *k-inductive*) if there is an ordering $v_1, \ldots, v_n$ of its nodes such that the number of neighbors of v_i in $\{v_{i+1}, \ldots, v_n\}$ is at most k. Many interesting classes of graphs are k-degenerate for some small k. Besides planar graphs (with $k = 5$), another example is given by k-trees [40]: by definition, a k-tree is a degenerate graph with an ordering such that every v_i (except for the last k nodes of the ordering) has exactly k neighbors in $\{v_{i+1}, \ldots, v_n\}$ and these k neighbors form a clique. Since graphs of treewidth k are subgraphs of k-trees [40], they are also k-degenerate. In particular, trees are 1-degenerate. We give here a direct proof and illustration of a star decomposition for k-degenerate graphs:

▶ **Theorem 14.** *For every k-degenerate graph, there is a truthful mechanism with approximation ratio $2k + 2$.*

Proof. Consider a k-degenerate graph G. It suffices to show that it admits a star decomposition with contention number $k + 1$. Let $v_1, \ldots, v_n$ be an inductive ordering of the nodes of G. We consider the star covering $\{T_2, \ldots, T_n\}$ where T_i is the star with root v_i and leaves all its neighbors in $\{v_1, \ldots, v_{i-1}\}$. Note that stars are created in the opposite direction of the inductive order (see Figure 2). This star decomposition has contention number $k + 1$ since every node belongs to at most one star as a root and to at most k stars as a leaf. ◀

▶ **Corollary 15.** *There exist truthful mechanisms with approximation ratio at most 4 for trees, and generally of ratio at most $2k + 2$ for graphs of treewidth k.*

3.3 Lower Bounds for Graph Balancing

In this subsection, we show corresponding negative results for the positive results of the previous subsection. We first observe that the natural candidate mechanisms for the Graph Balancing problem have very poor performance, in stark contrast to the Hybrid Mechanism.

▶ **Theorem 16.** *All local mechanisms for stars, including VCG, affine minimizers and task-independent mechanisms, have approximation ratio at least $\sqrt{m} = \sqrt{n-1}$.*

[2] This characterization of the orientation number $o(G)$ implies that a truthful mechanism with constant approximation ratio exists for any minor-closed class of graphs, because for every class of graphs with forbidden minors, there exists some constant γ that satisfies the property (see Theorems 7.2.3, 7.2.4 and Lemma 12.6.1. in [19]). We are grateful to an anonymous referee for pointing this out.

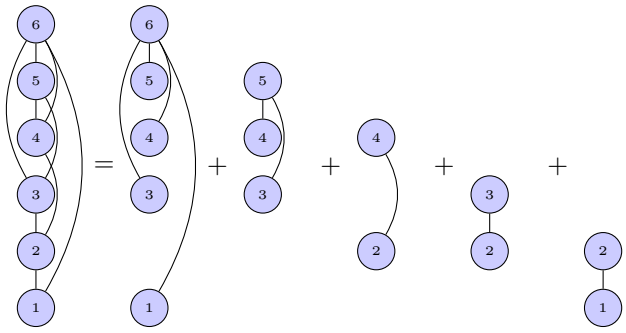


Figure 2 The star decomposition used in Theorem 14 of a 2-degenerate graph. The inductive order is upwards, while the stars are "pointing" downwards.

Proof. Consider the following input

$$t = \begin{pmatrix} \frac{1}{\sqrt{m}} & \frac{1}{\sqrt{m}} & \cdots & \frac{1}{\sqrt{m}} \\ 1 & \infty & \cdots & \infty \\ \infty & 1 & \cdots & \infty \\ \infty & \infty & \cdots & 1 \end{pmatrix}.$$

If, in the allocation of the mechanism, the root player takes all the tasks, then this allocation has approximation $\sqrt{m}$, as the optimal allocation is to assign the tasks to the leaves with makespan equal to 1. Otherwise, assume that (at least) one of the tasks, is given to some other player, say w.l.o.g. task 1 is given to player 1. By a series of applications of Lemma 4, and by exploiting the locality of the mechanism, we set the value of the owner of task j to 0 for every $j \neq 1$.

In particular, let S be the set of tasks assigned to the root player, and $M \setminus S$ be the tasks assigned to their respective leaf-player. Let $t^1 = (r', \ell_1, \ldots, \ell_m)$, with r' defined as follows for some arbitrarily small ϵ.

$$r'_j = \begin{cases} 0 & j \in S \\ \frac{1}{\sqrt{m}} + \epsilon & \text{otherwise.} \end{cases}$$

By applying Lemma 4, the root player receives again the set S, and therefore, the set $M \setminus S$ is assigned to the leaves. We proceed by changing the bids of the leaf-players for the tasks in $M \setminus S$ to 0, i.e., defining a sequence t^j for $j \in M \setminus S$, with $t^j = (r', \ell'_j = 0, \ell_{-j}^{j-1})$

Again, by Lemma 4 and by locality, we get that the allocation of the tasks remains the same for the leaf j, *and* for all the other players as well.

We end up with an instance t' where player 1 still takes the first task, while the rest of the tasks are assigned to a player with 0 processing time. For t', the optimal makespan is $1/\sqrt{m}$, while the mechanism achieves makespan equal to 1. We illustrate the case when $S = \emptyset$, that is, the allocation gives all the tasks to the leaves of the star.

$$t = \begin{pmatrix} \frac{1}{\sqrt{m}} & \frac{1}{\sqrt{m}} & \cdots & \frac{1}{\sqrt{m}} \\ \textcircled{1} & \infty & \cdots & \infty \\ \infty & \textcircled{1} & \cdots & \infty \\ \infty & \infty & \cdots & \textcircled{1} \end{pmatrix} \rightarrow t' = \begin{pmatrix} \frac{1}{\sqrt{m}} & \frac{1}{\sqrt{m}} & \cdots & \frac{1}{\sqrt{m}} \\ \textcircled{1} & \infty & \cdots & \infty \\ \infty & \textcircled{0} & \cdots & \infty \\ \infty & \infty & \cdots & \textcircled{0} \end{pmatrix}$$

◀

In the previous subsection, we showed that the Hybrid Mechanism outperforms all known mechanisms and has approximation ratio at most 2. The next theorem shows that this ratio is the best possible among all possible mechanisms for stars.

▶ **Theorem 17.** *There is no deterministic mechanism for stars that can achieve an approximation ratio better than 2.*

This is a special case of a more general lower bound for the L^p-norm objective (Theorem 30), but we give the proof here anyway, since it will be an ingredient of the proof of the following theorem (Theorem 18).

Proof. Let's assume that the mechanism takes an input where the processing time of the root player is $r_j = a^{j-1}$, for each task j, where $a > 1$ is a parameter, and the processing time of the corresponding leaf player for task j is $\ell_j = a^j$, as also shown in the following table.

$$t = \begin{pmatrix} 1 & a & \cdots & a^{m-2} & a^{m-1} \\ a & \infty & \cdots & \infty & \infty \\ \infty & a^2 & \cdots & \infty & \infty \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ \infty & \infty & \cdots & a^{m-1} & \infty \\ \infty & \infty & \cdots & \infty & a^m \end{pmatrix}$$

If the mechanism assigns all tasks to the root player, then the makespan for this input is $(a^m-1)/(a-1)$, while the optimal makespan is a^{m-1}, yielding a ratio of $(a^m-1)/((a-1)a^{m-1})$. Otherwise, let X be the nonempty set of tasks assigned to the leaf players. Let k be the task with the maximum index in X. Since it is processed by the leaf player, its processing time is a^k. Now consider the input in which we change the processing times of the root player to

$$r'_j = \begin{cases} 0 & j \notin X \\ r_j + \epsilon & \text{otherwise} \end{cases}$$

for some arbitrarily small $\epsilon > 0$. By weak monotonicity (Lemma 4), the set of tasks assigned to the root player remains the same, and as a result the whole allocation stays the same. Therefore task k is still assigned to the leaf player k and the makespan of the mechanism is at least a^k. Notice that the optimum allocation for this input is $a^{k-1} + \epsilon$ which yields an approximation ratio of a, as ϵ tends to 0.

In conclusion, the approximation ratio is $\min\{(a^m-1)/((a-1)a^{m-1}), a\}$, for every $a > 1$. By choosing $a = 2$, we see that the ratio is $2 - 1/2^{m-1}$, which shows that for the class of stars no mechanism can have approximation ratio better than 2. For fixed m, the lower bound is slightly better than $2 - 1/2^{m-1}$, by selecting a to be the positive root of the equation $(a^m - 1)/((a-1)a^{m-1}) = a$. ◀

We now show how to extend the previous result to get a lower bound of $1+\varphi \approx 2.618$ for trees, and thus for graphs. This matches the best lower bound for the Nisan-Ronen setting [29] that was known until the recent improvements [24, 20, 11], suggesting that studying the special case of scheduling in graphs may be useful in attacking the Nisan-Ronen conjecture.

▶ **Theorem 18.** *No mechanism for trees can achieve approximation ratio* $1 + \varphi \approx 2.618$.

Proof. The proof mimics the proof of Theorem 17 on the tree shown in Figure 3. The tree consists of a star with root 0 and leaves $1, \ldots, k$ in which we add a new node $\overline{v}$ for each node v of the star and connect it to v. These new nodes (players), which we call dummy will not

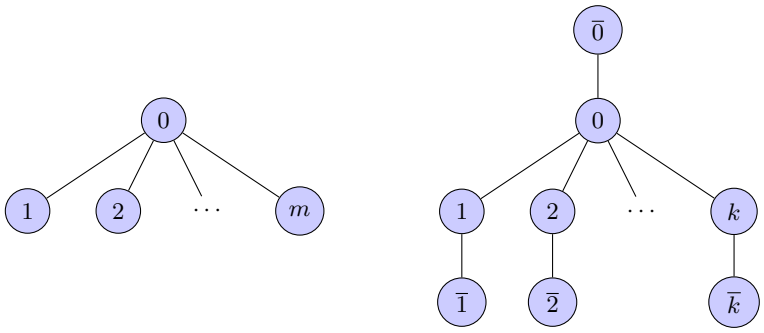


Figure 3 A star with root 0 and leaves $1, \ldots, m$ and its extension to a tree with dummy nodes.

be assigned any task by any efficient mechanism since we set their processing times to an arbitrarily high value H. The processing times of the edges of the star are exactly the same as in the proof of Theorem 17: $r_j = a^{j-1}$ and $\ell_j = a^j$, for some $a > 1$. The processing times for all edges are given below:

$$r_j = a^{j-1} \qquad \ell_j = a^j \qquad j = 1, \ldots, k$$
$$\overline{r} = 0 \qquad \overline{\ell}_j = 0$$

where $\overline{r}$ and $\overline{\ell}_j$ are the processing times of the star vertices of their respective dummy tasks. The dummy nodes themselves have a very large processing time $H \gg 1$ on these tasks.

We consider two cases. In the first case, all tasks of the star are assigned to the root player 0. We then consider a new instance in which we slightly lower the processing time of the root on the tasks of the star (i.e., $r_j = a^{j-1} - \epsilon$ for some $\epsilon > 0$) and increase the processing time of its dummy task $\overline{r} = a^k$. By weak monotonicity (Lemma 4), the r-player will take this task and all tasks of the star with a total processing time slightly less than $1 + a + \ldots + a^k = (a^{k+1} - 1)/(a - 1)$. It is easy to see that the optimal allocation for this instance is a^k, and the approximation ratio $(a^{k+1} - 1)/((a - 1)a^k)$.

In the second case, at least one task of the star is allocated to a leaf. Let p be the star task allocated to a leaf with the maximum index (that is, task p of the star is allocated to leaf-player p and tasks $p + 1, \ldots, k$ are allocated to the root). We consider the instance in which we change the processing times of the root player as follows: all processing times of the tasks allocated to the root become 0 and all processing times of the root player for the remaining tasks increase slightly. By weak monotonicity (Lemma 4), the r-player will still get the same set of tasks. We now create a new instance by increasing the processing time of the p-th dummy task: $\overline{\ell}_p = a^{p-1}$ and slightly decreasing the processing time of the leaf p for its task in the star: $\ell_p = a^p - \epsilon$, for some $\epsilon > 0$. Then again by weak monotonicity (Lemma 4), player p will get these two tasks. Although the allocation of the other tasks may change, the cost for the mechanism is at least $a^p + a^{p-1} - \epsilon$, while the optimal allocation has cost a^{p-1}. Therefore, in this case the mechanism has approximation ratio $(a^p + a^{p-1})/a^{p-1} = a + 1$, as $\epsilon \to 0$. In any case, the mechanism has approximation ratio $\min\{((a^{k+1} - 1)/((a - 1)a^k), a + 1\}$. By selecting $a = \varphi$, we get a ratio at least $1 + \varphi$ (as $k \to \infty$). ◀

Closing the gap between the above lower bound 2.618 of Theorem 18 and the upper bound 4 (Corollary 15) for mechanisms for trees is a crisp intriguing question.

4 Hybrid Mechanisms

Here we provide the general definitions related to Hybrid Mechanisms, and show necessary and sufficient conditions for truthfulness on stars (and hyper-stars[3]). We emphasize that this is a multi-dimensional mechanism design setting. Each leaf j has a single dimensional valuation, given by the scalar ℓ_j but a root has multi-dimensional preferences, given by the vector of values. For the sake of convenience, we call non-decreasing real functions *increasing*, and non-increasing functions *decreasing*. We say *strictly increasing/decreasing* if we want to emphasize strict monotonicity.

It is known, that an allocation rule can be equipped by a truthful payment scheme iff it is *weakly monotone* [41]. The next two propositions give a characterization of the weak monotonicity property in our case, for the leaf-players, and for the root player, respectively:

▶ **Proposition 19.** *An allocation rule is weakly monotone for a leaf-player i, iff for every r and every ℓ_{-i}, whenever leaf-player i gets task i with bid ℓ_i, then he also gets the task with every smaller bid $\ell_i' < \ell_i$.*

▶ **Proposition 20.** *An allocation rule is weakly monotone for the root player if and only if for every fixed bid vector ℓ of the other players, and every $T \subseteq M$ a constant $g_T(\ell)$ (i.e., independent of r) exists, such that for every r the root player is allocated a set $S \in \arg\min_T \{r(T) + g_T(\ell)\}$.*

The canonical choice for truthful payments to the r-player is then $P_S^0(\ell) = g_\emptyset(\ell) - g_S(\ell)$, and all other truthful payments can be obtained by an additive shift by an arbitrary $c(\ell)$.

We assume w.l.o.g. that for every fixed ℓ the payments P_S^0 correspond to an increasing set-function of S,[4] because a set of tasks with higher cost and less payments can not be allocated to player 0 by a truthful mechanism.[5] Motivated by Proposition 20 we restrict our search for truthful mechanisms on star graphs as follows:

▶ **Definition 21** (Hybrid Mechanism). *Assume that an m-variate function $g_T : \mathbb{R}^m \to \mathbb{R}$ is given for every $T \subseteq M$, so that for every fixed vector $\ell \geq 0$ the values $\{g_T(\ell)\}_{T \subseteq M}$ correspond to a decreasing setfunction of T. For any input (r, ℓ), a* Hybrid Mechanism (for the functions $\{g_T\}_{T \subseteq M}$) *allocates a set S to the root player such that*

$$S \in \arg\min_T \{r(T) + g_T(\ell)\};$$

if there are more than one such sets S, the mechanism breaks ties according to the lexicographic order over all subsets of M. The items in $M \setminus S$ are assigned to the leaves.

Now for any $i \in M$ fix all bids in the input except for r_i, i.e., fix the vectors r_{-i} and ℓ. The following function $\psi_i[r_{-i}, \ell]$ defines the so called *critical value* for the bid r_i. We omit the argument r_{-i}, ℓ whenever they are obvious from the context.

▶ **Definition 22.**

$$\psi_i = \psi_i[r_{-i}, \ell] = \min_{T: i \notin T} \{r(T) + g_T(\ell)\} - \min_{T: i \in T} \{r(T \setminus \{i\}) + g_T(\ell)\}$$

[3] For simplicity of presentation we give here all definitions and lemmata for the case of stars, and discuss the necessary changes for hyper-stars in the full version.

[4] We call a setfunction P *increasing*, if $P(S') \leq P(S)$ whenever $S' \subset S$; we call it *strictly increasing* if the inequality is strict.

[5] See also the *virtual payments* in [12].

The next lemma states that ψ_i is nonnegative, and is, indeed, a critical value function. The proofs are straightforward, and due to space limitations are deferred to the full version.

▶ **Lemma 23.** *Let $i \in M$, and arbitrary nonnegative bid vectors r_{-i} and ℓ be fixed. Then $\psi_i[r_{-i}, \ell] \geq 0$, furthermore for every $r_i < \psi_i$ the root player receives task i, and for every $r_i > \psi_i$ the leaf player with bid ℓ_i receives task i.*

The following lemma provides various necessary or sufficient conditions for the truthfulness of Hybrid Mechanisms in terms of monotonicity of the critical value function ψ_i as a function of ℓ_i. For the proof of the lemma see the full version. There we also present an example mechanism showing that conditions (b) and (c) are both not necessary for the Hybrid Mechanism to be truthful.

▶ **Lemma 24.** *For the truthfulness of the Hybrid Mechanism with given $\{g_T\}_{T \subseteq M}$ functions (i.e., for a truthful payment scheme to exist),*

(a) *it is necessary that for every $i \in M$ and every fixed (r_{-i}, ℓ_{-i}) the function $\psi_i(\ell_i) = \psi_i[r_{-i}, \ell_{-i}](\ell_i)$ is an increasing function of ℓ_i;*

(b) *it is sufficient that for every $i \in M$ and every fixed (r_{-i}, ℓ_{-i}) the function $\psi_i(\ell_i) = \psi_i[r_{-i}, \ell_{-i}](\ell_i)$ is a* strictly *increasing function of ℓ_i;*

(c) *it is sufficient that for every i and ℓ_{-i} the $g_T(\ell_i, \ell_{-i})$ is an increasing function of ℓ_i whenever $i \notin T$, and decreasing function of ℓ_i whenever $i \in T$.*

▶ **Corollary 25.** *The Hybrid Mechanism for Graph Balancing and the Hybrid L^p Mechanism on stars are truthful.*

Proof. The first statement follows from the fact that the Hybrid Mechanism for Graph Balancing fulfils (c). Clearly, $g_T(\ell) = \max_{i \notin T}\{\ell_i\} = \max_{i \in M \setminus T}\{\ell_i\}$ is an increasing setfunction of the sets $M \setminus T$, and therefore a decreasing setfunction of the sets T, for fixed ℓ. For fixed T, $\max_{i \notin T}\{\ell_i\}$ is an increasing function of ℓ_i for every $i \notin T$, and it is independent of ℓ_i (constant function) if $i \in T$. Finally, it is easy to see that the Hybrid L^p Mechanism (see Section 5) fulfils (b) as well as (c). ◀

5 Mechanisms for L^p-norm optimization

In this section we generalize some of the results of Section 3 to the objective of minimizing the L^p-norm of the values of the agents, i.e., minimizing, over all allocations X the expression

$$\left(\sum_{i=1}^{n} t_i(X)^p \right)^{1/p}. \tag{2}$$

The makespan scheduling problem is the special case of $p = \infty$. We consider all positive values of p, but we deal separately with the case $p \geq 1$, in which L^p is a proper norm, and the case $p \in (0, 1)$, where the L^p function is not subadditive (i.e., the triangle inequality does not hold). Due to space limitations we postpone most of the results and their proofs to the full version of the paper. There we also consider the *maximization* case, which for $p = 1$ corresponds to auctions.

Consider an instance of the Unrelated Graph Balancing problem on a star of n nodes and set of tasks M. Notice that for stars the objective of minimizing the L^p-norm corresponds to minimizing $(r(T)^p + \sum_{i \notin T} \ell_i^p)^{\frac{1}{p}}$ over all task sets $T \subseteq M$ given to the r-player.

▶ **Definition 26** (Hybrid L^p Mechanism for stars). *For a given $0 < p \leq \infty$, and an instance of the Unrelated Graph Balancing problem on a star of n nodes and set M of tasks, let*

$$S \in \arg\min_{T \subseteq M} \left\{ r(T) + \left(\sum_{i \notin T} \ell_i^p \right)^{1/p} \right\}. \tag{3}$$

The mechanism assigns S to the root and the remaining tasks to leaves. Ties are broken in a deterministic way (e.g., lexicographically).

The argmin expression that defines the Hybrid L^p Mechanism coincides with the VCG mechanism for $p = 1$ and with the Hybrid Mechanism of Section 3 for $p \to \infty$. As it is shown in Corollary 25, the Hybrid L^p mechanism is truthful.

Next we show two upper bound results for the approximation ratio (for the L^p-norm objective) separately in case $p \geq 1$, and in case $0 < p \leq 1$, respectively. We summarize here the inequalities that we will use:

▶ **Lemma 27.** *For any $p \geq 1$ it holds*

$$\sum_{i=1}^{k} x_i^p \leq \left(\sum_{i=1}^{k} x_i \right)^p \leq k^{p-1} \sum_{i=1}^{k} x_i^p. \tag{4}$$

Similarly for any $0 < p \leq 1$ it holds

$$\sum_{i=1}^{k} x_i^{1/p} \leq \left(\sum_{i=1}^{k} x_i \right)^{1/p} \leq k^{1/p-1} \sum_{i=1}^{k} x_i^{1/p}. \tag{5}$$

▶ **Theorem 28.** *For the problem of minimizing the L^p-norm, the Hybrid L^p Mechanism for stars has approximation ratio of at most $2^{(p-1)/p}$, when $p \geq 1$, and $2^{(1-p)/p}$, when $0 < p < 1$.*

Proof. Let $S^* = \arg\min_{T \subseteq M} (r(T)^p + \sum_{i \notin T} \ell_i^p)^{\frac{1}{p}}$ be the subset assigned to the root in the optimal allocation, S be the subset assigned to the root by the L^p Mechanism, OPT be the optimal L^p-norm, and ALG be the L^p-norm achieved by the Hybrid L^p Mechanism.

We first consider the case $p \geq 1$. We have

$$\begin{aligned} ALG = \left(r(S)^p + \sum_{i \notin S} \ell_i^p \right)^{1/p} &\leq r(S) + \left(\sum_{i \notin S} \ell_i^p \right)^{1/p} \\ &\leq r(S^*) + \left(\sum_{i \notin S^*} \ell_i^p \right)^{1/p} \\ &\leq 2^{(p-1)/p} \left(r(S^*)^p + \sum_{i \notin S^*} \ell_i^p \right)^{1/p} = 2^{(p-1)/p} OPT, \end{aligned}$$

where the first inequality follows from the triangle inequality, the second from the definition of the Hybrid L^p Mechanism, while the last one from Jensen's inequality (Lemma 27, Equation (4)) for $k = 2$, $x_1 = r(S^*)$, and $x_2 = (\sum_{i \notin S^*} \ell_i^p)^{1/p}$.

The case of $p < 1$, is essentially the same, but the proof is slightly different.

$$\begin{aligned} ALG = \left(r(S)^p + \sum_{i \notin S} \ell_i^p \right)^{1/p} &\leq 2^{\frac{1}{p}-1} \left(r(S) + \left(\sum_{i \notin S} \ell_i^p \right)^{1/p} \right) \\ &\leq 2^{\frac{1}{p}-1} \left(r(S^*) + \left(\sum_{i \notin S^*} \ell_i^p \right)^{1/p} \right) \\ &\leq 2^{\frac{1}{p}-1} \left(r(S^*)^p + \sum_{i \notin S^*} \ell_i^p \right)^{1/p} = 2^{\frac{1}{p}-1} OPT. \end{aligned}$$

The first inequality follows from Jensen's inequality for $x_1 = (r(S))^p$, and $x_2 = \sum_{i \notin S} \ell_i^p$; the second from the definition of the L^p Mechanism, while the last one from the fact that $(\alpha + \beta)^p \leq \alpha^p + \beta^p$, when $0 < p \leq 1$. ◀

As in the case of makespan, we can use the mechanism to other domains by decomposing them. We can apply the Star-Cover mechanism (Definition 10) to get good approximation ratios for general domains:

▶ **Theorem 29.** *For $p \geq 1$, the Star-Cover mechanism for a given multigraph G that uses the Hybrid L^p Mechanism on every star of a fixed star decomposition $T = \{T_1, \ldots, T_k\}$ is truthful and has an approximation ratio at most $(2c(T))^{(p-1)/p}$ of the L^p-norm of the machines' costs, where $c(T)$ is the star contention number of the decomposition.*

We also provide corresponding negative results for mechanisms. For the case, of $p \geq 1$, the next theorem shows that the Hybrid L^p Mechanism has optimal approximation ratio.

▶ **Theorem 30.** *For any $p \geq 1$, there is no deterministic mechanism for stars that can achieve an approximation ratio better than $2^{1-1/p}$ for the L^p-objective.*

We point out that all known (local) mechanisms perform much worse that the Hybrid Mechanism. Observe that for $p = 1$, the VCG is optimal, but for large p the inefficiency of all local mechanisms grows and tends to $\sqrt{m}$:

▶ **Theorem 31.** *For minimizing the L^p-norm on stars, all* local *mechanisms, including affine minimizers and task-independent mechanisms, have approximation ratio of at least $m^{\frac{1}{2}(1-1/p)} = (n-1)^{\frac{1}{2}(1-1/p)}$, when $p \geq 1$.*

The lower bound that we give for the case of $p < 1$ does not match exactly the upper bound, which leaves open the possibility that there exists a mechanism with better approximation ratio than the Hybrid L^p Mechanism. Notice that the following approximation ratio tends to infinity as p tends to 0.

▶ **Theorem 32.** *For any $0 < p \leq 1$ and every $a > 1$, there is no deterministic mechanism for stars that can achieve an approximation ratio better than*

$$\min\left\{a, \frac{(a+1)^{1/p}}{a^{1/p} + a)}\right\}. \tag{6}$$

By selecting an appropriate a, this is $\Omega(p^{-1}/\ln(p^{-1}))$.

References

1 Aaron Archer and Éva Tardos. Truthful mechanisms for one-parameter agents. In *Proc. of the 42nd IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 482–491, 2001.

2 Yuichi Asahiro, Jesper Jansson, Eiji Miyano, and Hirotaka Ono. Graph orientation to maximize the minimum weighted outdegree. *Int. J. Found. Comput. Sci.*, 22(3):583–601, 2011. doi:10.1142/S0129054111008246.

3 Yuichi Asahiro, Eiji Miyano, and Hirotaka Ono. Graph classes and the complexity of the graph orientation minimizing the maximum weighted outdegree. *Disc. Appl. Math.*, 159(7):498–508, 2011.

4 Itai Ashlagi, Shahar Dobzinski, and Ron Lavi. Optimal lower bounds for anonymous scheduling mechanisms. *Mathematics of Operations Research*, 37(2):244–258, 2012. doi:10.1287/moor.1110.0534.

5 Vincenzo Auletta, George Christodoulou, and Paolo Penna. Mechanisms for scheduling with single-bit private values. *Theory Comput. Syst.*, 57(3):523–548, 2015. doi:10.1007/s00224-015-9625-5.

6 Nikhil Bansal and Maxim Sviridenko. The Santa Claus problem. In *STOC*, pages 31–40. ACM, 2006. doi:10.1145/1132516.1132522.

7 Sushil Bikhchandani, Shurojit Chatterji, Ron Lavi, Ahuva Mu'alem, Noam Nisan, and Arunava Sen. Weak monotonicity characterizes deterministic dominant strategy implementation. *Econometrica*, 74(4), 2006.

8 Shuchi Chawla, Jason D. Hartline, David L. Malec, and Balasubramanian Sivan. Prior-independent mechanisms for scheduling. In *STOC'13*, pages 51–60. ACM, 2013. doi:10.1145/2488608.2488616.

9 Xujin Chen, Donglei Du, and Luis Fernando Zuluaga. Copula-based randomized mechanisms for truthful scheduling on two unrelated machines. *Theory Comput. Syst.*, 57(3):753–781, 2015.

10 George Christodoulou, Elias Koutsoupias, and Annamária Kovács. Mechanism design for fractional scheduling on unrelated machines. *ACM Transactions on Algorithms*, 6(2), 2010. doi:10.1145/1721837.1721854.

11 George Christodoulou, Elias Koutsoupias, and Annamária Kovács. On the Nisan-Ronen conjecture. *CoRR*, abs/2011.14434, 2020. arXiv:2011.14434.

12 George Christodoulou, Elias Koutsoupias, and Annamária Kovács. On the Nisan-Ronen conjecture for submodular valuations. In *STOC*, pages 1086–1096. ACM, 2020.

13 George Christodoulou, Elias Koutsoupias, and Angelina Vidali. A lower bound for scheduling mechanisms. *Algorithmica*, 55(4):729–740, 2009. doi:10.1007/s00453-008-9165-3.

14 George Christodoulou and Annamária Kovács. A deterministic truthful ptas for scheduling related machines. *SIAM J. Comput.*, 42(4):1572–1595, 2013. doi:10.1137/120866038.

15 Edward H. Clarke. Multipart pricing of public goods. *Public Choice*, 8, 1971.

16 Richard Cole and Vasilis Gkatzelis. Approximating the Nash social welfare with indivisible items. *SIAM Journal on Computing*, 47(3):1211–1236, January 2018.

17 Constantinos Daskalakis and S. Matthew Weinberg. Bayesian truthful mechanisms for job scheduling from bi-criterion approximation algorithms. In *SODA*, pages 1934–1952. SIAM, 2015. doi:10.1137/1.9781611973730.130.

18 Peerapong Dhangwatnotai, Shahar Dobzinski, Shaddin Dughmi, and Tim Roughgarden. Truthful approximation schemes for single-parameter agents. *SIAM J. on Computing*, 40(3):915–933, 2011. doi:10.1137/080744992.

19 Reinhard Diestel. *Graph Theory, 4th Edition*. Springer, 2012.

20 Shahar Dobzinski and Ariel Shaulker. Improved lower bounds for truthful scheduling. *CoRR*, abs/2007.04362, 2020. arXiv:2007.04362.

21 Tomás Ebenlendr, Marek Krcál, and Jirí Sgall. Graph balancing: A special case of scheduling unrelated parallel machines. *Algorithmica*, 68(1):62–80, 2014. doi:10.1007/s00453-012-9668-9.

22 Leah Epstein, Asaf Levin, and Rob van Stee. A unified approach to truthful scheduling on related machines. In *Proc. SODA*, pages 1243–1252, 2013.

23 Paul Erdős and András Hajnal. On chromatic number of graphs and set-systems. *Acta Mathematica Hungarica*, 17(1-2):61–99, 1966.

24 Yiannis Giannakopoulos, Alexander Hammerl, and Diogo Poças. A new lower bound for deterministic truthful scheduling. In *SAGT 2020*, volume 12283, pages 226–240. Springer, 2020. doi:10.1007/978-3-030-57980-7_15.

25 Yiannis Giannakopoulos and Maria Kyropoulou. The VCG mechanism for bayesian scheduling. *ACM Trans. Economics and Comput.*, 5(4):19:1–19:16, 2017. doi:10.1145/3105968.

26 Theodore Groves. Incentives in teams. *Econometrica*, 41(4):617–631, 1973.

27 Chien-Chung Huang and Sebastian Ott. A combinatorial approximation algorithm for graph balancing with light hyper edges. In *ESA 2016*, volume 57 of *LIPIcs*, pages 49:1–49:15, 2016. doi:10.4230/LIPIcs.ESA.2016.49.

28 Klaus Jansen and Lars Rohwedder. Local search breaks 1.75 for graph balancing. In *ICALP 2019*, volume 132 of *LIPIcs*, pages 74:1–74:14, 2019.

29 Elias Koutsoupias and Angelina Vidali. A lower bound of $1+\phi$ for truthful scheduling mechanisms. *Algorithmica*, pages 1–13, 2012.

30 Ron Lavi and Chaitanya Swamy. Truthful mechanism design for multidimensional scheduling via cycle monotonicity. *Games and Economic Behavior*, 67(1):99–124, 2009. doi:10.1016/j.geb.2008.08.001.

31 Kangbok Lee, Joseph Y.-T. Leung, and Michael L. Pinedo. A note on graph balancing problems with restrictions. *Inf. Process. Lett.*, 110(1):24–29, 2009. doi:10.1016/j.ipl.2009.09.015.

32 Jan Karel Lenstra, David B Shmoys, and Eva Tardos. Approximation algorithms for scheduling unrelated parallel machines. *Mathematical programming*, 46(1-3):259–271, 1990.

33 Stefano Leucci, Akaki Mamageishvili, and Paolo Penna. No truthful mechanism can be better than n approximate for two natural problems. *Games and Economic Behavior*, 111:64–74, 2018. doi:10.1016/j.geb.2018.05.003.

34 Pinyan Lu. On 2-player randomized mechanisms for scheduling. In *WINE*, volume 5929 of *Lecture Notes in Computer Science*, pages 30–41. Springer, 2009. doi:10.1007/978-3-642-10841-9_5.

35 Pinyan Lu and Changyuan Yu. An improved randomized truthful mechanism for scheduling unrelated machines. In *STACS*, volume 1 of *LIPIcs*, pages 527–538, 2008.

36 Pinyan Lu and Changyuan Yu. Randomized truthful mechanisms for scheduling unrelated machines. In *WINE*, pages 402–413, 2008. doi:10.1007/978-3-540-92185-1_46.

37 Hadi Minooei and Chaitanya Swamy. Truthful mechanism design for multidimensional covering problems. In *WINE 2012*, volume 7695 of *LNCS*, pages 448–461. Springer, 2012. doi:10.1007/978-3-642-35311-6_33.

38 Ahuva Mu'alem and Michael Schapira. Setting lower bounds on truthfulness. *Games and Economic Behavior*, 110:174–193, 2018.

39 Noam Nisan and Amir Ronen. Algorithmic mechanism design. *Games and Economic Behavior*, 35:166–196, 2001.

40 Donald J Rose. On simple characterizations of k-trees. *Disc. Math.*, 7(3-4):317–322, 1974.

41 Michael E. Saks and Lan Yu. Weak monotonicity suffices for truthfulness on convex domains. In *Proceedings 6th ACM Conference on Electronic Commerce (EC)*, pages 286–293, 2005. doi:10.1145/1064009.1064040.

42 José Verschae and Andreas Wiese. On the configuration-lp for scheduling on unrelated machines. *J. Scheduling*, 17(4):371–383, 2014. doi:10.1007/s10951-013-0359-4.

43 William Vickrey. Counterspeculations, auctions and competitive sealed tenders. *Journal of Finance*, 16:8–37, 1961.

44 Chao Wang and René Sitters. On some special cases of the restricted assignment problem. *Inf. Process. Lett.*, 116(11):723–728, 2016.

45 Changyuan Yu. Truthful mechanisms for two-range-values variant of unrelated scheduling. *Theoretical Computer Science*, 410(21-23):2196–2206, 2009. doi:10.1016/j.tcs.2009.02.001.

Towards the k-Server Conjecture: A Unifying Potential, Pushing the Frontier to the Circle

Christian Coester
CWI, Amsterdam, The Netherlands

Elias Koutsoupias
University of Oxford, UK

Abstract

The k-server conjecture, first posed by Manasse, McGeoch and Sleator in 1988, states that a k-competitive deterministic algorithm for the k-server problem exists. It is conjectured that the work function algorithm (WFA) achieves this guarantee, a multi-purpose algorithm with applications to various online problems. This has been shown for several special cases: $k = 2$, $(k + 1)$-point metrics, $(k + 2)$-point metrics, the line metric, weighted star metrics, and $k = 3$ in the Manhattan plane.

The known proofs of these results are based on potential functions tied to each particular special case, thus requiring six different potential functions for the six cases. We present a *single* potential function proving k-competitiveness of WFA for all these cases. We also use this potential to show k-competitiveness of WFA on multiray spaces and for $k = 3$ on trees. While the DoubleCoverage algorithm was known to be k-competitive for these latter cases, it has been open for WFA. Our potential captures a type of lazy adversary and thus shows that in all settled cases, the worst-case adversary is lazy. Chrobak and Larmore conjectured in 1992 that a potential capturing the lazy adversary would resolve the k-server conjecture.

To our major surprise, this is not the case, as we show (using connections to the k-taxi problem) that our potential fails for three servers on the circle. Thus, our potential highlights laziness of the adversary as a fundamental property that is shared by all settled cases but violated in general. On the one hand, this weakens our confidence in the validity of the k-server conjecture. On the other hand, if the k-server conjecture holds, then we believe it can be proved by a variant of our potential.

2012 ACM Subject Classification Theory of computation → K-server algorithms

Keywords and phrases Online algorithms, competitive analysis, k-server, work function algorithm

Digital Object Identifier 10.4230/LIPIcs.ICALP.2021.57

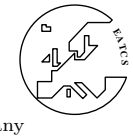
Category Track A: Algorithms, Complexity and Games

Related Version *Full Version*: `https://arxiv.org/abs/2102.10474` [14]

Funding *Christian Coester*: Supported by NWO VICI grant 639.023.812.

1 Introduction

The k-server problem, introduced by Manasse, McGoech and Sleator [23], is one of the most fundamental problems in online optimization and contains other problems like paging or weighted paging as important special cases. It is defined as follows: k servers are located in a metric space. One by one, points of the metric space are requested, and each request must be served upon arrival by moving one of the servers to the requested point. The problem is typically considered *online*, where the choice of this server has to be made without knowledge of future requests. The goal is to minimize the total distance traveled by all servers.



When Manasse, McGeoch and Sleator [23] introduced the k-server problem, they showed that on any metric space with $n \geq k+1$ points[1], every deterministic online algorithm has competitive ratio at least k. They showed that this lower bound is tight when $k = 2$ or $k = n - 1$ by giving a k-competitive algorithm for these cases and boldly conjectured that a k-competitive online algorithm exists for the general case. This conjecture became known as the famous *k-server conjecture* and has been a driving force in online optimization, making the k-server problem perhaps the most studied problem in the field. It has often been referred to as "the holy grail of competitive analysis", and many techniques developed for the k-server problem have later found applications to other problems.

Chrobak, Karloff, Payne, and Vishwanathan [10] designed the elegant *Double Coverage* algorithm to achieve the optimal competitive ratio of k on the line metric. Shortly after, Chrobak and Larmore [11] extended this algorithm to tree metrics, again matching the lower bound of k. The first algorithm for general metrics with a competitive ratio depending only on k was found by Fiat, Rabani and Ravid [16], achieving a competitive ratio exponential in k. Significant progress was made by Koutsoupias and Papadimitriou [21], showing that a competitive ratio of $2k - 1$ is achievable on general metric spaces.

While this reduces the gap between the upper and lower bound to a factor of 2, it remains open to determine the exact competitive ratio. The lack of a proof of the k-server conjecture is even more puzzling given that the algorithm conjectured to achieve the competitive ratio of k has been known for 30 years: The *work function algorithm (WFA)*. It is this algorithm that achieves the aforementioned upper bound of $2k - 1$ [21]. Its definition is generic[2], with applications reaching far beyond the k-server problem. For instance, WFA achieves the optimal competitive ratio for metrical task systems [7, 6], the closely related *generalized WFA* has been applied successfully to the weighted k-server problem [3], the generalized 2-server problem [26] and layered graph traversal [9], and work functions have also played a crucial role in recent breakthroughs for convex body chasing [1, 25]. Given these connections, an exact understanding of the WFA for the k-server problem is likely to have a wider impact on online optimization in general.

WFA is known to achieve the tight competitive ratio of k for the following special cases, which impose restrictions on the number of servers and/or the type of metric space:

- $k = 2$ [12]
- $k = n - 1$ (folklore; see e.g. [19])
- $k = n - 2$ [20, 4]
- line metric [4]
- weighted star metrics [4]
- $k = 3$ in the Manhattan plane [5]

While there has been a lack of progress on the k-server conjecture for about two decades, tremendous progress has been achieved for the *randomized* k-server problem in recent years [2, 8, 22], leading to algorithms with polylogarithmic competitive ratios.

1.1 Our contribution

Our contribution consists of three parts.

(a) The known proofs of the aforementioned six special cases where WFA is k-competitive all use a different potential function, and thus do not seem to point towards a potential function that can solve the k-server conjecture in the general case. We present a single potential function that proves the k-server conjecture for all these cases.

[1] On metric spaces with $n \leq k$ points, the k-server problem is trivial.

[2] WFA always chooses the action that would be best if the future were a mirror image of the past.

(b) Tree metrics are the only special case of the k-server problem where WFA is not known to be k-competitive but another algorithm is (namely, the Double Coverage algorithm [11]). In [4], the question whether WFA is k-competitive on trees was raised as an intermediate step towards solving the k-server conjecture. In this direction, we use our potential function to show that WFA is k-competitive on multiray spaces (a type of tree metrics that generalizes the line and weighted star metrics) and for $k = 3$ on general trees. Our proofs employ the quasi-convexity property of work functions in several new ways.

(c) Chrobak and Larmore [12] formulated three conjectures which say, essentially, that the "adversary is lazy" in the sense that at any time, the worst-case continuation of the request sequence begins with many requests to the k offline server locations (forcing any sensible algorithm to converge to this configuration) before other points are requested. They verified their conjectures on tens of thousands of small metric spaces. In [5], a stronger statement was considered (ignoring the question what kind of work functions are "reachable"), which fails in general but which they conjectured to be true on the circle metric. We reject all these conjectures by showing that for $k = 3$, our potential captures exactly this lazy adversary (and a more restricted adversary for general k), but that it fails on the circle by giving an explicit request sequence as a counterexample. This highlights an important conceptual separation between all cases where k-competitiveness of WFA has been shown and the general case. We believe this property constitutes the main difficulty in resolving the k-server conjecture, and it suggests the circle as the main testing ground for further progress. Our method of constructing the counterexample is based on a connection with the k-taxi problem [13], which we use to generate phenomena of large metric spaces on a much smaller metric space.

1.2 Overview

We provide various definitions and lemmas in Section 2. In Section 3 we formally define our potential in two equivalent ways and show the basic way to use it to prove k-competitiveness. In Section 4, we relate our potential to the lazy adversary potential that was defined implicitly by Chrobak and Larmore. We prove k-competitiveness on multiray spaces in Section 5. This is our most involved proof, and implies the previously known k-competitiveness on the line and weighted stars as special cases. The proof for $k = 3$ on trees and proofs for previously known special cases using our potential can be found in the full version of our paper [14]. In Section 6, we describe ideas of a counter-example to our potential for $k = 3$ on the circle, implying that the adversary is not lazy in this case, unlike the cases where WFA is known to be k-competitive. Details of this construction are given in the full version [14].

2 Preliminaries

Basic notation and abuse of notation. We use (M, d) to denote the metric space, where d is the distance function. We denote by $n = |M|$ its size and by $\Delta = \max_{x,y} d(x, y)$ its diameter. For $x, y \in M$, we will often use the shorthand notation $xy := d(x, y)$. A multiset $C \subseteq M$ of k points is called a *configuration*, representing the location of k servers. We denote by $\mathcal{C}_M^k$ the set of all configurations. For two configurations, X and Y, we denote by $d(X, Y)$ the value of their minimum matching. For notational convenience, we often use the empty space as a union operator on elements of M. For example, we often write $x_1x_2 \dots x_i$ instead of $\{x_1, x_2, \dots, x_i\}$ when it is clear from the context that the set is meant. Similarly, given also a multiset C, we may write $Cx_1 \dots x_i$ instead of $C \cup \{x_1, \dots, x_i\}$. For $x \in M$ and $i \in \mathbb{N}_0$, we write x^i for the multiset containing i copies of x.

For a set $S \subseteq M$, let clique(S) be the sum of pairwise distances of the points in S.

The k-server problem. An instance of the k-server problem is defined by a metric space (M, d), an initial configuration $C_0 \in \mathcal{C}_M^k$ and a sequence $r_1, r_2, \ldots, r_T \in M$ of requests. A feasible solution is a sequence $C_1, C_2, \ldots C_T$ of configurations such that $r_t \in C_t$ for all $t = 1, \ldots, T$. The cost of this solution is the sum $\sum_{t=1}^{T} d(C_{t-1}, C_t)$.

The work function algorithm (WFA). Given an instance of the k-server problem, the *work function* w_t at time t is the function that maps any configuration C to the minimal cost of serving the first t requests and subsequently ending in configuration C. Formally,

$$w_t(C) := \min_{\substack{C_1,\ldots,C_t \\ \forall \tau\colon r_\tau \in C_\tau}} \sum_{\tau=1}^{t} d(C_{\tau-1}, C_\tau) + d(C_t, C).$$

The work function algorithm (WFA) selects $C_t \ni r_t$ so as to minimize $d(C_{t-1}, C_t) + w_t(C_t)$, with ties broken arbitrarily.

Quasiconvexity. A function $w\colon \mathcal{C}_M^k \to \mathbb{R}$ is called *quasiconvex* if for any configurations X and Y there exists a bijection $\mu\colon X \to Y$ such that for any $A \subseteq X$,

$$w(X) + w(Y) \geq w(A \cup \mu(X \setminus A)) + w(\mu(A) \cup (X \setminus A)).$$

It was shown in [21] that if w is quasiconvex, then μ can be chosen such that $\mu(x) = x$ for all $x \in X \cap Y$. More importantly, it was shown in [21] that any work function is quasiconvex.

Fundamentals about work functions. A function $w\colon \mathcal{C}_M^k \to \mathbb{R}$ is 1-*Lipschitz* if

$$w(X) - w(Y) \leq d(X, Y) \tag{1}$$

for all configurations X and Y. By triangle inequality, every work function is 1-Lipschitz.

Let $\mathcal{Q}_M^k$ be the set of functions $w\colon \mathcal{C}_M^k \to \mathbb{R}$ that are quasiconvex. Let $\mathcal{W}_M^k \subseteq \mathcal{Q}_M^k$ be the subset of functions that are additionally 1-Lipschitz. We may drop k and/or M from the notation when they are clear from the context or immaterial. For $w \in \mathcal{W}$ and configurations X and Y, we say that Y *supports* X if (1) holds with equality. Note that if Y supports X in w_t, then the cheapest way of serving the first t requests and ending in configuration X is equal to the cheapest way of serving the first t requests and then first going to Y and then to X. Thus, if Y supports X, then there is no reason for an offline algorithm to be in configuration X because it is at least as good to be in configuration Y and delay the move from Y to X until later.

The *support* of w, denoted supp(w), is the set of all configurations that are *not* supported by any other configuration. Intuitively, supp(w_t) are the possible configurations where an optimal offline algorithm might be at time t. Clearly,

$$w(X) = \min_{Y \in \mathrm{supp}(w)} w(Y) + d(X, Y)$$

for any configuration X. In particular, any work function is fully specified by its support and the values it takes on support configurations.

For $r \in M$, let $\mathcal{W}_M^k(r) \subseteq \mathcal{W}_M^k$ be the subset of 1-Lipschitz, quasiconvex functions with the property that every support configuration contains r. Again, we may drop k and/or M from the notation. Note that the work function w_t at time t is in $\mathcal{W}(r_t)$.

There exists a simple update rule to compute the new work function when a new request is issued. For $w \in \mathcal{W}_M$ and $r \in M$, the updated work function $w \wedge r \in \mathcal{W}(r)$ is defined by

$$w \wedge r(C) = \min_{X \ni r} w(X) + d(X, C).$$

It is easy to see that $w_t = w_{t-1} \wedge r_t$. A basic observation is that if $r_t \in C$, then $w_{t-1}(C) = w_t(C)$. Another basic property is that $w_t(C) \geq w_{t-1}(C)$.

2.1 Extended cost, minimizers and duality

The following lemma was proved by Chrobak and Larmore [12] (see also [19]):

▶ **Lemma 1** (Extended cost lemma). *If for every k-server instance on a metric space M it holds that*

$$\sum_{t=1}^{T} \max_X [w_t(X) - w_{t-1}(X)] \leq (\rho + 1) \cdot \min_X w_T(X) + c_M$$

for some constant c_M independent of the request sequence, then WFA is ρ-competitive on M.

The power of this lemma is that it reduces the task of proving competitiveness of WFA to a property of work functions. In particular, we do not need to keep track of the actual configurations of the online and offline algorithm. The quantity $\max_X [w_t(X) - w_{t-1}(X)]$ is also called the *extended cost* of the tth request, and the proof of the lemma is based on the fact that the total extended cost over all requests is an upper bound on the sum of WFA's cost and the optimal offline cost.

For a work function $w \in \mathcal{W}_M^k$ and a point $y \in M$, we call a configuration $X \in \arg\min w(X) - d(y^k, X)$ a *minimizer of w with respect to y*. There is a direct connection between minimizers and the configurations X maximizing the extended cost. This is captured by the duality lemma, which was first proved in [21]. We give a slightly stronger version of the duality lemma by stating it as an equivalence rather than an implication.

▶ **Lemma 2** (Duality lemma). *Let $w \in \mathcal{W}_M$ and $r \in M$. Define $w' = w \wedge r$. Then $A \in \arg\min_X w(X) - d(r^k, X)$ if and only if the following two conditions hold:*

$$A \in \arg\max_X w'(X) - w(X) \tag{2}$$

$$A \in \arg\min_X w'(X) - d(r^k, X) \tag{3}$$

Proof. The "only if" direction is the duality lemma of [21], where it was shown that if $A \in \arg\min_X w(X) - d(r^k, X)$ then for every configuration B

$$w'(A) + w(B) \geq w(A) + w'(B), \quad \text{and} \tag{4}$$

$$w'(B) - d(r^k, B) \geq w'(A) - d(r^k, A). \tag{5}$$

By summing these two constraints we get $w(B) - d(r^k, B) \geq w(A) - d(r^k, A)$, which shows the other direction. ◀

It is interesting that the proof of the duality lemma does not use the fact that d is a distance, i.e., it satisfies the triangle inequality.

2.2 Additional properties of work functions

In this section, we provide additional properties of work functions that follow from the quasiconvexity property. We will use these properties to prove k-competitiveness on multiray spaces and for $k = 3$ on trees.

The notion of quasiconvex or quasiconcave functions appears in many different areas and was discovered independently a few times. As a result, they appear with different terminology in literature. For example, in the early 1980s Celso and Crawford [17] defined a related notion as a sufficient condition to the existence of Walrasian Equilibria and called a similar notion gross substitute functions[3]; in 1990, Dress and Wenzel [15] related them to a variant of the greedy algorithm and called them valuated matroids; Koutsoupias and Papadimitriou [21] defined them in the context of online algorithms for the k-server problem and called them quasiconvex. They have also played a central role in discrete optimization [24].

▶ **Lemma 3.** *Let $w \in \mathcal{Q}$. Let $X \in \arg\min w(X)$, and let $x \in X$. Then there exists $Y \in \arg\min_{Y \not\ni x} w(Y)$ such that $X - x \subset Y$.*

Proof. Let Y be chosen such that $X \cap Y$ is maximal under inclusion. Suppose towards a contradiction that there exists $x' \in (X - x) \setminus Y$. By quasiconvexity, there exists $y' \in Y \setminus X$ such that $w(X) + w(Y) \geq w(X - x' + y') + w(Y - y' + x')$. By choice of X, we have $w(X - x' + y') \geq w(X)$. Combining these last two inequalities, we get $w(Y) \geq w(Y - y' + x')$. But $Y - y' + x' \not\ni x$ and $X \cap Y \subsetneq X \cap (Y - y' + x')$, so this contradicts the choice of Y. ◀

▶ **Lemma 4.** *Let $w \in \mathcal{Q}^k_M$. Let $X \in \arg\min w(X)$, and let $A \subset M$ be a (multi)set of cardinality $|A| < k$. Then there exists $Y \in \arg\min_{Y \supset A} w(Y)$ such that $Y - A \subseteq X - A$.*

Proof. Let Y be chosen such that $(Y - A) \setminus (X - A)$ is minimal under inclusion and suppose towards a contradiction that there exists $y \in (Y - A) \setminus (X - A)$. By quasiconvexity, there exists $x \in X \setminus Y$ such that $w(X) + w(Y) \geq w(X - x + y) + w(Y - y + x)$. By choice of X, we have $w(X - x + y) \geq w(X)$. Combining these inequalities, we get $w(Y) \geq w(Y - y + x)$. But this contradicts the choice of Y since we would rather have chosen $Y - y + x$. ◀

▶ **Lemma 5.** *Let $w \in \mathcal{W}^k_M(r)$, let $X \subseteq M$ be a k-point multiset and $x, y \in X$. If X resolves[4] from x in w, then also $X - y + x$ resolves from x in w.*

Proof. Suppose that instead, $X - y + x$ resolves from some $z \in X - y - x$. Consider the $(k-3)$-point multiset $C := X - y - x - z$. Then

$$\begin{aligned} w(X) + w(X - y + x) &= w(Cxyz) + w(Cx^2z) \\ &= w(Cyzr) + w(Cx^2r) + rx + rz \\ &\geq w(Cxyr) + w(Cxzr) + rx + rz \\ &\geq w(Cxyz) + w(Cx^2z), \end{aligned}$$

where the first inequality is by quasiconvexity and the last by 1-Lipschitzness of w. Since the second and the last expression are the same, we have equality in all steps. But then the last step shows that Cx^2z resolves from x. Since $Cx^2z = X - y + x$, the lemma follows. ◀

[3] Gross substitute functions are real functions defined for all subsets of a ground set V, whose restriction to subsets of each size k are quasiconvex.

[4] When $w(X) = w(X - x + y) + xy$, we say that X "resolves from x to y". If $y = r$ is the last request, we simply say that X resolves from x.

3 The potential

We provide two different, but equivalent definitions of our potential function. The first formulation views the potential through the lens of the *m-evader problem*, which is equivalent to the k-server problem when $m = n - k$. Thereafter, we will give a more compact and equivalent formulation of the same potential in the k-server view based on extending the metric space by adding antipodal points.

3.1 The evader potential

The m-evader problem is defined similarly to the k-server problem, but instead of k servers there are m evaders in the metric space, which must occupy m different points at all times. When a point r is requested, rather than moving a server towards r, an evader that might be located at r has to move to a different point. The equivalence between the k-server problem and the $(n-k)$-evader problem follows by identifying a server configuration C with the evader configuration $M \setminus C$.[5] Given a k-server work function w, we denote by $\hat{w}$ the corresponding evader work function, defined by $\hat{w}(C) := w(M \setminus C)$.

In the evader view, the potential $\hat{\Phi}$ is defined as follows. Let $y = (y_1, \dots, y_n)$ be a permutation of the points of the metric space M. Let

$$\hat{\Phi}_y(\hat{w}) := \operatorname{clique}(y_1 \dots y_{n-k-1}) + \sum_{i=n-k}^{n} \min_{\substack{C \subseteq \{y_1,\dots,y_i\} \\ |C|=n-k}} \left(\hat{w}(C) + d(C, y_i^{n-k})\right)$$

$$\hat{\Phi}(\hat{w}) := \min_y \hat{\Phi}_y(\hat{w}). \tag{6}$$

▶ **Theorem 6.** *Let (M, d) be an n-point metric space. If for every $r \in M$ and every work function $w \in \mathcal{W}_M^k(r)$ it holds that $\hat{\Phi}(\hat{w}) = \hat{\Phi}_y(\hat{w})$ for a permutation y of M with $y_n = r$, then WFA is k-competitive on M.*

Proof. Consider a k-server instance on M with a request sequence $r_1, \dots, r_T$ and associated sequence of work functions $w_0, \dots, w_T$. We first show that at each time t, the change in potential is an upper bound on the extended cost.

By the premise of the lemma, $\hat{\Phi}(\hat{w}_t) = \hat{\Phi}_y(\hat{w}_t)$ for some y with $y_n = r_t$. Thus,

$$\begin{aligned}
\hat{\Phi}(\hat{w}_t) - \hat{\Phi}(\hat{w}_{t-1}) &\geq \hat{\Phi}_y(\hat{w}_t) - \hat{\Phi}_y(\hat{w}_{t-1}) \\
&\geq \min_{\substack{C \subseteq M \\ |C|=n-k}} \left(\hat{w}_t(C) + d(C, r_t^{n-k})\right) - \min_{\substack{C \subseteq M \\ |C|=n-k}} \left(\hat{w}_{t-1}(C) + d(C, r_t^{n-k})\right) \\
&= \min_{\substack{X \subseteq M \\ |X|=k}} \left(w_t(X) - d(X, r_t^k)\right) - \min_{\substack{X \subseteq M \\ |X|=k}} \left(w_{t-1}(X) - d(X, r_t^k)\right) \\
&= \max_X w_t(X) - w_{t-1}(X),
\end{aligned}$$

where the first inequality uses $\hat{\Phi}(\hat{w}_{t-1}) \leq \hat{\Phi}_y(\hat{w}_{t-1})$, the second inequality uses $y_n = r_t$ and the fact that $\hat{w}_{t-1}(C) \leq \hat{w}_t(C)$ for each C, the first equation translates evader work functions to server work functions and uses $d(C, r_t^{n-k}) = d(M, r_t^n) - d(M \setminus C, r_t^k)$, and the second equation is due to the duality lemma, which says that the same X can be chosen in both minima and the maximum. So indeed, the potential change upper bounds the extended cost.

[5] This identification requires the server configuration to be a set rather than a multiset. This is no restriction on the power of k-server algorithms (online or offline).

Now, we can bound the total extended cost by

$$\sum_{t=1}^{T} \max_X [w_t(X) - w_{t-1}(X)] \le \hat{\Phi}(\hat{w}_T) \le (k+1) \cdot \min_X w_T(X) + c_M,$$

where the last inequality is due to the fact that $\hat{\Phi}(\hat{w}_T)$ is a sum of distances (which are absorbed by the constant c_M) and $k+1$ work function values, each of which differs from $\min_X w_T(X)$ by at most k times the diameter of M due to 1-Lipschitzness of w_T (and the diameters are also absorbed by c_M). The theorem follows from the extended cost lemma. ◀

3.2 The k-server potential

We now derive an equivalent but simpler expression for the aforementioned potential. To formulate it, we need the notion of antipodal points.

Let Δ be the diameter of M. A point $\bar{p} \in M$ is called the *antipode* of another point $p \in M$ if for each $x \in M$, $px + x\bar{p} = p\bar{p} = \Delta$. On some metric spaces such as the circle, each point has an antipode. As mentioned in [18], every metric space can be extended so that each point has an antipode: To achieve this, add to M another copy of the same points, $\bar{M} = \{\bar{p} \colon p \in M\}$, and define distances by $\bar{p}\bar{q} = pq$ and $\bar{p}q = 2\Delta - pq$ for $p, q \in M$. It is easy to check that $M \cup \bar{M}$ is a metric space where $\bar{p}$ and p are antipodes of each other.

Consider a metric space M where every point has an antipode. Let $x_1, \dots, x_k \in M$. We define the k-server potential Φ via

$$\Phi_{x_1,\dots,x_k}(w) := \sum_{i=0}^{k} w(\bar{x}_i^i x_{i+1} \dots x_k)$$
$$\Phi(w) := \min_{x_1,\dots,x_k} \Phi_{x_1,\dots,x_k}(w). \tag{7}$$

The following lemma states that the two potential functions differ by a fixed constant depending on M and are therefore equivalent.

▶ **Lemma 7.** *Let M be a pseudo-metric space of diameter Δ where every point has an antipode and there are k copies of each point.*[6] *For any work function $w \in \mathcal{W}_M^k$ and any permutation $y = (y_1, \dots, y_n)$ of M,*

$$\Phi_{y_{n-k+1}\dots y_n}(w) = \hat{\Phi}_y(\hat{w}) - \text{clique}(M) + \frac{k(k+1)}{2}\Delta.$$

Proof. Subtracting clique(M) from the evader potential and using server work functions instead of evader work functions, we have

$$\hat{\Phi}_y(\hat{w}) - \text{clique}(M) = \sum_{i=n-k}^{n} \min_{\substack{C \supseteq \{y_{i+1},\dots,y_n\} \\ |C|=k}} \left(w(C) - \sum_{p \in C \cap \{y_1,\dots,y_i\}} p y_i \right).$$

[6] It is only a pseudo-metric because the distance between two copies of the same point is 0. We use the assumption of several copies of the same point because the definition of $\Phi_{x_1,\dots,x_k}$ allows points to repeat, whereas Φ_y requires y to be a permutation.

Notice that the minimum in the summand for i is achieved when $C \setminus \{y_{i+1}, \dots, y_n\}$ consists of $k - n + i$ copies of the antipodal point $\bar{y}_i$. Thus, the expression is equal to

$$\sum_{i=n-k}^{n} \left(w(\bar{y_i}^{k-n+i} y_{i+1} \dots y_n) - (k-n+i)\Delta\right) = \sum_{i=n-k}^{n} w(\bar{y_i}^{k-n+i} y_{i+1} \dots y_n) - \frac{k(k+1)}{2}\Delta$$
$$= \Phi_{y_{n-k+1} \dots y_n}(w) - \frac{k(k+1)}{2}\Delta. \quad \blacktriangleleft$$

▶ **Corollary 8.** *Let (M, d) be a metric space where every point has an antipode. If for every $r \in M$ and every work function $w \in \mathcal{W}_M^k(r)$ it holds that $\Phi(w) = \Phi_{x_1 \dots x_k}(w)$ for some $x_1, \dots x_k \in M$ with $x_k = r$, then WFA is k-competitive on M.*

4 Interpretation as a lazy adversary potential

4.1 The implicitly defined potential by Chrobak and Larmore

Chrobak and Larmore [12] gave an *implicit* definition of a potential that they conjectured to prove the k-server conjecture. This potential captures exactly a type of lazy adversary. To give a precise definition, we first need some additional notation.

For $r \in M$ and a work function $w \in \mathcal{W}$, denote by $\nabla(w, r) := \max_A (w \wedge r)(A) - w(A)$ the extended cost of request r on w. For a request sequence $\rho = (r_1, \dots, r_T) \in M^*$, let

$$\nabla(w, \rho) := \sum_{t=1}^{T} \nabla(w_{t-1}, r_t)$$

be the total extended cost, where $w_t = w \wedge r_1 \wedge r_2 \wedge \dots \wedge r_t$ is the updated work function after the first t requests. The potential conjectured by Chrobak and Larmore is given by

$$\tilde{\Phi}(w) := \min_X \tilde{\Phi}_X(w)$$

where the maximum is taken over configurations X and

$$\tilde{\Phi}_X(w) := -\operatorname{clique}(X) + (k+1)w(X) - \sup_{\rho \in X^*} \nabla(w, \rho).$$

Because of the term $\sup_{\rho \in X^*} \nabla(w, \rho)$, this potential captures exactly the worst-case extended cost when the future request sequence consists only of points in X, until the work function is a cone[7] with support $\{X\}$. An adversary constructing such a request sequence can be thought of as "lazy" because it wants to force the online algorithm to the offline configuration X before it requests different points. The additional term $\operatorname{clique}(X)$ is needed because of extended cost being incurred when passing from one cone to a different cone. The definition of $\tilde{\Phi}$ is only implicit because of the supremum over request sequences $\rho \in X^*$. It was conjectured in [12] that $\tilde{\Phi}(w \wedge r) - \tilde{\Phi}(w) \geq \nabla(w, r)$ for any (reachable) work function w and request r. This would imply the k-server conjecture similarly to the proof of Theorem 6. They also conjectured that $\tilde{\Phi}_X(w \wedge r)$ is minimized for a configuration X containing r, and more specifically that it is minimized by a configuration $X \in \operatorname{supp}(w \wedge r)$. This would imply the previous conjectures. We show that for $k = 3$, the potential $\tilde{\Phi}$ matches our potential Φ. For $k \geq 4$, our potential captures a more restricted type of lazy adversary. As we will

[7] A work function is a *cone* if its support contains only a single configuration.

show in Section 6 that our potential fails to bound the extended cost for $k = 3$ on the circle, this disproves the conjectures from [12] and yields the surprising insight that the worst-case adversary on the circle is not lazy – unlike the adversary for all cases where WFA is known to be k-competitive.

4.2 Relationship to our potential

Our next lemma shows that our potential Φ captures a more restricted adversarial strategy, where the configuration X is ordered as $x_1, \dots, x_k$ and the next request in ρ is always to the point x_i with i maximal that leads to a change of the work function. We will show later that for $k = 3$, this imposes no additional restriction.

For fixed $x_1, \dots, x_k \in M$ and a work function $w \in \mathcal{W}_M^k$, define a request sequence $r_1, r_2, \dots, r_T$ as follows. Let $w_t = w \wedge r_1 \wedge r_2 \wedge \dots \wedge r_t$ be the updated work function after the first t requests. We define $r_t = x_i$ for i maximal such that $w_{t-1} \wedge x_i \neq w_{t-1}$; if no such i exists, the request sequence ends, $T = t - 1$, and w_T is a cone with support $\{\{x_1, \dots, x_k\}\}$.

▶ **Lemma 9.**

$$\Phi_{x_1,\dots,x_k}(w) = \frac{k(k+1)}{2}\Delta - \text{clique}(x_1, \dots, x_k) + (k+1)w(x_1 \dots x_k) - \sum_{t=1}^{T} \nabla(w_{t-1}, r_t).$$

Proof. It suffices to show

$$\Phi_{x_1,\dots,x_k}(w_t) = \Phi_{x_1,\dots,x_k}(w_{t-1}) + \nabla(w_{t-1}, r_t) \tag{8}$$

$$\Phi_{x_1,\dots,x_k}(w_T) = (k+1)w(x_1 \dots x_k) + \frac{k(k+1)}{2}\Delta - \sum_{1 \le i < j \le k} x_i x_j. \tag{9}$$

For equation (9), we have

$$\begin{aligned}
\Phi_{x_1,\dots,x_k}(w_T) &= \sum_{j=0}^{k} w_T(\bar{x}_j^j x_{j+1} \dots x_k) \\
&= (k+1)w_T(x_1, \dots, x_k) + \sum_{1 \le i \le j \le k} x_i \bar{x}_j \\
&= (k+1)w(x_1, \dots, x_k) + \sum_{1 \le i \le j \le k} (\Delta - x_i x_j) \\
&= (k+1)w(x_1, \dots, x_k) + \frac{k(k+1)}{2}\Delta - \sum_{1 \le i < j \le k} x_i x_j.
\end{aligned}$$

We now show equation (8). Let i be such that $r_t = x_i$. Then,

$$\begin{aligned}
\Phi_{x_1,\dots,x_k}(w_t) &= \sum_{j=0}^{k} (w_{t-1} \wedge x_i)(\bar{x}_j^j x_{j+1} \dots x_k) \\
&= \sum_{j=0}^{i-1} w_{t-1}(\bar{x}_j^j x_{j+1} \dots x_k) + \sum_{j=i}^{k} (w_{t-1} \wedge x_i)(\bar{x}_j^j x_{j+1} \dots x_k).
\end{aligned} \tag{10}$$

By maximality of i, $x_{j+1} \dots x_k$ is contained in every support configuration of w_{t-1}. Thus, $\bar{x}_i^i x_{i+1} \dots x_k$ is a minimizer of w_{t-1} with respect to x_i and hence

$$(w_{t-1} \wedge x_i)(\bar{x}_i^i x_{i+1} \dots x_k) = w_{t-1}(\bar{x}_i^i x_{i+1} \dots x_k) + \nabla(w_{t-1}, r_t) \tag{11}$$

by the duality lemma.

We claim that

$$(w_{t-1} \wedge x_i)(\bar{x}_j^j x_{j+1} \dots x_k) = w_{t-1}(\bar{x}_j^j x_{j+1} \dots x_k) \qquad \forall j = i+1, \dots, k. \tag{12}$$

Assuming this is true, we obtain (8) by substituting (11) and (12) into (10).

It remains to show (12). Since $w_{t-1} \wedge x_i \geq w_{t-1}$, the direction "$\geq$" is obvious. For the other direction, since $x_j x_{j+1} \dots x_k$ is contained in every support configuration of w_{t-1},

$$\begin{aligned} w_{t-1}(\bar{x}_j^j x_{j+1} \dots x_k) &= w_{t-1}(\bar{x}_j^{j-1} x_j x_{j+1} \dots x_k) + \bar{x}_j x_j \\ &\geq w_{t-1}(\bar{x}_j^{j-1} x_i x_{j+1} \dots x_k) - x_i x_j + \bar{x}_j x_j \\ &= (w_{t-1} \wedge x_i)(\bar{x}_j^{j-1} x_i x_{j+1} \dots x_k) + x_i \bar{x}_j \\ &\geq (w_{t-1} \wedge x_i)(\bar{x}_j^j x_{j+1} \dots x_k). \end{aligned}$$

◀

▶ **Lemma 10.** *Let $X \subset M$ with $|X| = 3$ and $r \in X$ be fixed and let $w \in \mathcal{W}_M^3(r)$. For a bijection $\pi \colon \{1, \dots, 3\} \to X$, write $\Phi_\pi := \Phi_{\pi(1)\pi(2)\pi(3)}$. Then*

$$\min_{\pi \colon \pi(3)=r} \Phi_\pi(w) = \min_{\pi} \Phi_\pi(w).$$

Proof. Let π be a minimizer of the right hand side. If $\pi(k) = r$, we are done. The case $\pi(k-1) = r$ is also easy, using the fact that r is contained in every support configuration. The remaining case $\pi(k-2) = r$ is non-trivial. Let $y := \pi(k-1)$ and $z := \pi(k)$. We will construct a permutation π' with $\pi'(3) = r$ and $\Phi_\pi(w) \geq \Phi_{\pi'}(w)$. This will only affect the last three terms in the sum of the definition of Φ,

$$w(\bar{r}^{k-2} yz) + w(\bar{y}^{k-1} z) + w(\bar{z}^k).$$

If $w(\bar{y}^{k-1} z) = w(\bar{y}^{k-2} rz) + \bar{y}r$, then

$$\begin{aligned} w(\bar{r}^{k-2} yz) + w(\bar{y}^{k-1} z) + w(\bar{z}^k) &= w(\bar{r}^{k-2} yz) + w(\bar{y}^{k-2} rz) + w(\bar{z}^k) + \bar{y}r \\ &\geq w(\bar{y}^{k-2} rz) + w(\bar{r}^{k-1} z) + w(\bar{z}^k) \end{aligned}$$

where the inequality uses $\bar{y}r = y\bar{r}$. This corresponds to a permutation with r in the next-to-last position, and it is easy to push it from there to the last position.

So we can assume $w(\bar{y}^{k-1} z) = w(\bar{y}^{k-1} r) + zr$. Thus

$$\begin{aligned} w(\bar{r}^{k-2} yz) + w(\bar{y}^{k-1} z) + w(\bar{z}^k) &= w(\bar{r}^{k-2} yz) + w(\bar{y}^{k-1} r) + w(\bar{z}^{k-1} r) + zr + \bar{z}r \\ &= w(\bar{r}^{k-2} yz) + w(\bar{y}^{k-1} r) + w(\bar{z}^{k-1} r) + \Delta. \end{aligned} \tag{13}$$

In the last expression, y and z are symmetric, so we can assume

$$w(\bar{r}^{k-2} yz) = w(\bar{r}^{k-2} rz) + yr. \tag{14}$$

By quasi-convexity and Lipschitzness of the work function (and $\bar{y}\bar{r} = yr$, $\bar{r}r = \Delta$),

$$\begin{aligned} w(\bar{y}^{k-1} r) + w(\bar{r}^{k-2} rz) &\geq w(\bar{y}^{k-2} zr) + w(\bar{r}^{k-2} \bar{y}r) \\ &\geq w(\bar{y}^{k-2} zr) + w(\bar{r}^k) - yr - \Delta \end{aligned} \tag{15}$$

Combining (13), (14) and (15), we get

$$w(\bar{r}^{k-2} yz) + w(\bar{y}^{k-1} z) + w(\bar{z}^k) \geq w(\bar{y}^{k-2} zr) + w(\bar{z}^{k-1} r) + w(\bar{r}^k),$$

corresponding to the permutation $(\pi(1), \pi(2), \pi(3)) = (y, z, r)$. ◀

We remark (without proof) that the above lemma fails for $k = 4$.

By the following corollary, for $k = 3$ it holds that Φ is an explicit expression for the implicit potential of [12].

▶ **Corollary 11.** *For $k = 3$,*

$$\Phi(w) = 6\Delta + \min_{\substack{X\colon |X|=3 \\ r_1,\dots,r_T \in X}} \left[4w(X) - \text{clique}(X) - \sum_{t=1}^{T} \nabla(w \wedge r_1 \dots r_{t-1}, r_t) \right]. \tag{16}$$

Proof. The direction "$\geq$" follows from Lemma 9. For direction "$\leq$", select X and $r_1, \dots, r_T$ to minimize the right hand side. Let $w_t = w \wedge r_1 \dots r_t$. By minimality of the right hand side, w_T is a cone at X. Let $\Phi_X = \min \Phi_{x_1x_2x_3}$, with the minimum taken over permutations x_1, x_2, x_3 of X. By Lemma 10, we have $\Phi_X(w_t) = \Phi_{xyr_t}(w_t)$ for some $x, y \in X$. Thus,

$$\begin{aligned} \Phi_X(w_t) - \Phi_X(w_{t-1}) &\geq \Phi_{xyr_t}(w_t) - \Phi_{xyr_t}(w_{t-1}) \\ &= w_t(\bar{r}_t^3) - w_{t-1}(\bar{r}_t^3) \\ &= \nabla(w_{t-1}, r_t). \end{aligned}$$

Hence,

$$\begin{aligned} \Phi(w) \leq \Phi_X(w) = \Phi_X(w_T) - \sum_{t=1}^{T} [\Phi_X(w_t) - \Phi_X(w_{t-1})] \\ \leq \Phi_X(w_T) - \sum_{t=1}^{T} \nabla(w_{t-1}, r_t), \end{aligned}$$

which is equal to the right hand side of (16) by Lemma 9 and since w_T is a cone at X. ◀

5 Multi-ray spaces

A multi-ray space is a tree of depth 1 whose edges have infinite length and where requests can appear at arbitrary locations along the edges. We call these edges rays.

We will show in this section that WFA is k-competitive on multiray spaces. Note that a multiray space with only 2 rays is equal to the line metric. A subset of a multi-ray space containing only one point from each ray is a weighted star. Our proof therefore recovers the known proofs that WFA is k-competitive on the line and on weighted stars as special cases.

We denote by c the center/root of the multi-ray space, i.e., the origin of the rays. We can assume that every ray has finite length by considering only a sufficiently long part that all requests fall into. We call the endpoint of a ray that is not the center a *leaf*. Denote by $\mathcal{L}$ the set of leaves. For $w \in \mathcal{W}^k$, define $m_w(X) := w(X) - d(c^k, X)$. Note that m_w is also quasiconvex. As we use the server definition of the potential, we augment the multi-ray space by adding antipodes as discussed earlier. In the definition (6), we require the points $x_1, \dots, x_k$ to be chosen from the original metric space M. This corresponds to requiring the permutation in the evader potential to end with k points from the original metric space, which does not affect the proof of Theorem 6.

The proof that WFA is k-competitive on multi-ray spaces proceeds along the following three main steps:

1. First we establish properties of $\Phi_{x_1\dots x_k}$ when $x_i = \ell_i$ are leaves. In particular, we express $\Phi_{\ell_1\dots\ell_k}$ in terms of m_w, and show that $\ell_1, \dots, \ell_k$ can be permuted under certain conditions.
2. We then show by induction on k that $\Phi_{x_1\dots x_k}(w)$ is indeed minimized when $x_1, \dots, x_k$ are leaves and $\min_X m_w(X) = m_w(x_1 \dots x_k)$.
3. Finally, we show that $\Phi_{x_1\dots x_k}(w)$ is also minimized for some $x_1, \dots, x_k$ where only $x_1, \dots, x_{k-1}$ are leaves whereas $x_k = r$ is the last request.

Step 1: Properties of $\Phi_{x_1 \dots x_k}$ when x_i are leaves

▶ **Lemma 12.** *Let $w \in \mathcal{W}^k$. There exist leaves $\ell_1, \dots, \ell_k$ such that $\min_X m_w(X) = m_w(\ell_1 \dots \ell_k)$.*

Proof. Follows from the fact that since w is 1-Lipschitz, $m_w(X)$ cannot increase when a point in X moves away from c towards a leaf. ◀

▶ **Lemma 13.** *For any $w \in \mathcal{W}^k$, a leaf $\ell \in \mathcal{L}$ and $x_{i+1}, \dots, x_k \in M$,*

$$w(\bar{\ell}^i x_{i+1} \dots x_k) = \min_{\substack{X \supseteq x_{i+1} \dots x_k : \\ X - x_{i+1} \dots x_k \subseteq \mathcal{L} - \ell}} m_w(X) + i(\Delta - c\ell) + \sum_{j=i+1}^{k} c x_j.$$

Proof.

$$\begin{aligned} w(\bar{\ell}^i x_{i+1} \dots x_k) &= \min_{X \supseteq x_{i+1} \dots x_k} w(X) + d(X - x_{i+1} \dots x_k, \bar{\ell}^i) \\ &= \min_{X \supseteq x_{i+1} \dots x_k} w(X) + i\Delta - d(X - x_{i+1} \dots x_k, \ell^i) \end{aligned}$$

We claim that the minimum is achieved by some X with $X - x_{i+1} \dots x_k \subseteq \mathcal{L} - \ell$: Indeed, if there is some $x \in X - x_{i+1} \dots x_k$ that is not in $\mathcal{L} - \ell$, sliding x away from ℓ along a path to some other leaf increases $d(X - x_{i+1} \dots x_k, \ell^i)$ by the distance moved, and it increases $w(X)$ by at most this distance, so the whole term cannot increase.

This also means that every distance in $d(X - x_{i+1} \dots x_k, \ell^i)$ goes across the center, i.e.,

$$d(X - x_{i+1} \dots x_k, \ell^i) = i \cdot c\ell + d(X - x_{i+1} \dots x_k, c^i).$$

The lemma now follows by definition of m_w. ◀

As a consequence of Lemma 13, we obtain the following expression for $\Phi_{\ell_1 \dots \ell_k}$ whenever $\ell_1, \dots, \ell_k$ are leaves:

$$\Phi_{\ell_1 \dots \ell_k}(w) = \frac{k(k+1)}{2}\Delta + \sum_{i=0}^{k} \min_{\substack{X_i \supseteq \ell_{i+1} \dots \ell_k : \\ X_i - \ell_{i+1} \dots \ell_k \subseteq \mathcal{L} - \ell_i}} m_w(X_i). \tag{17}$$

The following symmetry and monotonicity properties allow us to reorder $\ell_1, \dots, \ell_k$ under certain circumstances.

▶ **Lemma 14** (Symmetry and Monotonicity Lemma)**.** *Let $w \in \mathcal{W}^k$ and let $\ell_1, \dots, \ell_k$ be leaves such that $\min_X m_w(X) = m_w(\ell_1 \dots \ell_k)$. The following properties hold:*
Symmetry: *$\Phi_{\ell_1 \dots \ell_k}(w)$ is constant under permutation of $\ell_1, \dots, \ell_k$.*
Monotonicity: *For any leaf ℓ, $\Phi_{\ell_1 \dots \ell_{k-1} \ell}(w) \geq \Phi_{\ell \ell_1 \dots \ell_{k-1}}(w) \geq \Phi_{\ell_1 \dots \ell_k}(w)$.*

Proof. For the symmetry property and the first inequality of the monotonicity property, we proceed by induction on k. The base case $k = 1$ is trivial. For the induction step, it suffices to show that $\Phi_{\ell_1 \dots \ell_{k-1} \ell}(w) \geq \Phi_{\ell_1 \dots \ell_{k-2} \ell \ell_{k-1}}(w)$, with equality if $\ell = \ell_k$. The lemma then follows by invoking the induction hypothesis on $\tilde{w} = w(\,\cdot\, \ell_{k-1}) \in \mathcal{W}^{k-1}$, observing that $\min_X m_{\tilde{w}}(X) = m_{\tilde{w}}(\ell_1 \dots \ell_{k-2} \ell_k)$, and that $\Phi_{x_1 \dots x_{k-1} \ell_{k-1}}(w) - \Phi_{x_1 \dots x_{k-1}}(\tilde{w}) = w(\bar{\ell}^k_{k-1})$ is a constant function of $x_1, \dots, x_{k-1}$ (and thus $\Phi_{x_1 \dots x_{k-1} \ell_{k-1}}(w)$ and $\Phi_{x_1 \dots x_{k-1}}(\tilde{w})$ are affected in the same way when $x_1, \dots, x_{k-1}$ are permuted).

Note that only the two terms involving X_{k-1} and X_k in (17) are affected when the last two leaves are swapped. For two leaves y and z, let

$$f(y,z) := \min_{\substack{Y \ni z:\\ Y-z \subseteq \mathcal{L}-y}} m_w(Y) + \min_{Z \subseteq \mathcal{L}-z} m_w(Z)$$

We only need to show that $f(\ell_{k-1},\ell) \geq f(\ell,\ell_{k-1})$, and that this holds with equality if $\ell = \ell_k$. Assume $\ell_{k-1} \neq \ell$ as otherwise there is nothing to show. Then

$$\begin{aligned}
&f(\ell_{k-1},\ell) - f(\ell,\ell_{k-1})\\
&\quad= \min_{\substack{Y_1 \ni \ell:\\ Y_1 \subseteq \mathcal{L}-\ell_{k-1}}} m_w(Y_1) + \min_{Z_1 \subseteq \mathcal{L}-\ell} m_w(Z_1) - \min_{\substack{Y_2 \ni \ell_{k-1}:\\ Y_2 \subseteq \mathcal{L}-\ell}} m_w(Y_2) - \min_{Z_2 \subseteq \mathcal{L}-\ell_{k-1}} m_w(Z_2)\\
&\quad\geq \min_{Z_1 \subseteq \mathcal{L}-\ell} m_w(Z_1) - \min_{\substack{Y_2 \ni \ell_{k-1}:\\ Y_2 \subseteq \mathcal{L}-\ell}} m_w(Y_2)\\
&\quad= 0,
\end{aligned}$$

where the last equation follows by applying Lemma 3 to the quasi-convex function m_w. If $\ell = \ell_k$, then the same argument shows that the inequality can be replaced by equality.

It remains to show the second inequality of the monotonicity property. Due to the symmetry property, and by a renaming of leaves, it suffices to show that $\Phi_{\ell\ell_2\dots\ell_k}(w) \geq \Phi_{\ell_1\ell_2\dots\ell_k}(w)$. Assume $\ell \neq \ell_1$, otherwise we are done. In (17), the only terms affected when the first leaf is replaced are the ones involving X_0 and X_1. Let X_0 and X_1 be these sets in $\Phi_{\ell_1\ell_2\dots\ell_k}(w)$ and X_0' and X_1' those in $\Phi_{\ell\ell_2\dots\ell_k}(w)$. Then $X_0 = \ell_1\dots\ell_k$, $X_0' = \ell\ell_2\dots\ell_k$, and since $\min_X m_w(X) = m_w(\ell_1\dots\ell_k)$, we can choose $X_1' = \ell_1\dots\ell_k$. Moreover, X_0' satisfies the requirements of X_1 (apart from minimality, possibly), hence $m_w(X_1) \leq m_w(X_0')$. Thus,

$$\Phi_{\ell\ell_2\dots\ell_k}(w) - \Phi_{\ell_1\ell_2\dots\ell_k}(w) = m_w(X_0') + m_w(X_1') - m_w(X_0) - m_w(X_1) \geq 0. \qquad \blacktriangleleft$$

Step 2: $x_1, \dots, x_k$ are indeed leaves

▶ **Lemma 15.** *Let $w \in \mathcal{W}^k$. Let $\ell_1,\dots,\ell_k$ be leaves such that $\min_X m_w(X) = m_w(\ell_1\dots\ell_k)$. Then $\Phi(w) = \Phi_{\ell_1\dots\ell_k}(w)$.*

Proof. By induction on k. The base case $k = 0$ is trivial. For the induction step, fix x such that $\Phi(w) = \Phi_{x_1\dots x_{k-1}x}(w)$ for some $x_1,\dots,x_{k-1}$. Consider the function $\tilde{w} = w(\cdot x) \in \mathcal{W}^{k-1}$. By Lemma 4, $\min m_{\tilde{w}}(X) = m_{\tilde{w}}(\ell_1\dots\ell_k - \ell')$ for some $\ell' \in \ell_1\dots\ell_k$. By Lemma 14, we can assume without loss of generality that $\ell' = \ell_k$, i.e., $\min m_{\tilde{w}}(X) = m_{\tilde{w}}(\ell_1\dots\ell_{k-1})$. By the induction hypothesis, $\Phi(\tilde{w}) = \Phi_{\ell_1\dots\ell_{k-1}}(\tilde{w})$. Hence,

$$\begin{aligned}
\Phi(w) &= \min_{x_1\dots x_{k-1}} \Phi_{x_1\dots x_{k-1}x}(w)\\
&= \min_{x_1\dots x_{k-1}} \Phi_{x_1\dots x_{k-1}}(\tilde{w}) + w(\bar{x}^k)\\
&= \Phi_{\ell_1\dots\ell_{k-1}}(\tilde{w}) + w(\bar{x}^k)\\
&= \Phi_{\ell_1\dots\ell_{k-1}x}(w).
\end{aligned}$$

We will now transform the last expression in several steps with the goal of eventually replacing x by ℓ_k.

Denote by ℓ the leaf below x. The goal of the following transformations is to replace x by ℓ. For some $a \in \{0,1,\dots,k\}$ we have

$$w(\bar{x}^k) = w(\bar{\ell}^a \ell^{k-a}) + a \cdot x\ell + (k-a) \cdot x\bar{\ell}.$$

The symmetry property of Lemma 14 allows us to assume that $\ell_1, \dots, \ell_{s-1}$ are all different from ℓ and $\ell_s = \ell_{s+1} = \cdots = \ell_{k-1} = \ell$ for some $s \in \{1, \dots, k\}$.

As an intermediate step, we show by (backwards) induction on $j = k, k-1, \dots, \max\{s, a\}$ that

$$\Phi(w) \geq \sum_{i=0}^{j-1} w(\bar{\ell}_i^i \ell_{i+1} \dots \ell_{k-1} x) + \sum_{i=j}^{k-1} w(\bar{\ell}_i^{i+1} \ell_{i+1} \dots \ell_{k-1}) + w(\bar{\ell}^a \ell^{j-a} \ell_j \dots \ell_{k-1}) + a \cdot x\ell + (j-a) \cdot x\bar{\ell} \tag{18}$$

The base case $k = j$ follows from the previous equation. Suppose now that (18) holds for some $j > \max\{s, a\}$. From $\ell_{j-1} = \ell$ we get

$$w(\bar{\ell}_{j-1}^{j-1} \ell_j \dots \ell_{k-1} x) \geq w(\bar{\ell}_{j-1}^{j} \ell_j \dots \ell_{k-1}) - x\bar{\ell}$$

and

$$w(\bar{\ell}^a \ell^{j-a} \ell_j \dots \ell_{k-1}) = w(\bar{\ell}^a \ell^{j-1-a} \ell_{j-1} \dots \ell_{k-1}).$$

The induction step of (18) follows by plugging these in to (18).

If $a \geq s$, then (18) for $j = a$ yields

$$\begin{aligned}
\Phi(w) &\geq \sum_{i=0}^{a-1} w(\bar{\ell}_i^i \ell_{i+1} \dots \ell_{k-1} x) + \sum_{i=a}^{k-1} w(\bar{\ell}_i^{i+1} \ell_{i+1} \dots \ell_{k-1}) + w(\bar{\ell}^a \ell_a \dots \ell_{k-1}) + a \cdot x\ell \\
&\geq \sum_{i=0}^{a-1} w(\bar{\ell}_i^i \ell_{i+1} \dots \ell_{k-1} \ell) + \sum_{i=a}^{k-1} w(\bar{\ell}_i^{i+1} \ell_{i+1} \dots \ell_{k-1}) + w(\bar{\ell}^a \ell_a \dots \ell_{k-1}) \\
&= \Phi_{\ell_1 \dots \ell_{k-1} \ell}(w),
\end{aligned}$$

where we have used that $\ell_i = \ell$ for $i \geq a \geq s$. The lemma then follows from the monotonicity property of Lemma 14.

Otherwise, $a < s$, and from (18) for $j = s$ we get

$$\begin{aligned}
\Phi(w) &\geq \sum_{i=0}^{s-1} w(\bar{\ell}_i^i \ell_{i+1} \dots \ell_{k-1} x) + \sum_{i=s}^{k-1} w(\bar{\ell}_i^{i+1} \ell_{i+1} \dots \ell_{k-1}) \\
&\qquad + w(\bar{\ell}^a \ell^{s-a} \ell_s \dots \ell_{k-1}) + a \cdot x\ell + (s-a) \cdot x\bar{\ell} \\
&\geq \sum_{i=0}^{s-a-1} w(\bar{\ell}_i^i \ell_{i+1} \dots \ell_{k-1} x) + \sum_{i=s-a}^{s-1} w(\bar{\ell}_i^i \ell_{i+1} \dots \ell_{k-1} \ell) + \sum_{i=s+1}^{k} w(\bar{\ell}^i \ell^{k-i}) \\
&\qquad + w(\bar{\ell}^a \ell^{s-a} \ell_s \dots \ell_{k-1}) + (s-a) \cdot x\bar{\ell}.
\end{aligned}$$

By Claim 16 below, replacing a by $a+1$ does not increase the latter quantity. Inductively we may therefore replace a by s to obtain

$$\begin{aligned}
\Phi(w) &\geq \sum_{i=0}^{s-1} w(\bar{\ell}_i^i \ell_{i+1} \dots \ell_{k-1} \ell) + \sum_{i=s+1}^{k} w(\bar{\ell}^i \ell^{k-i}) + w(\bar{\ell}^s \ell^{k-s}) \\
&= \Phi_{\ell_1 \dots \ell_{k-1} \ell}(w).
\end{aligned}$$

The monotonicity property of Lemma 14 completes the proof. ◀

▷ Claim 16. Let $0 \leq a < s \leq k$ and $w \in \mathcal{W}^k$. Let $\ell_{s-a-1}, \dots, \ell_{k-1}$ and ℓ be leaves such that $\ell_i \neq \ell$ for $i < s$, and let x be a point on the ray of ℓ. Then

$$\begin{aligned}
&w(\bar{\ell}_{s-a-1}^{s-a-1} \ell_{s-a} \dots \ell_{k-1} x) + w(\bar{\ell}^a \ell^{s-a} \ell_s \dots \ell_{k-1}) + x\bar{\ell} \\
&\qquad \geq w(\bar{\ell}_{s-a-1}^{s-a-1} \ell_{s-a} \dots \ell_{k-1} \ell) + w(\bar{\ell}^{a+1} \ell^{s-a-1} \ell_s \dots \ell_{k-1}).
\end{aligned}$$

Proof. Consider the bijection from the definition of quasiconvexity between the two configurations on the left hand side.[8] By the pigeonhole principle, at least one of the $s-a$ copies of ℓ in the second configuration maps to some point $p \in \ell_{s-a} \dots \ell_{s-1} x$ in the first configuration. Quasiconvexity gives

$$w(\bar{\ell}^{s-a-1}_{s-a-1}\ell_{s-a}\dots\ell_{k-1}x) + w(\bar{\ell}^{a}\ell^{s-a}\ell_s\dots\ell_{k-1}) \\ \geq w(\bar{\ell}^{s-a-1}_{s-a-1}\ell_{s-a}\dots\ell_{k-1}\ell x - p) + w(\bar{\ell}^{a}\ell^{s-a-1}\ell_s\dots\ell_{k-1}p).$$

By 1-Lipschitzness of w, we get

$$w(\bar{\ell}^{s-a-1}_{s-a-1}\ell_{s-a}\dots\ell_{k-1}\ell x - p) \geq w(\bar{\ell}^{s-a-1}_{s-a-1}\ell_{s-a}\dots\ell_{k-1}\ell) - px$$

and

$$w(\bar{\ell}^{a}\ell^{s-a-1}\ell_s\dots\ell_{k-1}p) \geq w(\bar{\ell}^{a+1}\ell^{s-a-1}\ell_s\dots\ell_{k-1}) - p\bar{\ell}.$$

Since $\ell_i \neq \ell$ for $i < s$ and $p \in \ell_{s-a}\dots\ell_{s-1}x$, the point x lies on the path from p to ℓ, i.e., $px + x\ell = p\ell$. Equivalently, $x\bar{\ell} = px + p\bar{\ell}$. The claim follows by combining these inequalities. ◁

Step 3: Alternatively, $x_k = r$

▶ **Lemma 17.** *For any $w \in \mathcal{W}^k(r)$, there exist leaves $\ell_1, \dots, \ell_{k-1}$ such that $\Phi(w) = \Phi_{\ell_1\dots\ell_{k-1}r}(w)$.*

Proof. Since $w \in \mathcal{W}^k(r)$, we can choose $X \in \arg\min_X m_w(X)$ of the form $X = r\ell_2\dots\ell_k$ for $\ell_2, \dots, \ell_k \in \mathcal{L}$. If $\ell := \ell_1$ is the leaf of the ray containing r, then clearly $\ell_1\dots\ell_k$ is also a minimizer of m_w and $\ell_1\dots\ell_k$ resolves from ℓ_1. Let $\ell_2, \dots, \ell_k$ be ordered such that $\ell = \ell_1 = \dots = \ell_s$ for some $s \geq 1$ and $\ell_i \neq \ell$ for $i > s$.

The main part of this proof is to show that there exists $a \in \{1, \dots, s\}$ such that

$$w(\bar{\ell}^i_i\ell_{i+1}\dots\ell_k) = \begin{cases} w(\bar{\ell}^i_i\ell_{i+2}\dots\ell_k r) + r\ell & \text{if } i < a \\ w(\bar{\ell}^{i-1}_i\ell_{i+1}\dots\ell_k r) + r\bar{\ell}_i & \text{if } i \geq a. \end{cases} \tag{19}$$

Before we prove this, let us see why it implies the lemma. By Lemma 15 and the fact that $\ell = \ell_1 = \dots = \ell_a$, we have

$$\begin{aligned} \Phi(w) &= \Phi_{\ell_1\dots\ell_k}(w) \\ &= \sum_{i=0}^{k} w(\bar{\ell}^i_i\ell_{i+1}\dots\ell_k) \\ &= \sum_{i=0}^{a-1} w(\bar{\ell}^i_i\ell_{i+2}\dots\ell_k r) + \sum_{i=a}^{k} w(\bar{\ell}^{i-1}_i\ell_{i+1}\dots\ell_k r) + a\cdot r\ell + \sum_{i=a}^{k} r\bar{\ell}_i \\ &= \sum_{i=0}^{k-1} w(\bar{\ell}^i_{i+1}\ell_{i+2}\dots\ell_k r) + w(\bar{\ell}^{a-1}\ell_{a+1}\dots\ell_k r) + (a-1)\cdot r\ell + \sum_{i=a+1}^{k} \bar{r}\ell_i + \Delta \\ &\geq \sum_{i=0}^{k-1} w(\bar{\ell}^i_{i+1}\ell_{i+2}\dots\ell_k r) + w(\bar{r}^k) \\ &= \Phi_{\ell_2\dots\ell_k r}(w). \end{aligned}$$

[8] We remark that earlier proofs about competitiveness of the work function algorithm only used a weaker form of quasi-convexity and did not actually use the existence of such a bijection.

It remains to show (19). We choose a maximal such that $\bar{\ell}_{a-1}^{a-1}\ell_a \dots \ell_k$ resolves from ℓ. Recall from the start of this proof that $\ell_1 \dots \ell_k$ resolves from ℓ_1, so $a \geq 1$. Moreover, $a \leq s$ since $\ell_i \neq \ell$ for $i > s$. So $a \in \{1, \dots, s\}$ as required. The case "$i < a$" of (19) now follows by backwards induction on i, where the induction step is due to Lemma 5.

Consider now some $i > s$. Letting $\tilde{w} = w(\cdot\, \ell_{i+1} \dots \ell_k) \in \mathcal{W}^i_{\mathcal{L}+r}$, we have

$$w(\bar{\ell}_i^i \ell_{i+1} \dots \ell_k) = \tilde{w}(\bar{\ell}_i^i) = \min_{X \subseteq \mathcal{L}+r-\ell_i} m_{\tilde{w}}(X) + i(\Delta - c\ell_i),$$

where the last equation is proved similarly to Lemma 13, but we may allow X to contain the non-leaf r since it is on a different ray than ℓ_i (thanks to $i > s$). Since $\min_X m_w(X) = m_w(r\ell_2 \dots \ell_k)$, we have $\min_X m_{\tilde{w}}(X) = m_{\tilde{w}}(r\ell_2 \dots \ell_i)$, so by Lemma 3 the minimum under the restriction $X \subseteq \mathcal{L} + r - \ell_i$ is achieved for some X with $r \in X$. Thus,

$$\begin{aligned}
w(\bar{\ell}_i^i \ell_{i+1} \dots \ell_k) &= \min_{Y \subseteq \mathcal{L}-\ell_i} \tilde{w}(Yr) - d(Y, c^{i-1}) - r\ell_i + i(\Delta - c\ell_i) \\
&= \min_{Y \subseteq \mathcal{L}-\ell_i} \tilde{w}(Yr) + d(Y, \bar{\ell}_i^{i-1}) + \bar{\ell}_i r \\
&\geq \tilde{w}(\bar{\ell}_i^{i-1} r) + \bar{\ell}_i r \\
&= w(\bar{\ell}_i^{i-1} r \ell_{i+1} \dots \ell_k) + r\bar{\ell}_i.
\end{aligned}$$

Note that the inequality between the first and last expression cannot be strict due to 1-Lipschitzness of w, and their equality reveals that $\bar{\ell}_i^i \ell_{i+1} \dots \ell_k$ resolves from $\bar{\ell}_i$, as desired.

Finally, consider $i \in \{a, a+1, \dots, s\}$. Then $\ell_i = \ell$, and we need to show that $\bar{\ell}^i \ell_{i+1} \dots \ell_k$ resolves from $\bar{\ell}$. Suppose that it instead resolves from ℓ_h for some $h > i$. Since $a \leq s$ was chosen maximal, we know that $\ell_h \neq \ell$. By Lemma 13,

$$\begin{aligned}
w(\bar{\ell}^i \ell_{i+1} \dots \ell_k) &= w(\bar{\ell}^i \ell_{i+1} \dots \ell_{h-1} \ell_{h+1} \dots \ell_k r) + r\ell_h \\
&= \min_{\substack{X \supseteq \ell_{i+1} \dots \ell_{h-1} \ell_{h+1} \dots \ell_k r \\ X - \ell_{i+1} \dots \ell_{h-1} \ell_{h+1} \dots \ell_k r \subseteq \mathcal{L} - \ell}} m_w(X) + i(\Delta - c\ell) + \sum_{\substack{j=i+1 \\ j \neq h}}^{k} c\ell_j + cr + r\ell_h. \\
&= \min_{Y \not\ni \ell} m_{\tilde{w}}(Y) + i(\Delta - c\ell) + r\ell_h,
\end{aligned}$$

where $\tilde{w} = w(\cdot\, \ell_{i+1} \dots \ell_{h-1} \ell_{h+1} \dots \ell_k r) \in \mathcal{W}^i_{\mathcal{L}}$. Since $\min_X m_w(X) = m_w(r\ell_2 \dots \ell_k)$, we have that $\min_X m_{\tilde{w}}(X) = m_{\tilde{w}}(\ell_2 \dots \ell_i \ell_h)$, so by Lemma 3 the minimum under the restriction $Y \not\ni \ell$ is achieved for some Y with $\ell_h \in Y$. Letting $Y' = Y - \ell_h$, we get

$$\begin{aligned}
w(\bar{\ell}^i \ell_{i+1} \dots \ell_k) &= m_{\tilde{w}}(Y) + i(\Delta - c\ell) + r\ell_h \\
&= w(Y' \ell_{i+1} \dots \ell_k r) - d(Y, c^i) + i(\Delta - c\ell) + (rc + c\ell_h) \\
&= w(Y' \ell_{i+1} \dots \ell_k r) - \sum_{y \in Y'} y\ell + i\Delta - c\ell + rc \\
&\geq w(Y' \ell_{i+1} \dots \ell_k r) + \sum_{y \in Y'} y\bar{\ell} + \Delta - r\ell \\
&\geq w(\bar{\ell}^{i-1} \ell_{i+1} \dots \ell_k r) + r\bar{\ell},
\end{aligned}$$

where the second equation uses that $\ell \neq \ell_h$ and therefore c lies on the path from r to ℓ_h, and the third equation uses that y and ℓ are different leaves and therefore $yc + c\ell = y\ell$. Again, the inequality between the first and last expression cannot be strict due to 1-Lipschitzness of w, and their equality reveals that $\bar{\ell}^i \ell_{i+1} \dots \ell_k$ resolves from $\bar{\ell}$, completing the proof. ◀

▶ **Theorem 18.** *WFA is k-competitive on multiray spaces.*

Proof. Follows from Lemma 17 and Corollary 8. ◀

6 Non-laziness of the worst-case adversary on the circle

▶ **Theorem 19.** *For $k = 3$ servers on the circle, there exists a reachable work function from where the worst-case adversarial continuation of the request sequence is not lazy. More precisely, there exists a request sequence such that the induced work functions w_t and w_{t+1} after time steps t and $t+1$ and the WFA configuration C_t after time step t satisfy $w_{t+1}(C_t) - w_t(C_t) > \Phi(w_{t+1}) - \Phi(w_t)$.*

In other words, the extended cost is strictly greater than the change in potential. Due to the interpretation of our potential (Section 4), this means that the worst-case continuation of the request sequence after time t is not lazy. If Theorem 19 could be strengthened such that the request sequence to reach w_t has extended cost equal to its induced change in potential, then this would disprove the premise of the extended cost lemma (because one could create a cyclic request sequence where extended cost is always at least the change in potential and exceeds it infinitely often; we remark that one can go from a cone work function to any other cone via a request sequence whose extended cost equals its potential change).

Note that Theorem 19 holds even if in the extended cost $\max_X w_{t+1}(X) - w_t(X)$ we replace X by the configuration C_t of WFA at time t. The significance of this is that the sum of the terms $w_{t+1}(C_t) - w_t(C_t)$ over all time steps is *equal* to the sum of WFA's cost and the optimal offline cost (up to a bounded additive error). Thus, proving violation of the premise of the extended cost lemma with X replaced by C_t would imply that WFA's competitive ratio is strictly greater than k.

The proof of Theorem 19 is given in the full version of our paper [14]. It is based on a tight connection between the k-server problem and the "easy" version of the k-taxi problem observed in [13]. The k-taxi problem is the generalization of the k-server problem where each request is not a single point, but a pair (s, t) of two points, representing the start s and destination t of a taxi request. To serve it, the algorithm has to select a server that first goes to s and then to t. In the "easy" version relevant for us, the cost is defined as the total distance traveled by servers.[9] As shown in [13], the easy k-taxi problem has exactly the same competitive ratio as the k-server problem. The idea of this reduction is that a k-taxi request (s, t) can be simulated by a sequence of many k-server requests along the shortest path from s to t. We extend this idea to show that we can use k-taxi requests to reach work functions that are arbitrarily close to work functions reachable via k-server requests. We then give an explicit counter-example using k-taxi requests.

We remark that up to symmetry and shift by an additive constant, for $k = 3$ there exist over 280,000 different work functions reachable by taxi requests with starts/destinations at the points on the circle considered in our construction (8 equally spaced points for the destinations and 16 equally spaced points for the starts – the aforementioned 8 points as well as the 8 intermediate points). Among these over 280,000 work functions, the pair of w_t and w_{t+1} from our construction is the *only* counterexample to laziness of the adversary. Using only k-server requests and no k-taxi requests, we were unable to find any counterexamples for n equally spaced points on the circle for the values of n that were computationally feasible for us to try. Of course, though, our approximability argument of k-taxi requests via k-server requests implies that such counterexamples do exist for n sufficiently large. Given the rarity of these counterexamples, it is not surprising that Chrobak and Larmore [12] who reported testing their conjecture on tens of thousands of small metric spaces in the early 90s did not find any counterexample.

[9] In contrast, the "hard" k-taxi problem defines the cost as only the overhead distance traveled while *not* carrying a passenger, i.e., the distance from s to t is excluded from the cost.

7 Conclusion

Our potential gives a unified perspective on all cases where WFA is known to be k-competitive. Unlike previous potentials, which were specific to their special case and had no clear intuition, our potential has a natural interpretation as capturing a lazy adversary. We remark that our potential also proves k-competitiveness on 6-point metric spaces. Since work functions, the WFA, and the generalized WFA are central to various online problems, similar potential functions may also prove useful to analyze different problems.

Since it was a major belief that a lazy adveresary would capture the worst case, our insights yield a qualitative explanation of the shortcomings of previous approaches and may point in a direction to overcome these shortcomings.

Our proof for $k = 3$ on trees relies on the fact that if (M, d) is a tree metric, then d is quasiconcave (i.e., $-d$ is quasiconvex). We are puzzled by the question whether this has any deeper connection to the quasiconvexity property of work functions and whether it is crucial for the existence of k-competitive algorithms. While the k-server problem is also k-competitive on some non-quasiconcave metrics (such as the cases $k = 2$ and $n = k - 2$), the reason for this might simply be due to the fact that the subspaces relevant in all proof steps are small (note that any 3-point metric is quasiconcave).

References

1. C. J. Argue, Anupam Gupta, Guru Guruganesh, and Ziye Tang. Chasing convex bodies with linear competitive ratio. In *Proceedings of the 2020 ACM-SIAM Symposium on Discrete Algorithms, SODA '20*, pages 1519–1524, 2020.
2. Nikhil Bansal, Niv Buchbinder, Aleksander Madry, and Joseph Naor. A polylogarithmic-competitive algorithm for the k-server problem. *J. ACM*, 62(5):40:1–40:49, 2015.
3. Nikhil Bansal, Marek Eliás, and Grigorios Koumoutsos. Weighted k-server bounds via combinatorial dichotomies. In *58th IEEE Annual Symposium on Foundations of Computer Science, FOCS '17*, 2017.
4. Yair Bartal and Elias Koutsoupias. On the competitive ratio of the work function algorithm for the k-server problem. *Theoretical Computer Science*, 324(2-3):337–345, 2004.
5. Wolfgang W. Bein, Marek Chrobak, and Lawrence L. Larmore. The 3-server problem in the plane. *Theor. Comput. Sci.*, 289(1):335–354, 2002.
6. Allan Borodin and Ran El-Yaniv. *Online computation and competitive analysis*. Cambridge University Press, 1998.
7. Allan Borodin, Nathan Linial, and Michael E. Saks. An optimal on-line algorithm for metrical task system. *J. ACM*, 39(4):745–763, 1992.
8. Sébastien Bubeck, Michael B. Cohen, Yin Tat Lee, James R. Lee, and Aleksander Madry. k-server via multiscale entropic regularization. In *Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2018*, pages 3–16, 2018.
9. William R. Burley. Traversing layered graphs using the work function algorithm. *J. Algorithms*, 20(3):479–511, 1996.
10. Marek Chrobak, Howard Karloff, Tom Payne, and Sundar Vishwanathan. New results on server problems. *SIAM Journal on Discrete Mathematics*, 4(2):172–181, 1991.
11. Marek Chrobak and Lawrence L. Larmore. An optimal on-line algorithm for k servers on trees. *SIAM Journal on Computing*, 20(1):144–148, 1991.
12. Marek Chrobak and Lawrence L Larmore. The server problem and on-line games. In *On-line Algorithms, volume 7 of DIMACS Series in Discrete Mathematics and Theoretical Computer Science*. Citeseer, 1992.
13. Christian Coester and Elias Koutsoupias. The online k-taxi problem. In *Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing, STOC '19*, 2019.

14 Christian Coester and Elias Koutsoupias. Towards the k-server conjecture: A unifying potential, pushing the frontier to the circle. *CoRR*, abs/2102.10474, 2021. `arXiv:2102.10474`.

15 Andreas W.M. Dress and Walter Wenzel. Valuated matroids: a new look at the greedy algorithm. *Applied Mathematics Letters*, 3(2):33–35, 1990.

16 Amos Fiat, Yuval Rabani, and Yiftach Ravid. Competitive k-server algorithms. *J. Comput. Syst. Sci.*, 48(3), 1994.

17 Alexander S. Kelso Jr and Vincent P. Crawford. Job matching, coalition formation, and gross substitutes. *Econometrica: Journal of the Econometric Society*, pages 1483–1504, 1982.

18 Elias Koutsoupias. Weak adversaries for the k-server problem. In *40th Annual Symposium on Foundations of Computer Science, FOCS '99*, pages 444–449, 1999.

19 Elias Koutsoupias. The k-server problem. *Computer Science Review*, 3(2):105–118, 2009.

20 Elias Koutsoupias and Christos Papadimitriou. The 2-evader problem. *Information Processing Letters*, 57(5):249–252, 1996.

21 Elias Koutsoupias and Christos H. Papadimitriou. On the k-server conjecture. *J. ACM*, 42(5), 1995.

22 James R. Lee. Fusible HSTs and the randomized k-server conjecture. In *Proceedings of the 59th Annual IEEE Symposium on Foundations of Computer Science*, FOCS '18, pages 438–449, 2018.

23 Mark S. Manasse, Lyle A. McGeoch, and Daniel Dominic Sleator. Competitive algorithms for on-line problems. In *Proceedings of the 20th Annual ACM Symposium on Theory of Computing, STOC '88*, 1988.

24 Kazuo Murota. *Discrete Convex Analysis*. Society for Industrial and Applied Mathematics, January 2003.

25 Mark Sellke. Chasing convex bodies optimally. In *Proceedings of the 2020 ACM-SIAM Symposium on Discrete Algorithms, SODA '20*, pages 1509–1518, 2020.

26 René Sitters. The generalized work function algorithm is competitive for the generalized 2-server problem. *SIAM J. Comput.*, 43(1), 2014.

Haystack Hunting Hints and Locker Room Communication

Artur Czumaj
Department of Computer Science and DIMAP, University of Warwick, Coventry, UK

George Kontogeorgiou
Mathematics Institute, University of Warwick, Coventry, UK

Mike Paterson
Department of Computer Science and DIMAP, University of Warwick, Coventry, UK

Abstract

We want to efficiently find a specific object in a large unstructured set, which we model by a *random n-permutation*, and we have to do it by revealing just a single element. Clearly, without any help this task is hopeless and the best one can do is to select the element at random, and achieve the success probability $\frac{1}{n}$. Can we do better with some small amount of advice about the permutation, even without knowing the object sought? We show that by providing advice of just one integer in $\{0, 1, \ldots, n-1\}$, one can improve the success probability considerably, by a $\Theta(\frac{\log n}{\log \log n})$ factor.

We study this and related problems, and show asymptotically matching upper and lower bounds for their optimal probability of success. Our analysis relies on a close relationship of such problems to some intrinsic properties of random permutations related to the rencontres number.

2012 ACM Subject Classification Mathematics of computing → Permutations and combinations; Theory of computation → Communication complexity

Keywords and phrases Random permutations, Search, Communication complexity

Digital Object Identifier 10.4230/LIPIcs.ICALP.2021.58

Category Track A: Algorithms, Complexity and Games

Related Version *Full Version*: `https://arxiv.org/abs/2008.11448`

Funding *Artur Czumaj*: Research partially supported by the Centre for Discrete Mathematics and its Applications (DIMAP), by an IBM Faculty Award, and by EPSRC award EP/V01305X/1.
George Kontogeorgiou: Research partially supported by an EPSRC Doctoral Training Partnership.

1 Introduction

Understanding basic properties of random permutations is an important concern in modern data science. For example, a preliminary step in the analysis of a very large data set presented in an unstructured way is often to model it assuming the data is presented in a random order. Understanding properties of random permutations would guide the processing of this data and its analysis. In this paper, we consider a very natural problem in this setting. You are given a set of n objects ($[n-1]$, say[1]) stored in locations $x_0, \ldots, x_{n-1}$ according to a random permutation σ of $[n-1]$. This is the *haystack*, and you want to find one specific object, not surprisingly called the *needle*, by drawing from just one location.

Clearly, the probability of finding the right object in a single draw is always $\frac{1}{n}$ (whichever location you choose, since the permutation σ is random, the probability that your object is there is exactly $\frac{1}{n}$). But can I give you any advice or *hint* about σ – *without knowing which object you are seeking* – to improve the chance of you finding the specific object? If I could

[1] Throughout the paper we use the standard notation $[n-1] := \{0, \ldots, n-1\}$, and we write log for $\log_2$.

tell you the entire σ (which can be encoded with $\log(n!) = \Theta(n \log n)$ bits) then this task is trivial and you would know the location of the object sought. But what if I give you just a small hint (on the basis of σ), one number $\mathfrak{h}$ from $[n-1]$ (or equivalently, one $\log n$-bit sequence) – even when I know nothing about the object sought?

Formally, the goal is to design a strategy to choose a *hint* $\mathfrak{h} = \mathfrak{h}(\sigma)$ and an *index* $\mathfrak{i} = \mathfrak{i}(\mathfrak{h}, \mathfrak{s})$, with both $\mathfrak{h}, \mathfrak{i} \in [n-1]$, such that for a given $\mathfrak{s} \in [n-1]$, $\mathbf{Pr}[\sigma(\mathfrak{i}) = \mathfrak{s}]$ is maximized, where the probability is over the random choice of σ and over the randomness in the choice of the strategy (since $\mathfrak{h} = \mathfrak{h}(\sigma)$ and $\mathfrak{i} = \mathfrak{i}(\mathfrak{h}, \mathfrak{s})$ may be randomized functions), see also Section 2.2.

1.1 Related puzzle: *communication in the locker room*

The *needle in a haystack* problem is closely related to the following *locker room* problem (see Figure 1): The locker room has n lockers, numbered $0, \ldots, n-1$. A set of n cards, numbered $0, \ldots, n-1$, is inserted in the lockers according to a uniformly random permutation σ. Alice and Bob are a team with a task. Alice enters the locker room, opens all the lockers and can swap the cards between just two lockers, or may choose to leave them unchanged. She closes all the lockers and leaves the room. Bob is given a number $\mathfrak{s} \in [n-1]$ and his task is to find card $\mathfrak{s}$. He can open at most two lockers. Before the game begins, Alice and Bob may communicate to decide on a strategy. What is their optimal strategy, and how efficient is it?

As in the *needle in a haystack* problem, without help from Alice, Bob can do no better than open lockers at random. If he opens one locker his probability of success is $\frac{1}{n}$ and if he opens two lockers this probability is $\frac{2}{n}$. With the help of Alice, he can do better when opening one locker. E.g., their strategy could be that Bob will open locker $\mathfrak{s}$, where $\mathfrak{s}$ is his given number. Alice would then try to increase the number of fixed points in the permutation above the expected number of 1. If there is a transposition she can reverse it, increasing the number of fixed points by two, and if not she can produce one more fixed point (unless the permutation is the identity). This strategy succeeds with probability just under $\frac{12}{5n}$. When Bob can open two lockers, the challenge is to increase the success probability to $\omega(\frac{1}{n})$.

The answer involves viewing Bob's first locker opening in a different way: not as looking for his card but as receiving a communication from Alice. The interest is in finding what kind of information Alice can send about the permutation which could help Bob in his search.

Now, we invite the reader to stop for a moment: to think about this puzzle, to find any strategy that could ensure the success probability would be $\omega(\frac{1}{n})$.

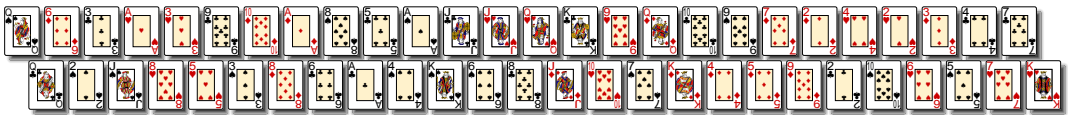


Figure 1 Consider the following randomly shuffled deck, one card per locker. What advice should Alice give to Bob – just by swapping the locations of at most one pair of cards – to increase the probability that Bob will find his randomly chosen card by opening at most two lockers?

It is easy to see that a solution to the *needle in a haystack* search problem immediately yields a solution to the *locker room* problem: Alice just takes the card corresponding to the advice and swaps it into the first locker. For example, the shuffled deck from Figure 1 corresponds to the following permutation σ of 52 numbers:

$$\begin{aligned}\sigma(0, 1, \ldots, 51) = \langle &49, 17, 1, 38, 27, 7, 21, 25, 45, 3, 51, 9, 35, 36, 11, 33, 23,\\ &8, 46, 18, 13, 28, 26, 14, 2, 5, 10, 39, 48, 32, 29, 40, 19, 4,\\ &50, 43, 6, 22, 34, 44, 24, 15, 16, 20, 0, 47, 30, 42, 31, 37\rangle\end{aligned}$$

with mapping: ♣: 0–12 (in order *2,3,4,5,6,7,8,9,10,J,Q,K,A*), ♢: 13–25, ♡: 26–38, ♠: 39–51. We see, for example, that ♠Q, card number 49 is in locker 0. If in the *needle in a haystack* search problem the advice is a number $\mathfrak{h} \in [n-1]$, then Alice swaps the contents of locker 0 and the locker containing the card corresponding to number $\mathfrak{h}$. This way, Bob gets the advice $\mathfrak{h}$ by opening locker 0.

For the strategy we propose in Theorem 5, Alice would swap ♠Q and ♡5. But can we do better?

1.2 Results for the *needle in a haystack* and *locker room* problems

We present a tight analysis of the *needle in a haystack* search problem. While some basic examples suggest that it is difficult to ensure success probability $\omega(\frac{1}{n})$, we will show that one can improve this probability considerably. Our main results are tight (up to lower order terms) lower and upper bounds for the maximum probability that with a single number hint one can find the object sought. First, we will show that for any strategy, the probability that one can find the sought object is at most $\frac{(1+o(1))\log n}{n\log\log n}$ (Theorem 5). Next, as the main result of this paper, we will complement this by designing a simple strategy that with a hint ensures that the sought object is found with probability at least $\frac{(1+o(1))\log n}{n\log\log n}$ (Theorem 6).

Further, we demonstrate essentially the same results for the *locker room* problem. Theorem 6 for the *needle in a haystack* search problem immediately implies that there is a simple strategy for Alice and Bob which ensures that Bob finds his card with probability at least $\frac{(1+o(1))\log n}{n\log\log n}$. We will complement this claim and extend in Theorem 21 the result from Theorem 5 for the *needle in a haystack* search problem to prove that for any strategy for Alice and Bob, the probability that Bob finds the required card is at most $O\left(\frac{\log n}{n\log\log n}\right)$.

Techniques. Our analysis exploits properties of random permutations to ensure that some short advice can reveal information about the input permutation, which can be used to increase the success probability substantially. Our approach relies on a close relationship between the *needle in a haystack* search problem and some intrinsic properties of random permutations related to the *rencontres number*, the number of n-permutations with a given number of fixed points. The two parts of our analysis for the *needle in a haystack* problem, the upper bound and the lower bound for the success probability, use quite different techniques.

To show the upper bound for the success probability (Theorem 5), we first observe that every deterministic strategy corresponds to a unique partition of $\mathbb{S}_n$ (set of all permutations of $[n-1]$) into n parts, with part $\mathfrak{h}$ containing exactly those permutations that cause the choice of hint $\mathfrak{h}$. By a careful analysis of the properties of this partition, we devise a metric for the best possible accuracy of the prediction counting instances in each part of the partition in which a permutation maps a given choice $\mathfrak{i}$ to $\mathfrak{s}$. By combining these estimation with the bounds for the rencontres number, we prove the desired upper bound for the success probability in the *needle in a haystack* search problem. An application of Yao's principle shows that our results are also valid for randomized strategies.

To show the lower bound for the success probability (Theorem 6), we first present a simple *shift strategy*, and then provide a non-trivial analysis of random permutations that demonstrates desirable properties of this strategy. The analysis here is related to the maximum load problem for balls and bins, where one allocates n balls into n bins, chosen independently and uniformly at random (*i.u.r.*). However, the dependencies between locations of distinct elements in the random permutations make this analysis more complex (cf. Remark 11 for more detailed discussion).

Finally, while a solution to the *needle in a haystack* search problem immediately yields a solution to the *locker room* problem with the same success probability, we complement our analysis by showing (Theorem 21) that no strategy of Alice and Bob can do much better. We show that Alice can do little more than just to send a few numbers to Bob, which is essentially the setup of the *needle in a haystack* search problem.

1.3 Background: Permutations, puzzles, and locker rooms

Our *locker room* problem follows a long line of the study of combinatorial puzzles involving the analysis of properties of permutations. One such example is the following locker problem involving prisoners and lockers: There are n lockers into which a random permutation of n cards are inserted. Then n prisoners enter the locker room one at a time and are allowed to open half of the lockers in an attempt to find their own card. The team of prisoners wins if every one of them is successful. The surprising result is that there is a strategy which wins with probability about $1 - \ln 2$. This problem was initially considered by Peter Bro Miltersen and appeared in his paper with Anna Gál [7], which won a best paper award at ICALP 2003. In that paper they refer to a powerful strategy approach suggested by Sven Skyum but it was left to the readers to find it for themselves. This is the idea of using the number contained in each locker as a pointer to another locker. Thus using a sequence of such steps corresponds to following a cycle in the permutation. Solutions to these problems are of a *combinatorial and probabilistic flavor* and involve an *analysis of the cycle structure of random permutations*. The original paper [7] stimulated many subsequent papers considering different variants (see, e.g., [4, 8]), including a matching upper bound provided in [5]. An early version giving the problem where each prisoner can open half of the lockers was published by [14] (see also [15, p. 18]). If each prisoner begins with the locker corresponding to the number they seek then they will all succeed provided that there is no cycle in the permutation which is longer than $\frac{n}{2}$. It is easy to show that a helpful prison warder, Alice, can always find an appropriate transposition of the contents of two lockers so that the resulting permutation has no cycle longer than $\frac{n}{2}$. We were told of this observation recently by Kazuo Iwama and this stimulated the current paper, in which we subvert the locker problem tradition with a problem which has little to do with the cycle structure of permutations and is more concerned with some basic communication complexity and rather different properties of permutations.

Various results about permutations have found diverse applications in computer science, especially for sorting algorithms (for example, see [10, Chapter 5]). In this paper, we are particularly interested in two such questions. Firstly, to apply known results concerning the asymptotic growth of the rencontres numbers, in order to approximate the optimal success probabilities in both the *needle in a haystack* problem and the *locker room* problem. Secondly, to use the concept of the rencontres numbers to examine the way in which the sizes of "shift sets" (sets of elements which a permutation displaces by the same number of positions "to the right") are distributed in permutations of $\mathbb{S}_n$ for a fixed natural number n. In particular, to determine the mean size of the largest shift set of a permutation chosen uniformly at random from $\mathbb{S}_n$, as well as to show that it is typical, i.e., that the variance of the size of the largest shift set is small. These results are useful for providing a concrete optimal strategy for both of the titular search problems.

2 Preliminaries

In this section, we prepare a framework for the study of strategies to prove an upper bound for the success probability for the *needle in a haystack* search problem (cf. Section 3). For the simplicity of the analysis, we will consider (in Sections 2, 3, and 6) the setting when $\mathfrak{s}$

is chosen i.u.r. from $[n-1]$; see Section 2.2 for justification that this can be done without loss of generality. First, let us rephrase the original problem in a form of an equivalent communication game between Alice and Bob: Bob, the *seeker*, has as his input a (random) number $\mathfrak{s} \in [n-1]$. Alice, the *adviser*, sees a permutation σ chosen i.u.r. from $\mathbb{S}_n$, and uses σ to send advice to Bob in the form of a number $\mathfrak{h} \in [n-1]$. Bob does not know σ, but on the basis of $\mathfrak{s}$ and $\mathfrak{h}$, he picks some $\mathfrak{i} \in [n-1]$ trying to maximize the probability that $\sigma(\mathfrak{i}) = \mathfrak{s}$.

First we will consider *deterministic strategies* (we will later argue separately that randomized strategies cannot help much here). Since we consider deterministic strategies, the advice sent is a function $\mathbb{S}_n \to [n-1]$, which can be defined by a partition of $\mathbb{S}_n$ into n sets. This naturally leads to the following definition of a *strategy*.

▶ **Definition 1.** *A* ***strategy*** $\mathbb{C}$ *for* $\mathbb{S}_n$ *is a partition of* $\mathbb{S}_n$ *into* n *sets* $C_0, C_1, \ldots, C_{n-1}$*. Such a strategy* $\mathbb{C}$ *is denoted by* $\mathbb{C} = \langle C_0, C_1, \ldots, C_{n-1} \rangle$*.*

Given a specific strategy $\mathbb{C}$, we examine the success probability. Let $\mathcal{V}$ be the event that the sought number is found, $\mathcal{A}_\mathfrak{h}$ the event that $\mathfrak{h}$ is the received advice, and $\mathcal{B}_\mathfrak{s}$ the event that $\mathfrak{s}$ is the sought number. Notice that for every $\mathfrak{h} \in [n-1]$ we have $\mathbf{Pr}[\mathcal{A}_\mathfrak{h}] = \frac{|C_\mathfrak{h}|}{n!}$ and for every $\mathfrak{s} \in [n-1]$ we have $\mathbf{Pr}[\mathcal{B}_\mathfrak{s}] = \frac{1}{n}$. Therefore, since the events $\mathcal{A}_\mathfrak{h}$ and $\mathcal{B}_\mathfrak{s}$ are independent,

$$\mathbf{Pr}[\mathcal{V}] = \sum_{\mathfrak{s}=0}^{n-1} \sum_{\mathfrak{h}=0}^{n-1} \mathbf{Pr}[\mathcal{V}|\mathcal{A}_\mathfrak{h} \cap \mathcal{B}_\mathfrak{s}] \cdot \mathbf{Pr}[\mathcal{A}_\mathfrak{h} \cap \mathcal{B}_\mathfrak{s}] = \sum_{\mathfrak{s}=0}^{n-1} \sum_{\mathfrak{h}=0}^{n-1} \mathbf{Pr}[\mathcal{V}|\mathcal{A}_\mathfrak{h} \cap \mathcal{B}_\mathfrak{s}] \cdot \mathbf{Pr}[\mathcal{A}_\mathfrak{h}] \cdot \mathbf{Pr}[\mathcal{B}_\mathfrak{s}]$$
$$= \frac{1}{n} \sum_{\mathfrak{h}=0}^{n-1} \frac{|C_\mathfrak{h}|}{n!} \cdot \sum_{\mathfrak{s}=0}^{n-1} \mathbf{Pr}[\mathcal{V}|\mathcal{A}_\mathfrak{h} \cap \mathcal{B}_\mathfrak{s}] \ . \tag{1}$$

▶ **Definition 2.** *Let* $\mathbb{C} = \langle C_0, C_1, \ldots, C_{n-1} \rangle$ *be a strategy. The* ***magneticity*** *of an element* i *for an element* k *in the class* C_j *is defined as* $\mathsf{mag}(C_j, i, k) = |\{\sigma \in C_j : \sigma(i) = k\}|$*.*

The element with the greatest magneticity for k *in the class* C_j *is called the* ***magnet in*** C_j ***of*** k *and is denoted* $\mathsf{max\text{-}mag}(C_j, k)$*; ties are broken arbitrarily. The magneticity of* $\mathsf{max\text{-}mag}(C_j, k)$ *is called the* ***intensity of*** k ***in*** C_j*, denoted by* $\mathsf{int}(C_j, k)$*; that is,* $\mathsf{int}(C_j, k) = \max_{i \in [n-1]} \{\mathsf{mag}(C_j, i, k)\}$*.*

This can be extended in a natural way to any $C = \langle A_0, A_1, \ldots, A_{n-1} \rangle$ of n subsets of $\mathbb{S}_n$.

Let us discuss the intuitions. Firstly, the *magneticity* in the class C_j of an element i for an element k, $\mathsf{mag}(C_j, i, k)$, denotes the number of permutations in C_j with k in position i. Therefore, the *magnet* in C_j of k is an index $i \in [n-1]$ such that among all permutations in C_j, k is most likely to be in position i. The *intensity* in C_j of k denotes just the number of times (among all permutations in C_j) that k appears in the position of the magnet i.

In the *needle in a haystack* search problem, Alice sends to Bob a message $\mathfrak{h}$ which points to a class $C_\mathfrak{h}$ of their agreed strategy $\mathbb{C}$, and Bob has to choose a number $\mathfrak{i}$ in order to find whether $\sigma(\mathfrak{i})$ is the number $\mathfrak{s} \in [n-1]$ which he seeks. The maximum probability that they succeed is $\frac{\mathsf{int}(C_\mathfrak{h}, \mathfrak{s})}{|C_\mathfrak{h}|}$, realized if Bob opts for the magnet of $\mathfrak{s}$ in $C_\mathfrak{h}$. Thus, by (1), we obtain

$$\mathbf{Pr}[\mathcal{V}] \leq \frac{1}{n} \cdot \frac{1}{n!} \sum_{\mathfrak{s},\mathfrak{h} \in [n-1]} \mathsf{int}(C_\mathfrak{h}, \mathfrak{s}) \ .$$

▶ **Definition 3.** *Let the* ***field*** *of* $\mathbb{S}_n$ *be* $F(n) = \max_{\mathbb{C}=\langle C_0, C_1, \ldots, C_{n-1} \rangle} \sum_{\mathfrak{s},\mathfrak{h} \in [n-1]} \mathsf{int}(C_\mathfrak{h}, \mathfrak{s})$*.*

With this definition, a strategy which yields the field of $\mathbb{S}_n$ is called *optimal*, and

$$\mathbf{Pr}[\mathcal{V}] \leq \frac{1}{n} \cdot \frac{1}{n!} \sum_{\mathfrak{s},\mathfrak{h} \in [n-1]} \mathsf{int}(C_\mathfrak{h}, \mathfrak{s}) \leq \frac{1}{n} \cdot \frac{F(n)}{n!} \ . \tag{2}$$

We will use this bound to prove Theorem 5 in Section 3, that whatever the strategy, we always have $\mathbf{Pr}[\mathcal{V}] \leq \frac{(1+o(1)) \cdot \log n}{n \log \log n}$.

2.1 Derangements

We use properties of random permutations related to derangements and rencontres numbers.

▶ **Definition 4.** *A permutation $\sigma \in \mathbb{S}_n$ with no fixed points is called a* ***derangement****. The number of derangements in $\mathbb{S}_n$ is denoted D_n. A permutation $\sigma \in \mathbb{S}_n$ with exactly r fixed points is called an* ***r-partial derangement****. The number of r-partial derangements in $\mathbb{S}_n$ (also known as the* rencontres number*) is denoted $D_{n,r}$.*

Definition 4 yields $D_{n,0} = D_n$ and it is easy to see that $D_{n,r} = \binom{n}{r} \cdot D_{n-r}$. It is also known (see, e.g., [9, p. 195]) that $D_n = \lfloor \frac{n!}{e} + \frac{1}{2} \rfloor$, and hence one can easily show $D_{n,r} \leq \frac{n!}{r!}$.

2.2 Formal framework and justification about worst-case vs. random $\mathfrak{s}$

We consider the problem with two inputs: a number $\mathfrak{s} \in [n-1]$ and a permutation $\sigma \in \mathbb{S}_n$. We are assuming that σ is a random permutation in $\mathbb{S}_n$; no assumption is made about $\mathfrak{s}$.

For the *needle in a haystack* search problem (a similar framework can be easily set up for the *locker room* problem), a *strategy* (or an *algorithm*) is defined by a pair of (possibly randomized) functions, $\mathfrak{h} = \mathfrak{h}(\sigma)$ and $\mathfrak{i} = \mathfrak{i}(\mathfrak{h}, \mathfrak{s})$, with both $\mathfrak{h}, \mathfrak{i} \in [n-1]$.

For a fixed strategy, let $\mathfrak{p}(\mathfrak{s})$ be the success probability for a given $\mathfrak{s}$ and for a randomly chosen $\sigma \in \mathbb{S}_n$. That is,

$$\mathfrak{p}(\mathfrak{s}) = \mathbf{Pr}[\sigma(\mathfrak{i}) = \mathfrak{s}] \ ,$$

where the probability is over σ taken i.u.r. from $\mathbb{S}_n$, and over the randomness in the choice of the strategy (since both $\mathfrak{h} = \mathfrak{h}(\sigma)$ and $\mathfrak{i} = \mathfrak{i}(\mathfrak{h}, \mathfrak{s})$ may be randomized functions).

The goal is to design an algorithm (find a strategy) that will achieve some success probability for every $\mathfrak{s} \in [n-1]$. That is, we want to have a strategy which maximizes

$$\mathbf{Pr}[\mathcal{V}] = \min_{\mathfrak{s} \in [n-1]} \{\mathfrak{p}(\mathfrak{s})\} \ .$$

In our analysis for the upper bounds in Sections 2 and 3 (Theorem 5) and Section 6 (Theorem 21), for simplicity, we will be making the assumption that $\mathfrak{s}$, the input to the *needle in a haystack* search problem and to the *locker room* problem, is random, that is, is chosen i.u.r. from $[n-1]$. (We do not make such assumption in the lower bound in Section 4 (Theorem 6), where the analysis is done explicitly for arbitrary $\mathfrak{s}$.) Then the main claim (Theorem 5) is that if we choose $\mathfrak{s}$ i.u.r. then $\mathfrak{p}(\mathfrak{s}) \leq \frac{(1+o(1)) \log n}{n \log \log n}$, though in fact, one can read this claim as that $\sum_{\mathfrak{s} \in [n-1]} \frac{\mathfrak{p}(\mathfrak{s})}{n} \leq \frac{(1+o(1)) \log n}{n \log \log n}$. However, notice that this trivially yields

$$\mathbf{Pr}[\mathcal{V}] = \min_{\mathfrak{s} \in [n-1]} \{\mathfrak{p}(\mathfrak{s})\} \leq \sum_{\mathfrak{s} \in [n-1]} \frac{\mathfrak{p}(\mathfrak{s})}{n} \ ,$$

and therefore Theorem 5 yields $\mathbf{Pr}[\mathcal{V}] \leq \frac{(1+o(1)) \log n}{n \log \log n}$, as required.

Observe that such arguments hold only for the upper bound. Indeed, since $\min_{\mathfrak{s} \in [n-1]} \{\mathfrak{p}(\mathfrak{s})\}$ may be much smaller than $\sum_{\mathfrak{s} \in [n-1]} \frac{\mathfrak{p}(\mathfrak{s})}{n}$, in order to give a lower bound for the success probability, Theorem 6 proves that there is a strategy that ensures that $\mathfrak{p}(\mathfrak{s}) \geq \frac{(1+o(1)) \log n}{n \log \log n}$ for every $\mathfrak{s} \in [n-1]$; this clearly yields $\mathbf{Pr}[\mathcal{V}] \geq \frac{(1+o(1)) \log n}{n \log \log n}$, as required.

3 Upper bound for the success probability for needle in a haystack

We will use the framework set up in the previous section, in particular the tools in Definition 2 and inequality (2) and that $\mathfrak{s}$ is chosen i.u.r. from $[n-1]$, to bound from above the best possible success probability in the *needle in a haystack* search problem.

▶ **Theorem 5.** *For any strategy in the* needle in a haystack *problem, the success probability is*

$$\mathbf{Pr}[\mathcal{V}] \leq \frac{(1+o(1))\cdot \log n}{n \log\log n} \ .$$

Proof. We will first consider only *deterministic* strategies and, only at the end, we will argue that this extends to randomized strategies.

Consider an optimal strategy $\mathbb{C} = \langle C_0, \ldots, C_{n-1}\rangle$. First, we will modify sets $C_0, \ldots, C_{n-1}$ to ensure that each C_j has n distinct magnets.

Fix $j \in [n-1]$. Suppose that there are two elements $k_1 < k_2 \in [n-1]$ with the same magnet i_1 in C_j. Since there are exactly n elements and n possible magnets, there is some $i_2 \in [n-1]$ which is not a magnet in C_j of any element. For every $\sigma \in C_j$ with $\sigma(i_1) = k_2$, calculate $\sigma' = \sigma(i_1 i_2)$ (that is, σ' is the same as σ, except that the images of i_1 and i_2 are exchanged). Now, if $\sigma' \notin C_j$, then remove σ from C_j and add σ' to C_j. We notice the following properties of the resulting set C'_j in the case that some σ is replaced by σ':

(i) $|C'_j| = |C_j|$.

(ii) i_2 can be chosen as the new magnet of k_2. Indeed, for every $i \neq i_1, i_2$, we have

$$\mathsf{mag}(C'_j, i_2, k_2) > \mathsf{mag}(C_j, i_1, k_2) = \mathsf{int}(C_j, k_2) \geq \mathsf{mag}(C_j, i, k_2) = \mathsf{mag}(C'_j, i, k_2), \text{ so}$$
$$\mathsf{mag}(C'_j, i_2, k_2) = \mathsf{int}(C'_j, k_2) > \mathsf{int}(C_j, k_2) \ .$$

(iii) None of the intensities decreases. Indeed the only differences are due to changes to permutations $\sigma \in C_j$ with $\sigma(i_1) = k_2$. Such a permutation where $\sigma(i_2) = k_3$, say, is replaced by σ', where $\sigma'(i_2) = k_2$ and $\sigma'(i_1) = k_3$, if σ' is not already in C_j. As shown in (ii), the intensity of k_2 increases. For k_3, only $\mathsf{mag}(C_j, i_2, k_3)$ decreases, but since i_2 was not a magnet in C_j, the magnet in C'_j of k_3, and hence $\mathsf{int}(C_j, k_3)$, is unchanged.

We repeat this operation for every remaining pair of elements which share a magnet in C_j until we arrive at a set of permutations which has n distinct magnets. Then, we perform the same process for every other class in $\mathbb{C}$.

To see that this algorithm indeed terminates, (ii) shows that if in any iteration the magnet of an element i changes, then $\mathsf{int}(C'_j, i) > \mathsf{int}(C_j, i)$. As the maximum intensity of any element within a class C_j is $|C_j|$ and the minimum is 1, the algorithm terminates after $n \cdot n!$ iterations.

Let us consider the collection $C = \langle A_0, \ldots, A_{n-1}\rangle$ obtained. From (i), we see that the sets of C contain a total of $n!$ permutations of $\mathbb{S}_n$. Permutations belonging to the same set A_j are necessarily distinct, but two different sets of C may have non-trivial intersection. Hence, C may not be a strategy. Every A_j has n distinct magnets, one for each element of $[n-1]$. Most importantly, by (iii), we have

$$\sum_{i,j\in[n-1]} \mathsf{int}(A_j, i) \geq \sum_{i,j\in[n-1]} \mathsf{int}(C_j, i) = F(n) \ .$$

Hence, calculating an upper bound for $\sum_{i,j\in[n-1]} \mathsf{int}(A_j, i)$ yields an upper bound for $F(n)$.

The set A_j has exactly n magnets, one for each element of $[n-1]$. For a permutation $\sigma \in A_j$ to contribute r to $\sum_{i\in[n-1]} \mathsf{int}(A_j, i)$, σ^{-1} must map exactly r elements to their magnets in A_j. Hence, (cf. Definition 4) there are at most $D_{n,r}$ permutations in A_j which contribute exactly r to $\sum_{i\in[n-1]} \mathsf{int}(A_j, i)$. Recall that $D_{n,r} \leq \frac{n!}{r!}$ and thus for any natural ℓ,

$$\sum_{i\in[n-1]} \mathsf{int}(A_j, i) \leq \ell \cdot |A_j| + \sum_{r=\ell+1}^{n} r \cdot D_{n,r} = \ell \cdot |A_j| + \sum_{r=\ell+1}^{n} \frac{n!}{(r-1)!} \leq \ell \cdot |A_j| + \frac{en!}{\ell!} \ .$$

We will choose some $\ell = \frac{(1+o(1))\log n}{\log\log n}$ to ensure that $\ell! = \omega(n)$, giving

$$F(n) \le \sum_{i,j\in[n-1]} \mathsf{int}(A_j, i) \le \sum_{j\in[n-1]} (\ell \cdot |A_j| + o((n-1)!)) = (\ell + o(1))n! = \frac{(1+o(1))\log n}{\log\log n} n! \quad (3)$$

We can combine (2) and (3) to obtain the following,

$$\mathbf{Pr}[\mathcal{V}] \le \frac{1}{n} \cdot \frac{F(n)}{n!} \le \frac{(1+o(1))\log n}{n\log\log n} \ .$$

The upper bound of $\frac{(1+o(1))\log n}{n\log\log n}$ is valid not only for deterministic strategies, but also for *randomized strategies*. Let $c(\mathbb{C}, (\sigma, i))$ be the indicator function of the event that the strategy $\mathbb{C}$ fails to guess the image of i under the permutation σ. Let us consider a probability measure P over the set D of all deterministic strategies, and the distribution $U = (U_{\mathbb{S}_n}, U_{[n-1]})$ over $\mathbb{S}_n \times [n-1]$, where U_S denotes the uniform probability measure over the set S. Let S be a random strategy chosen according to P, and let X be a random set-up chosen according to U. Then, by Yao's principle, $\max_{(\sigma,i)\in\mathbb{S}_n\times[n-1]} \mathbf{E}[c(S,(\sigma,i))] \ge \min_{\mathbb{C}\in D} \mathbf{E}[c(\mathbb{C}, X)]$. That is, the probability that a randomized strategy fails for the worst-case input exceeds the probability that an optimal deterministic strategy fails. Hence, the worst-case probability that a randomized strategy succeeds is also bounded above by $\frac{(1+o(1))\log n}{n\log\log n}$. ◀

4 Lower bound: solution for the needle in a haystack search problem

In Theorem 5, we showed that whatever strategy we use in the *needle in a haystack* problem, the best success probability we can hope for is $\frac{(1+o(1))\log n}{n\log\log n}$. In this section we will show that such success probability is achievable by a simple strategy, which we call the *shift strategy*.

- Let $\mathfrak{h} \in [n-1]$ maximize $|\{\ell \in [n-1] : \ell = \sigma(\ell + \mathfrak{h} \pmod n)\}|$.
- In order to find number $\mathfrak{s} \in [n-1]$ in σ, check $\sigma(\mathfrak{s} + \mathfrak{h} \pmod n)$.

(Our choice of $\mathfrak{h}$ is equivalent to maximizing $|\{\ell \in [n-1] : (\ell - \mathfrak{h} \pmod n) = \sigma(\ell)\}|$.)

We will prove that the shift strategy ensures a success probability of at least $\frac{(1+o(1))\log n}{n\log\log n}$. Notice that this is equivalent to saying that $\mathbf{Pr}[\sigma(\mathfrak{s} + \mathfrak{h} \pmod n) = \mathfrak{s}] \ge \frac{(1+o(1))\log n}{n\log\log n}$, and hence, by the definition of $\mathfrak{h}$, that with probability $1 - o(1)$,

$$\max_{s\in[n-1]} \left\{ \left| \{\ell \in [n-1] : \ell - \sigma(\ell) = s \pmod n\} \right| \right\} \ge \frac{(1+o(1))\log n}{\log\log n} \ .$$

This also implies, by Theorem 5 in Section 3, that the shift strategy is asymptotically optimal.

▶ **Theorem 6.** *For any $\mathfrak{s} \in [n-1]$, the shift strategy satisfies $\mathbf{Pr}[\mathcal{V}] \ge \frac{(1+o(1))\log n}{n\log\log n}$.*

In order to prove Theorem 6, we introduce some notation. For every $i \in [n-1]$, let $v(i) = i - \sigma(i) \pmod n$. Since σ is random, $v(i)$ has uniform distribution over $[n-1]$.

Let $S_\ell = |\{i \in [n-1] : v(i) = \ell\}|$. Notice that in the shift strategy $\mathbb{C} = \langle C_0, C_1, \ldots, C_{n-1}\rangle$, if $\sigma \in C_\mathfrak{h}$ then $S_\mathfrak{h} = \max_{\ell\in[n-1]}\{S_\ell\}$. Therefore, our goal is to study basic properties of the distribution of $S_\mathfrak{h}$, and in particular, to estimate the largest value of S_j over all $j \in [n-1]$.

▶ **Example 7.** Using the example presented in Figure 1 with

$$\sigma(0,1,\ldots,51) = \langle 49,17,1,38,27,7,21,25,45,3,51,9,35,36,11,33,23,8,46,18,13,28,26,$$
$$14,2,5,10,39,48,32,29,40,19,4,12,41,50,43,6,22,34,44,24,15,16,20,0,47,30,42,31,37\rangle,$$

we have

$$v(0,1,\ldots,51) = \langle 3,36,1,17,29,50,37,34,15,6,11,2,29,29,3,34,45,9,24,1,7,45,48,9,$$
$$22,20,16,40,32,49,1,43,13,29,22,46,38,46,32,17,6,49,18,28,28,25,46,0,18,7,19,14\rangle.$$

Then

$$S_{0,1,2,\ldots,50,51} = \langle 1,3,1,2,0,0,2,2,0,2,0,1,0,1,1,1,1,2,2,1,1,0,2,0,1,1,$$
$$0,0,2,4,0,0,2,0,2,0,1,1,1,0,1,0,0,1,0,2,3,0,1,2,1,0\rangle,$$

so $\mathfrak{h} = 29$ and $S_{\mathfrak{h}} = 4$. Alice delivers this hint to Bob by exchanging cards ♡5 and ♠Q. Then, over all $\mathfrak{s} \in [n-1]$, $\mathbf{Pr}[\sigma(\mathfrak{s}+29 \pmod{52}) = \mathfrak{s}] = \frac{4}{52}$. ⊠

Let us first notice the following simple auxiliary lemma which should give the *intuition* behind our approach (see the end of this section for a standard and elementary proof).

▶ **Lemma 8.** *The expected number of values $j \in [n-1]$ with $S_j \geq \frac{(1+o(1))\cdot\log n}{\log\log n}$ is at least one.*

Lemma 8 tells us that in expectation, there is at least one value j such that $S_j \geq \frac{(1+o(1))\log n}{\log\log n}$. Notice however that in principle, we could have that the expectation is high but only because with small probability the random variable takes a very high value. Therefore the bound in Lemma 8 is fairly weak. We will now prove, using the second moment method, that with high probability there is some j such that $S_j \geq \frac{(1+o(1))\log n}{\log\log n}$. This yields Theorem 6.

▶ **Lemma 9.** *With probability $1-o(1)$ there is some $j \in [n-1]$ such that $S_j \geq \frac{(1+o(1))\log n}{\log\log n}$.*

Proof. Let Z_j^t be the indicator random variable that $S_j = t$. Let $R_t = \sum_{j=0}^{n-1} Z_j^t$. With this notation, our goal is to show that $R_t = 0$ is unlikely for our choice of some $t = \frac{(1+o(1))\log n}{\log\log n}$ (since if $R_t > 0$ then $\max_{j\in[n-1]} S_j \geq t$, and hence $\mathbf{Pr}\big[\max_{j\in[n-1]} S_j \geq t\big] \geq \mathbf{Pr}[R_t > 0]$). We use the second moment method relying on a standard implication of Chebyshev's inequality,

$$\mathbf{Pr}\big[\max_{j\in[n-1]} S_j < t\big] \leq \mathbf{Pr}\big[R_t = 0\big] \leq \frac{\mathbf{Var}[R_t]}{\mathbf{E}[R_t]^2} \ . \tag{4}$$

Let us recall that

$$\mathbf{Var}[R_t] = \mathbf{Var}[\sum_{j=0}^{n-1} Z_j^t] = \sum_{j=0}^{n-1} \mathbf{Var}[Z_j^t] + \sum_{i,j\in[n-1],i\neq j} \mathbf{Cov}[Z_i^t, Z_j^t] \ . \tag{5}$$

Next, since every Z_j^t is a 0-1 random variable, we obtain the following,

$$\mathbf{Var}[Z_j^t] = \mathbf{Pr}[Z_j^t = 1]\cdot\mathbf{Pr}[Z_j^t = 0] \leq \mathbf{Pr}[Z_j^t = 1] = \mathbf{E}[Z_j^t] \ . \tag{6}$$

Our main technical claim is that the covariance of random variables Z_j^t, Z_i^t is small. Although the proof of Lemma 10 is the *main technical contribution* of this section, for the clarity of the presentation, we defer its proof to Section 5.

▶ **Lemma 10.** *Let $t \leq O(\log n)$. Then, the following holds for any $i \neq j$, $i,j \in [n-1]$:*

$$\mathbf{Cov}[Z_i^t, Z_j^t] = \mathbf{E}\big[Z_i^t \cdot Z_j^t\big] - \mathbf{E}\big[Z_i^t\big]\cdot\mathbf{E}[Z_j^t] \leq o(1)\cdot\mathbf{E}\big[Z_i^t\big]\cdot\mathbf{E}[Z_j^t] \ . \tag{7}$$

Therefore, if we combine (6) and Lemma 10 in identity (5), then (assuming $t \le O(\log n)$)

$$\begin{aligned}\mathbf{Var}[R_t] &= \sum_{j=0}^{n-1} \mathbf{Var}[Z_j^t] + \sum_{i,j\in[n-1],i\neq j} \mathbf{Cov}[Z_i^t, Z_j^t] \le \sum_{j=0}^{n-1} \mathbf{E}[Z_j^t] + o(1) \sum_{i,j\in[n-1],i\neq j} \mathbf{E}\big[Z_i^t\big]\mathbf{E}[Z_j^t] \\ &= \mathbf{E}[R_t] + o(1)\cdot \mathbf{E}[R_t]^2 \ .\end{aligned}$$

If we plug this in (4), then we will get the following (assuming $t \le O(\log n)$),

$$\mathbf{Pr}\big[R_t = 0\big] \le \frac{\mathbf{Var}[R_t]}{\mathbf{E}[R_t]^2} \le \frac{1}{\mathbf{E}[R_t]} + o(1) \ . \tag{8}$$

Therefore, if for some $\varsigma > 0$ we have $\mathbf{E}[R_t] \ge \varsigma$ (with $t \le O(\log n)$) then the bound above yields $\mathbf{Pr}\big[\max_{i\in[n-1]} S_i < t\big] \le \frac{1}{\varsigma} + o(1)$. Hence we can combine this with (9) to obtain $\mathbf{E}[R_t] = \sum_{j=0}^{n-1} \mathbf{E}[Z_j^t] = \sum_{j=0}^{n-1} \mathbf{Pr}[S_j = t] > \frac{n}{2et!}$, which is $\omega(1)$ for any t such that $t! = o(n)$. This in particular holds for some $t = \frac{(1+o(1))\log n}{\log\log n}$, and thus concludes Lemma 9. ◀

▶ Remark 11. A reader may notice a close similarity of the problem of estimating $\max_{i\in[n-1]} S_i$ to the maximum load problem for balls and bins, where one allocates n balls into n bins i.u.r. Indeed, random variables $S_0, \ldots, S_{n-1}$ have similar distribution to the random variables $B_0, \ldots, B_{n-1}$, where B_i represents the number of balls allocated to bin i. However, the standard approaches used in the analysis of balls-and-bins processes seem to be more complicated in our setting. The main reason is that while every single random variable S_i has approximately Poisson distribution with mean 1, as has B_i too, the analysis of $\max_{i\in[n-1]} S_i$ is more complicated than the analysis of $\max_{i\in[n-1]} B_i$ because of the intricate *correlation* of random variables $S_0, \ldots, S_{n-1}$. For example, one standard approach to show that $\max_{i\in[n-1]} B_i \ge \frac{(1+o(1))\log n}{\log\log n}$ with high probability relies on the fact that the load of a set of bins B_i with $i \in I$ decreases if we increase the load of bins B_j with $j \in J$, $I \cap I = \emptyset$. However, the same property holds only *approximately* for $S_0, \ldots, S_{n-1}$ (and in fact, the $o(1)$ error term in Lemma 10 corresponds to this notion of "approximately"; for balls and bins the covariance is known to be always non-positive). To see the difficulty (see also the classic reference for permutations [13, Chapters 7–8]), notice that, for example, if $\sigma(i) = i + \ell$ then we cannot have $\sigma(i+1) = i+\ell$, meaning that there is a special correlation between S_ℓ (which counts i with $\sigma(i) = i + \ell$) and $S_{\ell-1}$ (which counts i with $\sigma(i+1) = i + \ell$). In particular, from what we can see, random variables $S_0, \ldots, S_{n-1}$ are not negatively associated [6]. In a similar way, we do not expect the Poisson approximation framework from [1] (see also [11, Chapter 5.4]) to work here. Our approach is therefore closer to the standard second moment method, see, e.g., [2, Chapter 3] and [12].

Elementary proof of Lemma 8. Let us recall Definition 4 for derangements and r-partial derangements. The probability that a random permutation in $\mathbb{S}_n$ is a derangement is $D_n/n! = \lfloor \frac{n!}{e} + \frac{1}{2} \rfloor / n! \sim \frac{1}{e}$. Let $u(n) = \lfloor \frac{n!}{e} + \frac{1}{2} \rfloor / \frac{n!}{e}$ and note that $D_n = u(n)\, n!/e$, that $u(n) = 1 + o(1)$, and $u(n) > 0.9$ for all $n > 1$. Since the permutation $\sigma \in \mathbb{S}_n$ is chosen i.u.r.,

$$\mathbf{Pr}[S_0 = k] = \frac{D_{n,k}}{n!} = \frac{\binom{n}{k} D_{n-k}}{n!} = \frac{\binom{n}{k}\frac{(n-k)!}{e} u(n-k)}{n!} = \frac{u(n-k)}{ek!} \ .$$

The same bound can be obtained for S_j for every $j \ge 0$. For any permutation $\sigma \in \mathbb{S}_n$ and any integer $\ell \in [n-1]$, define permutation $\sigma_\ell \in \mathbb{S}_n$ such that $\sigma_\ell(i) = \sigma(i) + \ell \pmod n$. For any permutation $\sigma \in \mathbb{S}_n$ and any ℓ, the operator $\sigma \mapsto \sigma_\ell$ is a bijection from $\mathbb{S}_n$ to $\mathbb{S}_n$, and a

permutation $\sigma \in \mathbb{S}_n$ with $\ell \in [n-1]$ has exactly k fixed points if and only if permutation σ_ℓ has exactly k points with $\sigma_\ell(i) = i + \ell \pmod n$. Hence for every $j, j' \in [n-1]$ and $k \in [n]$, we have $\mathbf{Pr}[S_j = k] = \mathbf{Pr}[S_{j'} = k]$. Therefore, for any integers $j \in [n-1]$ and $k \in [n-2]$,

$$\mathbf{Pr}[S_j = k] = \frac{u(n-k)}{ek!} > \frac{1}{2ek!} \ . \tag{9}$$

Let $k(n)$ be the largest k such that $2ek! \le n$. Then $\mathbf{Pr}[S_j = k(n)] > 1/n$. Hence, if we let Q_j be the indicator random variable that $S_j = k(n)$, then $\mathbf{Pr}[Q_j = 1] > 1/n$, and hence $\mathbf{E}[\sum_{j=0}^{n-1} Q_j] = \sum_{j=0}^{n-1} \mathbf{E}[Q_j] = \sum_{j=0}^{n-1} \mathbf{Pr}[Q_j = 1] > 1$. Therefore, in expectation, there is at least one value j such that $S_j = k(n)$. It is easy to show that $k(n) = \frac{\log n}{\log \log n}(1 + o(1))$. ◀

5 Proof of Lemma 10: bounding the covariance of Z_i^t and Z_j^t

The main technical part of the analysis of the lower bound for the *needle in a haystack* search problem in Section 4 (cf. Theorem 6) relies on the proof Lemma 9. This proof, in turn, is quite simple except for one central claim, Lemma 10, bounding the covariance of Z_i^t and Z_j^t. The proof of Lemma 10 is rather lengthly, and because of space considerations, some proofs are deferred to the full version of the paper (cf. `https://arxiv.org/abs/2008.11448`).

Let Z_j^t be the indicator random variable that $S_j = t$. Since Z_i^t and Z_j^t are 0-1 random variables, we have $\mathbf{E}\left[Z_i^t \cdot Z_j^t\right] = \mathbf{Pr}[S_i = t, S_j = t]$, $\mathbf{E}\left[Z_i^t\right] = \mathbf{Pr}[S_i = t]$ and $\mathbf{E}[Z_j^t] = \mathbf{Pr}[S_j = t]$. Since $\mathbf{Pr}[S_i = t] = \mathbf{Pr}[S_j = t] = \frac{u(n-t)}{et!} = \frac{1+o(1)}{et!}$ by (9), to complete the proof of Lemma 10, we only have to show that, for $i \neq j$,

$$\mathbf{Pr}[S_i = t, S_j = t] \le (1 + o(1)) \cdot \frac{1}{(et!)^2} \ . \tag{10}$$

We will prove this claim in Lemma 19 in Section 5.2.4 below.

5.1 Notation and key intuitions

For any set $I \subseteq [n-1]$ and any integer $\ell \in [n-1]$, let $\mathcal{F}_{I,\ell} = \{\sigma \in \mathbb{S}_n : \sigma(i) = i + \ell \pmod n \text{ iff } i \in I\}$ and $\mathcal{F}^*_{I,\ell} = \{\sigma \in \mathbb{S}_n : \forall_{i \in I}\ \sigma(i) = i + \ell \pmod n\}$. Notice that $\mathcal{F}_{I,\ell} \subseteq \mathcal{F}^*_{I,\ell}$. Further, $|\mathcal{F}_{I,\ell}| = D_{n-t,0}$ where $t = |I|$, and

$$\mathbf{Pr}[S_i = t] = \frac{|\bigcup_{I \subseteq [n-1], |I|=t} \mathcal{F}_{I,i}|}{n!} = \frac{\sum_{I \subseteq [n-1], |I|=t} |\mathcal{F}_{I,i}|}{n!} = \frac{\binom{n}{t} \cdot D_{n-t,0}}{n!} \ .$$

Next, with this notation and for $i \neq j$, we also have

$$\mathbf{Pr}[S_i = t, S_j = t] = \frac{1}{n!} \left| \bigcup_{I,J \subseteq [n-1], |I|=|J|=t} \mathcal{F}_{I,i} \cap \mathcal{F}_{J,j} \right| = \frac{1}{n!} \sum_{I,J \subseteq [n-1], |I|=|J|=t} |\mathcal{F}_{I,i} \cap \mathcal{F}_{J,j}| \ .$$

Notice that in the sum above one can restrict attention only to $I \cap J = \emptyset$, since $\mathcal{F}_{I,i} \cap \mathcal{F}_{J,j} = \emptyset$ otherwise. In view of this, our goal is to estimate $|\mathcal{F}_{I,i} \cap \mathcal{F}_{J,j}|$ for disjoint sets $I, J \subseteq [n-1]$.

In what follows, we will consider sets S_i and S_j with $i = 0$ and $j = s$ for some $s \in [n-1] \setminus \{0\}$. By symmetry, we can consider the first shift to be 0 without loss of generality; s is an arbitrary non-zero value. As required in our analysis (cf. Lemma 10), we will consider $t \le O(\log n)$.

Our approach now is to focus on a typical pair I and J, and consider some atypical pairs separately. We will show in Lemma 13 that almost all pairs of disjoint sets I and J are so-called *compatible for shift* s. As a result, the contribution of pairs I and J that

are not compatible for s is negligible, and so we will focus solely on pairs compatible for s. Then, for the pair of indices I and J we will estimate $|\mathcal{F}_{I,i} \cap \mathcal{F}_{J,j}|$ using the Principle of Inclusion-Exclusion. For that, we will have to consider the contributions of all possible sets $K \subseteq [n-1] \setminus (I \cup J)$ to the set of permutations in $\mathcal{F}^*_{I,i} \cap \mathcal{F}^*_{J,j}$. As before, contributions of some sets are difficult to be captured and so we will show in Lemma 15 that almost all sets $K \subseteq [n-1] \setminus (I \cup J)$ are so-called *feasible for I, J, and s*. As a result, the contribution of sets K that are not feasible for I, J, and s is negligible, and so we will focus on sets that are feasible for I, J, and s. The final simplification follows from the fact that we do not have to consider all such sets K, but only small sets K, of size $O(\log n)$. Once we have prepared our framework, we will be able to use the Principle of Inclusion-Exclusion to estimate $|\bigcup_{I,J\subseteq[n-1],|I|=|J|=t} \mathcal{F}_{I,i} \cap \mathcal{F}_{J,j}|$ in Lemmas 18 and 19.

5.2 The analysis

For any integer ℓ and any subset $L \subseteq [n-1]$ we write $L + \ell$ to denote the set of elements in L shifted by ℓ, in the arithmetic modulo n, that is, $L + \ell = \{i + \ell \pmod n : i \in L\}$. Similarly, $L - \ell = \{i - \ell \pmod n : i \in L\}$.

Let $\Phi_{0,s}(I, J) = \mathcal{F}_{I,0} \cap \mathcal{F}_{J,s} = \{\sigma \in \mathbb{S}_n : \sigma(i) = i \text{ iff } i \in I \text{ and } \sigma(j) = j+s \pmod n \text{ iff } j \in J\}$. Let $\Phi^*_{0,s}(I, J) = \mathcal{F}^*_{I,0} \cap \mathcal{F}^*_{J,s} = \{\sigma \in \mathbb{S}_n : \forall_{i\in I}\ \sigma(i) = i \text{ and } \forall_{j\in J}\ \sigma(j) = j + s \pmod n\}$.

It is easy to compute the size of $\Phi^*_{0,s}(I, J)$. Notice first that if $I \cap J \neq \emptyset$ or $I \cap (J + s) \neq \emptyset$, then $\Phi^*_{0,s}(I, J) = \Phi_{0,s}(I, J) = \emptyset$. Otherwise, if $I \cap J = \emptyset$ and $I \cap (J + s) = \emptyset$, then $|\Phi^*_{0,s}(I, J)| = (n - |I \cup J|)!$ (see also Lemma 12).

However, our main goal, that of computing the size of $\Phi_{0,s}(I, J)$, is significantly more complicated, because this quantity cannot be reduced to an intersection test and a simple formula over n, $|I|$, $|J|$, and s.

5.2.1 Disjoint sets $I \subseteq [n-1]$ and $J \subseteq [n-1] \setminus I$ compatible for shift s

Let I and J be two arbitrary subsets of $[n-1]$ of size t each. We say *I and J are compatible for shift s* if the four sets I, J, $I - s$, and $J + s$ are all pairwise disjoint. With this notation, we have the following lemma.

▶ **Lemma 12.** *If I and J are compatible for shift s then $\Phi_{0,s}(I, J) \neq \emptyset$ and $|\Phi^*_{0,s}(I, J)| = (n - |I \cup J|)!$.*

Proof. If I and J are compatible for shift s then any permutation $\sigma \in \mathbb{S}_n$ with $\sigma(i) = i$ for all $i \in I$, $\sigma(j) = j + s \pmod n$ for all $j \in J$ and complemented by an arbitrary permutation $[n-1] \setminus (I \cup J)$ is in $\Phi^*_{0,s}(I, J)$. Hence the claim follows from the fact that since I, J, and $J + s$ are pairwise disjoint, such permutations always exist. ◀

The following lemma shows that almost all pairs of disjoint sets of size $t \leq O(\log n)$ are compatible.

▶ **Lemma 13.** *Let s be an arbitrary non-zero integer in $[n-1]$. If we choose two disjoint sets $I, J \subseteq [n-1]$ of size t i.u.r., then the probability that I and J are compatible for shift s is at least $\left(1 - \frac{4t}{(n-2t)}\right)^{2t}$. In particular, if $t \leq O(\log n)$, then this probability is at least $1 - O\left(\frac{\log^2 n}{n}\right)$.*

Because of Lemma 13, our goal will be to compute the sizes of sets $\Phi_{0,s}(I, J)$ only for compatible sets I and J. Next, for given disjoint sets I and J compatible for shift s, we will consider all sets $K \subseteq [n-1] \setminus (I \cup J)$ and argue about their contributions to $|\Phi^*_{0,s}(I, J)|$ using the Principle of Inclusion-Exclusion.

5.2.2 Properties of sets $K \subseteq [n-1]$ feasible for I, J, and s

Define $\mathcal{P}_{I,J,0,s}(K) = \{\sigma \in \Phi^*_{0,s}(I,J) : \text{ for every } \ell \in K \text{ it holds that: } \sigma(\ell) \in \{\ell, \ell+s \pmod n\}\}$. While it is difficult to study $\mathcal{P}_{I,J,0,s}(K)$ for all sets $K \subseteq [n-1] \setminus (I \cup J)$, we will want to focus our attention only on subsets with some good properties. We call a *set* $K \subseteq [n-1]$ *feasible for* I, J, *and* s, if I and J are compatible for shift s, $K \cap (K+s) = \emptyset$, and $K \cap (I \cup J \cup (I-s) \cup (J+s)) = \emptyset$.

To justify the definition of feasible sets, we begin with the following simple lemma.

▶ **Lemma 14.** *If $K \subseteq [n-1]$ is feasible for I, J, and s, then $|\mathcal{P}_{I,J,0,s}(K)| = 2^{|K|} \cdot (n - |I \cup J \cup K|)!$.*

Next, similarly to Lemma 13, we argue that almost all suitably small sets are feasible for pairs of disjoint small sets.

▶ **Lemma 15.** *Let s be an arbitrary non-zero integer in $[n-1]$. Let I and J be a pair of compatible sets for s with $|I| = |J| = t$. Let k be a positive integer with $2k \le n - 4t$. If we choose set $K \subseteq [n-1] \setminus (I \cup J)$ of size k i.u.r., then the probability that K is feasible for I, J, and s is at least $\left(1 - \frac{2t+k}{n-2t-k}\right)^k$. In particular, if $t, k \le O(\log n)$, then this probability is at least $1 - O\left(\frac{\log^2 n}{n}\right)$.*

5.2.3 Approximating $|\Phi_{0,s}(I,J)|$ for compatible sets I, J for s

In this section we will complete our analysis to provide a tight bound for the size of $\Phi_{0,s}(I,J)$ for any pair I and J of compatible sets for shift s with $|I| = |J| \le O(\log n)$. Our analysis relies on properties of sets feasible for I, J, and s, as proven in Lemmas 14 and 15.

We begin with the two auxiliary claims. For both, let r be the smallest integer such that $2r \ge \log n$ and let $t = |I| = |J| \le O(\log n)$.

▷ Claim 16.

$$\sum_{k=1}^{2r} (-1)^{k+1} \sum_{\substack{K \subseteq [n-1]\setminus(I\cup J), |K|=k \\ K \text{ feasible for } I,\ J,\text{ and } s}} |\mathcal{P}_{I,J,0,s}(K)| \ge \left(1 - O\left(\frac{\log^2 n}{n}\right)\right) \cdot (n-2t)! \cdot (1 - e^{-2}) \ . \quad (11)$$

▷ Claim 17.

$$\sum_{k=1}^{2r} (-1)^{k+1} \sum_{\substack{K \subseteq [n-1]\setminus(I\cup J), |K|=k \\ K \textit{ not} \text{ feasible for } I,\ J,\text{ and } s}} |\mathcal{P}_{I,J,0,s}(K)| \ge -O\left(\frac{\log^2 n}{n}\right) \cdot (n-2t)! \ .$$

In order to approximate the size of $\Phi_{0,s}(I,J)$ for compatible sets I and J for shift s, let us first notice that

$$\Phi_{0,s}(I,J) = \Phi^*_{0,s}(I,J) \setminus \bigcup_{\ell \in [n-1]\setminus(I\cup J)} \mathcal{P}_{I,J,0,s}(\{\ell\}) \ . \quad (12)$$

Therefore, since we know that $|\Phi^*_{0,s}(I,J)| = (n - (|I| + |J|))!$ by Lemma 12, we only have to approximate $|\bigcup_{\ell \in [n-1]\setminus(I\cup J)} \mathcal{P}_{I,J,0,s}(\{\ell\})|$; we need a good lower bound.

We compute $|\bigcup_{\ell\in[n-1]\setminus(I\cup J)} \mathcal{P}_{I,J,0,s}(\{\ell\})|$ using the Principle of Inclusion-Exclusion,

$$\begin{aligned}
|\bigcup_{\ell\in[n-1]\setminus(I\cup J)} \mathcal{P}_{I,J,0,s}(\{\ell\})| &= \sum_{K\subseteq[n-1]\setminus(I\cup J),K\neq\emptyset} (-1)^{|K|+1}|\bigcap_{\ell\in K} \mathcal{P}_{I,J,0,s}(\{\ell\})| \\
&= \sum_{K\subseteq[n-1]\setminus(I\cup J),K\neq\emptyset} (-1)^{|K|+1}|\mathcal{P}_{I,J,0,s}(K)| \\
&= \sum_{k=1}^{n-(|I|+|J|)} (-1)^{k+1} \sum_{K\subseteq[n-1]\setminus(I\cup J),|K|=k} |\mathcal{P}_{I,J,0,s}(K)| \ .
\end{aligned}$$

We will make further simplifications; since computing $|\mathcal{P}_{I,J,0,s}(K)|$ for arbitrary non-empty sets $K \subseteq [n-1] \setminus (I \cup J)$ is difficult, we restrict our attention only to *small* sets K which are *feasible for I, J, and s*. For that, we will need to show that by restricting only to small sets K feasible for I, J, and s, we will not make too big errors in the calculations.

Let r be the smallest integer such that $2r \geq \log n$. We can use the Bonferroni inequality [3] to obtain the following,

$$\begin{aligned}
|\bigcup_{\ell\in[n-1]\setminus(I\cup J)} \mathcal{P}_{I,J,0,s}(\{\ell\})| &\geq \sum_{k=1}^{2r}(-1)^{k+1} \sum_{K\subseteq[n-1]\setminus(I\cup J),|K|=k} |\mathcal{P}_{I,J,0,s}(K)| \\
&= \sum_{k=1}^{2r}(-1)^{k+1}\left(\sum_{\substack{K\subseteq[n-1]\setminus(I\cup J),|K|=k \\ K \text{ feasible for } I,\ J, \text{ and } s}} |\mathcal{P}_{I,J,0,s}(K)| + \sum_{\substack{K\subseteq[n-1]\setminus(I\cup J),|K|=k \\ K \textit{ not} \text{ feasible for } I,\ J, \text{ and } s}} |\mathcal{P}_{I,J,0,s}(K)| \right) \\
&\geq -O\left(\frac{\log^2 n}{n}\right)(n-2t)! + \left(1 - O\left(\frac{\log^2 n}{n}\right)\right)(n-2t)!\cdot(1-e^{-2}) \\
&= \left(1 - O\left(\frac{\log^2 n}{n}\right)\right)(n-2t)!\cdot(1-e^{-2}) \ , \qquad (13)
\end{aligned}$$

where the last inequality follows from the auxiliary Claims 16 and 17.

If we combine (12) and (13), then we get the following lemma.

▶ **Lemma 18.** *If I and J are compatible for shift s and $|I| = |J| = t = O(\log n)$, then*

$$|\Phi_{0,s}(I,J)| = |\Phi^*_{0,s}(I,J)| - |\bigcup_{\ell\in[n-1]\setminus(I\cup J)} \mathcal{P}_{I,J,0,s}(\{\ell\})| \leq \frac{(n-2t)!}{e^2}\left(1 + O\left(\frac{\log^2 n}{n}\right)\right) \ .$$

Proof. Indeed, by (12), we have

$$|\Phi_{0,s}(I,J)| = |\Phi^*_{0,s}(I,J)| - |\bigcup_{\ell\in[n-1]\setminus(I\cup J)} \mathcal{P}_{I,J,0,s}(\{\ell\})| \ ,$$

by Lemma 12 we get

$$|\Phi^*_{0,s}(I,J)| = (n - (|I| + |J|))! \ ,$$

and by (13) we have

$$|\bigcup_{\ell\in[n-1]\setminus(I\cup J)} \mathcal{P}_{I,J,0,s}(\{\ell\})\ | \geq \left(1 - O\left(\frac{\log^2 n}{n}\right)\right)\cdot(n-2t)!\cdot(1-e^{-2}) \ .$$

Putting these three bounds together yields the promised bound. ◀

5.2.4 Completing the proof of inequality (10)

Now, with Lemma 18 at hand, we are ready to complete our analysis in the following lemma.

▶ **Lemma 19.** *For any $i, j \in [n-1]$, $i \neq j$, and for $t \leq O(\log n)$, we have,*

$$\mathbf{Pr}[S_i = t, S_j = t] \leq \left(1 + O\left(\frac{\log^2 n}{n}\right)\right) \frac{1}{(et!)^2} \ .$$

Proof. Without loss of generality we assume that $i = 0$ and $j \in [n-1] \setminus \{0\}$.

First, let us recall the following

$$\sum_{I,J\subseteq[n-1],|I|=|J|=t,I\cap J=\emptyset} |\mathcal{F}_{I,0} \cap \mathcal{F}_{J,j}| = \sum_{I,J\subseteq[n-1],|I|=|J|=t,I\cap J=\emptyset} |\Phi_{0,j}(I,J)|$$
$$= \sum_{\substack{I,J\subseteq[n-1],|I|=|J|=t,I\cap J=\emptyset \\ I \text{ and } J \textit{ not} \text{ compatible for } j}} |\Phi_{0,j}(I,J)| + \sum_{\substack{I,J\subseteq[n-1],|I|=|J|=t \\ I \text{ and } J \text{ compatible for } j}} |\Phi_{0,j}(I,J)| \ .$$

Next, let us notice that if I and J are *not* compatible for shift j and $I \cap J = \emptyset$, then we clearly have $|\Phi_{0,s}(I,J)| \leq (n-2t)!$ (since once we have fixed $2t$ positions, we can generate at most $(n-2t)!$ distinct n-permutations). Further, by Lemma 18, we know that if I and J are compatible for shift j, then $|\Phi_{0,s}(I,J)| \leq \frac{(n-2t)!}{e^2} \cdot \left(1 + O\left(\frac{\log^2 n}{n}\right)\right)$. Next, we notice that by Lemma 13, we have,

$$|\{I, J \subseteq [n-1] : |I| = |J| = t, I \cap J = \emptyset \text{ and } I, J \textit{ not} \text{ compatible for } j\}|$$
$$= O\left(\frac{\log^2 n}{n}\right) \left|\{I, J \subseteq [n-1] : |I| = |J| = t, I \cap J = \emptyset\}\right| = O\left(\frac{\log^2 n}{n}\right) \binom{n}{t}\binom{n-t}{t} \ .$$

This immediately gives,

$$\sum_{\substack{I,J\subseteq[n-1],|I|=|J|=t,I\cap J=\emptyset \\ I \text{ and } J \textit{ not} \text{ compatible for } j}} |\Phi_{0,j}(I,J)| \leq O\left(\frac{\log^2 n}{n}\right) \binom{n}{t}\binom{n-t}{t}(n-2t)! = O\left(\frac{\log^2 n}{n}\right) \frac{n!}{(t!)^2}$$

and

$$\sum_{\substack{I,J\subseteq[n-1],|I|=|J|=t \\ I \text{ and } J \text{ compatible for } j}} \Phi_{0,j}(I,J)| \leq \binom{n}{t}\binom{n-t}{t} \frac{(n-2t)!}{e^2} \left(1 + O\left(\frac{\log^2 n}{n}\right)\right)$$
$$= \left(1 + O\left(\frac{\log^2 n}{n}\right)\right) \frac{n!}{(et!)^2} \ .$$

Therefore,

$$\sum_{\substack{I,J\subseteq[n-1],I\cap J=\emptyset \\ |I|=|J|=t}} |\mathcal{F}_{I,0} \cap \mathcal{F}_{J,j}| = \sum_{\substack{I,J\subseteq[n-1],|I|=|J|=t,I\cap J=\emptyset \\ I \text{ and } J \textit{ not} \text{ compatible for } j}} |\Phi_{0,j}(I,J)| + \sum_{\substack{I,J\subseteq[n-1],|I|=|J|=t \\ I \text{ and } J \text{ compatible for } j}} |\Phi_{0,j}(I,J)|$$
$$\leq \left(1 + O\left(\frac{\log^2 n}{n}\right)\right) \frac{n!}{(et!)^2} \ .$$

Hence, we can conclude that for $i \neq j$ we have,

$$\mathbf{Pr}[S_i = t, S_j = t] = \frac{1}{n!} \sum_{\substack{I,J\subseteq[n-1],I\cap J=\emptyset \\ |I|=|J|=t}} |\mathcal{F}_{I,i} \cap \mathcal{F}_{J,j}| \leq \left(1 + O\left(\frac{\log^2 n}{n}\right)\right) \cdot \frac{1}{(et!)^2} \ . \quad \blacktriangleleft$$

6 Analysis of the *communication in the locker room* setting

A lower bound for the success probability in the *locker room* problem is provided by a straightforward adaptation of the *shift strategy*: Alice enters her message relaying the most common shift $\mathfrak{h}$ to locker 0, and Bob opens locker 0 and uses Alice's message to check location $(\mathfrak{s}+\mathfrak{h})$ mod n for his card. This strategy ensures a success probability of $\frac{(1+o(1))\log n)}{n\log\log n}$.

As in Sections 2 and 3, we will consider the case when $\mathfrak{s}$ is chosen i.u.r. from $[n-1]$(cf. Section 2.2). In order to obtain an upper bound for the success chance in the *locker room* problem, we shall introduce some intermediate settings, or "protocols". In the *CLR* protocol Alice views the contents of all the lockers, interchanges the contents of two lockers, then Bob is given a number and can open two lockers in search of it (i.e., the *CLR* protocol is the set of rules which govern the *locker room* problem). In the *NH* protocol Alice views the contents of all the lockers, communicates a message of length $\log n$ to Bob, then Bob is given a number and can open one locker in search of it (i.e., the *NH* protocol is the set of rules which govern the *needle in a haystack* game). Moreover, we can append the modifier "-with-r-bits" to NH, which substitutes r for $\log n$ in the above description.

We write $\mathbf{Pr}[\mathcal{V}(\mathcal{P})]$ for the optimal probability of success in protocol $\mathcal{P}$ and $\mathbf{Pr}[\mathcal{V}(\mathbb{C},\mathcal{P})]$ for the probability of success for strategy $\mathbb{C}$ in protocol $\mathcal{P}$. For example, we have already shown that $\mathbf{Pr}[\mathcal{V}(NH)] = \frac{(1+o(1))\log n}{n\log\log n}$.

▶ **Lemma 20.** $\mathbf{Pr}[\mathcal{V}(CLR)] \leq \mathbf{Pr}[\mathcal{V}(\textit{NH-with-}4\log n\textit{-bits})]$.

Proof. We will interpolate between CLR and NH-with-$4\log n$-bits with two other protocols.

In the protocol *CLR0*, Alice views the contents of all the lockers, interchanges the contents of two lockers, then Bob is given a number and can open two lockers in search of it, and he can recognize upon seeing the content of the first locker whether it has been altered by Alice.

In the protocol *CLR1*, Alice views the contents of all the lockers, interchanges the contents of two lockers, leaves these two lockers open with their contents visible to Bob, then Bob is given a number and can open one locker in search of it.

Also, let *Sim* be the strategy in *NH-with-*$4\log n$*-bits* in which Alice uses her message to communicate to Bob the cards whose positions she would exchange, and the positions of these cards, if she encountered the permutation σ while working in the *CLR1* protocol, simulating an optimal strategy $\mathbb{C}$ in *CLR1*. Since this is an ordered quadruple in $[n-1]^4$, it can indeed be communicated in at most $4\log n$ bits.

The proof is in four parts:

(i) $\mathbf{Pr}[\mathcal{V}(CLR)] \leq \mathbf{Pr}[\mathcal{V}(CLR0)]$,
(ii) $\mathbf{Pr}[\mathcal{V}(CLR0)] \leq \mathbf{Pr}[\mathcal{V}(CLR1)] + O(\frac{1}{n})$,
(iii) $\mathbf{Pr}[\mathcal{V}(CLR1)] \leq \mathbf{Pr}[\mathcal{V}(\textit{Sim, NH-with-4logn-bits})]$,
(iv) $\mathbf{Pr}[\mathcal{V}(\textit{Sim, NH-with-4logn-bits})] \leq \mathbf{Pr}[\mathcal{V}(\textit{NH-with-4logn-bits})]$.

(i), (iii), (iv) are straightforward and so we only have to show (ii). Let p_t be the maximum probability that Bob finds his sought number in the t^{th} locker that he opens, $t \in \{1,2\}$.

Firstly, we bound p_1. Suppose that Alice and Bob have settled on a specific strategy. Let $e_{x,w}$ be the probability that σ is such that Alice's transposition sends the locker w to card x. Evidently, $0 \leq e_{x,w} \leq \frac{n-1}{n}$ for all $x, w \in [n-1]$ and $\sum_{x,w\in[n-1]} e_{x,w} \leq 2$.

Having received his number $\mathfrak{s}$, Bob has to open a specific locker, let us say $b = b(\mathfrak{s})$. The probability that Bob happens upon the card $\mathfrak{s}$ in the locker b is at most $e_{\mathfrak{s},b(\mathfrak{s})} + \frac{1}{n}$ (either Alice substitutes the content of $b(\mathfrak{s})$ for $\mathfrak{s}$, or the content of $b(\mathfrak{s})$ is initially $\mathfrak{s}$ and Alice does not interfere). Thus, choosing $\mathfrak{s}$ i.u.r. from $[n-1]$, the probability that Bob finds $\mathfrak{s}$ at his first try is at most $\frac{1}{n}(\sum_{\mathfrak{s},b\in[n-1]} e_{\mathfrak{s},b(\mathfrak{s})} + \frac{1}{n}) < \frac{3}{n} = O(\frac{1}{n})$.

Then, we bound p_2. If Bob opens first one of the lockers whose contents have been altered by Alice, then there is one remaining locker for him to open, and he has at most as much information as in the *CLR0* protocol. Hence, in this case, $p_2 \leq \mathbf{Pr}[\mathcal{V}(CLR0)]$.

Alternatively, Bob first opens one of the lockers whose contents have not been altered by Alice. This requires a more detailed analysis of the *CLR0* protocol.

Alice's choice of a transposition is informed solely by the initial permutation σ of the cards inside the lockers. Hence, there should be a function $a : \mathbb{S}_n \to \binom{[n-1]}{2}$ which directs Alice to a pair of lockers. Then, Bob's choice of a first locker to open is informed only by his sought number. Thus, there should be a function $b : [n-1] \to [n-1]$ which directs Bob to his first locker. Finally, Bob chooses his second locker by considering his sought number and the content of the first locker, so there should be a function $b' : \{0,1\} \times [n-1]^2 \to [n-1]$ which directs Bob to his second locker (the binary factor distinguishes whether Bob's first locker has had its content altered by Alice or not). The strategy which Alice and Bob employ in the *CLR0* protocol can therefore be identified with a triple $[a, b, b']$.

Let $E_{u,v} = a^{-1}(\{u, v\})$ be the event that Alice transposes the contents of the u^{th} and v^{th} lockers, and let $F_w = b^{-1}(w)$ be the event that Bob opens the w^{th} locker first. Let $s(y, w) \subseteq \mathbb{S}_n$ be the permutations which map w to y, and let G_y be the event that the initial content of Bob's first locker is y. Notice that $\mathbf{Pr}[E_{u,v}|F_w \cap G_y] = \frac{|a^{-1}(\{u,v\}) \cap s(y,w)|}{(n-1)!}$, $\mathbf{Pr}[F_w] = \frac{|b^{-1}(w)|}{n}$, and $\mathbf{Pr}[G_y] = \frac{1}{n}$. Then, the probability that Bob finds his sought number in his second attempt given that his first locker was not altered by Alice is

$$p_2 \leq \sum_{\substack{u,v,w,y \in [n-1] \\ u,v,w \text{ distinct}}} \mathbf{Pr}[E_{u,v}|F_w \cap G_y] \cdot \mathbf{Pr}[F_w] \cdot \mathbf{Pr}[G_y] \cdot \mathbf{Pr}[\mathcal{V}(CLR0)|E_{u,v} \cap F_w \cap G_y] \ .$$

Observe that

$$\mathbf{Pr}[\mathcal{V}(CLR0)|E_{u,v} \cap F_w \cap G_y] \leq \frac{(n-2)!}{\left|\left(\mathbb{S}_n \setminus \bigcup_{\ell \in [n-1]} a^{-1}(\{w, \ell\})\right) \cap s(y, w)\right|} + \frac{2}{n} \ .$$

This holds because, barring the $\frac{2}{n}$ probability for Bob's sought card to be in a locker whose content was changed by Alice, Bob is only going to find his sought card in his second locker if the permutation σ maps both w to y and Bob's second locker to his sought card. There are exactly $(n-2)!$ such permutations, which yields the numerator. For the denominator, when Bob opens the locker w and views the card inside, he sees that its content is y and that it has not been touched by Alice, so he knows that σ is a permutation which maps w to y and which does not prompt Alice to transpose y with some other card, and there are exactly $\left|\left(\mathbb{S}_n \setminus \bigcup_{\ell \in [n-1]} a^{-1}(\{w, \ell\})\right) \cap s(y, w)\right|$ such permutations.

Also, note that

$$\bigcup_{\substack{u,v \in [n-1] \\ u,v,w \text{ distinct}}} (a^{-1}(\{u, v\}) \cap s(y, w)) = (\mathbb{S}_n \setminus \bigcup_{\ell \in [n-1]} a^{-1}(\{w, \ell\})) \cap s(y, w) \Rightarrow$$

$$\sum_{\substack{u,v \in [n-1] \\ u,v,w \text{ distinct}}} |a^{-1}(\{u, v\}) \cap s(y, w)| = |(\mathbb{S}_n \setminus \bigcup_{\ell \in [n-1]} a^{-1}(\{w, \ell\})) \cap s(y, w)| \ .$$

Combining the above, we obtain that

$$p_2 \leq \sum_{\substack{u,v,w,y \in [n-1] \\ u,v,w \text{ distinct}}} \frac{1}{n} \cdot \frac{|a^{-1}(\{u, v\}) \cap s(y, w)|}{(n-1)!} \cdot \frac{|b^{-1}(w)|}{n} \cdot \left(\mathbf{Pr}[\mathcal{V}(CLR0)|E_{u,v} \cap F_w \cap G_y] + \frac{2}{n}\right)$$

$$\leq \sum_{w,y \in [n-1]} \frac{1}{n} \cdot \frac{1}{n-1} \cdot \frac{|b^{-1}(w)|}{n} + \frac{2}{n} = \frac{1}{n-1} + \frac{2}{n} \ .$$

Thus, in this case, $p_2 \leq \frac{4}{n}$.

Ultimately, $p_2 \leq \mathbf{Pr}[\mathcal{V}(CLR1)] + \frac{4}{n}$, and hence $\mathbf{Pr}[\mathcal{V}(CLR0)] \leq p_1 + p_2 \leq \mathbf{Pr}[\mathcal{V}(CLR1)] + O(\frac{1}{n})$, concluding the proof. ◀

▶ **Theorem 21.** $\mathbf{Pr}[\mathcal{V}(CLR)] \leq \frac{(4+o(1))\log n}{n \log\log n}$.

Proof. We use Lemma 20 along with the fact that $\mathbf{Pr}[\mathcal{V}(\textit{NH-with-4logn-bits})] \leq \frac{(4+o(1))\log n}{n\log\log n}$, which can be immediately derived from Theorem 22 in Section 7.1 by setting $m = n^4$. ◀

7 Conclusions and generalizations

In this paper we presented a new search problem and provided a comprehensive analysis of its optimal strategy. The core of our analysis is a novel study of properties of random permutations with a given number of fixed points.

There are several natural generalizations of the problem studied in this paper and related questions about properties of random permutations, which we will discuss in the full version of the paper (cf. `https://arxiv.org/abs/2008.11448`). Here we discuss a couple of them.

7.1 Simple generalization: longer message

In the *needle in a haystack* problem, when Alice sends the message $\mathfrak{h}$ to Bob, there is no reason why she must choose a number in $[n-1]$; instead, she could transmit a number $\mathfrak{h} \in [m-1]$ for an arbitrary integer m. One can easily generalize the analysis from Theorems 5 and 6 in this setting for a large range of m.

Let us denote the maximum attainable sum of intensities received from partitioning $\mathbb{S}_n$ to m parts the *m-field* of $\mathbb{S}_n$, and denote it by $F(n,m)$. Fields are simply diagonal m-fields (fields of the form $F(n,n)$).

We have $F(n,1) = n!$ (yielding a success probability of $\frac{1}{n}$, corresponding to not receiving advice) and $F(n,m) = n \cdot n!$ for every $m \geq n!$ (yielding a success probability of 1, corresponding to obtaining full information). For other values of m we can follow the approach used in Theorem 5. First, notice that there is $\ell = \frac{(1+o(1))\log m}{\log\log m}$, such that $\frac{m}{\ell!} = o(1)$. Then, using the techniques from the proof of Theorem 5, we obtain

$$\begin{aligned} F(n,m) &\leq \sum_{i\in[n-1], j\in[m-1]} \mathsf{int}(A_j, i) \leq \sum_{j\in[m-1]} \left(\ell \cdot |A_j| + \sum_{r=\ell+1}^{n} r \cdot D_{n,r}\right) \\ &\leq \sum_{j\in[m-1]} \left(\ell \cdot |A_j| + \frac{(1+o(1))n!}{\ell!}\right) \leq n! \cdot \left(\ell + \frac{(1+o(1))m}{\ell!}\right) \\ &\leq \ell \cdot n! \cdot (1+o(1)) = \frac{(1+o(1))\log m}{\log\log m} \cdot n! \ .r \end{aligned}$$

By (2), this yields the success probability of $\frac{(1+o(1))\log m}{n\log\log m}$, giving the following theorem.

▶ **Theorem 22.** *If Alice can choose a number $\mathfrak{h} \in [m]$, then the maximum attainable success probability is at most $\frac{(1+o(1))\log m}{n\log\log m}$. In particular, if $m = poly(n)$, then the maximum attainable success probability is at most $O\left(\frac{\log n}{n\log\log n}\right)$.*

Observe that Theorem 22 implies that since for the algorithm presented in Theorem 6, that is, one using the shift strategy with hint $\mathfrak{h} \in [n]$, the success probability is already $\Omega\left(\frac{\log n}{n\log\log n}\right)$, the shift strategy is asymptotically optimal to within a constant factor for

any hint $\mathfrak{h}$ polynomial in n. A similar conclusion holds also for the communication in the locker room setting: even if Alice leaves Bob a message by altering the contents of a constant number c of lockers rather than just one, this message is $c \log n$ bits long, and hence the success probability is still at most $O(\frac{\log n}{n \log \log n})$.

Asymptotic results for several other interesting domains of m could be found in a similar way. However, for super-polynomial domains, the upper bound derived in the above manner is far away from the lower bound that we currently can provide in Theorem 6. Determining some properties of the rate of growth of $F(n, m)$ for fixed n would be a good step towards determining its values. With this in mind, we have the following natural conjecture.

▶ **Conjecture 23.** *For any fixed n, the function $f(m) = F(n, m)$ is concave.*

7.2 Optimal strategies

Although we have successfully calculated the maximum field and the maximum success probability for the *needle in a haystack* problem, the problem of determining a characterization of, or at least some major properties for, optimal strategies remains. Indeed, the only optimal strategy that we have explicitly described so far is the shift strategy (which is in fact a set of different strategies, since, for permutations which have several $S_{\mathfrak{h}}$'s of maximum size, there are multiple legitimate options for their class). A natural generalization of shift strategies are *latin strategies*; in these, Alice and Bob decide on a $n \times n$ latin square S, and Alice's message indicates the row of S which coincides with σ at the maximum number of places.

We present a couple of interesting questions concerning the optimal strategies for $\mathbb{S}_n$ in *needle in a haystack*.

▶ **Conjecture 24.** *For every natural number n, there is an optimal strategy for $\mathbb{S}_n$ whose parts all contain exactly $(n-1)!$ permutations.*

▶ **Conjecture 25.** *Optimal strategies are exactly latin strategies.*

References

1 Micah Adler, Soumen Chakrabarti, Michael Mitzenmacher, and Lars Eilstrup Rasmussen. Parallel randomized load balancing. In *Proceedings of the 27th Annual ACM Symposium on Theory of Computing (STOC)*, pages 238–247, 1995.

2 Béla Bollobás. *Random Graphs*. Cambridge University Press, Cambridge, UK, 2nd edition, 2001.

3 Carlo Emilio Bonferroni. Teoria statistica delle classi e calcolo delle probabilità. *Pubblicazioni del R Istituto Superiore di Scienze Economiche e Commerciali di Firenze*, 8:3–62, 1936.

4 Joe P. Buhler. Hat tricks. *The Mathematical Intelligencer*, 24(4):44–49, 2002.

5 Eucene Curtin and Max Warshauer. The locker puzzle. *The Mathematical Intelligencer*, 28(1):28–31, March 2006.

6 Devdatt P. Dubhashi and Desh Ranjan. Balls and bins: A study in negative dependence. *Random Structures and Algorithms*, 13(2):99–124, 1998.

7 Anna Gál and Peter Bro Miltersen. The cell probe complexity of succinct data structures. In *Proceedings of the 30th Annual International Colloquium on Automata, Languages and Programming (ICALP)*, pages 332–344, 2003.

8 Navin Goyal and Michael E. Saks. A parallel search game. *Random Structures and Algorithms*, 27(2):227–234, 2005.

9 Ronald L. Graham, Donald E. Knuth, and Oren Patashnik. *Concrete Mathematics: A Foundation for Computer Science*. Addison-Wesley, Reading, MA, 2nd edition, 1994.

10 Donald E. Knuth. *The Art of Computer Programming: Sorting and Searching*, volume III. Addison-Wesley, Reading, MA, 2nd edition, 1998.

11 Michael Mitzenmacher and Eli Upfal. *Probability and Computing: Randomized and Probabilistic Techniques in Algorithms and Data Analysis*. Cambridge University Press, Cambridge, UK, 2nd edition, 2017.

12 Martin Raab and Angelika Steger. "Balls into bins" – A simple and tight analysis. In *Proceedings of the 2nd International Workshop on Randomization and Approximation Techniques in Computer Science (RANDOM)*, pages 159–170, 1998.

13 John Riordan. *An Introduction to Combinatorial Analysis*. John Wiley & Sons, Inc., New York, NY, 1958.

14 Peter Winkler. Names in boxes puzzle. *College Mathematics Journal*, 37(4):260, 285, 289, September 2006.

15 Peter Winkler. *Mathematical Mind-Benders*. A K Peters, Ltd., Wellesley, MA, 2007.

Improved Approximation Factor for Adaptive Influence Maximization via Simple Greedy Strategies

Gianlorenzo D'Angelo ✉
Gran Sasso Science Institute, L'Aquila, Italy

Debashmita Poddar ✉
Gran Sasso Science Institute, L'Aquila, Italy

Cosimo Vinci ✉
Gran Sasso Science Institute, L'Aquila, Italy

Abstract

In the *adaptive influence maximization problem*, we are given a social network and a budget k, and we iteratively select k nodes, called seeds, in order to maximize the expected number of nodes that are reached by an influence cascade that they generate according to a stochastic model for influence diffusion. The decision on the next seed to select is based on the observed cascade of previously selected seeds. We focus on the *myopic feedback model*, in which we can only observe which neighbors of previously selected seeds have been influenced and on the *independent cascade* model, where each edge is associated with an independent probability of diffusing influence. While adaptive policies are strictly stronger than non-adaptive ones, in which all the seeds are selected beforehand, the latter are much easier to design and implement and they provide good approximation factors if the adaptivity gap, the ratio between the adaptive and the non-adaptive optima, is small. Previous works showed that the adaptivity gap is at most 4, and that simple adaptive or non-adaptive greedy algorithms guarantee an approximation of $\frac{1}{4}\left(1-\frac{1}{e}\right) \approx 0.158$ for the adaptive optimum. This is the best approximation factor known so far for the adaptive influence maximization problem with myopic feedback.

In this paper, we directly analyze the approximation factor of the non-adaptive greedy algorithm, without passing through the adaptivity gap, and show an improved bound of $\frac{1}{2}\left(1-\frac{1}{e}\right) \approx 0.316$. Therefore, the adaptivity gap is at most $\frac{2e}{e-1} \approx 3.164$. To prove these bounds, we introduce a new approach to relate the greedy non-adaptive algorithm to the adaptive optimum. The new approach does not rely on multi-linear extensions or random walks on optimal decision trees, which are commonly used techniques in the field. We believe that it is of independent interest and may be used to analyze other adaptive optimization problems. Finally, we also analyze the adaptive greedy algorithm, and show that guarantees an improved approximation factor of $1-\frac{1}{\sqrt{e}} \approx 0.393$.

2012 ACM Subject Classification Theory of computation → Stochastic approximation; Theory of computation → Online algorithms; Theory of computation → Probabilistic computation; Theory of computation → Graph algorithms analysis

Keywords and phrases Adaptive Optimization, Influence Maximization, Submodular Optimization, Stochastic Optimization

Digital Object Identifier 10.4230/LIPIcs.ICALP.2021.59

Category Track A: Algorithms, Complexity and Games

Related Version *Full Version*: https://arxiv.org/abs/2007.09065

Funding This work has been partially supported by the Italian MIUR PRIN 2017 Project ALGADIMAR "Algorithms, Games, and Digital Markets".

1 Introduction

In the Influence Maximization (IM) problem, we are given a social network, a stochastic model for diffusion of influence over the network, and a budget k, and we ask to find a set of k nodes, called *seeds*, that maximize their *spread of influence*, which is the expected number of nodes reached by a cascade of influence diffusion generated by the seeds according to the given diffusion model. One of the most studied models for influence diffusion is the Independent Cascade (IC), where each edge is associated with an independent probability of transmitting influence from the source node to the tail node. In the IC model the spread of influence is a monotone submodular function of the seed set, therefore a greedy algorithm, which iteratively selects a seed with maximum marginal gain, guarantees a $1 - \frac{1}{e}$ approximation factor for the IM problem [34]. Since its definition by Domingos and Richardson [19, 40] and formalization as an optimization problem by Kempe et al. [33, 34], the IM problem and its variants have been extensively investigated, motivated by applications in viral marketing [13], adoption of technological innovations [24], and outbreak or failure detection [35]. See [10, 36] for surveys on the IM problem.

Recently, Golovin and Krause [25] initiated the study of the IM problem under the framework of adaptive optimization, where, instead of selecting all the seeds at once at the beginning of the process, we can select one seed at a time and observe, to some extent, the portion of the network reached by a new selected seed. The advantage is that the decision on the next seed to choose can be based on the observed spread of previously selected seeds, usually called *feedback*. Two main feedback models have been introduced: in the *full-adoption* feedback the whole spread from each seed can be observed, while, in the *myopic* feedback, one can only observe the direct neighbors of each seed.

Golovin and Krause [25] considered the Independent Cascade model and showed that, under full-adoption feedback, the objective function satisfies the property of *adaptive submodularity* (introduced in the same paper) and therefore an adaptive greedy algorithm achieves a $1 - \frac{1}{e}$ approximation factor for the adaptive IM problem. In their arXiv version, they retracted a claim (appeared in their previous conference version) in which they mistakenly showed that the adaptive submodularity holds even under the myopic feedback model, and this property would have guaranteed that the adaptive greedy is a constant factor approximation algorithm (under the myopic feedback model). Anyway, they conjectured that there exists a constant factor approximation algorithm for the myopic feedback model, which indeed has been found by Peng and Chen [39] who showed that both the adaptive and non-adaptive greedy algorithms guarantee a $\frac{1}{4}\left(1 - \frac{1}{e}\right)$-approximation. In particular, they showed that the adaptivity gap, which is the supremum, over all possible inputs, of the ratio between the spread of an optimal adaptive policy and that of an optimal non-adaptive one, is upper-bounded by 4. By combining this bound with the approximation factor of both the adaptive and non-adaptive greedy algorithms for the non-adaptive problem they obtain a $\frac{1}{4}\left(1 - \frac{1}{e}\right)$-approximation factor. To prove their upper-bound on the adaptivity gap, Peng and Chen use an approach that is inspired by Bradac et al. [6], which in turn is based on the approach introduced by Gupta et al. [29, 30] in the context of stochastic probing. To relate a non-adaptive solution to an optimal adaptive one, they consider the decision tree of an optimal adaptive solution and construct a non-adaptive policy by performing a random root-leaf walk in the tree, according to a probability distribution induced by the tree. Note that computing the non-adaptive policy that guarantees the upper-bound on the adaptivity gap would require to know an optimal decision tree. However, this approach only requires to show the existence of such a non-adaptive policy since it allows us to bound

the adaptivity gap and non-adaptive approximation factor is obtained by combining the non-adaptive approximation factor with the adaptivity gap. In the same paper, Peng and Chen showed that both the adaptive greedy and the non-adaptive greedy algorithms cannot achieve a factor better than $\frac{e^2+1}{(e+1)^2} < 0.607 < 1 - \frac{1}{e}$ of the adaptive optimum, and that the adaptivity gap is at least $\frac{e}{e-1} \approx 1.582$.

In [11], the same authors showed some upper and lower bounds on the adaptivity gap in the case of full-adoption feedback, still under independent cascade, for some particular graph classes. In order to show these bounds, they followed an approach introduced by Asadpour and Nazerzadeh [3] which consists in transforming an adaptive policy into a non-adaptive one by means of multilinear extensions, and constructing a Poisson process to relate the influence spread of the non-adaptive policy to that of the adaptive one. For general graphs, a non-constant upper bound for the adaptivity gap under the full-adoption feedback has been recently shown in [16].

Our Contribution

In this paper, we focus on the myopic model and analyze the approximation factor of the non-adaptive greedy algorithm without passing through the adaptivity gap. We show that the algorithm achieves at least a fraction of $\frac{1}{2}\left(1 - \frac{1}{e}\right) \approx 0.316$ of the adaptive optimum (Theorem 5). By definition, this implies that the adaptivity gap is at most $\frac{2e}{e-1} \approx 3.164$ (Remark 12). For both approximation ratio and adaptivity gap we obtain a substantial improvement with respect to the upper bounds obtained in [39], which are $\frac{1}{4}\left(1 - \frac{1}{e}\right) \approx 0.158$ and 4, respectively.

Non-adaptive policies are strictly weaker than adaptive ones, since the latter can implement the former by simply ignoring any kind of feedback. On the other hand, adaptive policies are difficult to implement as they require to probe suitable seeds and to observe the corresponding feedback, which can be expensive and error-prone. Moreover, they may consist of exponentially-large decision trees that are hard to compute and store. In contrast, non-adaptive policies are easy to design and implement and are independent from the feedback. In particular, the non-adaptive greedy algorithm has been extensively studied and successfully applied in the field of influence maximization. For the non-adaptive setting, several efficient implementation of the greedy algorithm have been devised that allows us to use it in large real-world networks [14, 27, 35, 37, 47, 48]. Our results show that the simple non-adaptive greedy algorithm performs well, even in the adaptive setting where we compare it with the adaptive optimum.

To show our bounds, we introduce a new approach that relate the non-adaptive greedy policy to an optimal adaptive solution. The new approach is not based on multilinear extensions and poisson processes (like, e.g. [3, 7, 9, 11]) neither on random walks on the optimal decision trees (like, e.g. [6, 29, 30, 39]), which are the main tools used so far to relate adaptive and non-adaptive policies, and to bound adaptivity gaps. Previous techniques derive adaptive approximation factors by combining non-adaptive approximation ratios with a bound on the adaptivity gap which is obtained by showing the existence of a "good" non-adaptive policy. However such a policy is hard to compute as it is usually constructed by using an optimal adaptive policy. Our approach, instead, directly analyzes a non-adaptive policy and therefore provides the exact policy that gives the desired adaptivity gap and adaptive approximation factor. We believe that our approach is of independent interest and may be used to bound approximation factors and adaptivity gaps of different adaptive optimization problems.

Our new approach consists in defining a simple randomized non-adaptive policy whose performance is not higher than that guaranteed by the greedy algorithm, and to relate such randomized non-adaptive policy with the optimal adaptive policy. In order to recover good properties of the objective function (like, e.g. submodularity) that usually guarantee good approximations when adopting greedy strategies, we introduce an artificial diffusion process in which each seed has two chances to influence its neighbours. A similar process was introduced by Peng and Chen [39], who consider a diffusion model in which the seeds appear in multiple copies of the influence graphs, so that, roughly speaking, each node has several chances to influence the neighbours, and the main machinery they consider to relate optimal adaptive strategies with optimal non-adaptive ones is that in [6]. Our direct and more refined analysis of the non-adaptive greedy algorithm improves at the same time both the approximation ratio and the adaptivity gap.

To illustrate our approach, in Section 2 we first apply our machinery to the simpler setting of adaptive monotone submodular maximization under cardinality constraint. As observed by Asadpour and Nazerzadeh [3], a constant factor approximation of the non-adaptive greedy policy applied to such setting can be obtained by combining the approximation ratio over the non-adaptive optimum and the adaptivity gap (which is equal to $\left(1-\frac{1}{e}\right)$ [3]); this leads to an approximation guarantee of $\left(1-\frac{1}{e}\right)^2 \approx 0.399$. We give a more refined analysis of the non-adaptive greedy policy and we show that its approximation ratio is at least $\frac{1}{2}\left(1-\frac{1}{e^2}\right) \approx 0.432$. Asadpour and Nazerzadeh [3] also showed that a so-called continuous greedy policy [7, 9] achieves an approximation ratio of $1-\frac{1}{e}-\epsilon$ (in polynomial time w.r.t. $\frac{1}{\epsilon}$). Since the continuous greedy policy is a non-adaptive policy, this bound is strictly better than ours. However, the non-adaptive greedy policy is simpler and deterministic, while the continuous greedy policy is randomized and more complex.

Finally, in Section 5, we analyze the adaptive version of the greedy algorithm applied to the adaptive influence maximization problem; by resorting again to an artificial diffusion process, we show that such adaptive algorithm guarantees an approximation ratio of $1-\frac{1}{\sqrt{e}} \approx 0.393$; thus, we further improve the upper bound shown for the non-adaptive greedy algorithm, and we also give a more refined analysis of the adaptive greedy algorithm than that of Peng and Chen [39], who showed a $\frac{1}{4}\left(1-\frac{1}{e}\right) \approx 0.158$ approximation factor.

Due to the lack of space, some proofs are deferred to the full version of the paper [15].

Related Work

Adaptive Influence Maximization. The adaptive influence maximization problem under the independent cascade model has been studied by [11, 12, 31, 39, 45, 46, 49, 51, 52, 53]. These include studies on several classes of graphs and different feedback models.

The most studied feedback model is the full-adoption feedback, in which the entire influence spread generated by each selected node is observed. Golovin and Krause [25] show that the full-adoption feedback model satisfies the adaptive submodularity property and, by exploiting such property, provide a $(1-1/e)$ approximation adaptive algorithm for the problem of finding the best adaptive policy. Chen and Peng [11] study the adaptivity gap in the full-adoption feedback model under certain restrictions on the graph topology. In particular, they show that the adaptivity gap of in-arborescence (resp. out-arborescence) graphs belongs to $[e/(e-1), 2e/(e-1)]$ (resp. $[e/(e-1), 2]$). For general graphs with n nodes (resp. in-arborescence graphs), an upper bound of $O(n^{1/3})$ (resp. $2e^2/(e^2-1)$) for the adaptivity gap under the full-adoption feedback has been recently shown in [16].

Golovin and Krause [25] conjectured that the influence maximization problem under the myopic feedback model admits a constant approximation algorithm and a constant adaptivity gap, despite the adaptive submodularity does not hold under such feedback model. Since

then, several studies have been conducted on the myopic feedback model. Some recent works include that of Salha et al. [42], in which they consider a modified version of the independent cascade model which gives multiple chances to the seeds to activate their neighbours, and consider a different utility function which needs to maximized. They demonstrate that the myopic feedback model is adaptive submodular under such modified diffusion model, and provide an adaptive greedy policy that achieves a $1 - 1/e$ approximation ratio for the problem of finding the best adaptive policy. The work of Peng and Chen [39] is the first one that provides a constant upper bound on the adaptivity gap under the myopic feedback model. They introduce a policy in which each seed appears in multiple copies of the original graph; furthermore, this hybrid policy connects the adaptive and the non-adaptive policies via a machinery used by [29, 30, 6] in the context of stochastic probing. They show that the adaptivity gap is in $[e/(e-1), 4]$, and the upper bound is turned into $(1 - 1/e)/4$ approximation algorithm.

Other diffusion and feedback models have been also studied, e.g., the multi-round diffusion model [45], the general feedback model [49], and the partial feedback model [53]. Han et al. [31] conduct a study on the batch selection of seeds at each step of the diffusion process. Tong et al. [50] introduce the dynamic independent cascade model, which captures the dynamic nature of real-world social networks. Finally, Singer et al. [5, 41, 43, 44], in their line of research on adaptivity gaps, studied a two-stage process called adaptive seeding, that exhibits some similarities with the influence maximization problem under the myopic feedback model.

Other Topics in Adaptive Optimization. Beyond influence maximization problems, the adaptive optimization and the adaptivity gap have been generally studied for many other stochastic settings [1, 2, 3, 4, 6, 8, 17, 18, 20, 21, 22, 23, 25, 26, 28, 29, 30, 32, 38].

A general adaptive optimization framework deals with the fact that an item will reveal its actual state only when it has been irrevocably included in the final solution, and the main goal is to optimize an objective function under such uncertainty. Stochastic variants of packing integer programs, 0/1 knapsack problems, and covering problems, have been studied under the perspective of adaptive optimization in [17, 18, 23], respectively.

Asadpour et al. [3, 4] study the adaptivity gap of the stochastic submodular maximization problem under a matroid constraint, in which the goal is to select a subset of items satisfying a matroid constraint, that maximizes the value of a monotone submodular value function defined on the random states of the selected items. In [3], the authors consider an adaptive greedy policy (often denoted as myopic policy) to approximate the optimal value of the best adaptive policy, and they show that it achieves a $1/2$ approximation ratio under general matroid constraint, and a $1 - \frac{1}{e}$ approximation ratio under cardinality constraint. Interesting variants or extensions of the above optimization problem have been considered in [6, 29, 30].

Chan and Farias [8] study the efficiency of adaptive greedy policies applied to a general class of stochastic optimization problems, called stochastic depletion problems, in which the adaptive policy, at each step, chooses an action that generates a reward and depletes some of the items; they show that, under certain structural properties, a simple adaptive greedy policy guarantees a constant factor approximation of the best adaptive policy. Hellerstein et al. [32] use an optimal decision tree to build a connection between the adaptive and the non-adaptive policies, and show that the adaptivity gap of stochastic submodular maximization under cardinality constraint is $1 - \frac{1}{e^\tau}$, where τ is the minimum value of the probability that an item is in some state.

Organization of the Paper

In the next section we introduce our new approach by applying it to a simpler setting. In Section 3 we introduce the adaptive influence maximization problem along with the necessary notation and definitions. In Section 4 we give the main results of the paper, that is the improved approximation factor for the non-adaptive greedy algorithm and the upper bound on the adaptivity gap for adaptive influence maximization. In Section 5 we show an improved approximation ratio for the adaptive greedy algorithm. In Section 6 we outline possible research directions.

2 Overview of the Approach: Stochastic Submodular Maximization

In this section, we illustrate our machinery by applying part of it to the problem of maximizing a stochastic submodular set function under cardinality constraints [3].

For two integers h and k, $h \leq k$, let $[k]_h := \{h, h+1, \ldots, k\}$ and $[k] := [k]_1$. A function $f : \mathbb{R}^n_{\geq 0} \to \mathbb{R}_{\geq 0}$ is *a monotone submodular value function* if, for any vectors $x, y \in \mathbb{R}_{\geq 0}$, we get $f(x \vee y) + f(x \wedge y) \leq f(x) + f(y)$, where $x \wedge y$ denotes the componentwise minimum and $x \vee y$ denotes the componentwise maximum.

Let $[n]$ be a finite set of n items, and let $\boldsymbol{\theta} = (\boldsymbol{\theta}_1, \ldots, \boldsymbol{\theta}_n)$ be a vector of n real, non-negative, and independent *state* random variables, where each $\boldsymbol{\theta}_i$ returns the state $\theta_i \in \mathbb{R}_{\geq 0}$ associated to each item i following a certain probability distribution. Let $f : \mathbb{R}^n_{\geq 0} \to \mathbb{R}_{\geq 0}$ be the *objective function*, that is a monotone submodular value function. For any $S \subseteq [n]$, let $\boldsymbol{\theta}(S) := (\boldsymbol{\theta}_1(S), \ldots, \boldsymbol{\theta}_n(S))$ be the *partial state* random variable, that is a random vector defined as $\boldsymbol{\theta}_i(S) = \boldsymbol{\theta}_i$ if $i \in S$, $\boldsymbol{\theta}_i(S) = 0$ otherwise. With a little abuse of notation, we assume that vector $\boldsymbol{\theta}(S)$ gives also information on the set S which $\boldsymbol{\theta}(S)$ is based on.

For a given integer $k \geq 0$, we aim at selecting a subset $S \subseteq [n]$ subject to cardinality constraint $|S| = k$, that maximizes the expected value $\mathbb{E}_{\boldsymbol{\theta}}[f(\boldsymbol{\theta}(S))]$. To guarantee a (possibly) better solution we can resort to an *adaptive policy*, that, at each step, observes the partial state $\xi \sim \boldsymbol{\theta}(U)$, where U denotes the set of items previously selected, and selects another further item $\pi(\xi) \in [n] \setminus U$; after k iterations, the policy returns a set $U_{\boldsymbol{\theta},k}(\pi) \subseteq [n]$ with $|U_{\boldsymbol{\theta},k}(\pi)| = k$, which is a random set depending on the state of $\boldsymbol{\theta}$. The *stochastic monotone submodular maximization problem* (SMSM) takes as input a set of items $[n]$, a random vector $\boldsymbol{\theta}$, a monotone submodular value function f, and an integer $k \in [n]$, and asks to find an adaptive policy π that maximizes the expected value $\mathbb{E}_{\boldsymbol{\theta}}[f(\boldsymbol{\theta}(U_{\boldsymbol{\theta},k}(\pi)))]$.

In general, computing an optimal adaptive strategy is a computationally hard problem. Furthermore, in many contexts it is difficult to implement adaptive strategies, and non-adaptive strategies (in which the solution is chosen without observing the states of the random variables) is a more feasible choice. Asadpour and Nazerzadeh [3] show that a non-adaptive randomized continuous greedy algorithm guarantees a $\left(1 - \frac{1}{e} - \epsilon\right)$ approximation for the SMSM problem (in polynomial time w.r.t. $1/\epsilon$); however, the proposed approach resorts to a quite sophisticated randomized algorithm. They also consider a simpler and deterministic *non-adaptive greedy algorithm* as approximation algorithm, that starts from an empty set $S := \emptyset$ and, at each iteration $t \in [k]$, adds in S the item $i \in [n] \setminus S$ maximizing $\mathbb{E}_{\boldsymbol{\theta}}[f(\boldsymbol{\theta}(S \cup \{i\}))]$. They show that the greedy algorithm guarantees an approximation factor of $\frac{1}{2}\left(1 - \frac{1}{e}\right) \approx 0.316$ if the chosen subsets are subject to a matroid constraint, that can be reduced to $\left(1 - \frac{1}{e}\right)^2 \approx 0.399$ when considering a cardinality constraint $|S| \leq k$ only (i.e., uniform matroid constraint).

In the following theorem, we give a better analysis of the greedy algorithm under cardinality constraints, and we show that the approximation factor increases to $\frac{1}{2}\left(1 - \frac{1}{e^2}\right) \approx 0.432$.

▶ **Theorem 1.** *The non-adaptive greedy algorithm is a* $\frac{1}{2}\left(1-\frac{1}{e^2}\right)$ *approximation algorithm for the SMSM problem.*

For the proof of Theorem 1 we relate the non-adaptive greedy solution with the optimal adaptive solution. We assume w.l.o.g that $k \geq 2$, otherwise the approximation ratio is equal to 1. For any $t \in [k]_0$, let S_t be the set of t items computed by the greedy algorithm at iteration t (where $S_0 := \emptyset$). Let π be an optimal adaptive policy, and let $x = (x_1, \ldots, x_n) \in [0,1]^n$ be the vector such that x_i is the probability that node $i \in [n]$ is selected by policy π. Let $OPT_A(k)$ denote the value of policy π (i.e., the optimal value of the problem). Given $i \in [n]$, let $\boldsymbol{e}^i := (\boldsymbol{e}^i_1, \ldots, \boldsymbol{e}^i_n)$ be a random vector where $\boldsymbol{e}^i_j$ has the same distribution of $\boldsymbol{\theta}_i$ if $i = j$, and $\boldsymbol{e}^i_j = 0$ otherwise; given a partial state ξ, let $\Delta(i|\xi) := \mathbb{E}_{\boldsymbol{e}^i}[f(\xi \vee \boldsymbol{e}^i) - f(\xi)]$, i.e., the expected increment of the objective function when adding an item i under partial state ξ.

For any $t \in [k-1]_0$, as intermediate step of our analysis, we consider a *randomized non-adaptive policy* Rand_t that, starting from the greedy solution S_t, computes a random set $S_{\boldsymbol{\rho},t} := S_t \cup \{\boldsymbol{\rho}\}$, where $\boldsymbol{\rho} \in [n]$ is a random item such that $\mathbb{P}[\boldsymbol{\rho} = i] = x_i/k$ for any $i \in [n]$ and selected independently from any other event. Observe that the above random variable is well-defined, as $\sum_{v \in V}(x_v/k) = k/k = 1$; furthermore, we observe that the expected value of f under Rand_t is $\mathbb{E}_{\boldsymbol{\theta},\boldsymbol{\rho}}[f(\boldsymbol{\theta}(S_{\boldsymbol{\rho},t})]$. Furthermore, for any $t \in [k-1]_0$, we consider a *hybrid adaptive policy* Hyb_t, that first runs the adaptive policy π, and then merges the items of S_t with the items of $U_{\hat{\boldsymbol{\theta}},k}(\pi)$ selected by π, where $\hat{\boldsymbol{\theta}}$ is a random state that follows the same distribution of $\boldsymbol{\theta}$ but is independent from $\boldsymbol{\theta}$; finally, the expected value of f under Hyb_t is defined as $\mathbb{E}_{\boldsymbol{\theta},\hat{\boldsymbol{\theta}}}[f(\boldsymbol{\theta}(S_t) \vee \hat{\boldsymbol{\theta}}(U_{\hat{\boldsymbol{\theta}},k}(\pi) \cup S_t))]$. A similar hybrid adaptive policy has been also considered by Asadpour and Nazerzadeh [3], but in place of the randomized non-adaptive policy considered in our work, they resort to a non-adaptive strategy defined by a Poisson process based on the multilinear extension of the expected value function $\mathbb{E}_{\boldsymbol{\theta}}[f(*)]$. Before showing the theorem, we give some preliminary lemmas, which relate the randomized non-adaptive policy with the hybrid adaptive policy.

▶ **Lemma 2.** *We have that* $\mathbb{E}_{\boldsymbol{\theta},\boldsymbol{\rho}}[f(\boldsymbol{\theta}(S \cup \{\boldsymbol{\rho}\}))] - \mathbb{E}_{\boldsymbol{\theta}}[f(\boldsymbol{\theta}(S))] = \sum_{i \in [n]\setminus S} x_i \cdot \mathbb{E}_{\boldsymbol{\theta}}[\Delta(i|\boldsymbol{\theta}(S))]$ *for any* $S \subseteq [n]$.

▶ **Lemma 3.** *For any* $S \subseteq [n]$, *we have* $\mathbb{E}_{\boldsymbol{\theta},\hat{\boldsymbol{\theta}}}[f(\boldsymbol{\theta}(S) \vee \hat{\boldsymbol{\theta}}(U_{\hat{\boldsymbol{\theta}},k}(\pi) \cup S))] \leq \mathbb{E}_{\boldsymbol{\theta},\hat{\boldsymbol{\theta}}}[f(\boldsymbol{\theta}(S) \vee \hat{\boldsymbol{\theta}}(S)] + \sum_{i \in [n]\setminus S} x_i \cdot \mathbb{E}_{\boldsymbol{\theta}}[\Delta(i|\boldsymbol{\theta}(S))]$.

▶ **Lemma 4.** *We have that* $OPT_A(k) \leq \mathbb{E}_{\boldsymbol{\theta},\hat{\boldsymbol{\theta}}}[f(\boldsymbol{\theta}(S) \vee \hat{\boldsymbol{\theta}}(U_{\hat{\boldsymbol{\theta}},k}(\pi) \cup S))]$ *for any* $S \subseteq [n]$.

Armed with the above lemmas, we can prove Theorem 1.

Proof of Theorem 1. For any $t \in [k]_0$, let $GR_N(t) := \mathbb{E}_{\boldsymbol{\theta}}[f(\boldsymbol{\theta}(S_t))]$ denote the expected value of f obtained at the t-th iteration of the non-adaptive greedy algorithm. We have that

$$GR_N(t+1) - GR_N(t) = \max_{v \in V}\left[\mathbb{E}_{\boldsymbol{\theta}}[f(\boldsymbol{\theta}(\{v\} \cup S_t))]\right] - \mathbb{E}_{\boldsymbol{\theta}}[f(\boldsymbol{\theta}(S_t))] \tag{1}$$

$$\geq \overbrace{\mathbb{E}_{\boldsymbol{\theta},\boldsymbol{\rho}}[f(\boldsymbol{\theta}(S_t \cup \{\boldsymbol{\rho}\}))]}^{\text{Exp. value of } \mathsf{Rand}_t} - \mathbb{E}_{\boldsymbol{\theta}}[f(\boldsymbol{\theta}(S_t))]$$

$$\geq \frac{1}{k} \cdot \sum_{i \in [n]\setminus S_t} x_i \cdot \mathbb{E}_{\boldsymbol{\theta}}[\Delta(i|\boldsymbol{\theta}(S_t))] \tag{2}$$

$$\geq \frac{1}{k} \cdot \left(\overbrace{\mathbb{E}_{\boldsymbol{\theta},\hat{\boldsymbol{\theta}}}[f(\boldsymbol{\theta}(S_t) \vee \hat{\boldsymbol{\theta}}(U_{\hat{\boldsymbol{\theta}},k}(\pi) \cup S_t))]}^{\text{Exp. value of } \mathsf{Hyb}_t} - \mathbb{E}_{\boldsymbol{\theta},\hat{\boldsymbol{\theta}}}[f(\boldsymbol{\theta}(S_t) \vee \hat{\boldsymbol{\theta}}(S_t))] \right) \tag{3}$$

$$\geq \frac{1}{k} \cdot \Big(OPT_A(k) - \mathbb{E}_{\boldsymbol{\theta},\hat{\boldsymbol{\theta}}}[f(\boldsymbol{\theta}(S_t) \vee \hat{\boldsymbol{\theta}}(S_t))] \Big) \tag{4}$$

$$\geq \frac{1}{k} \cdot OPT_A(k) - \frac{1}{k} \cdot \left(\mathbb{E}_{\boldsymbol{\theta},\hat{\boldsymbol{\theta}}}[f(\boldsymbol{\theta}(S_t)) + f(\hat{\boldsymbol{\theta}}(S_t))]\right) \tag{5}$$

$$\geq \frac{1}{k} \cdot OPT_A(k) - \frac{1}{k} \cdot \left(\mathbb{E}_{\boldsymbol{\theta}}[f(\boldsymbol{\theta}(S_t))] + \mathbb{E}_{\hat{\boldsymbol{\theta}}}[f(\hat{\boldsymbol{\theta}}(S_t))]\right) = \frac{1}{k} \cdot OPT_A(k) - \frac{2}{k} \cdot \mathbb{E}_{\boldsymbol{\theta}}[f(\boldsymbol{\theta}(S_t)]$$

$$= \frac{1}{k} \cdot OPT_A(k) - \frac{2}{k} \cdot GR_N(t), \tag{6}$$

where (1) comes from the fact that the greedy strategy, at iteration $t+1$, adds to S_t the item i guaranteeing the best expected value of f, (2) comes from Lemma 2, (3) comes from Lemma 3, (4) comes from Lemma 4, and (5) holds since f is a monotone submodular value function (as $f(\boldsymbol{\theta}(S_t)\vee\hat{\boldsymbol{\theta}}(S_t)) \leq f(\boldsymbol{\theta}(S_t)\vee\hat{\boldsymbol{\theta}}(S_t))+f(\boldsymbol{\theta}(S_t)\wedge\hat{\boldsymbol{\theta}}(S_t)) \leq f(\boldsymbol{\theta}(S_t))+f(\hat{\boldsymbol{\theta}}(S_t))$). Thus, by (6) and some manipulations, we get $GR_N(t+1) \geq \frac{1}{k}\cdot OPT_A(k)+\left(1-\frac{2}{k}\right)\cdot GR_N(t)$ for any $t \in [k-1]_0$. By applying iteratively the above inequality, we get $GR_N(k) \geq \frac{1}{k}\cdot\sum_{t=0}^{k-1}\left(1-\frac{2}{k}\right)^t\cdot OPT_A(k) = \frac{1}{2}\left(1-\left(1-\frac{2}{k}\right)^k\right)\cdot OPT_A(k)$, that leads to $\frac{GR_N(k)}{OPT_A(k)} \geq \frac{1}{2}\left(1-\left(1-\frac{2}{k}\right)^k\right) \geq \frac{1}{2}\left(1-\frac{1}{e^2}\right)$, and this shows the claim. ◀

3 Adaptive Influence Maximization under the Myopic Feedback Model: Preliminaries

Independent Cascade Model. In the *independent cascade model* (IC), we have an *influence graph* $G = (V = [n], E, (p_{uv})_{(u,v)\in E})$, where edges are directed and $p_{uv} \in [0,1]$ is an *activation probability* associated to each edge $(u,v) \in E$. Given a set of *seeds* $S \subseteq V$ which are initially *active*, the diffusion process in the IC model is defined in $t \geq 0$ discrete steps as follows: (i) let A_t be the set of active nodes which are activated at each step $t \geq 0$; (ii) $A_0 := S$; (iii) given a step $t \geq 0$, for any edge (u,v) such that $u \in A_t$, node u can activate node v with probability p_{uv} independently from any other node, and, in case of success, v is included in A_{t+1}; (iv) the diffusion process ends at a step $r \geq 0$ such that $A_r = \emptyset$, i.e., no node can be activated at all. The size of $\bigcup_{t\leq r} A_t$, i.e. the number of nodes activated/reached by the diffusion process, is the *influence spread*.

The above diffusion process can be equivalently defined as follows. The *live-edge graph* $\boldsymbol{L} = (V, \boldsymbol{L}(E))$ is a random graph made from G, such that each edge $(u,v) \in E$ is included in $\boldsymbol{L}(E)$ with probability p_{uv}, independently from the other edges, i.e., $\mathbb{P}[\boldsymbol{L} = L] = \prod_{(u,v)\in L} p_{uv} \prod_{(u,v)\in E\setminus L}(1-p_{uv})$. With a little abuse of notation, we may denote $\boldsymbol{L}(E)$ with $\boldsymbol{L}$. Given $L \subseteq E$, let $R_L(S) := \{v \in V :$ there exists a path from u to v in L for some $u \in S\}$, i.e., the set of nodes reached by nodes in S in the graph L. Informally, if S is the set of seeds, and L is a realisation of the live-edge graph, $R_L(S)$ equivalently denotes the set of nodes which are reached/activated by the above diffusion process. Let $\sigma_L(S) := |R_L(S)|$ denote the *influence spread* generated by the set of seeds S if the realised live-edge graph is L, and let $\sigma(S) := \mathbb{E}_{\boldsymbol{L}}[\sigma_{\boldsymbol{L}}(S)]$ be the *expected influence spread* generated by S.

Non-adaptive Influence Maximization. The *non-adaptive influence maximization problem under the IC model* is the computational problem that, given an influence graph G and an integer $k \geq 1$, asks to find a set of seeds $S \subseteq V$ with $|S| \leq k$ such that $\sigma(S)$ is maximized. Without loss of generality, we assume that $k \in [n]$ and, since the objective function is monotone, $|S| = k$ for any solution S.

Kempe et al. [33, 34] showed that function σ is monotone and submodular, therefore the following greedy algorithm achieves a $1-\frac{1}{e}$ approximation factor: (i) start with an empty set of seeds $S := \emptyset$; (ii) at each iteration $t \in [k]$, add to S the node v that maximizes the expected influence spread $\sigma(S \cup \{v\})$. Note that the greedy algorithm requires at each

iteration to compute the value of function σ, for some set of seeds and this has been shown to be computationally intractable as it is $\#P$-hard [13]. However, standard Chernoff bounds allows us estimate the value of σ through a polynomial number of Monte-Carlo simulations by introducing an arbitrarily small additive error $\epsilon > 0$, which depends on the number of simulations [34]. In the reminder of the paper we will omit the additional term ϵ to avoid unnecessary complicated formulas. We will refer to this algorithm as the *non-adaptive greedy algorithm*.

Adaptive Influence Maximization. Differently from the non-adaptive setting, in which all the seeds are selected at the beginning, an *adaptive policy* activates the seeds sequentially in k steps, one seed at each step, and the decision on the next seed to select is based on the feedback resulting from the observed spread of previously selected nodes. The feedback model considered in this work is *myopic*: when a node is selected, the adaptive policy observes the state of its neighbours.

An adaptive policy under the myopic feedback model is formally defined as follows. Given $L \subseteq E$, the *realisation* $\phi_L : V \to 2^V$ associated to L assigns to each node $v \in V$ the value $\{z \in V : (v, z) \in L\} \cup \{v\}$, i.e., the set containing v and the neighbours activated by seed v when $\boldsymbol{L} = L$. Let Φ denote the *random realisation*, i.e., the random variable such that $\mathbb{P}[\Phi = \phi_L] = \mathbb{P}[\boldsymbol{L} = L]$ for any $L \subseteq E$. Given a set $S \subseteq V$, a *partial realisation* $\psi : S \to 2^V$ is the restriction to S of the domain of some realisation, i.e., there exists $L \subseteq E$ such that $\psi(v) = \phi_L(v)$ for any $v \in S$. Given a partial realisation $\psi : S \to 2^V$, let $dom(\psi) := S$, i.e., $dom(\psi)$ is the domain of partial realisation ψ, and let $Im(\psi) := \bigcup_{v \in dom(\psi)} \psi(v)$. A partial realisation ψ' is a *sub-realisation* of a partial realisation ψ (or, equivalently, $\psi' \subseteq \psi$), if $dom(\psi') \subseteq dom(\psi)$ and $\psi'(v) = \psi(v)$ for any $v \in dom(\psi')$. We observe that any partial realisation ψ can be equivalently represented as $\{(v, \phi_L(v)) : v \in dom(\psi)\}$ for some $L \subseteq E$.

An adaptive policy π takes as input a partial realisation ψ and, either returns a node $\pi(\psi) \in V$ and activates it as seed, or interrupts the activation of new seeds, e.g., by returning a string $\pi(\psi) := STOP$. In particular, an adaptive policy π can be run as follows: (i) start from an empty realisation $\psi := \emptyset$; (ii) if $\pi(\psi) \neq STOP$ set $\psi \leftarrow \psi \cup \{(v, \phi_L(v))\}$ and repeat (ii) until $\pi(\psi) = STOP$; (iii) at the end, return $\psi_\pi := \psi$. Let Ψ_π be the random partial realisation returned by the execution of policy π. The *expected influence spread* of an adaptive policy π is defined as $\sigma(\pi) := \mathbb{E}_{\boldsymbol{L}}[\sigma_{\boldsymbol{L}}(dom(\Psi_\pi))]$, i.e., it is the expected value of the number of nodes reached by the diffusion process at the end of the execution of policy π. We say that $|\pi| = k$ if policy π always return a partial realisation ψ_π with $|dom(\psi_\pi)| = k$. The *adaptive influence maximization problem (under IC and myopic feedback)* is the computational problem that, given an influence graph G and $k \in [n]$, asks to find an adaptive policy π subject to $|\pi| = k$ that maximizes the expected influence spread $\sigma(\pi)$.

Adaptivity gap. Given an influence graph G and an integer $k \geq 1$, let $OPT_N(G, k)$ (resp. $OPT_A(G, k)$) denote the optimal value of the non-adaptive (resp. adaptive) influence maximization problem with input G and k. Given an integer $k \in [n]$, the *k-adaptivity gap* of G is defined as $AG(G, k) := \frac{OPT_A(G,k)}{OPT_N(G,k)}$, and measures how much an adaptive policy outperforms a non-adaptive solution for the influence maximization problem applied to influence graph G, when the maximum number of seeds is k. The *adaptivity gap* of G is defined as $AG(G) := \sup_{k \in [n]} AG(G, k)$. We observe that for $k = 1$ the k-adaptivity gap and the approximation factors are trivially equal to 1, thus we omit such case in the following.

4 The Efficiency of the Non-adaptive Greedy Algorithm

In this section, we show that a simple non-adaptive algorithm guarantees an approximation ratio of $\frac{1}{2}\left(1-\frac{1}{e}\right) \approx 0.316$ for the adaptive influence maximization problem, thus improving the approximation ratio of $\frac{1}{4}\left(1-\frac{1}{e}\right) \approx 0.158$ given in [39]. The algorithm provided by [39] is the usual *non-adaptive greedy algorithm* given in [34] and reported in the previous section. We observe that such algorithm is non-adaptive, i.e., despite it is used for adaptive optimization, does not resort to the use of any adaptive policy and all the seeds are selected without observing any partial realisation.

▶ **Theorem 5.** *Given an influence graph G with n nodes and $k \in [n]_2$, the non-adaptive greedy algorithm is a $\frac{1}{2}\left(1-\left(1-\frac{1}{k}\right)^k\right) \geq \frac{1}{2}\left(1-\frac{1}{e}\right)$ approximation algorithm for the adaptive influence maximization problem (under IC and myopic feedback) applied to (G,k).*

In the proof of Theorem 5 (see Subsection 4.4) we relate the expected influence spread coming from the non-adaptive greedy algorithm with that of the optimal adaptive policy. We first need some notation and preliminary results. Let $G = (V = [n], E, (p_{uv})_{(u,v)\in E})$ be an influence graph, and let $k \in [n]_2$. For any $t \in [k]_0$, let S_t denote the set of the first t seeds selected by the greedy algorithm, so that $\sigma(S_k)$ is the expected influence spread of the solution returned by the algorithm. Let π be an optimal adaptive policy, and let $x = (x_1, \ldots, x_n)$ be the vector such that x_i is the probability that node i is selected by π. As in Section 2, we resort again to a randomized non-adaptive policy to relate the expected influence spread of the greedy algorithm with that of the adaptive policy.

Randomized Non-adaptive Policy. For any $t \in [k-1]_0$, as intermediate step of our analysis, we consider again the randomized non-adaptive policy Rand_t defined in Section 2: starting from the greedy solution S_t, we compute a random set $S_{\boldsymbol{\rho},t} := S_t \cup \{\boldsymbol{\rho}\}$, where $\boldsymbol{\rho} \in [n]$ is a random item such that $\mathbb{P}[\boldsymbol{\rho} = i] = x_i/k$ for any $i \in [n]$ and selected independently from any other event. We observe that the expected value of f under Rand_t is $\mathbb{E}_{\boldsymbol{\rho}}[\sigma(S_{\boldsymbol{\rho},t})]$.

As a support of our analysis, we also define a new diffusion model and a hybrid adaptive policy. In particular, the new diffusion model will be used to recover certain properties connected with the submodularity that, as in Section 2, will allow us to relate the expected influence spread of the randomized non-adaptive policy with that of the hybrid adaptive policy. Furthermore, following again the approach of Section 2, the hybrid adaptive policy is obtained by combining the greedy (non-adaptive) solution and the optimal adaptive one, and it will be used in our analysis to get an upper bound on the optimal adaptive spread.

2-level Diffusion Model. In the 2-level diffusion model each selected seed $u \in V$ has two chances to influence its neighbours, and all the non-seeds have one chance only, i.e., the activation probability for all the edges (u,v) is $1-(1-p_{u,v})^2$ if u is a seed, and $p_{u,v}$ otherwise. More formally, let $\hat{\boldsymbol{L}}$ be a live-edge graph distributed as $\boldsymbol{L}$ and independent from $\boldsymbol{L}$. Given a set of seeds S, the *2-level live-edge graph* can be defined as $\boldsymbol{L}^2(S) := \boldsymbol{L} \cup \{\hat{\boldsymbol{L}} \cap \{(u,v) \in E : u \in S\}\}$. Given a set of nodes S, let $\sigma^2_{\boldsymbol{L},\hat{\boldsymbol{L}}}(S) := \sigma_{\boldsymbol{L}^2(S)}(S)$ denote the influence spread induced by S in live-edge graph $\boldsymbol{L}^2(S)$, and let $\sigma^2(S) := \mathbb{E}_{\boldsymbol{L},\hat{\boldsymbol{L}}}[\sigma^2_{\boldsymbol{L},\hat{\boldsymbol{L}}}(S)]$ denote the expected influence spread induced by S under the 2-level diffusion model. Let Φ and $\hat{\Phi}$ denote the random realisations associated to live-edge graphs $\boldsymbol{L}$ and $\hat{\boldsymbol{L}}$, respectively.

2-Level Hybrid Adaptive Policy. For any $t \in [k-1]_0$, let Hyb^2_t be a hybrid adaptive policy defined as follows: (i) Hyb^2_t selects all the nodes in S_t as seeds; (ii) then, Hyb^2_t adds to S_t all the nodes that the optimal adaptive policy π would have select when starting from the

empty realisation, and observing, at each step, partial realisations coming from the live-edge graph $\hat{\boldsymbol{L}}$ only; in other words, for any seed $v \in V$ selected by the policy, the new set of nodes that the policy observes (to choose the next seed) is $\hat{\Phi}(v)$; (iii) finally, denoting with $\hat{\Psi}_\pi$ the random realisation returned by π, the expected influence spread of Hyb_t^2 is defined as $\mathbb{E}_{\boldsymbol{L},\hat{\boldsymbol{L}}}[\sigma^2_{\boldsymbol{L},\hat{\boldsymbol{L}}}(dom(\hat{\Psi}_\pi) \cup S_t)]$, i.e., it is the expected influence spread determined $dom(\hat{\Psi}_\pi) \cup S_t$, according to the 2-level live-edge graph $\boldsymbol{L}^2(dom(\hat{\Psi}_\pi) \cup S_t)$.

In what follows we give some technical results which are based on the above definitions and will be used in Subsection 4.4 to show the main theorem. In particular, we show that the 2-level diffusion model satisfies certain properties connected with submodular set functions (see Subsection 4.1); then, we use such properties to relate the expected influence spread in the ordinary diffusion model to that of the 2-level diffusion model (see Subsection 4.2); finally, by using a similar approach as in Section 2, we use the above relations to relate the efficiency of the randomized non-adaptive policy with that of the optimal adaptive policy, via the 2-level hybrid adaptive policy (see Subsection 4.3).

4.1 Adaptive Submodularity in the 2-Level Diffusion Model

The adaptive submodularity [25] is a property that extends the well-known concept of submodularity to the adaptive framework, and allows us to design efficient adaptive approximation algorithms. In particular, the adaptive submodularity states that, given two subrealisations $\psi \subseteq \psi'$ and a node $v \in V$, adding node v under partial realisation ψ' causes an expected increment of the influence spread that is not higher than that caused under partial realisation ψ. Unfortunately, as shown in the arXiv version of [25], the myopic feedback model, in general, does not satisfy the adaptive submodularity.

Anyway, by resorting to the 2-level diffusion model, we can recover a similar property as the adaptive submodularity. Given $S \subseteq V$, a partial realisation $\hat{\psi}$, and $v \in V$, let

$$\Delta^2_S(v|\hat{\psi}) := \mathbb{E}_{\boldsymbol{L},\hat{\boldsymbol{L}}}\left[\sigma_{\boldsymbol{L}^2(\{v\}\cup dom(\hat{\psi})\setminus S)}(\{v\} \cup S \cup dom(\hat{\psi})) - \sigma_{\boldsymbol{L}^2(dom(\hat{\psi})\setminus S)}(S \cup dom(\hat{\psi}))|\hat{\psi} \subseteq \hat{\Phi}\right]$$

denote the expected increment of the influence spread w.r.t. the 2-level diffusion model when adding seed v to the set of nodes $S \cup dom(\hat{\psi})$, but assuming that the nodes in S have a unique chance to influence their neighbors, and that partial realisation $\hat{\psi}$ has been observed. We say that the 2-level diffusion model is *adaptive submodular* if, for any $S \subseteq V$, any partial realisations $\hat{\psi}, \hat{\psi}'$ with $\hat{\psi} \subseteq \hat{\psi}'$, and any $v \in V$, we have that $\Delta^2_S(v|\hat{\psi}) \geq \Delta^2_S(v|\hat{\psi}')$.

▶ **Lemma 6.** *The 2-level diffusion model is adaptive submodular.*

4.2 From the Ordinary to the 2-level Diffusion Model

In this subsection, we see how to relate the 2-level diffusion model to the ordinary one.

▶ **Lemma 7.** *We have that $\sigma^2(S) \leq 2 \cdot \sigma(S)$ for any $S \subseteq V$.*

Now, given a set $S \subseteq V$ and $v \in V$, let $\Delta_S(v) := \sigma(S \cup \{v\}) - \sigma(S)$, that is the expected increment under the ordinary diffusion model when adding a new node to a set S, and let $\Delta^2_S(v) := \mathbb{E}_{\boldsymbol{L},\hat{\boldsymbol{L}}}[\sigma_{\boldsymbol{L}^2(\{v\})}(S \cup \{v\}) - \sigma_{\boldsymbol{L}}(S)]$, that is the above expected increment, with the further assumption that v has two chances to influence its neighbors.

▶ **Lemma 8.** *We have that $\Delta^2_S(v) \leq 2 \cdot \Delta_S(v)$ for any $S \subseteq V$ and $v \in V$.*

4.3 From the Adaptive to the Randomized Non-adaptive Policy

The following three lemmas (Lemma 9, Lemma 10, and 11) relate the randomized non-adaptive policy with the 2-level hybrid adaptive policy, and then with the optimal adaptive policy of the ordinary diffusion model. In particular, the proofs of Lemma 9, Lemma 10, and 11, resort to a similar approach of the proofs of Lemma 2, Lemma 3, and Lemma 4 of Section 2.

▶ **Lemma 9.** *For any $S \subseteq V$, we have that $k \cdot (\mathbb{E}_{\boldsymbol{\rho}}[\sigma(\{\boldsymbol{\rho}\} \cup S)] - \sigma(S)) \geq \sum_{v \in V \setminus S} x_v \cdot \Delta_S(v)$.*

The following lemma relates the adaptive setting with the non-adaptive one, and its proof resorts to the adaptive submodularity defined in Subsection 4.1.

▶ **Lemma 10.** *We have $\mathbb{E}_{\boldsymbol{L},\hat{\boldsymbol{L}}}[\sigma^2_{\boldsymbol{L},\hat{\boldsymbol{L}}}(dom(\hat{\Psi}_\pi) \cup S)] \leq \sigma^2(S) + \sum_{v \in V \setminus S} x_v \cdot \Delta^2_S(v)$ for any $S \subseteq V$.*

The following lemma shows that the optimal adaptive influence spread is upper bounded by that expected influence spread of the 2-level hybrid adaptive policy.

▶ **Lemma 11.** *We have $OPT_A(G,k) \leq \mathbb{E}_{\boldsymbol{L},\hat{\boldsymbol{L}}}[\sigma^2_{\boldsymbol{L},\hat{\boldsymbol{L}}}(dom(\hat{\Psi}_\pi) \cup S)]$ for any $S \subseteq V$.*

4.4 Proof of Theorem 5

Armed with the above results, we can now prove Theorem 5. For any $t \in [k]_0$, let $GR_N(G,t) := \sigma(S_t)$ denote the expected influence spread $\sigma(S_t)$ obtained when the first t seeds have been selected by the greedy algorithm. We have that

$$
\begin{aligned}
&GR_N(G,t+1) - GR_N(G,t) \\
&= \max_{v \in V} [\sigma(\{v\} \cup S_t)] - \sigma(S_t) \\
&\geq \overbrace{\mathbb{E}_{\boldsymbol{\rho}}[\sigma(\{\boldsymbol{\rho}\} \cup S_t)]}^{\text{Exp. value of } \mathsf{Rand}_t} - \sigma(S_t) && (7)\\
&\geq \frac{1}{k} \sum_{v \in V \setminus S_t} x_v \cdot \Delta_{S_t}(v) && (8)\\
&\geq \frac{1}{2k} \sum_{v \in V \setminus S_t} x_v \cdot \Delta^2_{S_t}(v) && (9)\\
&\geq \frac{1}{2k} (\overbrace{\mathbb{E}_{\boldsymbol{L},\hat{\boldsymbol{L}}}[\sigma^2_{\boldsymbol{L},\hat{\boldsymbol{L}}}(dom(\hat{\Psi}_\pi) \cup S_t)]}^{\text{Exp. value of } \mathsf{Hyb}^2_t} - \sigma^2(S_t)) && (10)\\
&\geq \frac{1}{2k} (OPT_A(G,k) - \sigma^2(S_t)) && (11)\\
&\geq \frac{1}{2k} (OPT_A(G,k) - 2 \cdot \sigma(S_t)) && (12)\\
&= \frac{1}{2k} OPT_A(G,k) - \frac{1}{k} GR_N(G,t), && (13)
\end{aligned}
$$

where (7) holds since the greedy strategy adds to S_t the node v maximizing $\sigma(S_t \cup \{v\})$, (8) comes from Lemma 9, (9) comes from Lemma 8, (10) comes from Lemma 10, (11) comes from Lemma 11, and (12) comes from Lemma 7. Thus, by (13) and some manipulations we get the following recursive relation: $GR_N(G,t+1) \geq \frac{1}{2k} \cdot OPT_A(G,k) + \left(1 - \frac{1}{k}\right) \cdot GR_N(G,t)$

for any $t \in [k-1]_0$. By applying iteratively the above inequality, we get $GR_N(G,k) \geq \frac{1}{2k} \cdot \sum_{t=0}^{k-1} \left(1-\frac{1}{k}\right)^t \cdot OPT_A(G,k) = \frac{1}{2}\left(1-\left(1-\frac{1}{k}\right)^k\right) \cdot OPT_A(G,k)$, that leads to $\frac{GR_N(G,k)}{OPT_A(G,k)} \geq \frac{1}{2}\left(1-\left(1-\frac{1}{k}\right)^k\right) \geq \frac{1}{2}\left(1-\frac{1}{e}\right)$, and this shows the claim. ◀

▶ Remark 12. By Theorem 5, we can easily show that, for any influence graph G with n nodes, the k-adaptivity gap of G is at most $2\left(1-\left(1-\frac{1}{k}\right)^k\right)^{-1} \leq \frac{2e}{e-1} \approx 3.164$.

5 The Efficiency of the Adaptive Greedy Algorithm

We show that the adaptive version of the greedy algorithm guarantees an even better approximation ratio of $1-\frac{1}{\sqrt{e}} \approx 0.393$ for the adaptive influence maximization problem. The *adaptive greedy algorithm* is an adaptive policy π_k^{GR} that selects k seeds in k steps, and at each step t selects the t-th seed that maximizes the expected influence spread conditioned by the observed realisation.

▶ **Theorem 13.** *Given an influence graph G with n nodes and $k \in [n]_2$, the adaptive greedy algorithm is a $1-\left(1-\frac{1}{2k}\right)^k \geq 1-\frac{1}{\sqrt{e}}$ approximation algorithm for the adaptive influence maximization problem (under IC and myopic feedback) applied to (G,k).*

In the proof of Theorem 13 we relate the expected influence spread coming from the adaptive greedy algorithm with that of the optimal adaptive policy, passing trough a new hybrid adaptive policy. In the same spirit of Theorem 5, we also consider a new diffusion model and a new notion of adaptive submodularity (slightly different from that of Subsection 4.1). To show the main theorem (see Subsection 5.1) we need some notation and preliminary results. Let $G = (V = [n], E, (p_{uv})_{(u,v)\in E})$ be an influence graph, and let $k \in [n]_2$. Let π be an optimal adaptive policy, and let $x = (x_1, \ldots, x_n)$ be the vector such that x_i is the probability that node i is selected by π.

Strong 2-level Diffusion Model. We define $\boldsymbol{L}$, $\hat{\boldsymbol{L}}$, $\boldsymbol{L}^2(S)$, Φ, $\hat{\Phi}$ as in the 2-level diffusion model considered in Section 4. Given a partial realisation ψ and a set $S \subseteq V$, let $\sigma^2_{\boldsymbol{L},\hat{\boldsymbol{L}},\psi}(S) := \sigma_{\boldsymbol{L}^2(S\setminus dom(\psi))}(S)$ denote the influence spread induced by S in live-edge graph $\boldsymbol{L}^2(S \setminus dom(\psi))$, and let $\sigma^2_\psi(S) := \mathbb{E}_{\boldsymbol{L},\hat{\boldsymbol{L}}}[\sigma^2_{\boldsymbol{L},\hat{\boldsymbol{L}},\psi}(S) | \psi \subseteq \Phi]$ denote the above influence spread in expectation, conditioned by partial realisation ψ.

Strong 2-level Hybrid Adaptive Policy. Given a partial realisation ψ, let Hyb^2_ψ be a hybrid adaptive policy defined as follows: (i) Hyb^2_ψ selects all the nodes in $dom(\psi)$ as seeds; (ii) then, Hyb^2_ψ adds to $dom(\psi)$ all the nodes that policy π would have select when starting from the empty realisation, and observing, at each step, partial realisations coming from the live-edge graph $\hat{\boldsymbol{L}}$ only; (iii) finally, denoting with $\hat{\Psi}_\pi$ the random realisation returned by policy π, the expected influence spread of Hyb^2_ψ is defined as $\mathbb{E}_{\boldsymbol{L},\hat{\boldsymbol{L}}}[\sigma^2_{\boldsymbol{L},\hat{\boldsymbol{L}},\psi}(dom(\psi) \cup dom(\hat{\Psi}_\pi)) | \ \psi \subseteq \ \Phi]$, i.e., it is the expected influence spread determined $dom(\psi) \cup dom(\hat{\Psi}_\pi)$ in the strong 2-level diffusion model, conditioned by partial realisation ψ. Differently from the 2-level hybrid policy defined in Section 4, the expected influence spread of the hybrid policy defined here is conditioned by partial realisation ψ and the unique seeds that have two chances to influence their neighbors are those in $dom(\hat{\Psi}_\pi) \setminus dom(\psi)$.

Strong Adaptive Submodularity of the Strong 2-level Diffusion Model. Given two partial realisations $\psi, \hat{\psi}$ and $v \in V$, let

$$\Delta^2_{\psi}(v|\hat{\psi}) := \mathbb{E}_{\boldsymbol{L},\hat{\boldsymbol{L}}}\left[\sigma^2_{\boldsymbol{L},\hat{\boldsymbol{L}},\psi}(\{v\} \cup dom(\psi) \cup dom(\hat{\psi})) - \sigma^2_{\boldsymbol{L},\hat{\boldsymbol{L}},\psi}(dom(\psi) \cup dom(\hat{\psi}))|\psi \subseteq \Phi, \hat{\psi} \subseteq \hat{\Phi}\right],$$

i.e., $\Delta^2_{\psi}(v|\hat{\psi})$ is the expected increment of the influence spread in the strong 2-level diffusion model when adding seed v to the set of nodes $dom(\hat{\psi}) \cup dom(\psi)$, conditioned by the observation of partial realisations $\hat{\psi}$ and ψ. We say that the strong 2-level diffusion model is *strongly adaptive submodular* if, for any partial realisations $\hat{\psi}, \hat{\psi}'$ with $\hat{\psi} \subseteq \hat{\psi}'$, any partial realisation ψ, and any $v \in V$, we have that $\Delta^2_{\psi}(v|\hat{\psi}) \geq \Delta^2_{\psi}(v|\hat{\psi}')$.

▶ **Lemma 14.** *The strong 2-level diffusion model is strongly adaptive submodular.*

From the Ordinary to the Strong 2-level Diffusion Model. Given a partial realisation ψ, and $v \in V$, let $\Delta_{\psi}(v) := \mathbb{E}_{\boldsymbol{L}}[\sigma_{\boldsymbol{L}}(\{v\} \cup dom(\psi)) - \sigma_{\boldsymbol{L}}(dom(\psi))|\psi \subseteq \Phi]$, that is the expected increment under the ordinary diffusion model when adding a new node to the nodes in $dom(\psi)$ conditioned by partial realisation ψ, and let $\Delta^2_{\psi}(v) := \Delta^2_{\psi}(v|\emptyset) = \mathbb{E}_{\boldsymbol{L},\hat{\boldsymbol{L}}}[\sigma_{\boldsymbol{L},\hat{\boldsymbol{L}},\psi}(\{v\} \cup dom(\psi)) - \sigma_{\boldsymbol{L},\hat{\boldsymbol{L}},\psi}(dom(\psi))|\psi \subseteq \Phi]$, that is the above conditional expectation, but w.r.t. the strong 2-level diffusion model. The following lemma can be shown analogously to Lemma 8.

▶ **Lemma 15.** *We have that $\Delta^2_{\psi}(v) \leq 2 \cdot \Delta_{\psi}(v)$ for any partial realisation ψ and $v \in V$.*

From the Optimal to the Greedy Adaptive Policy. The following lemma will be used to relate the strong 2-level hybrid adaptive policy with the adaptive greedy policy; its proof is similar to that of Lemma 10, but uses the concept of strong adaptive submodularity.

▶ **Lemma 16.** *For any partial realisation ψ, we have*

$$\mathbb{E}_{\boldsymbol{L},\hat{\boldsymbol{L}}}[\sigma^2_{\boldsymbol{L},\hat{\boldsymbol{L}},\psi}(dom(\psi) \cup dom(\hat{\Psi}_{\pi}))|\ \psi \subseteq \Phi] \leq \mathbb{E}_{\boldsymbol{L}}[\sigma_{\boldsymbol{L}}(dom(\psi))|\psi\ \subseteq\ \Phi] + \sum_{v \in V \setminus dom(\psi)} x_v \cdot \Delta^2_{\psi}(v).$$

The following lemma shows that the optimal adaptive influence spread is upper bounded by that expected influence spread of the strong 2-level hybrid adaptive policy, and its proof is completely analogue to that of Lemma 11.

▶ **Lemma 17.** *We have $OPT_A(G,k) \leq \mathbb{E}_{\boldsymbol{L},\hat{\boldsymbol{L}}}[\sigma^2_{\boldsymbol{L},\hat{\boldsymbol{L}},\psi}(dom(\psi) \cup dom(\hat{\Psi}_{\pi}))|\ \psi \subseteq \Phi]$ for any partial realisation ψ.*

5.1 Proof of Theorem 13

Armed with the above lemmas, and by using a similar approach as in the proof of Theorem 5, we can prove Theorem 13. Given $t \in [k]_0$, let $\boldsymbol{S}_t$ denote the (random) set of the first t seeds selected by the adaptive greedy policy π^{GR}_k, and let Ψ_t be the (random) partial realisation such that $dom(\Psi_t) = \boldsymbol{S}_t$ (i.e., the partial realisation observed by policy π^{GR}_k at the end of step t); let $GR_A(G,t) := \mathbb{E}_{\boldsymbol{L}}[\sigma_{\boldsymbol{L}}(\boldsymbol{S}_t)]$ denote the expected influence spread of the adaptive greedy policy after selecting the first t seeds. For any $t \in [k-1]_0$, we have that

$$
\begin{aligned}
&GR_A(G,t+1) - GR_A(G,t) \\
&= \mathbb{E}_{\boldsymbol{L}}[\sigma(\boldsymbol{S}_{t+1}) - \sigma(\boldsymbol{S}_t)] \\
&= \mathbb{E}_{\Psi_t}\left[\max_{v \in V \setminus dom(\Psi_t)} \mathbb{E}_{\boldsymbol{L}}[\sigma_{\boldsymbol{L}}(\{v\} \cup dom(\Psi_t)) - \sigma_{\boldsymbol{L}}(dom(\Psi_t))|\Psi_t]\right] && (14) \\
&\geq \mathbb{E}_{\Psi_t}\left[\sum_{v \in V \setminus dom(\Psi_t)} \frac{x_v}{k} \cdot \mathbb{E}_{\boldsymbol{L}}[\sigma_{\boldsymbol{L}}(\{v\} \cup dom(\Psi_t)) - \sigma_{\boldsymbol{L}}(dom(\Psi_t))|\Psi_t]\right] \\
&= \frac{1}{k} \cdot \mathbb{E}_{\Psi_t}\left[\sum_{v \in V \setminus dom(\Psi_t)} x_v \cdot \mathbb{E}_{\boldsymbol{L}}[\sigma_{\boldsymbol{L}}(\{v\} \cup dom(\Psi_t)) - \sigma_{\boldsymbol{L}}(dom(\Psi_t))|\Psi_t]\right] \\
&= \frac{1}{k} \cdot \mathbb{E}_{\Psi_t}\left[\sum_{v \in V \setminus dom(\Psi_t)} x_v \cdot \Delta_{\Psi_t}(v)\right] \\
&\geq \frac{1}{2k} \cdot \mathbb{E}_{\Psi_t}\left[\sum_{v \in V \setminus dom(\Psi_t)} x_v \cdot \Delta^2_{\Psi_t}(v)\right] && (15) \\
&\geq \frac{1}{2k} \cdot \mathbb{E}_{\Psi_t}\left[\mathbb{E}_{\boldsymbol{L},\hat{\boldsymbol{L}}}[\sigma^2_{\boldsymbol{L},\hat{\boldsymbol{L}},\Psi_t}(dom(\Psi_t) \cup dom(\hat{\Psi}_\pi))|\Psi_t] - \mathbb{E}_{\boldsymbol{L}}[\sigma_{\boldsymbol{L}}(dom(\Psi_t))|\Psi_t]\right] && (16) \\
&\geq \frac{1}{2k} \cdot \mathbb{E}_{\Psi_t}\left[OPT_A(G,k) - \mathbb{E}_{\boldsymbol{L}}[\sigma_{\boldsymbol{L}}(dom(\Psi_t))|\Psi_t]\right] && (17) \\
&\geq \frac{1}{2k} \cdot OPT_A(G,k) - \frac{1}{2k} \cdot \mathbb{E}_{\Psi_t}\left[\mathbb{E}_{\boldsymbol{L}}[\sigma_{\boldsymbol{L}}(dom(\Psi_t))|\Psi_t]\right] \\
&= \frac{1}{2k} \cdot OPT_A(G,k) - \frac{1}{2k} \cdot \mathbb{E}_{\boldsymbol{L}}(\sigma_{\boldsymbol{L}}(\boldsymbol{S}_t)) \\
&= \frac{1}{2k} \cdot OPT_A(G,k) - \frac{1}{2k} \cdot GR_A(G,t), && (18)
\end{aligned}
$$

where (14) holds since the adaptive greedy strategy at each step adds the node maximizing the expected influence spread (conditioned by the partial realisation coming from the previous selected seeds), (15) comes from Lemma 15, (16) comes from Lemma 16, (17) comes from Lemma 17. Thus, by (18) and some manipulations we get the following recursive relation:

$$
GR_A(G,t+1) \geq \frac{1}{2k} \cdot OPT_A(G,k) + \left(1 - \frac{1}{2k}\right) \cdot GR_A(G,t), \quad \forall t \in [k-1]_0. \tag{19}
$$

By applying iteratively (19), we get

$$
GR_A(G,k) \geq \frac{1}{2k} \cdot \sum_{t=0}^{k-1} \left(1 - \frac{1}{2k}\right)^t \cdot OPT_A(G,k) = 1 - \left(1 - \frac{1}{2k}\right)^k \cdot OPT_A(G,k),
$$

that leads to

$$
\frac{GR_A(G,k)}{OPT_A(G,k)} \geq 1 - \left(1 - \frac{1}{2k}\right)^k \geq 1 - \frac{1}{\sqrt{e}},
$$

and this shows the claim.

6 Conclusions and Future Work

In the context of adaptive optimization, we have introduced a new approach to relate the solution provided by a simple non-adaptive greedy policy with the adaptive optimum. The new approach allowed us to establish better bounds for the adaptive influence maximization problem under myopic feedback, specifically we improve both the approximation ratio of the non-adaptive greedy policy and the adaptivity gap.

Our results open several research directions in the context of influence maximization and in more general adaptive optimization settings. The approximation factor of the non-adaptive (resp. adaptive) greedy algorithm is between our lower bound of $\frac{1}{2}\left(1-\frac{1}{e}\right) \approx 0.316$ (resp. $1-\frac{1}{\sqrt{e}} \approx 0.393$) and the upper bound of $\frac{e^2+1}{(e+1)^2} \approx 0.606$ [39], and the adaptivity gap is between the lower bound of $\frac{e}{e-1} \approx 1.582$ [39] and our upper bound of $\frac{2e}{e-1} \approx 3.164$. The first problem left open by our result is to close these gaps. Furthermore, the techniques introduced in this paper to relate non-adaptive policies with adaptive ones might be useful to find better bounds in several variants of the adaptive influence maximization problem, like a combination of the following settings: different feedback models (e.g., the full-adoption feedback), different diffusion models (e.g., the general triggering model [33]), and different graph classes. Finally, as shown in Section 2, we believe that our new approach could be efficiently used to analyze non-adaptive greedy algorithms in other adaptive optimization problems, like e.g. the stochastic probing problem [6, 29, 30].

References

1 Marek Adamczyk, Maxim Sviridenko, and Justin Ward. Submodular stochastic probing on matroids. In *31st International Symposium on Theoretical Aspects of Computer Science, (STACS)*, pages 29–40, 2014. doi:10.4230/LIPIcs.STACS.2014.29.

2 Georgios Amanatidis, Federico Fusco, Philip Lazos, Stefano Leonardi, and Rebecca Reiffenhäuser. Fast adaptive non-monotone submodular maximization subject to a knapsack constraint. In *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems, (NeurIPS)*, 2020.

3 Arash Asadpour and Hamid Nazerzadeh. Maximizing stochastic monotone submodular functions. *Management Science*, 62(8):2374–2391, 2016.

4 Arash Asadpour, Hamid Nazerzadeh, and Amin Saberi. Stochastic submodular maximization. In *Internet and Network Economics, 4th International Workshop, (WINE)*, pages 477–489, 2008.

5 Ashwinkumar Badanidiyuru, Christos Papadimitriou, Aviad Rubinstein, Lior Seeman, and Yaron Singer. Locally adaptive optimization: Adaptive seeding for monotone submodular functions. *Proceedings of the Annual ACM-SIAM Symposium on Discrete Algorithms, (SODA)*, 1:414–429, 2016.

6 Domagoj Bradac, Sahil Singla, and Goran Zuzic. (near) optimal adaptivity gaps for stochastic multi-value probing. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques (APPROX/RANDOM)*, pages 49:1–49:21, 2019.

7 Gruia Călinescu, Chandra Chekuri, Martin Pál, and Jan Vondrák. Maximizing a monotone submodular function subject to a matroid constraint. *SIAM J. Comput.*, 40(6):1740–1766, 2011.

8 Carri W. Chan and Vivek F. Farias. Stochastic depletion problems: Effective myopic policies for a class of dynamic optimization problems. *Math. Oper. Res.*, 34(2):333–350, 2009. doi:10.1287/moor.1080.0364.

9 Chandra Chekuri, Jan Vondrák, and Rico Zenklusen. Submodular function maximization via the multilinear relaxation and contention resolution schemes. *SIAM J. Comput.*, 43(6):1831–1879, 2014.

10 Wei Chen, Laks V. S. Lakshmanan, and Carlos Castillo. *Information and Influence Propagation in Social Networks*. Synthesis Lectures on Data Management. Morgan & Claypool, 2013.

11 Wei Chen and Binghui Peng. On adaptivity gaps of influence maximization under the independent cascade model with full-adoption feedback. *30th International Symposium on Algorithms and Computation, (ISAAC)*, pages 24:1–24:19, 2019.

12 Wei Chen, Binghui Peng, Grant Schoenebeck, and Biaoshuai Tao. Adaptive Greedy versus Non-adaptive Greedy for Influence Maximization. In *The Thirty-Fourth AAAI Conference on Artificial Intelligence, (AAAI)*, 2020.

13 Wei Chen, Chi Wang, and Yajun Wang. Scalable influence maximization for prevalent viral marketing in large-scale social networks. In *Proceedings of the 16th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 1029–1038, 2010.

14 Edith Cohen, Daniel Delling, Thomas Pajor, and Renato F. Werneck. Sketch-based influence maximization and computation: Scaling up with guarantees. In *Proceedings of the 23rd ACM International Conference on Conference on Information and Knowledge Management, (CIKM)*, pages 629–638, 2014.

15 Gianlorenzo D'Angelo, Debashmita Poddar, and Cosimo Vinci. Improved approximation factor for adaptive influence maximization via simple greedy strategies. *CoRR*, abs/2007.09065, 2020. `arXiv:2007.09065`.

16 Gianlorenzo D'Angelo, Debashmita Poddar, and Cosimo Vinci. Better bounds on the adaptivity gap of influence maximization under full-adoption feedback. In *AAAI Conference on Artificial Intelligence, (AAAI)*, 2021. To appear.

17 Brian C. Dean, Michel X. Goemans, and Jan Vondrák. Adaptivity and approximation for stochastic packing problems. In *Proceedings of the Sixteenth Annual ACM-SIAM Symposium on Discrete Algorithms, (SODA)*, pages 395–404, 2005. URL: `http://dl.acm.org/citation.cfm?id=1070432.1070487`.

18 Brian C. Dean, Michel X. Goemans, and Jan Vondrák. Approximating the stochastic knapsack problem: The benefit of adaptivity. *Math. Oper. Res.*, 33(4):945–964, 2008. `doi:10.1287/moor.1080.0330`.

19 Pedro Domingos and Matt Richardson. Mining the network value of customers. *Proceedings of the Seventh ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 57–66, 2001.

20 Alina Ene and Huy L. Nguyen. Submodular maximization with nearly-optimal approximation and adaptivity in nearly-linear time. In Timothy M. Chan, editor, *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms, (SODA)*, pages 274–282, 2019.

21 Matthew Fahrbach, Vahab S. Mirrokni, and Morteza Zadimoghaddam. Non-monotone submodular maximization with nearly optimal adaptivity and query complexity. In Kamalika Chaudhuri and Ruslan Salakhutdinov, editors, *Proceedings of the 36th International Conference on Machine Learning, (ICML)*, pages 1833–1842, 2019.

22 Matthew Fahrbach, Vahab S. Mirrokni, and Morteza Zadimoghaddam. Submodular maximization with nearly optimal approximation, adaptivity and query complexity. In Timothy M. Chan, editor, *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms, (SODA)*, pages 255–273, 2019.

23 Michel X. Goemans and Jan Vondrák. Stochastic covering and adaptivity. In *Theoretical Informatics, 7th Latin American Symposium, (LATIN)*, pages 532–543, 2006. `doi:10.1007/11682462_50`.

24 Sharon Goldberg and Zhenming Liu. The diffusion of networking technologies. In *Proceedings of the Twenty-Fourth Annual ACM-SIAM Symposium on Discrete Algorithms, (SODA)*, pages 1577–1594, 2013.

25 Daniel Golovin and Andreas Krause. Adaptive submodularity: Theory and applications in active learning and stochastic optimization. *Journal of Artificial Intelligence Research*, 42:427–486, 2011.

26 Alkis Gotovos, Amin Karbasi, and Andreas Krause. Non-monotone adaptive submodular maximization. In *Proceedings of the 24th International Conference on Artificial Intelligence (IJCAI)*, page 1996–2003. AAAI Press, 2015.

27 Amit Goyal, Wei Lu, and Laks V. S. Lakshmanan. CELF++: optimizing the greedy algorithm for influence maximization in social networks. In *Proceedings of the 20th International Conference on World Wide Web, (WWW)*, pages 47–48, 2011.

28 Andrew Guillory and Jeff A. Bilmes. Interactive submodular set cover. In *Proceedings of the 27th International Conference on Machine Learning, (ICML)*, pages 415–422, 2010. URL: `https://icml.cc/Conferences/2010/papers/436.pdf`.

29 Anupam Gupta, Viswanath Nagarajan, and Sahil Singla. Algorithms and Adaptivity Gaps for Stochastic Probing. *Proceedings of the twenty-seventh annual ACM-SIAM symposium on Discrete algorithms, (SODA)*, 151:1731–1747, 2016.

30 Anupam Gupta, Viswanath Nagarajan, and Sahil Singla. Adaptivity gaps for stochastic probing: Submodular and XOS functions. *Proceedings of the Annual ACM-SIAM Symposium on Discrete Algorithms, (SODA)*, pages 1688–1702, 2017.

31 Kai Han, Keke Huang, Xiaokui Xiao, Jing Tang, Aixin Sun, and Xueyan Tang. Efficient algorithms for adaptive influence maximization. *Proceedings of the VLDB Endowment*, 11(9):1029–1040, 2018.

32 Lisa Hellerstein, Devorah Kletenik, and Patrick Lin. Discrete stochastic submodular maximization: Adaptive vs. non-adaptive vs. offline. In *Algorithms and Complexity - 9th International Conference, (CIAC)*, pages 235–248, 2015. `doi:10.1007/978-3-319-18173-8_17`.

33 David Kempe, Jon Kleinberg, and Éva Tardos. Maximizing the spread of influence through a social network. In *Proceedings of the Ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 137–146. ACM, 2003.

34 David Kempe, Jon Kleinberg, and Éva Tardos. Maximizing the spread of influence through a social network. *Theory of Computing*, 11:105–147, 2015.

35 Jure Leskovec, Andreas Krause, Carlos Guestrin, Christos Faloutsos, Jeanne M. VanBriesen, and Natalie S. Glance. Cost-effective outbreak detection in networks. In *Proceedings of the 13th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 420–429, 2007.

36 Y. Li, J. Fan, Y. Wang, and K. Tan. Influence maximization on social graphs: A survey. *IEEE Transactions on Knowledge and Data Engineering*, 30(10):1852–1872, 2018.

37 Hung T. Nguyen, My T. Thai, and Thang N. Dinh. Stop-and-stare: Optimal sampling algorithms for viral marketing in billion-scale networks. In *Proceedings of the 2016 International Conference on Management of Data, (SIGMOD)*, pages 695–710, 2016.

38 Srinivasan Parthasarathy. An analysis of the greedy algorithm for stochastic set cover. *CoRR*, abs/1803.07639, 2018. `arXiv:1803.07639`.

39 Binghui Peng and Wei Chen. Adaptive influence maximization with myopic feedback. *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing System, (NeurIPS)*, pages 5575–5584, 2019.

40 Matthew Richardson and Pedro M. Domingos. Mining knowledge-sharing sites for viral marketing. In *Proceedings of the Eighth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 61–70, 2002.

41 Aviad Rubinstein, Lior Seeman, and Yaron Singer. Approximability of adaptive seeding under knapsack constraints. *Proceedings of the 2015 ACM Conference on Economics and Computation, (EC)*, pages 797–814, 2015.

42 Guillaume Salha, Nikolaos Tziortziotis, and Michalis Vazirgiannis. Adaptive submodular influence maximization with myopic feedback. *Proceedings of the 2018 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining (ASONAM)*, pages 455–462, 2018.

43 Lior Seeman and Yaron Singer. Adaptive seeding in social networks. *Proceedings - Annual IEEE Symposium on Foundations of Computer Science, (FOCS)*, pages 459–468, 2013.

44 Yaron Singer. Influence maximization through adaptive seeding. *ACM SIGecom Exchanges*, 15(1):32–59, 2016.

45 Lichao Sun, Weiran Huang, Philip S. Yu, and Wei Chen. Multi-round influence maximization. *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 2249–2258, 2018.

46 Jing Tang, Laks V.S. Lakshmanan, Keke Huang, Xueyan Tang, Aixin Sun, Xiaokui Xiao, and Andrew Lim. Efficient approximation algorithms for adaptive seed minimization. *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 1096–1113, 2019.

47 Youze Tang, Yanchen Shi, and Xiaokui Xiao. Influence maximization in near-linear time: A martingale approach. In *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*, pages 1539–1554, 2015.

48 Youze Tang, Xiaokui Xiao, and Yanchen Shi. Influence Maximization : Near-Optimal Time Complexity Meets Practical Efficiency. *SIGMOD '14: Proceedings of the 2014 ACM SIGMOD International Conference on Management of Data*, pages 75–86, 2014.

49 Guangmo Tong and Ruiqi Wang. Adaptive Influence Maximization under General Feedback Models, 2019. `arXiv:1902.00192v3`.

50 Guangmo Tong, Ruiqi Wang, Zheng Dong, and Xiang Li. Time-constrained Adaptive Influence Maximization, 2020. `arXiv:2001.01742v2`.

51 Guangmo Tong, Weili Wu, Shaojie Tang, and Ding Zhu Du. Adaptive Influence Maximization in Dynamic Social Networks. *IEEE/ACM Transactions on Networking*, 25(1):112–125, 2017.

52 Sharan Vaswani and Laks V. S. Lakshmanan. Adaptive influence maximization in social networks: Why commit when you can adapt? *CoRR*, abs/1604.08171, 2016. `arXiv:1604.08171`.

53 Jing Yuan and Shaojie Tang. No time to observe: Adaptive influence maximization with partial feedback. *IJCAI International Joint Conference on Artificial Intelligence*, pages 3908–3914, 2017.

Approximation Algorithms for Min-Distance Problems in DAGs

Mina Dalirrooyfard
MIT, Cambridge, MA, USA

Jenny Kaufmann
Harvard University, Cambridge, MA, USA

Abstract

Graph parameters such as the diameter, radius, and vertex eccentricities are not defined in a useful way in Directed Acyclic Graphs (DAGs) using the standard measure of distance, since for any two nodes, there is no path between them in one of the two directions. So it is natural to consider the distance between two nodes as the length of the shortest path in the direction in which this path exists, motivating the definition of the min-distance. The min-distance between two nodes u and v is the minimum of the shortest path distances from u to v and from v to u.

As with the standard distance problems, the Strong Exponential Time Hypothesis [Impagliazzo-Paturi-Zane 2001, Calabro-Impagliazzo-Paturi 2009] leaves little hope for computing min-distance problems faster than computing All Pairs Shortest Paths, which can be solved in $\tilde{O}(mn)$ time. So it is natural to resort to approximation algorithms in $\tilde{O}(mn^{1-\epsilon})$ time for some positive ϵ. Abboud, Vassilevska W., and Wang [SODA 2016] first studied min-distance problems achieving constant factor approximation algorithms on DAGs, and Dalirrooyfard *et al* [ICALP 2019] gave the first constant factor approximation algorithms on general graphs for min-diameter, min-radius and min-eccentricities. Abboud *et al* obtained a 3-approximation algorithm for min-radius on DAGs which works in $\tilde{O}(m\sqrt{n})$ time, and showed that any $(2-\delta)$-approximation requires $n^{2-o(1)}$ time for any $\delta > 0$, under the Hitting Set Conjecture. We close the gap, obtaining a 2-approximation algorithm which runs in $\tilde{O}(m\sqrt{n})$ time. As the lower bound of Abboud *et al* only works for sparse DAGs, we further show that our algorithm is conditionally tight for dense DAGs using a reduction from Boolean matrix multiplication. Moreover, Abboud *et al* obtained a linear time 2-approximation algorithm for min-diameter along with a lower bound stating that any $(3/2-\delta)$-approximation algorithm for sparse DAGs requires $n^{2-o(1)}$ time under SETH. We close this gap for dense DAGs by obtaining a 3/2-approximation algorithm which works in $O(n^{2.350})$ time and showing that the approximation factor is unlikely to be improved within $O(n^{\omega-o(1)})$ time under the high dimensional Orthogonal Vectors Conjecture, where ω is the matrix multiplication exponent.

2012 ACM Subject Classification Mathematics of computing → Graph algorithms

Keywords and phrases Fine-grained complexity, Graph algorithms, Diameter, Radius, Eccentricities

Digital Object Identifier 10.4230/LIPIcs.ICALP.2021.60

Category Track A: Algorithms, Complexity and Games

Acknowledgements We thank our advisor, Virginia Vassilevska Williams, for many helpful suggestions.

1 Introduction

Among the most fundamental graph parameters that have been extensively studied are the diameter, radius and eccentricities [16, 24, 15, 21, 5, 17, 14, 20, 7, 8, 33, 34, 12, 22, 30, 28, 13, 2, 9] (and many others). The eccentricity of a vertex v is the largest distance between v and any other vertex. The diameter is the maximum eccentricity of a vertex in the graph, thus the distance between the two farthest nodes, and the radius is the minimum eccentricity, measuring the maximum distance to the most central node.

All of these parameters depend on the definition of the distance between two nodes. In undirected graphs, the distance between two vertices is just the shortest path distance $d(\cdot,\cdot)$ between them, which is symmetric. However, in directed graphs, this standard measure of distance d is not necessarily symmetric, since for two nodes, $d(u,v)$ may not equal $d(v,u)$.

Several notions of a "symmetric" distance for directed graphs have been studied. Cowen and Wagner [18] define the *roundtrip distance*, which for two vertices u and v is just $d(u,v)+d(v,u)$. Abboud, Vassilevska W., and Wang [3] define the *max-distance*, which is $\max\{d(u,v),d(v,u)\}$, and the *min-distance*, which is $\min\{d(u,v),d(v,u)\}$.

Each of these notions of distance has a particular application [19]. In this paper, we focus on the min-distance $d_{\min}(\cdot,\cdot)$. The min-distance characterizes a quantity of real-world relevance: for instance, a patient may visit a doctor or a doctor may visit a patient, and if they are in a hurry the min-distance between them may matter. Min-distance is a particularly natural notion of distance in directed acyclic graphs (DAGs), where the standard notion of distance is infinite in at least one direction for any given pair of vertices in a DAG. For example, in a topologically ordered DAG where the edges are directed from left to right, the min-diameter is simply the largest distance $d(u,v)$ where u is to the left of v.

More formally, for a vertex $v \in V$, the *min-eccentricity* $\epsilon(v)$ is $\max_{w\in V} d_{\min}(v,w)$, or in other words, the largest min-distance between v and any other vertex. The *min-diameter* of a graph is $\max_{v\in V}\epsilon(v)$. Note that the min-diameter is the only meaningful notion of diameter for DAGs: all other notions are infinite. The *min-radius* of a graph is $\min_{v\in V}\epsilon(v)$. A *center* is a vertex whose min-eccentricity is equal to the min-radius of the graph.

All-Pairs Shortest Paths (APSP) is the problem of computing the distance between u and v for every pair of vertices $u,v\in V$. In a graph G with m edges, n vertices, and nonnegative edge weights polynomial in n, APSP can easily be computed in $\tilde{O}(mn)$ time[1], by running Dijkstra's algorithm from every vertex[2]. Computing eccentricities, diameter, or radius with any of the notions of distance is no harder than computing APSP.

For the standard notion of distance, under the Strong Exponential Time Hypothesis (SETH) [25, 11], there is no *truly subquadratic* time algorithm for diameter (and thus nor for eccentricities) in unweighted graphs: that is, no such algorithm runs in time $O(m^{2-\epsilon})$ for $\epsilon>0$ [28]. This lower bound also holds for the other notions of diameter (and eccentricities) [19]. For radius, the same lower bound holds but under the Hitting Set Conjecture [3].

Since quadratic time is expensive on large graphs, we resort to approximation algorithms. Many constant factor approximation algorithms were known for all notions of diameter, eccentricities and radius, except for the min-distance notion until recently. For example, for the standard diameter and roundtrip diameter there is a folklore linear time 2-approximation algorithm, and for max-diameter and standard diameter, a conditionally tight 3/2-approximation algorithm is known in $\tilde{O}(m\sqrt{n})$ time [28].

Only recently Dalirrooyfard *et al* [19] showed constant factor approximation algorithms for min-distance problems in general graphs that run in $O(mn^{1-\epsilon})$ time for some fixed $\epsilon>0$. More specifically, they obtained a 3-approximation algorithm for min-diameter in $\tilde{O}(m\sqrt{n})$ time, a $(3+\delta)$-approximation algorithm for min-radius in $\tilde{O}(m\sqrt{n}/\delta)$ time, and a $(3+\delta)$-approximation algorithm for min-eccentricities in $\tilde{O}(m\sqrt{n}/\delta^2)$ time, for any $\delta>0$.

The reason it is hard to obtain approximation algorithms for min-diameter, min-radius, and min-eccentricities is that min-distance does not obey the triangle inequality. Hence the typical approaches to find algorithms that work for other notions of distance do not work for min-distance, as they crucially rely on the triangle inequality.

---

[1] The tilde hides polylogarithmic factors.

[2] Faster algorithms are known by Pettie [26] and Pettie and Ramachandran [27] for sparse graphs.

Table 1 Results on min-distance problems on DAGs. The $(*)$ marks lower bounds that are for dense DAGs. Our $(2-\delta)$ lower bound for min-radius is based on Triangle Detection and our $(\frac{3}{2}-\delta)$ lower bound for min-diameter is based on high dimensional OV. Our k and $(k+\delta)$-approximation algorithms are for any integer $k \geq 2$. Conditionally tight bounds are in bold.

Problem	Upper bound	Lower bound	Reference
min-diameter	2 in $O(m)$	$(\frac{\mathbf{3}}{\mathbf{2}}-\delta)$ needs $m^{2-o(1)}$	[3]
	$\frac{\mathbf{3}}{\mathbf{2}}$ in $O(n^{2.350})$ (dense, unweighted)	$(\frac{\mathbf{3}}{\mathbf{2}}-\delta)$ needs $n^{\omega-o(1)}$ $(*)$	this work
min-radius	3 in $\tilde{O}(m\sqrt{n})$	$(\mathbf{2}-\delta)$ needs $m^{2-o(1)}$	[3]
	2 in $\tilde{O}(\min(m\sqrt{n}, m^{2/3}n))$	$(\mathbf{2}-\delta)$ needs $n^{\omega-o(1)}$ $(*)$	this work
	k in $\tilde{O}(\min(mn^{1/k}, m^{\frac{2^{k-1}}{2^k-1}}n))$		this work
min-eccentri.	$3+\delta$ in $\tilde{O}(m\sqrt{n}/\delta^2)$		[19]
	$k+\delta$ in $\tilde{O}(\min(mn^{1/k}/\delta, m^{\frac{2^{k-1}}{2^k-1}}n/\delta))$		this work

On the bright side, since DAGs have more structure, it is easier to find algorithms for them. The best known subquadratic time algorithm for min-diameter in DAGs is a linear time 2-approximation algorithm, and the best subquadratic time algorithm for min-radius is a 3-approximation algorithm in $\tilde{O}(m\sqrt{n})$ time [3]. However, neither of these algorithms were proven to be conditionally tight.

Previously, the only known conditional lower bounds for these problems were due to Abboud, Vassilevska W., and Wang [3]. They showed that under the Orthogonal Vectors Conjecture from fine-grained complexity (and consequently under SETH [31]), there is no $(3/2-\delta)$-approximation algorithm for any $\delta > 0$ for min-diameter which runs in truly subquadratic time on sparse DAGs. Moreover, under the Hitting Set Conjecture, there is no $(2-\delta)$-approximation algorithm for any $\delta > 0$ for min-radius which runs in truly subquadratic time on sparse DAGs.

1.1 Our results

We obtain fast algorithms for min-diameter, min-eccentricities and min-radius with improved approximation factors. Our results can be seen in Table 1.

Min-Eccentricities and Min-Radius

We obtain the first known subquadratic time $(2+\delta)$-approximation algorithm for min-eccentricities in DAGs for any $\delta > 0$, and the first known subquadratic time 2-approximation algorithm for min-radius in DAGs. These algorithms run in time $\tilde{O}(\min(m\sqrt{n}/\delta, m^{2/3}n)/\delta)$ and $\tilde{O}(\min(m\sqrt{n}, m^{2/3}n))$ respectively. Note that our algorithms in this section are combinatorial: they do not exploit fast matrix multiplication and are potentially practical. Our results are *conditionally optimal* in both sparse and dense graphs: For sparse graphs, if the Hitting Set Conjecture is true, then our min-radius result is tight and our min-eccentricity result is essentially tight, in the sense that no approximation factor smaller than 2 can be achieved in subquadratic time for either of these problems [3]. For dense graphs, our 2-approximation algorithm works in $\tilde{O}(n^{7/3})$ time, and we show that there is no $(2-\delta)$-approximation algorithm for min-radius (and hence min-eccentricities) in $O(n^{\omega-\epsilon})$ for $\epsilon > 0$, if the best algorithm for Triangle Detection runs in time $\Omega(n^{\omega-o(1)})$. Here $\omega < 2.37286$ [6] is the exponent of matrix multiplication.

More generally, we obtain a series of algorithms trading off runtime and accuracy.

▶ **Theorem 1.** *For integer $k \geq 2$ and every $\delta > 0$, there is a $(k+\delta)$-approximation algorithm for min-eccentricities in DAGs which runs in $\tilde{O}(\min(mn^{1/k}/\delta, m^{2^{k-1}/(2^k-1)}n/\delta))$ time.*

For every integer $k \geq 2$, there is a k-approximation algorithm for min-radius in DAGs which runs in $\tilde{O}(\min(mn^{1/k}, m^{2^{k-1}/(2^k-1)}n))$ time.

As mentioned earlier, the case $k = 2$ gives a 2-approximation algorithm for min-radius running in time $\tilde{O}(\min(m\sqrt{n}, m^{2/3}n))$. For $m = \tilde{O}(n^{1.5})$, this matches the runtime and improves the approximation factor of the previous best known algorithm for this problem (from [3]). For $m = \omega(n^{1.5+o(1)})$, it improves both the approximation factor and the runtime.

Our min-eccentricity $(2+\delta)$-approximation algorithm borrows a key idea from the 3-approximation algorithm of [3] and combines it with a new binary search technique. The idea is to partition the DAG into intervals and do local APSP searches to find local paths, then combine these local paths with "outer" paths to guarantee a low enough min-distance to any vertex in the graph. In [3], these outer paths were found by using a clever choice of intervals; our algorithm instead applies binary search to find sets which can be used as jumping-off points for the outer paths, allowing us to shorten the lengths of these paths and also allowing us to approximate all min-eccentricities, not only min-radius. Our $(k+\delta)$-approximation algorithm is achieved by recursively running our approximation algorithm on the intervals instead of running local APSP, which allows us to improve the runtime.

For sparse graphs, Abboud, Vassilevska W., and Wang [3] already showed that a $(2-\delta)$-approximation for min-radius needs $\Omega(m^{2-o(1)})$ time under the Hitting Set Conjecture, so our 2-approximation algorithm is conditionally tight for sparse graphs. We show that the approximation factor of our algorithm is conditionally tight for the dense case as well by reducing Triangle Detection to $(2-\delta)$-approximation of min-radius for any $\delta > 0$. The best running time for Triangle Detection in n-node graphs is conjectured to be $\Omega(n^{\omega-o(1)})$ by many papers (see for example [1, 10]), where $\omega < 2.37286$ [6] is the exponent of fast matrix multiplication. Note that, since $m = O(n^2)$, our algorithm runs in $\tilde{O}(n^{7/3})$ time, which is faster than $O(n^\omega)$ for the current best bound on ω. Since the algorithm of Theorem 1 is combinatorial, if we restrict to combinatorial algorithms then there is no *truly subcubic* (meaning $O(n^{3-\epsilon})$ for $\epsilon > 0$) time $(2-\delta)$-approximation algorithm for min-radius provided that there is no truly subcubic time combinatorial algorithm for Boolean matrix multiplication (BMM). This is because BMM and Triangle Detection are subcubic equivalent [32]. Note that our reduction graph in Theorem 2 is an unweighted DAG.

▶ **Theorem 2.** *If there is a $T(n,m)$-time algorithm for $(2-\delta)$-approximation of min-radius in $O(n)$-node $\tilde{O}(m)$-edge DAGs for some $\delta > 0$, then there is an $\tilde{O}(T(n,m)+m)$-time algorithm for Triangle Detection on graphs with n nodes and m edges.*

▶ **Corollary 3.** *Assuming the best algorithm for Triangle Detection runs in time $\Omega(n^{\omega-o(1)})$, there is no algorithm for $(2-\delta)$-approximation of min-radius in n-node dense DAGs that runs in time $O(n^{\omega-\epsilon})$ for any $\delta, \epsilon > 0$.*

Moreover, there is no $O(n^{3-\epsilon})$-time combinatorial algorithm for $(2-\delta)$-approximation of min-radius in n-node dense DAGs with $\epsilon, \delta > 0$ if there is no $O(n^{3-\epsilon'})$-time combinatorial algorithm for BMM with $\epsilon' > 0$.

Improving the running time using Fast Matrix Multiplication

In DAGs with small integer edge weights, we further improve the running times for all k in Theorem 1 by applying a result of Zwick in [36] on the runtime of APSP in such graphs. We describe our result in more detail in Section 2. In particular, in DAGs with constant integer edge weights, including unweighted DAGs, our result in the case $k = 2$ is as follows:

▶ **Theorem 4.** *For every $\delta > 0$, there is an $\tilde{O}(\min(m\sqrt{n}/\delta, m^{0.605}n/\delta))$-time $(2+\delta)$-approximation algorithm for min-eccentricities in DAGs with constant integer edge weights.*

There is an $\tilde{O}(\min(m\sqrt{n}, m^{0.605}n))$-time 2-approximation algorithm for min-radius in DAGs with constant integer edge weights.

Min-Diameter

We obtain a 3/2-approximation algorithm for min-diameter in unweighted DAGs, where the approximation factor is conditionally optimal in dense graphs. Specifically, our algorithm improves on the standard APSP runtime for any graph with $m = \omega(n^{1+o(1)})$ edges. This is the first known 3/2-approximation algorithm for min-diameter in dense DAGs that runs faster than the best constant factor approximation algorithm for APSP, which runs in $\tilde{O}(n^\omega)$ time in unweighted directed graphs [36].

▶ **Theorem 5.** *There is an $O(m^{0.414}n^{1.522} + n^{2+o(1)})$-time 3/2-approximation algorithm for min-diameter in unweighted DAGs.*

This algorithm relies on the sparse matrix multiplication algorithm of Yuster and Zwick [35]. In dense graphs with $m = O(n^2)$, its runtime is $O(n^{2.350})$. In relatively sparse graphs, with $m = O(n^{1.154+o(1)})$, the second term dominates, so the runtime is $O(n^{2+o(1)})$.

Our techniques, which mix known diameter techniques with sparse matrix multiplication, are informally as follows: We first construct a covering set, which will intersect any sufficiently large set. We run BFS from all vertices in the covering set, and check whether any min-distances found were large. If not, then for each vertex u, we will define a set of vertices that are relatively "close" to u on its right; if this set is large it will intersect the covering set, allowing us to find paths from u to some vertices to its right, using a "close" vertex in the covering set as a jumping-off point. The remaining vertices w, for which this method did not construct a $u \to w$ path, must have the property that any $u \to w$ path must intersect a relatively small subset of the set of vertices "close" to u (note that this set may have been small to begin with, in which case we can skip the previous step). Symmetrically, for each vertex w we can construct the corresponding relatively small subset of vertices "close" to w on its left, and then to bound the min-distance between u and w we check whether these two small subsets share a vertex in common. We use sparse matrix multiplication to detect this set intersection.

The conditional lower bound of [3] says that if the Orthogonal Vectors Conjecture is true then min-diameter cannot be $(3/2-\delta)$-approximated in truly subquadratic time in sparse graphs. There is no known 3/2-approximation algorithm for min-diameter on DAGs that works faster than APSP, neither for dense graphs nor for sparse graphs. So the question is: Is 3/2 the right bound for inapproximability of min-diameter in DAGs? We answer this question in the affirmative for dense DAGs. Theorem 5 gives the first 3/2-approximation algorithm that works faster than APSP, and it is optimal conditioned on *high dimensional OV* using the same reduction as [3]. High dimensional OV can be used for obtaining lower bounds for dense graphs. In high dimensional OV, the dimension of the vectors can be as big as $O(n)$, and using a simple reduction to Boolean matrix multiplication, the best known algorithm for it is in time $O(n^\omega)$.

High dimensional OV gives a conditional lower bound of $\Omega(n^{\omega-o(1)})$ time for $(3/2-\delta)$-approximation of min-diameter for any $\delta > 0$. Our algorithm gives an upper bound of $O(n^{2.350})$ for $m = \Theta(n^2)$, which is faster than $O(n^\omega)$ for the current best bound on ω. We note while we provide tight results for dense DAGs, the gap between the lower bound and upper bound for computing min-diameter on sparse DAGs is still open.

1.2 Preliminaries

All graphs in this paper are directed graphs. Given a graph G, n denotes the number of vertices and m denotes the number of edges. We will assume $m \geq n-1$ since otherwise all min-eccentricities are infinite, a case that is easily checked. All edge weights are assumed to be nonnegative and polynomial in n; if w_{max} is the maximum edge weight and w_{min} is the minimum edge weight, we let $M = \max\{w_{max}, 1/w_{min}\}$. We write $G[S]$ to denote the subgraph of G induced by vertex set S. For a vertex v, we write $N_D^{\text{in}}(v)$ (respectively, $N_D^{\text{out}}(v)$) to denote the set of vertices u such that $d(u,v) \leq D$ (respectively, $d(v,u) \leq D$).

For $v \in V$ and $W \subseteq V$, we define $d_{\min}(W,v) = d_{\min}(v,W)$ as $\min_{w \in W} d_{\min}(v,w)$, and we define the min-eccentricity of W as $\epsilon(W) = \max_{v \in V} d_{\min}(W,v)$.

Given two sets $U, W \subseteq V$, if every $u \in U$ appears prior to (respectively, after) every $w \in W$ in a topological ordering of the vertices of G, we say that U is the left (respectively, right) of W with respect to the topological ordering. When U or W consists of a single vertex $\{x\}$, we omit the brackets. If $W \subseteq U \subseteq V$, we denote the subset of vertices in U that lie to the left (right) of W by $L_U(W)$ (respectively, $R_U(W)$). If $U = V$, we omit the subscript. A vertex set W is called *topologically consecutive* with respect to a topological ordering if its vertices are consecutive; i.e., if $W = V \setminus (L(W) \cup R(W))$. In general, the relevant topological ordering will be clear, and we will omit reference to it.

Let $\omega(1,r,1)$ be the exponent of the runtime of multiplying $n \times n^r$ by $n^r \times n$ matrices. Let $\omega = \omega(1,1,1)$ be the square matrix multiplication exponent. [6] showed that $\omega < 2.37286$.

For specifying lower bounds, we use the following problems with their corresponding running time conjectures.

Orthogonal Vectors (OV)

Given two lists A, B of n d-dimensional Boolean vectors, determine whether there are vectors $a \in A$ and $b \in B$ such that a and b are *orthogonal*; i.e. there is no $i \in [d]$ such that the ith bits of both a and b are 1. When $d = \Omega(\log n)$, the *OV Conjecture* [31] says that there is no algorithm that can solve the OV problem in time $O(n^{2-\epsilon})$ for any fixed $\epsilon > 0$. The OV Conjecture is implied by the Strong Exponential Time Hypothesis (SETH) [31].

High Dimensional Orthogonal Vectors

In high dimensional OV, the dimension d can be as high as $O(n)$. There is a simple reduction from high dimensional OV to matrix multiplication: Given two lists $A = \{a_1, \ldots, a_n\}, B = \{b_1, \ldots, b_n\}$ of d-dimensional Boolean vectors, let M and N be two $n \times d$ and $d \times n$ Boolean matrices, where $M[i,j] = 1$ if a_i is 1 in bit j, and $N[j,k] = 1$ if b_k is 1 in bit j, for $j = 1, \ldots, d$ and $i, k = 1, \ldots, n$. If MN has a zero entry, the vector pair corresponding to that entry are orthogonal. This gives a $O(n^\omega)$ algorithm for high dimensional OV, and there are no faster algorithms known for it up to polylogarithmic factors. Moreover, OV is equivalent to the problem of distinguishing diameter 2 vs 3 [28], and so high dimensional OV is equivalent to distinguishing diameter 2 vs 3 in dense graphs. A well-known open problem is whether diameter 2 vs 3 can be solved faster than matrix multiplication (see for example [5]). Hence, it is conjectured that high dimensional OV cannot be solved in $O(n^{\omega-\epsilon})$ time for any $\epsilon > 0$.

Hitting Set (HS)

Given two lists $A, B \in \{0,1\}^d$, determine whether there is a vector $a \in A$ that is not orthogonal to any vector $b \in B$. When $d = \Omega(\log n)$, the *Hitting Set Conjecture* [3] says that there is no algorithm that can solve the Hitting Set problem in time $O(n^{2-\delta})$ for any fixed $\delta > 0$.

Boolean Matrix Multiplication (BMM)

We abbreviate multiplying two Boolean $n \times n$ matrices over the (AND, OR)-semiring by BMM. It is conjectured that there is no combinatorial algorithm solving BMM in $O(n^{3-\epsilon})$ time for any fixed $\epsilon > 0$, and the best algebraic algorithm for it is in $O(n^{\omega+o(1)})$ time for $\omega < 2.37286$ [6].

Triangle Detection [32]

Given a tripartite graph $G(A, B, C, E)$ where A, B and C are the three parts of the vertex set and E is the edge set, determine if there are $a \in A$, $b \in B$, and $c \in C$ such that abc is a triangle. Vassilevska W. and Williams [32] showed that considering only combinatorial algorithms, Triangle Detection and BMM are subcubic equivalent, meaning that a truly subcubic combinatorial algorithm in one results in a truly subcubic combinatorial algorithm in the other. Moreover, the best (algebraic) algorithm for Triangle Detection is through BMM. Thus the best running time for Triangle Detection is $O(n^\omega)$, and it is conjectured (see for example [1, 10]) that there is no algorithm faster than $O(n^\omega)$ for detecting a triangle.

2 Min-Eccentricities and Min-Radius

We present two different versions of our min-eccentricity and min-radius approximation algorithms, one which works in general weighted DAGs and is combinatorial and one with a lower runtime upper bound which only works in DAGs with small integer edge weights. The algorithms are identical except in how they compute APSP; the former computes APSP in the standard combinatorial way, while the latter uses Zwick's fast APSP algorithm for graphs with small integer edge weights. Here, $\mu(t)$ is the value satisfying $\omega(1, \mu(t), 1) = 1 + 2\mu(t) - t$.

▶ **Theorem 6** ([36]). *APSP can be computed in $O(n^{2+\mu(t)})$ time in directed graphs with integer edge weights bounded by n^t, where $t < 3 - \omega$.*

Both versions of our algorithms use a common technique to compute min-distances to and from a vertex set. Given a graph G and a vertex set $W \subseteq V$, we construct a graph G' by adding a vertex y and adding weight-0 edges (w, y) for all $w \in W$. We then run Dijkstra into y in G'. We refer to this procedure as *running Dijkstra into W*. The symmetric procedure, in which the weight-0 edges point out of an added vertex y' and we run Dijkstra out of y', will be referred to as *running Dijkstra out of W*. Then for $x \in V$, $d_{\min}(x, W) = \min(d(x, y), d(y', x))$, a value which we can now compute. We added $|W|$ edges and ran Dijkstra in G', so in total the procedure takes time $O(|W| + m \log n) = O(m \log n)$.

Our min-eccentricity and min-radius approximation algorithms will be based on the following proposition. Let $c_k(\tau) = \frac{2^{k-2}(1+\tau)}{2^{k-1}(1+\tau)-\tau}$.

▶ **Proposition 7.** *For any $k \geq 2$, there is an $O(\min(mn^{1/k} \log^2 n, m^{2^{k-1}/(2^k-1)} n \log^2 n))$-time algorithm which takes as input a DAG G and a parameter r, and certifies for each vertex v that $\epsilon(v) > r$ or that $\epsilon(v) \leq kr$.*

In DAGs with integer edge weights bounded by n^t, where $t < 3 - \omega$, there is a version of this algorithm which runs in $O(\min(mn^{1/k} \log^2 n, m^{c_k(\mu(t))} n \log^2 n))$-time.

In [23], Le Gall and Urrutia showed that $\mu = \mu(0) < 0.529$. Thus in DAGs with constant integer edge weights (so that $t = 0$), the runtime of the algorithm of Proposition 7 is $\tilde{O}(\min(mn^{1/k}, m^{c_k(0.529)} n))$ time. When $k = 2$, $c_k(0.529) < 0.605$, leading to the special case stated in Theorem 4.

The algorithms of Proposition 7 will be described and proven correct in Subsection 2.1, and their runtimes will be analyzed in Lemma 13 in Subsection 2.2. Then by binary searching over $r \in [0, Mn]$, these algorithms can be used to obtain the min-eccentricity approximation algorithms of Theorems 8 and 9 and the min-radius approximation algorithms of Theorems 10 and 11.

▶ **Theorem 8.** *Let $k \geq 2$ be an integer. For any $\delta > 0$, there is an $\tilde{O}(\min(mn^{1/k}/\delta, m^{2^{k-1}/(2^k-1)}n/\delta))$-time algorithm which, given a DAG G, outputs for every vertex $v \in V$ an estimate $\epsilon'(v)$ such that $\epsilon(v) \leq \epsilon'(v) < (k+\delta)\epsilon(v)$.*

▶ **Theorem 9.** *Let $k \geq 2$ be an integer. For any $\delta > 0$, there is an $\tilde{O}(\min(mn^{1/k}/\delta, m^{c_k(\mu(t))}n/\delta))$ time algorithm which, given a DAG G with integer edge weights bounded by n^t for $t < 3 - \omega$, outputs for every vertex $v \in V$ an estimate $\epsilon'(v)$ such that $\epsilon(v) \leq \epsilon'(v) < (k+\delta)\epsilon(v)$.*

Proof. First we have all the vertices as "unmarked." We do binary search in $[0, Mn]$ by starting with $r = 1$ in Proposition 7 and incrementing $r' = (1 + \delta/k)r$ at each step. At each step, we run the algorithm given in Proposition 7, and for each unmarked v that is reported as having $\epsilon(v) \leq kr$, we set $\epsilon'(v) = kr$ and mark v. At the end we set $\epsilon'(v) = \infty$ for any remaining unmarked vertices.

Suppose a vertex v was marked at the step corresponding to r. Then $r/(1+\delta/k) < \epsilon(v) \leq kr$, so $\epsilon(v) \leq \epsilon'(v) = kr < (k+\delta)\epsilon(v)$. The binary search adds an $O(\log_{1+\delta/k} Mn) = O((\log Mn)/\delta)$ factor to the runtime. Since $\log Mn$ is polylogarithmic in n, this gives the time bounds stated. ◀

▶ **Theorem 10.** *Let $k \geq 2$ be an integer. There is an $\tilde{O}(\min(mn^{1/k}, m^{2^{k-1}/(2^k-1)}n))$-time algorithm which, given a DAG G, outputs an approximation R' such that if R is the min-radius of G, $R \leq R' < kR$.*

▶ **Theorem 11.** *Let $k \geq 2$ be an integer. There is an $\tilde{O}(\min(mn^{1/k}, m^{c_k(\mu(t))}n))$-time algorithm which, given a DAG G with integer edge weights bounded by n^t for $t < 3 - \omega$, outputs an approximation R' such that if R is the min-radius of G, $R \leq R' < kR$.*

Proof. We do binary search in $[0, Mn]$, running the algorithm given by Proposition 7 at each step as follows: We keep two numbers A_i and B_i at step i which are the lower bound and upper bound to the min-radius R. At step 1 we have $A_1 = 0$ and $B_1 = Mn$. At step i, we have A_i, B_i such that $A_i < R \leq B_i$. Let $C_i = B_i - kA_i$. If C_i is smaller than the minimum positive edge weight, then any path of length at most B_i must have length at most kA_i, so in this case we terminate the binary search and let $R' = kA_i$. We now have $R \leq R' < kR$ as desired.

If C_i is not smaller than the minimum positive edge weight, let $r = A_i + \frac{C_i}{k+1}$, and run the algorithm given by Proposition 7. If the algorithm reports that there is a vertex v with $\epsilon(v) < kr$, then let $A_{i+1} = A_i$ and $B_{i+1} = kr = kA_i + \frac{k}{k+1}C_i$, as we have the min-radius is between A_{i+1} and B_{i+1}. Note that in this case $C_{i+1} = B_{i+1} - kA_{i+1} = \frac{k}{k+1}C_i$. Otherwise, if the algorithm reports that every vertex has $\epsilon(v) \geq r$, then the min-radius is at least $A_{i+1} := r = A_i + \frac{C_i}{k+1}$ and is less than $B_{i+1} := B_i$. In this case $C_{i+1} = B_i - k(A_i + \frac{C_i}{k+1}) = \frac{k}{k+1}C_i$. Thus, at each step, the size of C_i shrinks by a factor of $\frac{k}{k+1}$. Hence, for constant k, the algorithm will in $O(\log Mn)$ steps find bounds A_i, B_i such that C_i is smaller than the minimum positive edge weight. ◀

2.1 Algorithm Description and Correctness

We now describe and prove the correctness of the algorithm of Proposition 7 by induction on k. For convenience, we use $k = 1$ as a base case; in this case we simply run an APSP computation. Our algorithm for $k > 1$ is as follows.

First, topologically sort the vertices and partition them into p consecutive sets $W_1, \dots W_p$ of size $|W_i| = n/p$. The runtime-minimizing value of p will be chosen later.

For each i, run Dijkstra to and from W_i. If $\epsilon(W_i) > r$, then we can report $\epsilon(w) > r$ for all $w \in W_i$. Otherwise, $\epsilon(W_i) \leq r$. In this case, we will apply Claim 12, below, twice. Recall that for $S \subseteq W \subseteq V$, $L_W(S)$ is the set of vertices in W that are to the left of all vertices in S in the topological ordering.

▷ Claim 12. Let $W \subseteq V$ be a topologically consecutive subset of a topologically ordered DAG G, and let r be a parameter such that $\epsilon(W) \leq r$. In $O(m \log^2 n)$ time, one can find a nonempty topologically consecutive subset $S \subseteq W$ such that:

(a) $\epsilon(S) \leq r$.
(b) If $w \in L_W(S)$, $\epsilon(w) > r$.
(c) If $|S| > 1$, all vertices $s \in S$ satisfy $\epsilon(s) > r$.

Proof. We will use a binary search argument to find S. We will induct on an index j. Let $S^0 = W$. Assume that $S^j \subseteq W$ is topologically consecutive, that $\epsilon(S^j) \leq r$, and that for every $w \in L_W(S^j)$, $\epsilon(w) > r$. These all hold for $j = 0$. If $S^j = \{s\}$ consists of a single vertex, let $S = S^j$; then we are done.

Otherwise, let S^j_L be the subset of S^j containing its first $|S^j|/2$ vertices in the topological ordering and let $S^j_R = S^j \setminus S^j_L$. So S^j_L and S^j_R are the left and right halves of S^j, respectively; hence both S^j_L and S^j_R are topologically consecutive. See Figure 1.

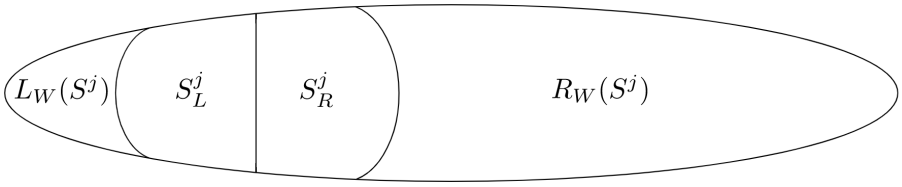


Figure 1 S^j is partitioned into two halves, S^j_L and S^j_R.

Run Dijkstra from S^j_L and from S^j_R. If either of these sets has min-eccentricity at most r, we will continue the induction: If $\epsilon(S^j_L) \leq r$, we let $S^{j+1} = S^j_L$. Then $L_W(S^{j+1}) = L_W(S^j)$, so for every $w \in L_W(S^{j+1})$, $\epsilon(w) > r$. Alternatively, if $\epsilon(S^j_L) > r$ but $\epsilon(S^j_R) \leq r$, we let $S^{j+1} = S^j_R$. Then $L_W(S^{j+1}) = L_W(S^j) \cup S^j_L$, so for every $w \in L_W(S^{j+1})$, $\epsilon(w) > r$.

Otherwise, $\epsilon(S^j_L) > r$ and $\epsilon(S^j_R) > r$. In this case we halt the induction and let $S = S^j$. Every $w \in L_W(S^j) \cup S^j$ satisfies $\epsilon(w) > r$, so S has the properties desired.

At each step, the size of the set S^j halves, so there are at most $\log |W| \leq \log n$ iterations. In each iteration, we perform a constant number of Dijkstras, so the runtime is $O(m \log^2 n)$. ◁

For each i such that $\epsilon(W_i) \leq r$, let S_i be the subset constructed by applying Claim 12 to the set $W = W_i$. For each $w \in L_{W_i}(S_i)$, we report that $\epsilon(w) > r$; this holds by Claim 12b. If S_i consists of a single vertex $\{s\}$, we can determine that for any $v \in L(W_i)$, $d_{\min}(v, s) \leq \epsilon(s) \leq r \leq kr$, by Claim 12a. Otherwise, $|S_i| > 1$, so we report that $\epsilon(s) > r$ for all $s \in S_i$; this holds by Claim 12c.

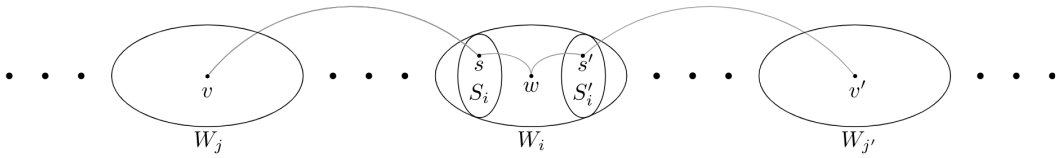


Figure 2 A representation of the $v \to w$ and $w \to v'$ paths, via the sets S_i and S'_i constructed with Claim 12. The outer subpaths are of length $\leq r$, and the inner subpaths are of length $\leq (k-1)r$.

Using a recursive application of our algorithm to the graph $G_i = G[W_i]$, we can certify, for every vertex $w \in W_i$, that $\epsilon_{G_i}(w) > r$ or that $\epsilon_{G_i}(w) \leq (k-1)r$. Consider any $w \in R_{W_i}(S_i)$. If we determined that $\epsilon_{G_i}(w) > r$, we report that $\epsilon(w) > r$; this holds since $\epsilon(w) \geq \epsilon_{G_i}(w)$. Otherwise, consider any $v \in L(W_i)$. Since $\epsilon(S_i) \leq r$, there is some $s \in S_i$ such that $d_{\min}(v, s) = d(v, s) \leq r$. Then since $\epsilon_{G_i}(w) \leq (k-1)r$ and since w is to the right of s in the topological ordering, we have $d_{\min}(v, w) \leq d(v, s) + d(s, w) \leq r + (k-1)r = kr$. See Figure 2.

Thus, our algorithm has certified for each $w \in W_i$ that $\epsilon(w) > r$ or that $d_{\min}(v, w) \leq kr$ for all $v \in L(W_i)$. By a symmetric argument, we can construct the set S'_i obtained by applying Claim 12 to the graph G with the edges reversed; see Figure 2. Then as above we can determine for each $w \in W_i$ that $\epsilon(w) > r$ or that $d_{\min}(w, v') \leq kr$ for all $v' \in R(W_i)$. Since W_i is a topologically consecutive set, $V \setminus W_i = L(W_i) \cup R(W_i)$. So for any $w \in W_i$, if we determine that $d_{\min}(w, v) \leq kr$ for all $v \in L(W_i)$ and for all $v \in R(W_i)$ we report that $\epsilon(w) \leq kr$; otherwise we report $\epsilon(w) > r$.

2.2 Runtime Analysis

In this section we analyze the runtime of the algorithm of Proposition 7, and we give full descriptions of how to prove Theorems 8-11 from Proposition 7 using binary search.

Recall that $c_k(\tau) = \frac{2^{k-2}(1+\tau)}{2^{k-1}(1+\tau)-\tau}$.

▶ **Lemma 13.** *The algorithm of Proposition 7 runs in time* $O(\min(mn^{1/k}\log^2 n, m^{2^{k-1}/(2^k-1)}n\log^2 n))$ *assuming APSP computations are done in* $\tilde{O}(mn)$ *time.*

On graphs with integer edge weights bounded by n^t *for* $t < 3-\omega$*, the algorithm runs in time* $O(\min(mn^{1/k}\log^2 n, m^{mc_k(\mu(t))}n\log^2 n))$*, assuming APSP computations are done in* $O(n^{2+\mu(t)})$ *time using Zwick's fast APSP algorithm [36].*

Proof. To simultaneously analyze both versions of the algorithm, our algorithm's runtime will be described in terms of a placeholder τ, such that APSP computations within the algorithm are done in $O(n^{2+\tau}\log n)$ time. To obtain the runtime bound for general weighted DAGs, we will let $\tau = 1$, and note $c_k(1) = \frac{2^{k-1}}{2^k-1}$. To obtain the runtime bound for DAGs with integer edge weights bounded by n^t for $t < 3-\omega$, we will let $\tau = \mu(t)$.

Topologically sorting the graph takes $O(m\log n)$ time which is absorbed into the final runtime.

In order to use $k = 1$ as a base case, our inductive hypothesis will assume a slightly weaker claim about the runtime: in the inductive step for k, we will assume there is an $O(\min(mn^{1/(k-1)}\log^2 n, n^{2c_{k-1}(\tau)+1}\log^2 n))$-time algorithm which certifies for each $v \in V$ that $\epsilon(v) > r$ or that $\epsilon(v) \leq (k-1)r$. Note that $n^{2c_{k-1}} \geq m^{c_{k-1}}$. Then in the base case where $k = 1$, APSP takes time $O(\min(mn\log n, n^{2+\tau}\log n))$, satisfying the inductive hypothesis.

Consider $k > 1$. Running Dijkstra to and from W_i for each i takes $O(mp \log n)$. It takes time $O(mp \log^2 n)$ to apply Claim 12 twice for each i, to construct sets S_i and symmetric sets S'_i (constructed in the same way as the sets S_i but with left and right swapped, pictured in Figure 2).

We also do recursive calls of our algorithm on at most p subgraphs, induced by sets W_i. Below, we analyze the runtime of the recursive calls in two different ways, giving us two upper bounds on the algorithm's runtime.

Analysis 1. Let $m_i = |E(G[W_i])|$; then note $\sum_i m_i \leq m$. For each i, the recursive call on W_i takes time $O(m_i(n/p)^{1/(k-1)} \log^2 n)$, so in total the recursive calls take time $O(m(n/p)^{1/(k-1)} \log^2 n)$. Let $p = n^{1/k}$, so that $mp = m(n/p)^{1/(k-1)}$. Then the runtime is $O(mn^{1/k} \log^2 n)$.

Analysis 2. Since $|W_i| = n/p$, a recursive call on $G[W_i]$ takes time $O((n/p)^{2c_{k-1}(\tau)+1} \log^2 n)$. We do at most p such calls, so the total runtime of the recursive calls is $O((n/p)^{2c_{k-1}(\tau)} n \log^2 n)$. Now, we choose p so that $mp = (n/p)^{2c_{k-1}(\tau)} n$. Then $m = (n/p)^{2c_{k-1}(\tau)+1}$. Recall that $c_k(\tau) = \frac{2^{k-2}(1+\tau)}{2^{k-1}(1+\tau)-\tau}$ and note that $2c_{k-1}(\tau) + 1 = \frac{2^{k-1}(1+\tau)-\tau}{2^{k-2}(1+\tau)-\tau} = \frac{2c_{k-1}(\tau)}{c_k(\tau)}$. Thus, $m^{c_k(\tau)} = (n/p)^{2c_{k-1}(\tau)}$. So the runtime of the algorithm is $O((n/p)^{2c_{k-1}(\tau)} \cdot n \log^2 n) = O(m^{c_k(\tau)} n \log^2 n)$. Since $m = O(n^2)$, this satisfies the inductive hypothesis. ◀

2.3 Lower Bounds

In this section, using an essentially linear time reduction, we reduce Triangle Detection to $(2-\delta)$-approximation of min-radius.

▶ **Reminder of Theorem 2 .** *If there is a $T(n,m)$-time algorithm for $(2-\delta)$-approximation of min-radius in $O(n)$-node $\tilde{O}(m)$-edge DAGs for some $\delta > 0$, then there is an $\tilde{O}(T(n,m)+m)$-time algorithm for Triangle Detection on graphs with n nodes and m edges.*

Proof. We are going to use two gadgets from previous works:

- DAG gadget [3]: Given a set X of n nodes $v_1, \ldots, v_n$ and a constant integer parameter $t \geq 2$, the gadget creates a DAG $DG_t(X)$ with at most $O(n)$ nodes and $O(n \log n)$ edges such that in the topological order of $DG_t(X)$, $v_i < v_{i+1}$, and for any two nodes of $DG_t(X)$ x, y where $x < y$ in the topological order, $d(x,y) \leq t+1$.
- Connectivity gadget [4]: Let $X = \{v_1, \ldots, v_n\}$, and let $X' = \{v'_1, \ldots, v'_n\}$ be a copy of X, where both X and X' are independent sets. Then we can add a connectivity gadget $U(X)$ along with edges from X to $U(X)$ and from $U(X)$ to X', such that $|U(X)| = O(\log n)$, for all $i \neq j$ we have $d(v_i, v'_j) = 2$, and there is no path from v_i to v'_i.

Now let $G = (A, B, C, E_G)$ be an instance of Triangle Detection, with n nodes and m edges. We create a DAG G^* such that if G has a triangle (YES case), the min-radius of G^* is $t+1$, and if G doesn't have a triangle (NO case), the min-radius of G^* is $2t$. We let t be an integer such that $2 - \delta/2 < \frac{2t}{t+1}$, so that a fast $(2-\delta)$-approximation algorithm is also a fast $(\frac{2t}{t+1} - \delta/2)$-approximation algorithm, and hence it can distinguish min-diameter $t+1$ vs $2t$.

We define G^* as follows: G^* has A, B, and C as part of its vertex set. Let $A'_1, A'_2, \ldots, A'_{t+1}$ be copies of A. Add $E_G(A,B)$ to G^* with edges directed from A to B, and add $E_G(B,C)$ with edges directed from B to C. For any $c \in C$ and $a \in A$, add an edge from c to $a' \in A'_2$ if a and c are attached in G, where a' is the copy of a in A'_2. For each $i = 1, \ldots, t$, connect the copy of a in A'_i to the copy of a in A'_{i+1} for all $a \in A$.

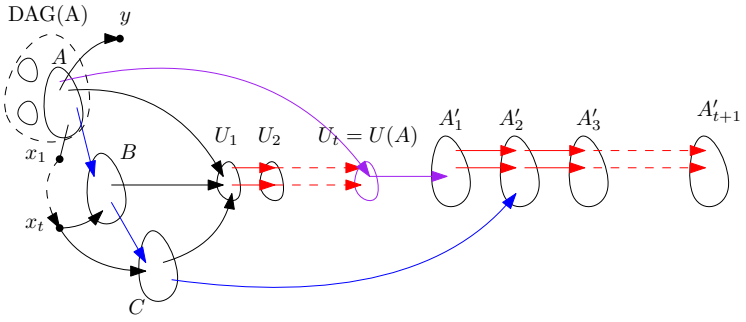


Figure 3 Graph G^* created from the Triangle Detection instance G. Blue edges are edges in G, red edges are between two nodes that are copies of the same vertex. Purple edges are part of the connectivity gadget. Dashed lines are subpaths.

Now we add the two gadgets. Add the connectivity gadget $U(A)$ between A and A'_1. Add two copies of $DG_t(A)$ sharing A, and denote the union of these copies by $DAG(A)$. Also add a node y, and add edges from all nodes in A to y; this guarantees that the center of G^* must be in $DAG(A)$.

To make all nodes in A at distance $t+1$ to A'_1, make $t-1$ copies of $U(A)$, $U_1, \ldots, U_{t-1}$. For each $i = 1, \ldots, t-1$, connect the copy of u in U_i to the copy of u in U_{i+1}, for any $u \in U(A)$, where $U_t = U(A)$. Add edges from all nodes in $A \cup B \cup C$ to all nodes in U_1.

To make all nodes in A at distance $t+1$ to B and C, let $x_1, \ldots, x_t$ be a path of length $t-1$. Connect all nodes of A to x_1, and connect x_t to all nodes of $B \cup C$. See Figure 3 for the construction. Note that G^* is a DAG, with the order of sets of vertices being $DAG(A), y, x_1, \ldots, x_t, B, C, U_1, \ldots, U_{t-1}, U(A), A'_1, \ldots, A'_{t+1}$. Moreover, $G^*[A \cup B \cup C \cup A_2]$ has m edges corresponding to the original edges of G^*, and besides those we only added $O(n \log n)$ edges to G^*. So G^* has $O(n)$ nodes and $O(m + n \log n)$ edges.

We will show that if the Triangle Detection instance is a YES instance, then there is a node $a \in A$ such that $\epsilon(a) = t+1$. If the Triangle Detection instance is a NO instance, then we show that for all nodes in G^*, their min-eccentricity is at least $2t$.

YES case. Let abc be a triangle in G. We show that $\epsilon(a) = t+1$. Note that $d_{\min}(a, \bar{a}) \leq t+1$ for all $\bar{a} \in DAG(A)$. We already know that $d(a, s) \leq t+1$ for any $s \in B \cup C \cup \{x_1, \ldots, x_t, y\}$. For any $u \in U_i$ for $i \leq t+1$, $d(a, u) \leq t+1$ using the path going through $U_1, \ldots U_{i-1}$. Since for any $z' \in A'_1$, there is a $u \in U(A)$ that has an edge to z', we have $d(a, z') \leq t+1$. Now for all $z' \in A'_2$ where z' is a copy of $z \in A$ and $z \neq a$, we have $d(a, z') = 3$ through $U(A)$ and A'_1 (using the edges of the connectivity gadget). For $z = a$, using the triangle edges going from A to B to C, we have that $d(a, z') = 3$. So for all $z' \in A'_2 \cup \ldots \cup A'_{t+1}$, we have $d(a, z') \leq t+1$.

NO case. Suppose that there is no triangle in G. First, note that the min-eccentricities of the vertices outside $DAG(A)$ are infinite, because there is no path between them and y. Moreover, if $z \in DAG(A) \setminus A$, it has a copy $z' \in DAG(A) \setminus A$ (in the other copy of $DG_t(A)$), and there is no path between z and z'. This is because this path must go through A, and since $DAG(A)$ consists of two copies of $DG_t(A)$ sharing A, the set of nodes in A that z has a path to (from) is exactly the same as the set of nodes in A that z' has a path to (from). So there is no $a \in A$ such that that z has a path to a and z' has a path from a.

Now it remains to compute the min-eccentricities of the vertices in A. Let $a \in A$, and let $a'_{t+1} \in A'_{t+1}$ be the copy of a. We show that $d(a, a'_{t+1}) = 2t$. Let P be a shortest path from a to a'_{t+1}. First note that any path from a to a'_{t+1} must go through $a'_2 \in A'_2$, where a'_2

is a copy of a, and we have $d(a'_2, a'_{t+1}) = t - 1$. We also know that there is no path from a to a'_2 using the edges from A to $U(A)$, because this path would need to contain a path between a and $a'_1 \in A_1$ in $G^*[A \cup U(A) \cup A'_1]$, and from the construction of the connectivity gadget there is no such path. If P does not use any $C \times A'_2$ edge, then the path must go through U_i for all i, and hence it is of length $2t$. So if the min-eccentricity of a is smaller than $2t$, the path P uses a $C \times A'_2$ edge ca'_2 for some $c \in C$. If x_1 is on the ac path, then the path goes through x_i for all i, and hence it is of length $2t$. Then x_1 is not on the path, so the ac path must go through B. In particular, there is a $b \in B$ such that $ab, bc \in E(G^*)$. Since $ca'_2 \in E(G^*)$, this implies that abc is a triangle in G, which is a contradiction. So $\epsilon(a) \geq 2t$. ◀

3 Min-diameter

Our min-diameter approximation algorithm relies on Yuster and Zwick's fast sparse matrix multiplication algorithm. Here, we define $\alpha = \max\{0 \leq r \leq 1 \mid \omega(1, r, 1) = 2\}$ and $\beta = \frac{\omega-2}{1-\alpha}$.

▶ **Theorem 14** ([35]). *If A and B are n by n matrices with at most l nonzero entries each, then A and B can be multiplied in $O(l^{\frac{2\beta}{\beta+1}} n^{\frac{2-\alpha\beta}{\beta+1}} + n^{2+o(1)})$ time.*[3]

This sparse matrix multiplication algorithm will be used to prove the following proposition.

▶ **Proposition 15.** *There is an $O(m^{\frac{2\beta}{3\beta+1}} n^{\frac{4\beta+2-\alpha\beta}{3\beta+1}+o(1)} + n^{2+o(1)})$-time algorithm which, given an unweighted DAG G and a parameter D', reports that the min-diameter D of G satisfies $D \leq \frac{3D'}{2}$ or that it satisfies $D > D'$.*

The algorithm of Proposition 15 will be described and proven to work in subsection 3.1, and its runtime will be analyzed in Lemma 18 in subsection 3.2. Then Proposition 15 allows us to obtain the min-diameter approximation algorithm given in Theorem 16 below.

▶ **Theorem 16.** *There is an $O(m^{\frac{2\beta}{3\beta+1}} n^{\frac{4\beta+2-\alpha\beta}{3\beta+1}+o(1)} + n^{2+o(1)})$-time algorithm which, given an unweighted DAG G, outputs an estimate D_0 for its min-diameter D such that $D \leq D_0 < \frac{3D}{2}$.*

Proof. To obtain our approximation D_0, we binary search over D' in $[0, n]$ by applying the algorithm of Proposition 15 logarithmically many times; note that polylogarithmic factors are $n^{o(1)}$ so they do not affect the runtime bound. Let C be the smallest value found in the binary search such that the algorithm reports that $D \leq \frac{3C}{2}$; then $D > C - 1$. Let $D_0 = \frac{3C}{2}$. Then $D \leq D_0 < \frac{3D}{2}$, as desired. ◀

Note that since $\alpha > 0.31389$ [23] and $\omega < 2.37286$ [6], we can use $\beta \simeq 0.5435$. This gives the runtime of $O(m^{0.414} n^{1.522} + n^{2+o(1)})$ stated in Theorem 5.

3.1 Algorithm Description and Correctness

Our algorithm takes as input an unweighted DAG G, an integer D', and a parameter $\epsilon \in [0, 1]$, and reports that $D > D'$ or that $D \leq \frac{3D'}{2}$. (The runtime-minimizing value of ϵ will be determined later.)

[3] To be precise, given known bounds $\alpha \geq a, \omega \leq c$, one can define $b = \frac{c-2}{1-a}$, and then equivalents of Theorem 14 hold for any such pair of values a, b, not just for the "true" values α, β. This is implicit in [35].

If at any point, a BFS finds a pair of vertices at min-distance more than D', the algorithm reports that $D > D'$; hence in what follows we will assume that this does not occur. We initially have all pairs of vertices "unmarked," and mark the pairs for which we know that there is a path from one to the other of length at most $\frac{3D'}{2}$.

The algorithm first takes two preliminary steps: it topologically sorts the graph, and it constructs for each vertex two topologically sorted lists, one of its in-neighbors and one of its out-neighbors.

Our algorithm will then use the greedy set cover algorithm, described in the following lemma. This lemma, and a related randomized version, are standard techniques used in graph distance algorithms (see for example [5, 28, 13, 3]). A proof may be found in [29].

▶ **Lemma 17.** *Let $|V| = n$, let $p = O(n)$, and let $X_1, \ldots X_p \subseteq V$ be sets of size $|X_i| \geq n^\epsilon$ for $\epsilon \in [0, 1]$. Then there is an $O(n^{1+\epsilon})$-time algorithm which constructs a set $S \subseteq V$ of size $\tilde{O}(n^{1-\epsilon})$ such that $S \cap X_i \neq \varnothing$ for all i.*

For any $u \in V$, if $|N^{\text{out}}_{D'/2}(u)| < n^\epsilon$ let $X_u = N^{\text{out}}_{D'/2}(u)$ and otherwise let X_u be the left-most n^ϵ vertices in $N^{\text{out}}_{D'/2}(u)$. So in particular, $|X_u| \leq n^\epsilon$. We can compute X_u as follows: we maintain a list of the $\leq n^\epsilon$ left-most vertices we have found so far that are at distance $< D'/2$ from u. At each step, for each vertex in the list, we consider its left-most out-neighbor that is not yet in our set; we add the left-most such out-neighbor to the set. We halt when there are no more such out-neighbors not in our set, or after adding n^ϵ vertices to our set. Likewise, for any $w \in V$, let $Y_w = N^{\text{in}}_{D'/2}(w)$ if $|N^{\text{in}}_{D'/2}(w)| < n^\epsilon$, and otherwise let Y_w consist of the right-most n^ϵ vertices in $N^{\text{in}}_{D'/2}(w)$. We can compute the sets Y_w in a manner symmetric to how we computed the sets X_u. Then we can use Lemma 17 to construct a set S of size $\tilde{O}(n^{1-\epsilon})$ such that for all u having $|N^{\text{out}}_{D'/2}(u)| \geq n^\epsilon$, $S \cap X_u$ is nonempty, and for all w having $|N^{\text{in}}_{D'/2}(w)| \geq n^\epsilon$, $S \cap Y_w$ is nonempty.

Run BFS into and out of every $s \in S$. We may assume that $d_{\min}(s, x) \leq D'$ for all $s \in S, x \in V$.

We will construct matrices A and B with rows and columns indexed by vertices in V, as follows: For each vertex $t \in X_u$, let $A[u, t] = 1$. For each vertex $t \in Y_w$, let $B[t, w] = 1$. Multiply A and B using the sparse matrix multiplication algorithm of Theorem 14.

Now, we will consider any pair of vertices (u, w) where u is to the left of w, $u \in R(N^{\text{in}}_{D'/2}(w) \cap S)$, and $w \in L(N^{\text{out}}_{D'/2}(u) \cap S)$. We have that $(A \cdot B)[u, w] > 0$ if and only if $d(u, w) \leq D'$. Indeed, if $d(u, w) \leq D'$, then there is some intermediate vertex t such that $d(u, x) \leq D'/2$ and $d(x, w) \leq D'/2$. Suppose that $t \notin X_u$. Then since X_u is defined as the left-most n^ϵ vertices in $N^{\text{out}}_{D'/2}(u)$, this implies that $|N^{\text{out}}_{D'/2}(u)| > n^\epsilon$ and hence that $|X_u| = n^\epsilon$. Then there is some $s \in S \cap X_u$. Since $t \notin X_u$, t is to the right of all vertices in X_u, and in particular t is to the right of s. This implies $t \notin L(N^{\text{out}}_{D'/2}(u) \cap S)$. But since $w \in L(N^{\text{out}}_{D'/2}(u) \cap S)$ and t lies between u and w, this is a contradiction. Thus, t must be in X_u, and by symmetry, t is in Y_w. So $A[u, t] = 1$ and $B[t, w] = 1$, meaning $(A \cdot B)[u, w] > 0$. Likewise, if $(A \cdot B)[u, w] > 0$, then there exists $t \in X_u \cap Y_w$ such that $d(u, t) \leq D'/2$ and $d(t, w) \leq D'/2$, so $d(u, w) \leq D'$. Therefore, we will mark all pairs (u, w) such that $(A \cdot B)[u, w] > 0$.

Now, consider any $u \in V$ and any $w \notin L(N^{\text{out}}_{D'/2}(u) \cap S)$ to the right of u. We mark the pair (u, w). If such a w exists, then there is some $s \in N^{\text{out}}_{D'/2}(u) \cap S$ such that s is to the left of or is equal to w. By assumption, $d(s, w) \leq D'$, so $d(u, w) \leq d(u, s) + d(s, w) \leq D'/2 + D' = \frac{3D'}{2}$. By a symmetric argument, for any $w \in V$ and any $u \notin R(N^{\text{in}}_{D'/2}(w) \cap S)$ to the left of w, we have that $d(u, w) \leq \frac{3D'}{2}$, so again we mark any such pair (u, w). Thus, since we have assumed that $\epsilon(s) \leq \frac{3D'}{2}$ for all $s \in S$, the algorithm will mark all pairs of vertices $u, w \in V$ except those for which we have simultaneously that $u \in R(N^{\text{in}}_{D'/2}(w) \cap S)$ and $w \in L(N^{\text{out}}_{D'/2}(u) \cap S)$.

Finally, check whether there exists an unmarked pair (u, w). If so, report that $D > D'$. Otherwise, report that $D \leq \frac{3D'}{2}$.

3.2 Runtime Analysis

Here we analyze the runtime of the algorithm of Proposition 15.

▶ **Lemma 18.** *The algorithm of Proposition 15 runs in time* $\tilde{O}(m^{\frac{2\beta}{3\beta+1}} n^{\frac{4\beta+2-\alpha\beta}{3\beta+1}+o(1)} + n^{2+o(1)})$.

Proof. Topologically sorting the graph takes $O(m \log n)$ time which is absorbed into the final runtime. Constructing for each vertex topologically ordered lists of its in-neighbors and out-neighbors can be done in time $\tilde{O}(n^2)$.

Computing the covering set S takes time $\tilde{O}(n^{1+\epsilon})$ and running BFS from its vertices takes time $O(n^{1-\epsilon} m \log n)$. Checking for each pair (u, w) whether $u \in L(N^{\text{out}}_{D'/2}(u) \cap S)$ and $w \in L(N^{\text{out}}_{D'/2}(u) \cap S)$ can be done in $\tilde{O}(n^2)$ time.

For a fixed u, to compute X_u, we maintain a list of the at most n^ϵ left-most vertices we have found that are at distance $< D'/2$ from u. For each vertex, we store its left-most out-neighbor that is not yet in our set. At each step, we find the left-most such out-neighbor of any vertex in the list; this takes time $O(n^\epsilon)$, and updating the list to reflect that this out-neighbor has been added to our set takes time $O(n^\epsilon)$. At each step we add a vertex to our set X_u, so there are at most $O(n^\epsilon)$ steps. Hence, constructing X_u for a fixed u takes $O(n^{2\epsilon})$ time. Then constructing all sets X_u, Y_w takes $O(n^{1+2\epsilon})$ time altogether.

Finally, note that there are at most n^ϵ 1s in each row of A, since we only set $A[u, t] = 1$ if $t \in X_u$. Thus, A contains at most $n^{1+\epsilon}$ 1s. By symmetry, the same holds for B. Then multiplying A and B can be done in time $O(n^{(1+\epsilon)\frac{2\beta}{\beta+1}+\frac{2-\alpha\beta}{\beta+1}+o(1)} + n^{2+o(1)})$, using Yuster and Zwick's fast sparse matrix multiplication (Theorem 14).

Then the total runtime is:

$$\tilde{O}(n^{1-\epsilon}m + n^{1+2\epsilon} + n^{(1+\epsilon)\frac{2\beta}{\beta+1}+\frac{2-\alpha\beta}{\beta+1}+o(1)} + n^{2+o(1)})$$

Let γ be the largest value such that $n^\gamma = O(m)$. Let $\epsilon = \frac{\alpha\beta+(\beta+1)(\gamma-1)}{3\beta+1}$; this value is chosen because it sets the first and third terms in the above runtime equal (up to $n^{o(1)}$ factors), hence asymptotically minimizing their sum. Substituting the value of ϵ and simplifying, the runtime of the algorithm is:

$$\tilde{O}(n^{\frac{2\beta}{3\beta+1}\gamma+\frac{4\beta+2-\alpha\beta}{3\beta+1}+o(1)} + n^{\frac{2\beta+2}{3\beta+1}\gamma+\frac{\beta-1+2\alpha\beta}{3\beta+1}} + n^{2+o(1)})$$

We note that $3\beta - 3\alpha\beta > 3(\omega - 2) \geq 0 > -1$, giving:

$$4\beta + 2 - \alpha\beta > 2 + (\beta - 1 + 2\alpha\beta) \geq 2\gamma + (\beta - 1 + 2\alpha\beta)$$

Thus, the first term of the above runtime dominates the second. Substituting $n^\gamma = O(m)$, and noting that the polylogarithmic factors in the runtime are of order $n^{o(1)}$, the runtime is $O(m^{\frac{2\beta}{3\beta+1}} n^{\frac{4\beta+2-\alpha\beta}{3\beta+1}+o(1)} + n^{2+o(1)})$, as desired. ◀

References

1 Amir Abboud, Arturs Backurs, and Virginia Vassilevska Williams. If the current clique algorithms are optimal, so is Valiant's parser. *SIAM Journal on Computing*, 47(6):2527–2555, 2018.

2 Amir Abboud, Fabrizio Grandoni, and Virginia Vassilevska Williams. Subcubic equivalences between graph centrality problems, APSP and diameter. In *Proceedings of the Twenty-Sixth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2015, San Diego, CA, USA, January 4-6, 2015*, pages 1681–1697, 2015.

3 Amir Abboud, Virginia Vassilevska Williams, and Joshua R. Wang. Approximation and fixed parameter subquadratic algorithms for radius and diameter in sparse graphs. In *Proceedings of the Twenty-Seventh Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2016, Arlington, VA, USA, January 10-12, 2016*, pages 377–391, 2016.

4 Udit Agarwal and Vijaya Ramachandran. Fine-grained complexity for sparse graphs. In *Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing*, pages 239–252, 2018.

5 D. Aingworth, C. Chekuri, P. Indyk, and R. Motwani. Fast estimation of diameter and shortest paths (without matrix multiplication). *SIAM J. Comput.*, 28(4):1167–1181, 1999.

6 Josh Alman and Virginia Vassilevska Williams. A refined laser method and faster matrix multiplication. In *Proceedings of the 32nd Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2021*, 2020.

7 B. Ben-Moshe, B. K. Bhattacharya, Q. Shi, and A. Tamir. Efficient algorithms for center problems in cactus networks. *Theoretical Computer Science*, 378(3):237–252, 2007.

8 P. Berman and S. P. Kasiviswanathan. Faster approximation of distances in graphs. In *Proc. WADS*, pages 541–552, 2007.

9 Michele Borassi, Pierluigi Crescenzi, Michel Habib, Walter A. Kosters, Andrea Marino, and Frank W. Takes. Fast diameter and radius BFS-based computation in (weakly connected) real-world graphs: With an application to the six degrees of separation games. *Theoretical Computer Science*, 586:59–80, 2015.

10 Karl Bringmann and Philip Wellnitz. Clique-based lower bounds for parsing tree-adjoining grammars. *arXiv preprint*, 2018. `arXiv:1803.00804`.

11 Chris Calabro, Russell Impagliazzo, and Ramamohan Paturi. The complexity of satisfiability of small depth circuits. In *International Workshop on Parameterized and Exact Computation*, pages 75–85. Springer, 2009.

12 T. M. Chan. All-pairs shortest paths for unweighted undirected graphs in $o(mn)$ time. *ACM Transactions on Algorithms*, 8(4):34, 2012.

13 Shiri Chechik, Daniel H. Larkin, Liam Roditty, Grant Schoenebeck, Robert Endre Tarjan, and Virginia Vassilevska Williams. Better approximation algorithms for the graph diameter. In *Proceedings of the Twenty-Fifth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2014, Portland, Oregon, USA, January 5-7, 2014*, pages 1041–1052, 2014.

14 V. Chepoi, F. Dragan, and Y. Vaxès. Center and diameter problems in plane triangulations and quadrangulations. In *Proc. SODA*, pages 346–355, 2002.

15 V. Chepoi and F. F. Dragan. A linear-time algorithm for finding a central vertex of a chordal graph. In *ESA*, pages 159–170, 1994.

16 F. R. K. Chung. Diameters of graphs: Old problems and new results. *Congr. Numer.*, 60:295–317, 1987.

17 D.G. Corneil, F.F. Dragan, M. Habib, and C. Paul. Diameter determination on restricted graph families. *Discr. Appl. Math.*, 113:143–166, 2001.

18 L. Cowen and C. Wagner. Compact roundtrip routing for digraphs. In *SODA*, pages 885–886, 1999.

19 Mina Dalirrooyfard, Virginia Vassilevska Williams, Nikhil Vyas, Nicole Wein, Yinzhan Xu, and Yuancheng Yu. Approximation algorithms for min-distance problems. In *46th International Colloquium on Automata, Languages, and Programming (ICALP 2019)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2019.

20 D. Dvir and G. Handler. The absolute center of a network. *Networks*, 43:109–118, 2004.

21 D. Eppstein. Subgraph isomorphism in planar graphs and related problems. *J. Graph Algorithms and Applications*, 3(3):1–27, 1999.

22 Silvio Frischknecht, Stephan Holzer, and Roger Wattenhofer. Networks cannot compute their diameter in sublinear time. In *Proceedings of the Twenty-Third Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1150–1162. SIAM, 2012.

23 François Le Gall and Florent Urrutia. Improved rectangular matrix multiplication using powers of the coppersmith-winograd tensor. In *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1029–1046. SIAM, 2018.

24 S.L. Hakimi. Optimum location of switching centers and absolute centers and medians of a graph. *Oper. Res.*, 12:450–459, 1964.

25 R. Impagliazzo and R. Paturi. On the complexity of k-SAT. *Journal of Computer and System Sciences*, 62(2):367–375, 2001.

26 Seth Pettie. A faster all-pairs shortest path algorithm for real-weighted sparse graphs. In *International Colloquium on Automata, Languages, and Programming*, pages 85–97. Springer, 2002.

27 Seth Pettie and Vijaya Ramachandran. Computing shortest paths with comparisons and additions. In *Proceedings of the thirteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 267–276, 2002.

28 Liam Roditty and Virginia Vassilevska Williams. Fast approximation algorithms for the diameter and radius of sparse graphs. In *Proceedings of the forty-fifth annual ACM symposium on Theory of computing*, pages 515–524, 2013.

29 Virginia Vassilevksa Williams, Nike Sun, and Nishith Khandwala. Lecture notes in graph algorithms (hitting sets, APSP), October 2016. URL: `http://theory.stanford.edu/~virgi/cs267/lecture5.pdf`.

30 O. Weimann and R. Yuster. Approximating the diameter of planar graphs in near linear time. In *Proc. ICALP*, 2013.

31 Ryan Williams. A new algorithm for optimal 2-constraint satisfaction and its implications. *Theoretical Computer Science*, 348(2-3):357–365, 2005.

32 Virginia Vassilevska Williams and R Ryan Williams. Subcubic equivalences between path, matrix, and triangle problems. *Journal of the ACM (JACM)*, 65(5):1–38, 2018.

33 C. Wulff-Nilsen. Wiener index, diameter, and stretch factor of a weighted planar graph in subquadratic time. *Technical report, University of Copenhagen*, 2008.

34 Raphael Yuster. Computing the diameter polynomially faster than APSP. *arXiv preprint*, 2010. `arXiv:1011.6181`.

35 Raphael Yuster and Uri Zwick. Fast sparse matrix multiplication. *ACM Trans. Algorithms*, 1(1):2–13, 2005. `doi:10.1145/1077464.1077466`.

36 U. Zwick. All pairs shortest paths using bridging sets and rectangular matrix multiplication. *J. ACM*, 49(3):289–317, 2002.

On Greedily Packing Anchored Rectangles

Christoph Damerius
University of Hamburg, Germany

Dominik Kaaser
University of Hamburg, Germany

Peter Kling
University of Hamburg, Germany

Florian Schneider
University of Hamburg, Germany

Abstract

Consider a set P of points in the unit square $\mathcal{U} = [1, 0)$, one of them being the origin. For each point $p \in P$ you may draw an axis-aligned rectangle in $\mathcal{U}$ with its lower-left corner being p. What is the maximum area such rectangles can cover without overlapping each other?

Freedman [18] posed this problem in 1969, asking whether one can always cover at least 50% of $\mathcal{U}$. Over 40 years later, Dumitrescu and Tóth [12] achieved the first constant coverage of 9.1%; since then, no significant progress was made. While 9.1% might seem low, the authors could not find any instance where their algorithm covers less than 50%, nourishing the hope to eventually prove a 50% bound. While we indeed significantly raise the algorithm's coverage to 39%, we extinguish the hope of reaching 50% by giving points for which its coverage stays below 43.3%.

Our analysis studies the algorithm's average and worst-case density of so-called tiles, which represent the staircase polygons in which a point can freely choose its maximum-area rectangle. Our approach is comparatively general and may potentially help in analyzing related algorithms.

2012 ACM Subject Classification Theory of computation → Computational geometry

Keywords and phrases lower-left anchored rectangle packing, greedy algorithm, charging scheme

Digital Object Identifier 10.4230/LIPIcs.ICALP.2021.61

Category Track A: Algorithms, Complexity and Games

Related Version *Full Version*: `https://arxiv.org/abs/2102.08181`

1 Introduction

The Lower-Left Anchored Rectangle Packing (LLARP) problem considers a finite set $P \subseteq \mathcal{U} := [0, 1)^2$ of *input points* with $(0, 0) \in P$. The goal is to find a set of non-empty, axis-aligned interior-disjoint rectangles $(r_p)_{p \in P}$ with p being the lower-left corner of $r_p \subseteq \mathcal{U}$ and such that their total area $\sum_{p \in P} |r_p|$ is maximized.

This problem was first introduced by Freedman [18, Unsolved Problem 11, page 345] in 1969. He asked the question whether, for any point set P, the rectangles can always be chosen such that they cover at least 50% of $\mathcal{U}$. It is easy to see that this is the best one can hope for, since putting n equally spaced points along the ascending diagonal of $\mathcal{U}$ yields a maximum coverable area of $1/2 + o(1)$ for $n \to \infty$.

Over the years, the LLARP problem reoccurred in the form of geometric challenges [14] and in miscellaneous books and journals about mathematical puzzles [19, 20, 21]. Still, it took more than 40 years until the first constant lower bound was established: Dumitrescu and Tóth [12] considered a natural greedy algorithm, called GreedyPacking, and proved that it achieves a coverage of 9.1%.

This caused a surge of interest in this old problem, resulting in numerous findings for variants or special cases of the problem (see Section 1.1). Since then, no further significant progress was made towards the original question[1], and even the question whether a maximum area covering can be found in polynomial time remains elusive.

While [12] themselves observed that "*a sizable gap to the conjectured* 50% *remains*", they were unable to find instances where their algorithm *does not* reach 50%. This led them and others to conjecture a much better quality of their algorithm, making it a natural candidate to answer Freedman's question positively, albeit [12] also mentioned that "*obtaining substantial improvements probably requires new ideas*".

Our results indeed attest the greedy algorithm a much better coverage of 39%. However, at the same time, we show that there are instances where the coverage stays below 43.3%.

1.1 Related Work

LLARP falls into the class of geometric packing problems, where a typical question is how much of a container can be covered using a set of geometric shapes in two or more dimensions. We concentrate on two-dimensional packing problems with rectangular containers and shapes.

Complexity of LLARP. The GreedyPacking algorithm by Dumitrescu and Tóth [12] considers the input points step by step from top-right to bottom-left, always choosing the maximum-area rectangle. They showed that this achieves the same worst-case coverage as an algorithm called TilePacking. The latter partitions the unit square into staircase-shaped *tiles*, one per input point, and chooses a maximal rectangle within each tile (see Section 2 for the formal algorithm description).

While the complexity of LLARP remains unknown, [12] also showed that there is an order of the input points for which the greedy algorithm achieves an optimal packing (albeit of unknown value); how to find that ordering remains unclear. [7] studied the combinatorial structure of optimal solutions, proving that the worst-case number of maximal rectangle packings is exponential in the number of input points.

LLARP Variants. After [12], a series of papers studied special cases and variants of LLARP. [6] allowed rectangles to be anchored in any of the four corners and showed that here the worst-case coverage lies in $[7/12, 2/3]$ and in $[5/32, 7/27]$ if the rectangles are restricted to squares. [4] showed that the union of all (possibly overlapping) squares covers at least $1/2$ and proved that finding a maximum corner-anchored square packing is NP-hard. Interestingly, there is only one other LLARP-variant known to be NP-hard, namely if the rectangle's anchors lie in their center [5]. Other results consider specific classes of input points, like points with certain ascending/descending structures [8] or points that lie on the unit square's boundary (for corner-anchored rectangles) [9].

Further Related Problems and Applications. Further related problems include the maximum weight independent set of rectangles problem [2, 10, 1] (which was used, e.g., in [5] to derive a PTAS for center-anchored rectangle packings) or geometric knapsack [3, 17] and strip packing problems [15]. In contrast to LLARP and its variants, the size of the objects to be packed is typically part of the input and object placement is less constrained.

[1] A very recent, still unpublished result slightly raised the greedy algorithm's coverage to 10.39% [13].

Note that LLARP-like problems are not of pure theoretical interest, but have applications in, e.g., map labeling. Here, rectangular text labels must be placed under certain constraints (e.g., labels might be scalable but require a fixed ratio and must be placed at a specific anchor) within a given container. We refer to the relatively recent survey [16] for details.

1.2 Our Contribution and Techniques

We analyze the greedy algorithm TilePacking from [12] (formally described in Section 2). From a high-level view, TilePacking partitions the unit square into staircase-shaped *tiles*, each anchored at an input point, and chooses an area-maximal rectangle in each tile. A natural way to analyze such an algorithm is to consider the tiles' densities (the ratio between their area-maximal rectangles and their own area) and prove a lower bound on the *average* tile density (which immediately yields the covering guarantee).

Dumitrescu and Tóth [12] follow this approach by defining suitable charging areas C_t for each tile t (trapezoids below/beneath the tile). We also use such a charging scheme, but rely on a much more complex charging area which we refer to as a tile's *crown*. But instead of directly analyzing a tile's charging area, we first extract the critical properties that determine the charging scheme's quality. This general approach (described in Section 3) requires a bound ξ on the tile's *charging ratio* $|C_t|/|t|$ together with some simple properties (basically a form of local convexity characterizing the average tile density).

We derive such a charging ratio bound ξ_s and describe simple, symmetric tiles for which it is tight (Figures 8 and 9). We then take an arbitrary tile and show how to gradually transform it into one of these tiles without increasing its charging ratio. This establishes that ξ_s is indeed a charging ratio bound and allows us to conclude the following theorem.

▶ **Theorem 1.** *Given any set of input points, TilePacking covers at least* 39% *of* $\mathcal{U}$.

While the aforementioned transformations to worst-case tiles require some care, we showcase the versatility of our approach by first proving a slightly weaker bound of only 25% (Section 4.2). Its analysis is not only much simpler but, in fact, takes us halfway to Theorem 1, as the used charging ratio bound ξ_w (Proposition 11) is tight for high-density tiles and all that remains is to refine our charging ratio bound for low-density tiles (Proposition 15).

Our second major result constructs an input instance (depicted in Figure 14) for which TilePacking covers significantly less than 50% of the unit square.

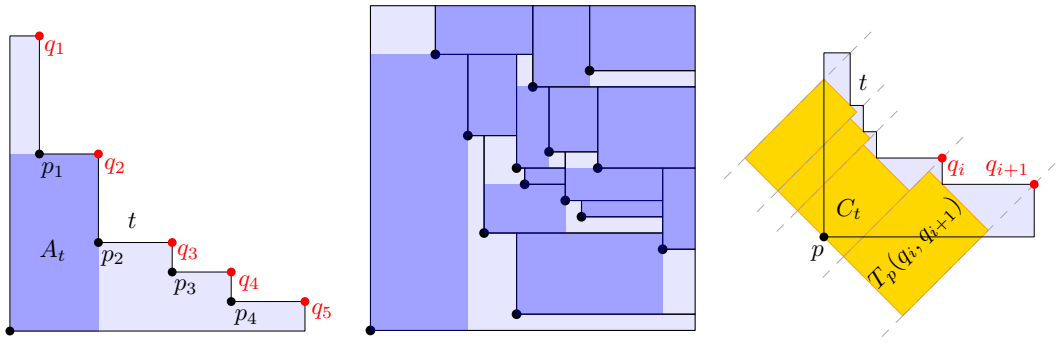
▶ **Theorem 2.** *There is a set P of input points for which TilePacking covers at most* 43.3% *of the unit square.*

By the aforementioned worst-case equivalence of TilePacking and GreedyPacking, both Theorems 1 and 2 also hold for the latter.

2 Preliminaries and Algorithm Description

Let $\mathcal{U} \coloneqq [0,1)^2$ denote the unit square. For a point $p \in \mathbb{R}^2$ define $x(p)$ and $y(p)$ as the x- and y-coordinates of p, respectively. For two points $p, p' \in \mathbb{R}^2$ we use the notation $p \preceq p'$ to indicate that $x(p) \leq x(p')$ and $y(p) \leq y(p')$. Similarly, $p \prec p'$ means that $x(p) < x(p')$ and $y(p) < y(p')$. The relations "$\succeq$" and "$\succ$" are defined analogously. For a set S we denote its closure by $\overline{S}$. If S is measurable, we use $|S|$ to denote its area.

To simplify some geometric arguments, we use the following line-notation: We define the line $\ell_q^{\diagup} \subseteq \mathbb{R}^2$ as the line through $q \in \mathbb{R}^2$ of slope $+1$. Similarly, we define the lines ℓ_q^{-}, $\ell_q^{\diagdown}$, and $\ell_q^{|}$ through q with slope 0, -1, and ∞, respectively. For lines of type $R \in \{ \diagup, | \}$, we



(a) Staircase points p_i, q_j and rectangle A_t.

(b) A tile packing produced by TilePacking.

(c) A tile t, its crown C_t, and a tower $T_p(q_i, q_{i+1})$.

Figure 1 Tiles, packings, crowns, and towers. In our figures, tiles are shaded light blue. Upper and lower stair case points are shown in red and black, respectively. A dark blue rectangle represents a (maximal) rectangle of a tile. Crowns are shown in yellow and towers are possibly labeled.

write $\ell_q^R < \ell_{q'}^R$ if $\ell_{q'}^R = \ell_q^R + (x, 0)$ (using element-wise addition) with $x > 0$ and say ℓ_q^R *is left of* $\ell_{q'}^R$. Similarly, for lines of type $R \in \{ \diagdown, - \}$ we write $\ell_q^R < \ell_{q'}^R$ if $\ell_{q'}^R = \ell_q^R + (0, y)$ with $y > 0$ and say ℓ_q^R *is below* $\ell_{q'}^R$. Analogous definitions apply for "$>$", "$\leq$", and "$\geq$".

Input Sets in General Position. Remember the problem description from Section 1. We say that the input set P is *in general position* if there are no two (different) points $p, p' \in P$ with $x(p) = x(p')$, $y(p) = y(p')$, or $x(p) + y(p) = x(p') + y(p')$. That is, no two points may share an x- or y-coordinate and may not lie on the same diagonal of slope -1. W.l.o.g., we restrict P to be in general position (see the full version [11] for why this is ok).

Tiles and Tile Packings. A *tile* t is a staircase polygon in $\mathcal{U}$ (see Figure 1a). More formally, t is defined using its *anchor* $p \in \mathcal{U}$ and a set of k *upper staircase points* $\Gamma_t := \{ q_1, q_2, \dots, q_k \} \subseteq \overline{\mathcal{U}}$ ordered by increasing x-coordinate and such that $q_i \succ p$ for all q_i as well as $q_i \not\preceq q_j$ for all $q_i \neq q_j$. With this we define $t = \{ q \in \mathcal{U} \mid q \succeq p \wedge \exists q' \in \Gamma_t \colon q \prec q' \}$. A point $p_i = \big(x(q_{i-1}), y(q_i)\big)$ is called a *lower staircase point*. We define $A_t \subseteq t$ as an (arbitrary) area-maximal rectangle in t and $\rho_t := |A_t|/|t|$ as the tile's *density*. For indexed upper staircase points q_i we often use the shorthands $x_i := x(q_i)$ and $y_i := y(q_i)$.

If p and Γ_t do not adhere to $q_i \succ p$ and $q_i \not\preceq q_j$, but only to the weaker requirements $q_i \succeq p$, $q_i \not\preceq q_j$ for all $q_i \neq q_j$), then we say that t is *degenerate*. We show in Lemma 13 that we can transform such tiles into non-degenerate ones without affecting our arguments.

The *hyperbola* of t is $h_t := \{ (x + x(p), y + y(p)) \in \mathbb{R}^2_{>0} \mid y = |A_t|/x \}$. Note that all upper staircase points lie between p and h_t. Moreover, the points from $\Gamma_t \cap h_t$ span all area-maximal rectangles in t. If, $p = (0, 0)$ and $|A_t| = 1$, then t is called *normalized*.

A *tile packing* of the unit square is a partition of $\mathcal{U}$ into tiles. In particular, $\sum_{t \in \mathcal{T}} |t| = |\mathcal{U}| = 1$. We use $A(\mathcal{T}) := \sum_{t \in \mathcal{T}} |A_t|$ to denote the area covered by choosing an area-maximal rectangle A_t for each tile t (the *area covered by* $\mathcal{T}$).

A Greedy Tile Packing Algorithm. Let us revisit the algorithm TilePacking by Dumitrescu and Tóth [12]. TilePacking processes the points P from top-right to bottom-left. More formally, it orders $P = \{ p_1, p_2, \dots, p_n \}$ such that $\ell^{\diagdown}_{p_i} \geq \ell^{\diagdown}_{p_{i+1}}$. It then defines for each $p_i \in P$ the tile $t_i := \{ q \in \mathcal{U} \mid q \succeq p_i \} \setminus \bigcup_{j=1}^{i-1} t_j$, yielding a tile packing $\mathcal{T} = \{ t_1, t_2, \dots, t_n \}$.

To build its solution to LLARP, TilePacking picks for each $p \in P$ the rectangle r_p as an (arbitrary) area-maximal rectangle $A_t \subseteq t$ in the tile t containing p. Thus, the total area covered by TilePacking is $A(\mathcal{T})$. Figure 1b illustrates the resulting tile packing.

Note that, by this construction, the lower staircase points of each tile t are input points. Moreover, as already mentioned in [12], for each tile we can define a certain *exclusive area* that does not contain an input point.

▶ Observation 3. Consider the tile packing $\mathcal{T}$ produced by TilePacking for a set P of input points. Fix a tile $t \in \mathcal{T}$ and let $p \in P$ denote its anchor point. Then the tile's *exclusive area* $E_t \coloneqq \{\, q \in \mathbb{R}^2 \mid \ell_q^\rangle > \ell_p^\rangle \wedge \exists q' \in \Gamma_t \colon q \prec q' \,\}$ does not contain any point from P.

This observation follows by noting that any such input point $p' \in E_t$ would be processed before p by TilePacking and "shield" at least one upper staircase point $q' \in \Gamma_t$ from p, preventing it from becoming an upper staircase point of tile t.

3 A General Approach for Lower Bounds

Here we present a general approach to derive lower bounds for the area covered by a given tile packing $\mathcal{T}$. Our approach relies on a suitable *charging scheme* $(c_t)_{t\in\mathcal{T}}$ that charges the area of each tile $t \in \mathcal{T}$ to a *charging area* $c_t > 0$.

▶ **Definition 4.** For a given charging scheme, we define $c^* \coloneqq \sum_{t\in\mathcal{T}} c_t$ as the total charged area and $c_t/|t|$ as the *charging ratio* of tile t. We call a function $\xi \colon (0,1] \to \mathbb{R}_{\geq 0}$ a *charging ratio bound* with *critical density* $\rho^* \in (0,1]$ if

1. ξ is point-convex at ρ^* with $\xi'(\rho^*) < 0$,
2. $\xi(\rho^*) \geq c^*$ and
3. for any $t \in \mathcal{T}$: $\xi(\rho_t) \leq c_t/|t|$.

Note that a function $f \colon I \to \mathbb{R}, I \subseteq \mathbb{R}$ is said to be *point-convex* at $x \in I$ if f is differentiable at x and the tangent t of f at x satisfies $t(x) \leq f(x)$ for all $x \in I$.

The following lemma uses a charging ratio bound to show that ρ^* is a lower bound on $A(\mathcal{T})$.

▶ **Lemma 5.** *Consider a tile packing $\mathcal{T}$ with a charging scheme $(c_t)_{t\in\mathcal{T}}$ together with a charging ratio bound ξ with critical density ρ^*. Then $A(\mathcal{T}) \geq \rho^*$.*

Proof. Since ξ is point-convex in ρ^*, the tangent $\tau(\rho) \coloneqq \xi(\rho^*) + \xi'(\rho^*) \cdot (\rho - \rho^*)$ of ξ in ρ^* satisfies $\tau(\rho) \leq \xi(\rho)$ for all $\rho \in (0,1]$. Using $A(\mathcal{T}) = \sum_{t\in\mathcal{T}} |A_t| = \sum_{t\in\mathcal{T}} |t| \cdot \rho_t$ we calculate

$$\tau\big(A(\mathcal{T})\big) = \tau\left(\sum_{t\in\mathcal{T}} |t| \cdot \rho_t\right) \leq \sum_{t\in\mathcal{T}} |t| \cdot \tau(\rho_t) \leq \sum_{t\in\mathcal{T}} |t| \cdot \xi(\rho_t) \leq \sum_{t\in\mathcal{T}} |t| \cdot \frac{c_t}{|t|} = c^* \leq \xi(\rho^*), \quad (1)$$

where the second inequality follows from applying Jensen's Inequality to the convex function τ. Combining $\tau\big(A(\mathcal{T})\big) = \xi(\rho^*) + \xi'(\rho^*) \cdot \big(A(\mathcal{T}) - \rho^*\big)$ with Inequality 1 and rearranging gives $\xi'(\rho^*) \cdot A(\mathcal{T}) \leq \xi(\rho^*) \cdot \rho^*$, which yields the desired result after dividing by $\xi(\rho^*) < 0$. ◀

4 Charging Scheme and Weak Covering Guarantee

This section introduces the charging scheme we will use to derive our lower bounds for TilePacking's coverage (via the approach presented in Section 3). Then we derive a weak charging ratio bound ξ_w, as described in Section 3. While comparatively simple, this already yields that TilePacking covers at least a quarter of the unit square, almost tripling the original guarantee from [12]. Section 5 will refine ξ_w to derive our main result (Theorem 1).

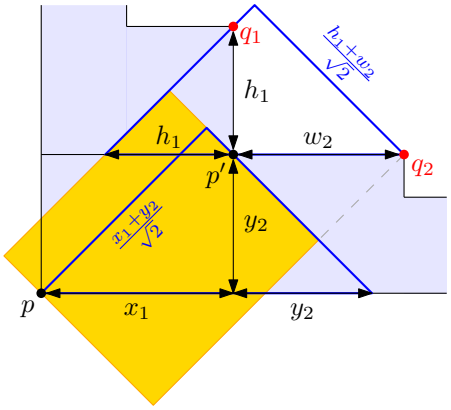


Figure 2 $|T_p(q_1, q_2)|$ is computed via the catheti of the blue triangles.

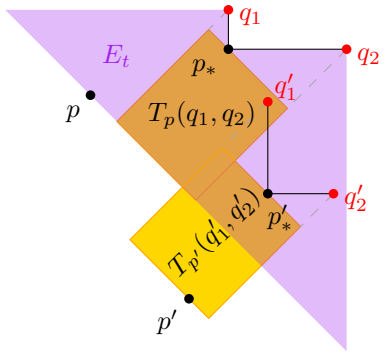


Figure 3 Example for Lemma 8. The shown tower overlap has $p'_* \in E_t$, violating t's exclusive area.

4.1 Charging Scheme

Given a tile packing $\mathcal{T}$ constructed by TilePacking, our charging scheme defines an area C_t for each tile $t \in \mathcal{T}$ and charges t's area to $c_t := |C_t|$. We first explain how C_t is constructed from t. Afterward, we prove useful properties about these areas and their relation to $\mathcal{T}$.

Construction of C_t. Consider three points $p, q_1 = (x_1, y_1), q_2 = (x_2, y_2) \in \mathbb{R}^2$ with $q_1, q_2 \succeq p$, $x_1 \leq x_2$, and $y_1 \geq y_2$. The *tower* $T_p(q_1, q_2)$ with *base point* p and *peak* $p_* = (x_1, y_2)$ is the interior of the rectangle enclosed by the lines $\ell_p^{\searrow}$ (the tower's *base*), $\ell_{q_1}^{\swarrow}$ (the tower's *left side*), $\ell_{q_2}^{\swarrow}$ (the tower's *right side*), and $\ell_{p_*}^{\searrow}$ (the tower's *top*). If the subscript p is omitted, the base point is assumed to be the origin $(0, 0)$.

For a tile t with anchor p and $\Gamma_t = \{q_1, q_2, \ldots, q_k\}$ being ordered by increasing x-coordinate, we define the charging area as the disjoint union of towers, i.e., $C_t := \bigcup_{i=1}^{k-1} T_p(q_i, q_{i+1})$. We refer to C_t as the *crown* of tile t. (See Figure 1c.)

The width and height of a tower $T_p(q_1, q_2)$ correspond to the side lengths of isosceles triangles (see Figure 2), which yields a formula for $|T_p(q_1, q_2)|$. By taking derivatives, we get formulas for the change of the tower's area when moving q_1 or q_2 horizontally or vertically.

▶ Observation 6. Consider $T_p(q_1, q_2)$ with $q_j - p = (x_j, y_j)$, $j \in \{1, 2\}$. Let $w_2 := x_2 - x_1$, and $h_1 := y_1 - y_2$. Then $|T_p(q_1, q_2)| = (x_1 + y_2) \cdot (w_2 + h_1)/2$.

▶ Observation 7. Consider $T_p(q_1, q_2)$ with $q_j - p = (x_j, y_j)$, $j \in \{1, 2\}$. Let $w_2 := x_2 - x_1$, and $h_1 := y_1 - y_2$. Fix $\alpha \in \mathbb{R}$ and consider the change of $|T_p(q_1, q_2)|$ if either q_1 or q_2 are moved horizontally or vertically as a linear function of ε:

(a) If either $q_1(\varepsilon) := q_1 + (0, \alpha \cdot \varepsilon)$ or $q_2(\varepsilon) := q_2 + (\alpha \cdot \varepsilon, 0)$, then $\partial |T_p(q_1, q_2)|/\partial \varepsilon = \alpha \cdot (x_1 + y_2)/2$ and $\partial^2 |T_p(q_1, q_2)|/\partial \varepsilon^2 = 0$.

(b) If either $q_1(\varepsilon) := q_1 + (\alpha \cdot \varepsilon, 0)$ or $q_2(\varepsilon) := q_2 + (0, \alpha \cdot \varepsilon)$, then $\partial |T_p(q_1, q_2)|/\partial \varepsilon = \alpha \cdot \big(w_2 + h_1 - (x_1 + y_2)\big)/2$ and $\partial^2 |T_p(q_1, q_2)|/\partial \varepsilon^2 = -\alpha^2$.

Properties of the Charging Scheme. The following results capture basic properties of our charging scheme. First, we show that the defined crowns are pairwise disjoint.

▶ **Lemma 8.** *Consider the tile packing $\mathcal{T}$ produced by algorithm TilePacking for a set P of input points. For any two different tiles $t, t' \in \mathcal{T}$, we have $C_t \cap C_{t'} = \emptyset$.*

Proof. Fix $t, t' \in \mathcal{T}$ and let $p, p' \in P$ denote their respective anchors. W.l.o.g., assume $\ell^{\searrow}_p > \ell^{\searrow}_{p'}$, such that TilePacking processes p before p'. As crowns consist of towers, it is sufficient to show $T_p(q_1, q_2) \cap T_{p'}(q'_1, q'_2) = \emptyset$ for consecutive $q_1, q_2 \in \Gamma_t$ and $q'_1, q'_2 \in \Gamma_{t'}$. Let $p_*, p'_* \in P$ be the respective peaks of these towers. W.l.o.g., we assume $\ell^{\swarrow}_{p_*} < \ell^{\swarrow}_{p'_*}$ (p_* lies left of $\ell^{\swarrow}_{p'_*}$); the other case follows symmetrically.

If $\ell^{\searrow}_{p'_*} < \ell^{\searrow}_p$, the towers are separated (the top of $T_{p'}(q'_1, q'_2)$ lies below the base of $T_p(q_1, q_2)$) and cannot intersect. So assume $\ell^{\searrow}_{p'_*} > \ell^{\searrow}_p$. Then we cannot have $p'_* \prec q_2$, since this would imply that p'_* lies in the exclusive area of t, violating Observation 3 (see Figure 3).

Let $\Delta_y := q'_1 - p'_*$ and note that $x(\Delta_y) = 0$. Define $\tilde{q}_1 := p_* - \Delta_y$ and note that $p'_* \not\succ \tilde{q}_1$, since otherwise $q'_1 = p'_* + \Delta_y \succ \tilde{q}_1 + \Delta_y = p_*$, which (together with $\ell^{\searrow}_{p_*} > \ell^{\searrow}_p > \ell^{\searrow}_{p'}$) would mean that p_* lies in the exclusive area of t' (again violating Observation 3).

So $\ell^{\swarrow}_{p'_*} > \ell^{\swarrow}_{p_*}$, $p'_* \not\prec q_2$, and $p'_* \not\succ \tilde{q}_1$. Together, these imply $x(p'_*) > x(q_2)$ and $y(p'_*) < y(\tilde{q}_1)$, which in turn imply $\ell^{\swarrow}_{p'_*} > \ell^{\swarrow}_{q_2 - \Delta_y}$. But then, the towers are separated, since $\ell^{\swarrow}_{q'_1} = \ell^{\swarrow}_{p'_* + \Delta_y} > \ell^{\swarrow}_{q_2 - \Delta_y + \Delta_y} = \ell^{\swarrow}_{q_2}$ ($T_p(q_1, q_2)$'s right side lies left of $T_{p'}(q'_1, q'_2)$'s left side). ◀

The next lemma's proof shows that all crowns lie inside a pentagon formed by $\mathcal{U}$ and two isosceles triangles left and below of $\mathcal{U}$ (see Figure 4). With Lemma 8 this implies that the total charging area is bounded by the pentagon's area.

▶ **Lemma 9.** *Consider the tile packing $\mathcal{T}$ produced by algorithm TilePacking for a set P of input points. The total charging area of $\mathcal{T}$ is $c^* \leq 3/2$. Moreover, this bound is tight, since for arbitrarily small $\varepsilon > 0$ there are input points P_ε with $c^* \geq 3/2 - \varepsilon$.*

Proof. Define the points SW $:= (0, 0)$, NW $:= (0, 1)$, and SE $:= (1, 0)$. Let $\pentagon$ denote the pentagon enclosed by the lines $\ell^{\searrow}_{\mathrm{SW}}$, $\ell^{\swarrow}_{\mathrm{NW}}$, $\ell^{\swarrow}_{\mathrm{SE}}$, ℓ^{-}_{NW}, and $\ell^{|}_{\mathrm{SE}}$ (see Figure 4). Since $|\pentagon| = 3/2$ and using Lemma 8, it is sufficient to show that $C_t \subseteq \pentagon$ for any $t \in \mathcal{T}$. For this, in turn, it is sufficient to show that any tower $T_p(q_1, q_2)$ of C_t lies in $\pentagon$.

Fix such a tower $T_p(q_1, q_2)$. Since $p \succeq \mathrm{SW}$, we have $\ell^{\searrow}_p \geq \ell^{\searrow}_{\mathrm{SW}}$ (the base of $T_p(q_1, q_2)$ lies above $\ell^{\searrow}_{\mathrm{SW}}$). Similarly, since $q_1, q_2 \in \overline{\mathcal{U}} \subseteq \pentagon$, we have $\ell^{\swarrow}_{q_1} \geq \ell^{\swarrow}_{\mathrm{NW}}$ (the left side of $T_p(q_1, q_2)$ lies right of the left side of $\pentagon$) and $\ell^{\swarrow}_{q_2} \leq \ell^{\swarrow}_{\mathrm{SE}}$ (the right side of $T_p(q_1, q_2)$ lies left of the right side of $\pentagon$). Finally, the topmost point $q_1 \in \overline{\mathcal{U}}$ of $T_p(q_1, q_2)$ lies below ℓ^{-}_{NW} and the rightmost point $q_2 \in \overline{\mathcal{U}}$ of $T_p(q_1, q_2)$ lies to the left of $\ell^{|}_{SE}$. Together, we get $T_p(q_1, q_2) \subseteq \pentagon$.

For the tightness of the bound, choose $\delta > 0, 1/\delta \in \mathbb{N}$ and define $P^\delta = \{\,\mathrm{SW}\,\} \cup \{\,(k \cdot \delta, 1 - k \cdot \delta^2), (1 - k \cdot \delta^2, k \cdot \delta) \mid k \in \{\,1, 2, \ldots, 1/\delta - 1\,\}\,\}$. As illustrated in Figure 4, the crown C_t of tile t with anchor SW converges towards $\pentagon$ as $\delta \to 0$. Therefore, for each $\varepsilon > 0$, we can choose δ such that, for point set $P_\varepsilon := P^\delta$, we have $c^* \geq |C_t| \geq 3/2 - \varepsilon$. ◀

4.2 Weak Covering Guarantee for Greedy Tile Packings

This section proves the following, slightly weaker version of Theorem 1:

▶ **Theorem 10.** *For any set of input points, TilePacking covers at least 25% of $\mathcal{U}$.*

Proving this not only serves as a warm-up to illustrate our approach before proving our main result, but – as we will see in Section 5 – brings us halfway towards proving Theorem 1.

So consider a tile packing $\mathcal{T}$ produced by algorithm TilePacking for some set P of input points. To prove Theorem 10, we follow the approach outlined in Section 3, using the charging scheme from Section 4.1. That is, the area of $t \in \mathcal{T}$ is charged to $c_t = |C_t|$, where C_t represents the crown of t. To this end, define $\rho^* := 1/4$ and the *weak charging ratio bound*

$$\xi_w \colon (0, 1] \to \mathbb{R}_{\geq 0}, \quad \xi_w(\rho) := 2 \cdot (1 - \rho).$$

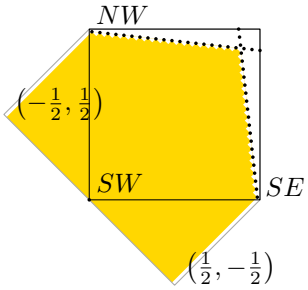


Figure 4 Pentagon $\pentagon$ and point set P_ε from Lemma 9.

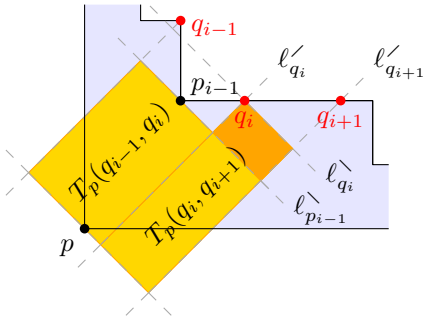


Figure 5 Removing superfluous points from Γ_t reduces $|C_t|$ by the rectangle between $\ell^{\swarrow}_{q_i}$, $\ell^{\swarrow}_{q_{i+1}}$, $\ell^{\searrow}_{q_i}$, and $\ell^{\searrow}_{p_{i-1}}$.

As a linear function, ξ_w is trivially point-convex in ρ^*. Moreover, $\xi_w(\rho^*) = 3/2$ and thus, by Lemma 9, $\xi_w(\rho^*) \geq c^*$. In the remainder of this section we prove the following Proposition 11, stating that ξ_w represents a lower bound on the charging ratio of any $t \in \mathcal{T}$. Once this is proven, Theorem 10 follows immediately by applying Lemma 5.

▶ **Proposition 11.** *For any tile t we have $c_t/|t| \geq \xi_w(\rho_t)$.*

A Lower Bound on the Charging Ratio. To prove that ξ_w bounds from below the charging ratio $c_t/|t|$ of any tile $t \in \mathcal{T}$, we gradually transform t into a "simpler" tile $\tilde{t}$. Our transformations ensure $\rho_{\tilde{t}} = \rho_t$ and $c_{\tilde{t}}/|\tilde{t}| \leq c_t/|t|$. Eventually, $\tilde{t}$ will be simple enough to directly prove $c_{\tilde{t}}/|\tilde{t}| \geq \xi_w(\rho_{\tilde{t}})$. The following notation expresses progress via such a transformation:

$$\tilde{t} \preceq t \quad :\Leftrightarrow \quad \rho_{\tilde{t}} = \rho_t \text{ and } c_{\tilde{t}}/|\tilde{t}| \leq c_t/|t|.$$

As a simple example, note that both a tile's density and charging-ratio are invariant under translation and concentric scaling w.r.t. its anchor. This gives rise to the following transformation, which allows us to restrict our analysis to normalized tiles.

▶ Observation 12. Translate a tile t such that it is anchored in the origin, then scale it by $1/|A_t|$ around the origin. We call the resulting tile $\tilde{t}$ *normalized*. Then $\tilde{t} \preceq t$.

Consider a tile t with anchor p. A transformation may move one of t's upper staircase points to the same x- or y-coordinate as another point from $\Gamma_t \cup \{ p \}$, resulting in a degenerate tile with superfluous points in Γ_t (see Section 2). The next lemma states that removing such superfluous points maintains an equal tile with a smaller crown.

▶ **Lemma 13.** *Consider a degenerate tile t. The tile $\tilde{t}$ with same anchor p but $\Gamma_{\tilde{t}} := \{ q \in \Gamma_t \mid q \succ p \text{ and } \nexists q' \in \Gamma_t \colon q \preceq q' \}$ covers the same points, is non-degenerate, and $\tilde{t} \preceq t$.*

Proof. Order $\Gamma_t = \{ q_1, q_2, \ldots, q_k \}$ by non-decreasing x-coordinate and let $q_0 = q_{k+1} = p$. W.l.o.g. assume there is some $i \in \{ 1, \ldots, k \}$ with $y(q_i) = y(q_{i+1})$; the case of identical x-coordinates follows analogously. Let $\tilde{t}$ denote the (possibly still degenerate) tile with anchor p and $\Gamma_{\tilde{t}} = \Gamma_t \setminus \{ q_i \}$. Note that $\{ q \in \mathcal{U} \mid q \succeq p \wedge q \prec q_i \} \subseteq \{ q \in \mathcal{U} \mid q \succeq p \wedge q \prec q_{i+1} \}$, which implies $\tilde{t} = t$ and, thus, $\rho_t = \rho_{\tilde{t}}$. Removing q_i affects the towers $T_p(q_i, q_{i+1})$ with peak q_i (only if $i < k$) and $T_p(q_{i-1}, q_i)$ with peak p_{i-1}. Figure 5 illustrates the situation.

We now show that $c_{\tilde{t}} \leq c_t$, such that $\tilde{t} \preceq t$; the lemma's statement then follows by iteration. If $i = k$, then $c_{\tilde{t}} = c_t - |T(q_{i-1}, q_i)| \leq c_t$. So assume $i < k$. Then $c_{\tilde{t}} = c_t - |\square| \leq c_t$, where $\square$ is the rectangle enclosed by the lines $\ell^{\swarrow}_{q_i}$, $\ell^{\swarrow}_{q_{i+1}}$, $\ell^{\searrow}_{q_i}$, and $\ell^{\searrow}_{p_{i-1}}$ (see Figure 5). ◀

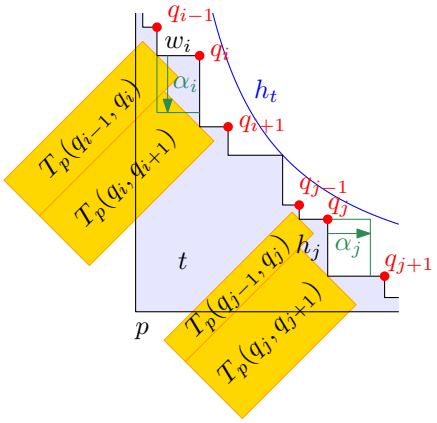


Figure 6 Moving inner points in Lemma 14. Note that $\alpha_j > 0$ and $\alpha_i < 0$ in this case.

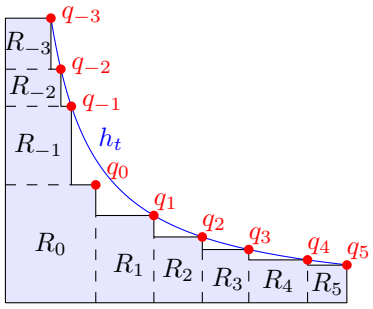


Figure 7 Notation for Proposition 11 with $l = 3$ and $m = 5$.

Note that $h_t \cap \Gamma_t$ consists of the upper staircase points that form maximum rectangles in the tile. We now prove that we can transform t such that at most one $q \in \Gamma_t$ lies not on h_t.

▶ **Lemma 14.** *A normalized tile t can be transformed into a tile $\tilde{t} \preceq t$ with $|\Gamma_{\tilde{t}} \setminus h_{\tilde{t}}| \leq 1$.*

Proof. Assume $|\Gamma_t \setminus h_t| > 1$ and order $\Gamma_t = \{ q_1, q_2, \ldots, q_k \}$ by increasing x-coordinate. To simplify border cases, define $q_0 = q_1$ and $q_{k+1} = q_k$. Choose $q_i, q_j \in \Gamma_t \setminus h_t$ with $i < j$. Consider the transformation $q_i(\varepsilon) := q_i + (0, \alpha_i \cdot \varepsilon)$ and $q_j(\varepsilon) := q_j + (\alpha_j \cdot \varepsilon, 0)$ with $\alpha_i, \alpha_j \in \mathbb{R}$. Then the tile and crown areas become functions $t(\varepsilon)$ and $c_t(\varepsilon)$ of ε. We show that there are non-zero α_i, α_j such that $t(\varepsilon)$ and, thus, $\rho_{t(\varepsilon)}$ remain constant and $c_t(\varepsilon)$ does not increase. Eventually, this results in a tile $\tilde{t} \preceq t$ that has an additional point on h_t or that has a degenerate staircase point which we can remove by Lemma 13. In both cases $|\Gamma_t \setminus h_t|$ is decreased and the lemma follows by iteration. Figure 6 illustrates the transformation.

The transformation changes only the towers $T(q_{i-1}, q_i)$, $T(q_i, q_{i+1})$, $T(q_{j-1}, q_j)$, and $T(q_j, q_{j+1})$. ($i < j$ ensures that these changes do not interfere with each other.) For $l \in \{ 0, 1, \ldots, k+1 \}$ let $q_l = (x_l, y_l)$ and define $w_l := x_l - x_{l-1}$ for $l \neq 0$ and $h_l := y_l - y_{l+1}$ for $l \neq k+1$. Then the transformation changes the area of the tile according to $\partial t(\varepsilon)/\partial\varepsilon = \alpha_i \cdot w_i + \alpha_j \cdot h_j$. To keep $|t|$ constant, we make this zero by setting $\alpha_j = -\alpha_i \cdot w_i / h_j$.

It remains to find a non-zero α_i such that $c_t(\varepsilon)$ is non-increasing in ε. For this let $T_i(\varepsilon) := |T(q_{i-1}, q_i)| + |T(q_i, q_{i+1})|$ and define $T_j(\varepsilon)$ analogously. By Observation 7, $\partial^2 T_l(\varepsilon)/\partial\varepsilon^2 = -\alpha_l^2$ for $l \in \{ i, j \}$. This yields $\partial^2 c_t(\varepsilon)/\partial\varepsilon^2 = -\alpha_i^2 - \alpha_j^2 < -\alpha_i^2$, which is a negative constant. This allows us to choose $\alpha_i \in \{ -1, +1 \}$ such that $c_t(\varepsilon)$ is non-increasing. ◀

With these results, we are ready to prove our first covering guarantee for TilePacking.

Proof of Proposition 11. Let t be a tile. By Observation 12 and Lemma 14, we can assume that t is normalized and $|\Gamma_t \setminus h_t| \leq 1$. If $|\Gamma_t \setminus h_t| = 1$, let $q_0 \in \Gamma_t \setminus h_t$, otherwise let $q_0 \in \Gamma_t$ arbitrary. Relabel the points $\Gamma_t = \{ q_{-l}, \ldots, q_m \}$ in increasing order of their x-coordinates. To simplify border cases, let $q_{-l-1} = q_{-l}$ and $q_{m+1} = q_m$. Let further $w_i := x_i - x_{i-1}$ for $i \neq l - 1$ and $h_i := y_i - y_{i+1}$ for $i \neq m + 1$ (using $q_i = (x_i, y_i)$). For $i = -l, \ldots, m$ inductively define the rectangles $R_i := \{ q \in t \mid q \prec q_i \} \setminus \bigcup_{|j|<|i|} R_j$. Finally, for $i = 1, \ldots, m$, let $T_i := T(q_{i-1}, q_i)$; for $i = -l, \ldots, -1$ let $T_i := T(q_i, q_{i+1})$. See Figure 7 for an illustration.

Note that $c_t = \sum_{i=-l}^{-1} |T_i| + \sum_{i=1}^{m} |T_i|$ and $|t| = \sum_{j=-l}^{m} |R_j|$. We will first show that for $i \in \{-l, \dots, m\} \setminus \{-1, 0, 1\}$ we have $|T_i| \geq 2|R_i|$. Afterward, we show $|T_{-1}| + |T_1| \geq 2(|R_{-1}| + |R_0| + |R_1| - 1)$. With these inequalities and since $\rho_t = |A_t|/|t| = 1/|t|$ due to the normalization, the desired statement follows via

$$c_t = \sum_{i=-l}^{-1} |T_i| + \sum_{i=1}^{m} |T_i| \geq \sum_{i=-l}^{m} 2|R_i| - 2 = 2|t| - 2 = 2|t| \cdot (1 - \rho_t) = |t| \cdot \xi_w(\rho_t).$$

We now show the above bounds, starting with $|T_i| \geq 2|R_i|$ for $|i| \geq 2$. W.l.o.g. we assume $i \geq 2$; the case $i \leq -2$ follows by symmetry. Note that $i \geq 2$ implies $q_i, q_{i-1} \in h_t$ and thus (since t is normalized) $y_j = 1/x_j$ for $j \in \{i-1, i\}$. This yields $x_{i-1}/y_i = x_{i-1} \cdot x_i$ as well as $w_i/h_{i-1} = (x_i - x_{i-1})/(y_{i-1} - y_i) = x_{i-1} \cdot x_i$. We use these identities together with $|R_i| = w_i \cdot y_i$ to bound the formula for $|T_i|$ from Observation 6:

$$\begin{aligned} |T_i| &= 1/2 \cdot (x_{i-1} + y_i)(w_i + h_{i-1}) = w_i \cdot y_i \cdot 1/2 \cdot (1 + x_{i-1}/y_i)(1 + h_{i-1}/w_i) \\ &= |R_i| \cdot \frac{1}{2} \cdot (1 + x_{i-1} \cdot x_i) \cdot \left(1 + \frac{1}{x_{i-1} \cdot x_i}\right) = |R_i| \cdot \frac{1}{2} \cdot \frac{(1 + x_{i-1} \cdot x_i)^2}{x_{i-1} \cdot x_i} \geq 2|R_i|, \end{aligned}$$

using that the function $x \mapsto (1+x)^2/x$ over $[0, \infty)$ has a minimum value of 4 at $x = 1$.

It remains to show that $|T_{-1}| + |T_1| \geq 2(|R_{-1}| + |R_{-1}| + |R_0| - 1)$. Note that, if $m = 0$, we have $q_{m+1} = q_m$ by definition and $|R_1| = 0$ and $|T_1| = 0$ hold. Similarly, if $l = 0$ then $|R_{-1}| = 0$ and $|T_{-1}| = 0$. We assume that $l > 0$ or $m > 0$, as otherwise $\xi_w(\rho_t) = \xi_w(1) = 0$ and the proposition becomes trivial. W.l.o.g. let $m > 0$; the other case follows symmetrically.

For $\alpha \in \{-1, +1\}$ (which we fix later) and $\varepsilon \geq 0$ define the transformation $y_0(\varepsilon) \coloneqq y_0 + \alpha \cdot \varepsilon$, where ε is chosen such that $y_0(\varepsilon) \in [y_1, 1/x_0]$ [2], which moves q_0 either up- or downward, depending on α. Thus, with $f(\varepsilon) \coloneqq |T_{-1}| + |T_1| - 2(|R_{-1}| + |R_1| + |R_0| - 1)$, where the T_i and R_i depend on y_0 and thus ε, our goal becomes to prove $f(0) \geq 0$. To this end, consider how $f(\varepsilon)$ changes with ε. The rectangles $|R_j|$ ($j \in \{-1, 0, 1\}$) change linearly or remain constant. By Observation 7, $\partial^2 |T_1|/\partial \varepsilon^2 = 0$. Similarly, if $l > 0$ we have $\partial^2 |T_{-1}|/\partial \varepsilon^2 = -\alpha^2 = -1$ by Observation 7, and if $l = 0$ we have $\partial^2 |T_{-1}|/\partial \varepsilon^2 = 0$ (because $|T_{-1}|$ remains zero). Thus, in all cases $\partial^2 f(\varepsilon)/\partial \varepsilon^2 \leq 0$, meaning its minimum $f_{\min}$ lies at one of the borders, where either $y_0 = y_1$ or $y_0 = 1/x_0$. We consider both possibilities and show that each time $f_{\min} \geq 0$ (which finishes the proof, since $f(0) \geq f_{\min} \geq 0$).

If at $f_{\min}$ we have $y_0 = 1/x_0$, let t_{high} denote the corresponding tile. Note that q_0 lies on the hyperbola h_t. But then $|R_0| = 1$ and, thus, $f_{\min} = |T_{-1}| + |T_1| - 2(|R_{-1}| + |R_1|)$. Moreover, with $q_0 \in h_t$ we can apply the calculations for $|i| > 1$ to get $|T_{-1}| \geq 2|R_{-1}|$ and $|T_1| \geq 2|R_1|$, such that $f_{\min} \geq 0$.

So assume that at $f_{\min}$ we have $y_0 = y_1$ and let t_{low} denote the corresponding tile. Note that R_0 and R_1 form a rectangle from the origin to the point q_1 on h_t, such that $|R_0| + |R_1| = 1$. Thus, $f_{\min} = |T_{-1}| + |T_1| - 2|R_{-1}|$. Define the (degenerate) tile t' with $\Gamma_{t'} = \{q_{-1}, q_0, q_1\}$ and anchor p, such that its crown area is $c_{t'} = |T_1| + |T_2|$. By Lemma 13, for the (non-degenerate) tile $\tilde{t}'$ with $\Gamma_{\tilde{t}'} = \Gamma_{t'} \setminus \{q_0\}$ we have $c_{\tilde{t}'} \leq c_{t'}$. The crown $c_{\tilde{t}'}$ consists of the single tower $T(q_{-1}, q_1)$. Since $q_{-1}, q_1 \in h_t$, we can apply the calculations for $|i| > 1$ to get $c_{\tilde{t}'} = |T(q_{-1}, q_1)| \geq 2|R_{-1}|$. Putting everything together we get

$$f_{\min} = |T_{-1}| + |T_1| - 2|R_{-1}| = c_{t'} - 2|R_{-1}| \geq c_{\tilde{t}'} - 2|R_{-1}| \geq 0. \qquad \blacktriangleleft$$

[2] These boundaries ensure that the tile remains valid and normalized. Note that if $l = 0$, moving q_0 upward also causes the dummy point q_{-1} to move upward, such that $|R_{-1}|$ and $|T_{-1}|$ remain zero.

5 Strong Covering Guarantee for Greedy Tile Packings

This section proves our strong covering guarantee for TilePacking, namely Theorem 1. We use the same approach as for our weak covering guarantee from Section 4.2 but derive a stronger charging ratio bound. More exactly, instead of ξ_w we use

$$\xi_s\colon (0,1] \to \mathbb{R}_{\geq 0}, \quad \xi_s(\rho) \coloneqq \begin{cases} 1-\rho\cdot\big(1+\sinh(1-1/\rho)\big) & \text{if } \rho \leq 1/2 \\ \xi_w(\rho) = 2\cdot(1-\rho) & \text{if } \rho > 1/2. \end{cases} \tag{2}$$

Most properties required for our approach from Definition 4 are easily verified for ξ_s (whose function graph can be seen in Figure 12). Indeed, for $\rho^* \coloneqq \xi_s^{-1}(3/2) \approx 0.3901$, we have $\partial\xi_s(\rho)/\partial\rho|_{\rho=\rho^*} \approx -5.1 < 0$. Moreover, ξ_s is point-convex at ρ^*, since it is convex on $(0,1/2]$ and on $(1/2,1]$ its tangent t_ξ at ρ^* lies below ξ_s (t_ξ is steeper and $t_\xi(1/2) \approx 0.94 < 1 = \xi_w(1/2)$). Also, by choice of ρ^* and by Lemma 9, we have $\xi_s(\rho^*) = 3/2 \geq c^*$ for the total charged area c^* of a tile packing $\mathcal{T}$ produced by algorithm TilePacking.

The following proposition states the remaining required property of Definition 4.

▶ **Proposition 15.** *For any tile t we have $c_t/|t| \geq \xi_s(\rho_t)$ and this bound is tight.*

With this, Theorem 1 follows by applying Lemma 5. The remainder of this section outlines the analysis of this proposition.

Transformation to Worst-case Tiles. For tiles t of density ρ_t larger than $1/2$, Proposition 15 follows from Proposition 11, since in this regime $\xi_s(\rho_t) = \xi_w(\rho_t)$. The tightness for such high densities follows since for any $\rho_t \in (1/2,1]$ there is a (symmetric) *step tile* $t = t_l(\rho_t)$ of density ρ_t (depicted in Figure 9) with $c_t/|t| = \xi_s(\rho_t)$. Thus, we restrict our further study to tiles of density at most $1/2$. We will show how to gradually transform any such tile t into a (symmetric) *hyperbola tile* $t_h(\rho_t) \preceq t$ (depicted in Figure 8). Again, the tightness follows from the existence of a tile $t = t_h(\rho_t)$ with $c_t/|t| = \xi_s(\rho_t)$.

Before we outline the transformation process into such worst-case low-density tiles, we need to cope with the fact that $t_h(\rho_t)$ is not a staircase polygon and, thus, not captured by our tile definition. However, one can see $t_h(\rho_t)$ as the result of defining Γ_t as k equally spaced points from the hyperbola $\{\,(x,y) \in [0,s) \mid y = 1/x\,\}$ and taking the limit $k \to \infty$. The next paragraph formalizes this intuition by introducing *generalized tiles* and some related notions.

Generalized Tiles and Crown Contribution. A *generalized tile* t is defined equivalently to "normal" tiles, with the only difference that Γ_t may be infinite. All other tile definitions (e.g., point set definition of t, maximum-area rectangle A_t, or density ρ_t) stay intact.

From now on the term *tile* always refers to a normalized and non-degenerate generalized tile. (Note that the points which cause a generalized tile to be degenerate, cannot reside in a slide. As such Observation 12 and Lemma 13 easily transfer to generalized tiles.) We require that the x-coordinates of Γ_t can be partitioned into k inclusion-wise maximal, closed intervals $I_1, I_2, \dots, I_k$, ordered by increasing x-coordinates. For $i \in \{\,1,2,\dots,k\,\}$ let $q_i^-, q_i^+ \in \Gamma_t$ denote the points realizing the left- and rightmost x-coordinate of I_i, respectively. Note that I_i may be a point interval, such that $q_i^- = q_i^+$. A *section* of Γ_t is a tuple as follows:

- a *step* (q_i^+, q_{i+1}^-), if $q_i^+, q_{i+1}^- \in h_t$;
- a *slide* (q_i^-, q_i^+), if $q_i^- \neq q_i^+$ and $\{\,q \in \Gamma_t \mid x(q) \in I_i\,\} \subseteq h_t$;
- a *double step* $(q_{i-1}^+, q_i^-, q_{i+1}^-)$, if $q_{i-1}^+, q_{i+1}^- \in h_t$ and $q_i^- = q_i^+ \notin h_t$; or
- the *corners* (q_1, q_2), if $q_1 \notin h_t$ and $q_2 \in h_t$ as well as (q_{k-1}, q_k) if $q_k \notin h_t$ and $q_{k-1} \in h_t$.

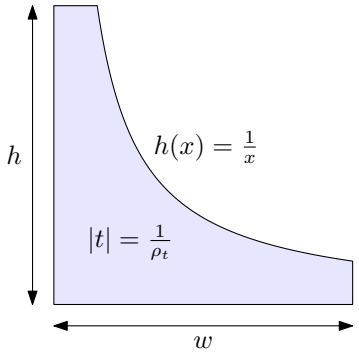


Figure 8 Symmetric ($w = h$) low-density tile $t = t_h(\rho_t)$ for which ξ_s is tight.

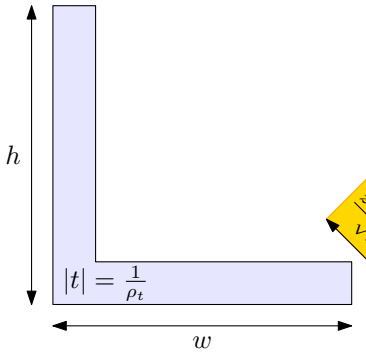


Figure 9 Symmetric ($w = h$) high-density tile $t = t_l(\rho_t)$ for which ξ_s is tight.

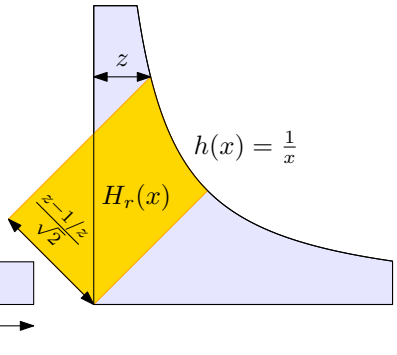


Figure 10 Crown contribution of a slide.

We further define $t(x_L, x_R) = \{ (x, y) \in t \mid x_L \leq x \leq x_R \}$. After applying Lemma 14, all tiles resulting from our transformations can be described as a sequence of such sections. Figure 13 illustrates generalized tiles and the different sections.

Note that a slide can be understood as the limit case of $k \to \infty$ equally spaced upper staircase points. As the slide is the only part of a generalized tile which differs from normal tiles, our charging scheme from Section 4.1 naturally extends to generalized tiles. This yields the following tower complement for slides:

▶ **Definition 16.** For a tile t with a slide (q_1, q_2), rotate the hyperbola by $\pi/4$ around the anchor of t, to obtain the rotated hyperbola $h_r(x) = \sqrt{x^2 + 2}$. The area under h_r between q_1 and q_2 will be denoted as $H(q_1, q_2)$.

▶ Observation 17. For a tile t with slide (q_1, q_2) we get $|H(q_1, q_2)| := \left[\frac{1}{4} \cdot (z^2 - z^{-2}) + \ln z\right]_{x_1}^{x_2}$.

Proof. $|H(q_1, q_2)|$ can be calculated via integration: The indefinite integral under $h_r(x) = \sqrt{x^2 + 2}$ is $H_r(x) := \int h_r(x)\, \mathrm{d}x = x/2 \cdot \sqrt{2 + x^2} + \operatorname{arsinh}(x/\sqrt{2})$, and for $x = (z - 1/z)/\sqrt{2}$, we get $H_r((z - 1/z)/\sqrt{2}) = 1/4 \cdot (z^2 - z^{-2}) + \ln z$, where $z = x_1$ or $z = x_2$ (see Figure 10). ◀

Overview of the Transformation Process.[3] Figure 11 gives an overview of how we gradually transform an arbitrary tile t with density $\rho_t \leq 1/2$ into a worst-case hyperbola tile $t_h(\rho_t)$. Starting with an arbitrary tile t (I), Lemma 18 either enforces $\Gamma_t \subseteq h_t$ or $\Gamma_t \setminus h_t = \{ q \}$.

▶ **Lemma 18.** *Let t be a tile. Then either there exists a tile $\tilde{t} \preceq t$ with $\Gamma_{\tilde{t}} \subset h_{\tilde{t}}$, or t contains a double step (q_1, q_2) with $x_1 \leq 1 \leq x_2$ or a corner (q_1, q_2) with $x_1 < 1$ (if $q_2 \notin h_t$, $x_2 > 1$ if $q_1 \notin h_t$).*

This lemma yields two different cases: In the first case (II), t contains a double-step (q_1, q, q_2) and we can enforce $x(q_1) \leq 1 \leq x(q_2)$. In the second case (III), q is part of a corner where $x(q') \geq 1$ or $x(q') \leq 1$ can be enforced for corners (q, q') or (q', q), respectively. (The case where $\Gamma_{\tilde{t}} \subset h_{\tilde{t}}$ will be dealt with later, in case (VIII).)

The next step from cases (II)/(III) to cases (IV)/(V) is based on smaller transformation/property statements about adjacent sections:

[3] Due to space limitations, the proofs for the following Lemmas 18–29 can only be found in the arXiv version of this paper (see [11]).

▶ **Lemma 19.** *Let t be a tile, $q_1, q_2, q_3 \in \Gamma_t$ the leftmost three points (in order) such that $q_1 = q_2$ or (q_1, q_2) is a step; and $q_2 = q_3$ or (q_2, q_3) is a slide. Then these sections can be replaced by up to one step $s = (q_1, q_2)$ and up to one slide $h = (q_2, q_3)$ such that*

1. *If s exists, then $x_1 x_2 \geq 1/\sqrt{2}$*
2. *If h exists, then $x_1 x_2 \leq 1/\sqrt{2}$*

giving us a tile $\tilde{t} \preceq t$.

Intuitively Lemma 19 states that the leftmost sections will only be a step and hyperbola when they satisfy precise properties and otherwise the tile can be transformed such that one of the sections vanished.

▶ **Lemma 20.** *Let t be a tile with a step (q_1, q_2) and a slide (q_2, q_3). If $x_1 \geq 1/x_3$, then the two sections can be replaced by a slide (q_1, q_4) and a step (q_4, q_2), resulting in a tile $\tilde{t} \preceq t$.*

Lemma 20 describes when "swapping" steps with an adjacent slide lowers c_t.

▶ **Lemma 21.** *Let t be a tile and (q_1, q_2),(q_2, q_3) be steps with $x_3 \leq 1$. Then the two steps can be replaced with a step (q_1, q_4) and a slide (q_4, q_3), resulting in a tile $\tilde{t} \preceq t$.*

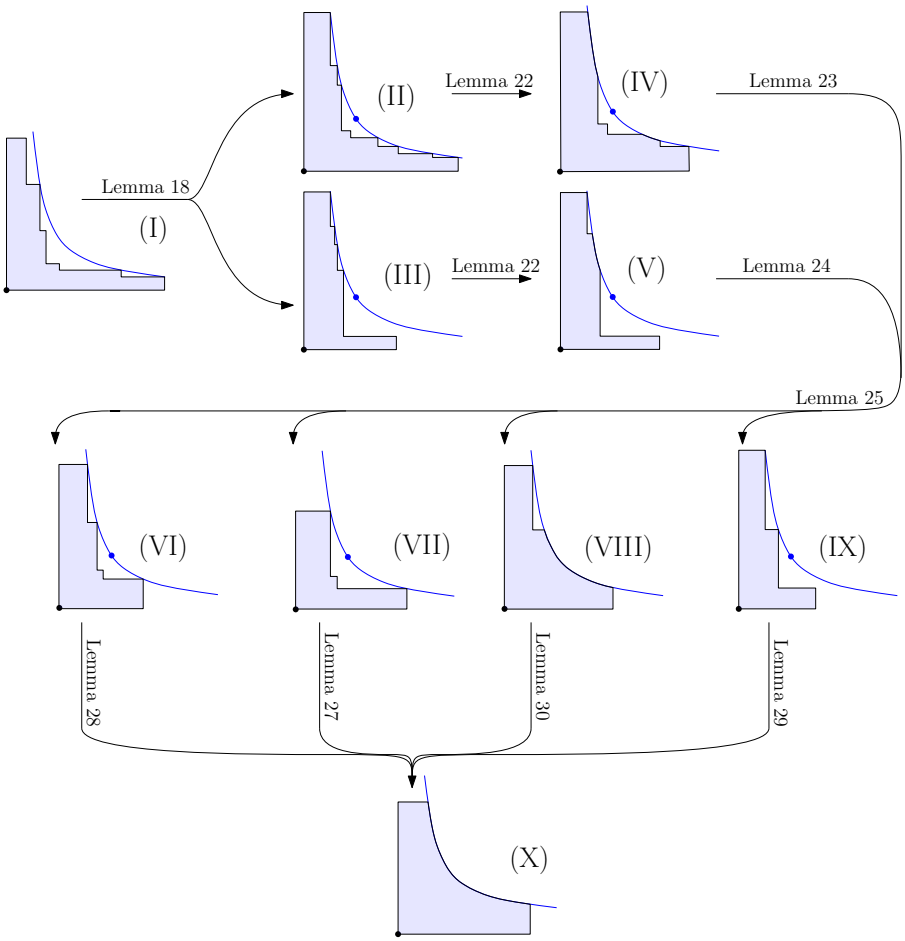


Figure 11 Transforming low-density tiles t with $\rho_t \leq 1/2$ to a corresponding worst-case hyperbola tile $t_h(s) \preceq t$. For the normalized tiles in Cases (II) and later, the blue dot marks the point $(1, 1)$.

Lemma 21 describes when "merging" two adjacent steps into a step and adjacent slide lowers c_t. Note that each of these lemmas hold up to reflection on the x=y axis, due to symmetry. Using the transformations/properties in these lemmas we can show the following Lemma 22, which allows us to only consider tiles containing a single step.

▶ **Lemma 22.** *Consider a tile t with $\rho_t \leq 1/2$. Then there exists a tile $\tilde{t} \preceq t$, containing at most one step. Furthermore, if $\Gamma_t \subseteq h_t$ then $\Gamma_{\tilde{t}} \subseteq h_{\tilde{t}}$.*

Now down to only one step, the following two lemmas (Lemmas 23 and 24) show that we can transform the tiles from (IV) and (V), respectively, such that either $\Gamma_t \subseteq h_t$ or t contains no slides:

▶ **Lemma 23.** *Let t be a tile with a double step (q_1, q_2, q_3) and a slide (q_3, q_4). We can replace both sections by a double step (q_1, q_4) or a sequence of steps and slides between q_1 and q_4, obtaining a tile $t' \preceq t$.*

▶ **Lemma 24.** *Let t be a tile with a slide (q_1, q_2) and a corner (q_2, q_3). We can replace both sections by a corner (q_1, q_2') or a sequence of steps and slides between q_1 and q_2', obtaining a tile $t' \preceq t$.*

These lemmas, together with the following Lemma 25, can then be used to reduce the remaining cases to those illustrated in (VI) to (IX):

▶ **Lemma 25.** *Let t be a tile with $\rho_t \leq 1/2$, then there exists a tile $\tilde{t} \preceq t$ with $\Gamma_{\tilde{t}} \subseteq h_{\tilde{t}}$ or $\tilde{t}$ consists of a step and a corner or a double step and possibly a step.*

The following Lemma 26 allows us to restrict ourselves to tiles t with $\rho_t = 1/2$ or where all points in Γ_t are on t's hyperbola, and is essential to proving the subsequent lemmas.

▶ **Lemma 26.** *Let t be a tile with $\rho_t \leq 1/2$. Then there exists a tile $\tilde{t}$ with $\Gamma_{\tilde{t}} \subseteq h_{\tilde{t}}$ or $\rho_{\tilde{t}} = 1/2$ such that, if $|C_{\tilde{t}}|/|\tilde{t}| \geq \xi_s(\rho_{\tilde{t}})$ then also $|C_t|/|t| \geq \xi_s(\rho_t)$.*

For each of the four cases we then separately show $t_h(\rho) \preceq t$ (Lemmas 27–30).

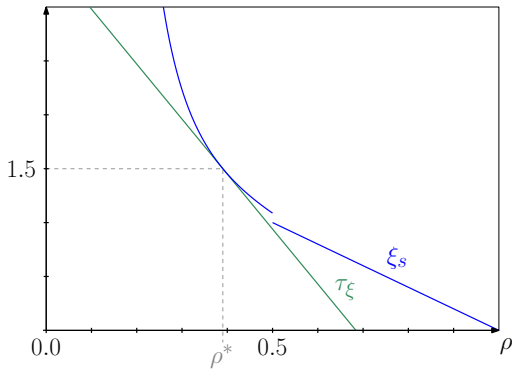


Figure 12 The charging ratio bound ξ_s from Section 5 and its tangent τ_ξ at $\rho^* = \xi_s^{-1}(3/2)$. This illustrates that ξ_s is point-convex at ρ^*.

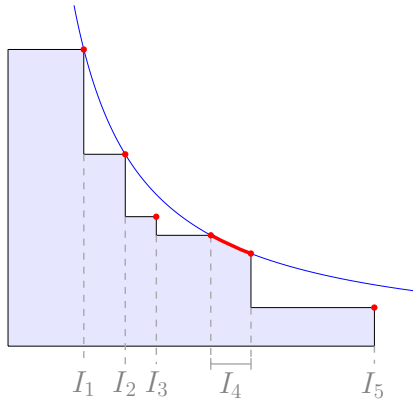


Figure 13 A generalized tile with four sections formed by the five intervals I_1 to I_5 (of which only I_4 is a proper interval). They form a step (between I_1 and I_2), a double-step (between I_2 and I_4), a slide (I_4), and a corner (I_4, I_5).

▶ **Lemma 27.** *Let t be a tile with $\rho_t \leq 1/2$ consisting only of a double-step (q_1, q_2, q_3). Then $|C_t|/|t| \geq \xi_s(\rho_t)$.*

▶ **Lemma 28.** *Let t be a tile with $\rho_t \leq 1/2$ consisting only of a double step (q_1, q_2, q_3) and a step (q_3, q_4). Then $|C_t|/|t| \geq \xi_s(\rho_t)$*

▶ **Lemma 29.** *Let t be a tile with $\rho_t \leq 1/2$ consisting only of a step (q_1, q_2) where $x_1 x_2 \geq 1/\sqrt{2}$ and a corner (q_2, q_3). Then there exists a tile $t' \preceq t$ only consisting of two steps.*

▶ **Lemma 30.** *Let t be a tile with $\rho_t \leq 1/2$ with $\Gamma_t \subseteq h_t$. Then $|C_t|/|t| \geq \xi_s(\rho_t)$.*

Proof. By Lemma 22 we can assume that t contains at most one step. Since $\Gamma_t \subseteq h_t$, t can only consist of steps and slides. Then t must contain at least one slide, as otherwise t consists of exactly one step, contradicting $\rho_t \leq 1/2$. Using Lemma 20, we can then ensure that t has at most one slide: Assuming this is not the case, t has a slide (q_1, q_2), a step (q_2, q_3) and another slide (q_3, q_4). Using the constraints of Lemma 20 we get $x_1 > 1/x_3 > 1/x_4 > x_2 > x_1$, a contradiction, so t has exactly one slide and possibly one step.

It is enough to show $\Delta := |t|\xi_s(\rho_t) - |C_t| < 0$, since the statement follows by rearranging. First assume that t's only section is a slide (q_1, q_2). Then $|t| = |t(0, x_1)| + |t(x_1, x_2)| = 1 + \ln(x_2/x_1)$, and we get:

$$\begin{aligned}
\Delta &= |t|\xi_s(\rho_t) - |H(q_1, q_2)| = |t| - 1 + \sinh(|t| - 1) - \left[1/4 \cdot (z^2 - z^{-2}) + \ln z\right]_{x_1}^{x_2} \\
&= \left(x_2^{-2} - x_2^2 + x_1^2 - x_1^{-2}\right)/4 + \sinh(|t| - 1) + |t| - (1 + \ln(x_2/x_1)) \\
&= \left(x_2^{-2} - x_2^2 + x_1^2 - x_1^{-2}\right)/4 + \sinh(\ln(x_2/x_1)) \\
&= 2\sinh((\ln(x_1) + \ln(x_2))/2)^2 \sinh(\ln(x_1/x_2)) < 0
\end{aligned}$$

where the last inequality directly follows from $x_1 < x_2$.

Now assume that t consists of a step (q_1, q_2) and a slide (q_2, q_3) (w.l.o.g. ordered in this way). In such a case we have $|t| = |t(x_0, x_1)| + |t(x_1, x_2)| + |t(x_2, x_3)| = 1 + (x_2 - x_1)/x_2 + \ln(x_3/x_2)$, or rearranged, $x_3 = x_2 e^{x_1/x_2 + |t| - 2}$. Again we calculate

$$\begin{aligned}
\Delta &= |t|\xi_s(\rho_t) - (|T(q_1, q_2)| + |H(q_2, q_3)|) \\
&= |t| - 1 + \sinh(|t| - 1) - \frac{1}{2}\left(\frac{1}{x_1} - \frac{1}{x_2} + x_2 - x_1\right)\left(x_1 + \frac{1}{x_2}\right) - \left[\frac{z^2 - z^{-2}}{4} + \ln z\right]_{x_2}^{x_3}.
\end{aligned}$$

Taking the derivative of Δ w.r.t. $|t|$ (after inserting $x_3 = x_2 e^{x_1/x_2 + |t| - 2}$), we obtain $\partial\Delta/\partial|t| = -2\cosh(x_1/x_2 + |t| - 2 + \ln x_2)^2 < 0$. This indicates that Δ is maximized for smallest $|t|$. So assume $|t| = 2$ now, or equivalently, $x_3 = x_2 e^{x_1/x_2}$. By Lemma 19 we can further assume that $x_2 = 1/(\sqrt{2}x_1)$. Using the substitution $x_1 = 2^{-3/4}\sqrt{u}$ we get

$$\Delta = \frac{1}{2\sqrt{2}}\left(-3 - \frac{\sqrt{2}}{e} + \sqrt{2}e + \frac{1 - e^u}{u} + u + \frac{u}{2e^u}\right) \quad \text{with the derivative}$$

$$\frac{\partial\Delta}{\partial u} = \frac{(1 - u)(u^2 + 2e^{2u} - 2e^u(1 + u))}{4\sqrt{2}u^2 e^u}.$$

The derivative above has only one zero, namely $u = 1$: From $0 < x_1^2 \leq x_1 x_2 = 1/\sqrt{2}$, we can deduce $u \in (0, 2]$ by the substitution. For the right factor of the derivative's numerator we then get $u^2 + 2e^{2u} - 2e^u(1 + u) > 2e^u(e^u - (1 + u)) > 0$ from the Taylor series of e^u. Hence checking Δ at u's boundaries and $u = 1$ is sufficient, where we get $\lim_{u\to 0}\Delta \approx -0.24 < 0$ (apply L'Hospital's rule on $(1 - e^u)/u$), $\Delta|_{u\to 1} \approx -0.07 < 0$ and $\Delta|_{u\to 2} \approx -0.26 < 0$. ◀

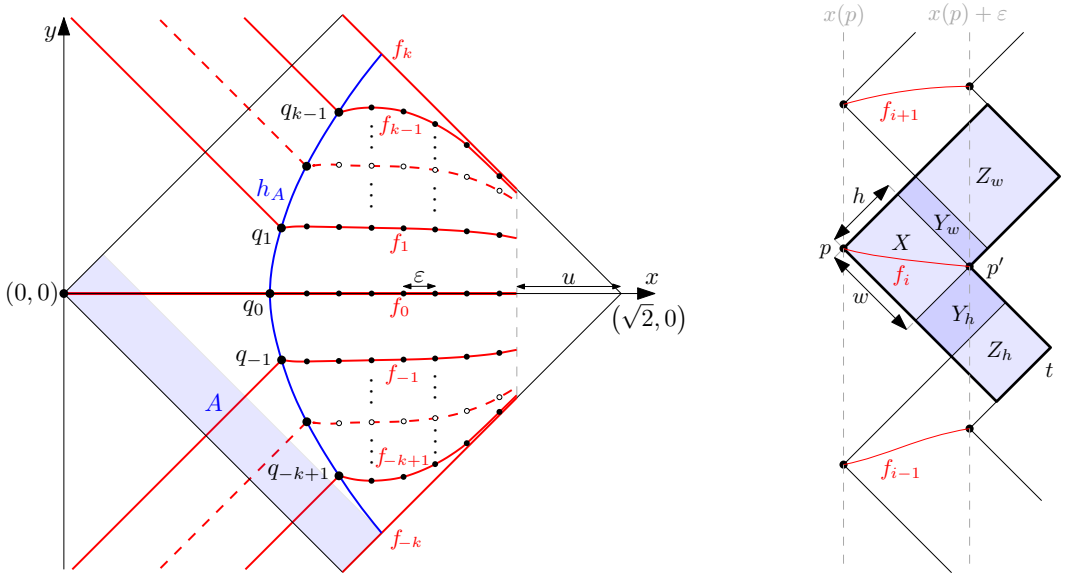


Figure 14 Left: Our construction. Right: A tile anchored at $p \in P_{\varepsilon,k} \setminus \{(0,0)\}$. For demonstration purposes, the areas are not shown to scale; in truth X, Z_w, Z_h are significantly smaller than Y_w, Y_h.

With all previous lemmas combined, we are now ready to give the proof for Proposition 15.

Proof of Proposition 15. By Proposition 11 we already get the result for $\rho_t > 1/2$. It remains to show the result for tiles with $\rho_t \leq 1/2$. Lemma 25 allows us to limit ourselves to certain tiles t: Tiles with $\Gamma_t \subseteq h_t$ (Lemma 30 yields the result), tiles with only a double step (use Lemma 27), tiles with a step and a double step (use Lemma 28) and tiles that consist of a step and a corner (use Lemma 29). As such the bound follows.

It remains to show the tightness. First note that for $\rho_t = 1$, the tightness is trivial (choose an arbitrary tile t with $|\Gamma_t| = 1$). For $\rho < 1$, we show that ξ_s exactly corresponds to the tiles $t_l(\rho)$ and $t_h(\rho)$ shown in Figures 8 and 9.

Let $t = t_l(\rho)$ with $\Gamma_t = \{q_1, q_2\} \subset h_t$, $x_1 = y_2$. We have $|t| = |t(0, x_1)| + |t(x_1, x_2)| = 1 + (1 - x_1^2)$, or rearranged $x_1 = \sqrt{2 - 1/\rho}$ and we get

$$|C_t|/|t| = |T(q_1, q_2)|/|t| = 1/2 \cdot (1/x_1 - x_1 + 1/x_1 - x_1)(x_1 + x_1)/\left(2 - x_1^2\right) = 2 - 2\rho.$$

Now let $t = t_h(\rho)$. t consists of a slide $(q_1, q_2) = ((x_1, 1/x_1), (1/x_1, x_1))$, hence $|t| = |t(0, x_1)| + |t(x_1, x_2)| = 1 + \ln((1/x_1)/x_1) = 1 - 2\ln x_1$, or rearranged, $x_1 = e^{(1-1/\rho)/2}$. So

$$|C_t|/|t| = |H(q_1, q_2)|\rho = \rho\left[\frac{1}{4}\left(z^2 - z^{-2}\right) + \ln z\right]_{e^{(1-1/\rho)/2}}^{e^{(1/\rho-1)/2}} = 1 - \rho(1 + \sinh(1 - \frac{1}{\rho})).$$

Note that t is only valid in the sense of generalized tiles. However, t can be arbitrarily well approximated by a non-generalized tile with area and crown size arbitrarily close to $|t|$ and $|C_t|$, respectively, by densely placing an increasing number of points on the hyperbola. ◀

6 Upper Bound

To show Theorem 2, we construct a point set where TilePacking covers at most roughly $(1 - e^{-2})/2$. Our goal is to construct a tile $\hat{t}$ at the origin where each maximal rectangle has the same size A. We therefore place $2k + 1$ points q_i into $\mathcal{U}$ densely on a hyperbola h_A centered at the origin. The remaining tiles will have a density close to $1/2$.

Such tiles can be realized when they have two nearly equal-sized maximal rectangles with minimum overlap. Hence, we add for each point q_i on h_A a set of (almost) evenly spaced points $p_{i,j}$ with distance roughly ε between each other. To get two maximal rectangles of roughly same area for each such tile, the exact coordinates of the points $p_{i,j}$ must be chosen carefully, as the placement of such points influences the size of maximal rectangles for surrounding points. That is why we place the points on arcs of functions f_i described by differential equations, where each f_i depends on the two neighboring curves f_{i-1} and f_{i+1}.

We may only use finitely many points $q_i \in h_A$ and $p_{i,j} \in f_i$. Both discretizations introduce an error term. Exploiting our choice of the functions f_i and how they relate to each other, we can show that both error terms vanish as k goes to infinity and ε goes to zero.

To aid the analysis, we rotate $\mathcal{U}$ by $\pi/4$ around the origin, obtaining a *rotated unit square* $\mathcal{U}_r$ (see Figure 14). Here, TilePacking processes the points from right to left.

We start by formally defining the point set $P_{\varepsilon,k}$ and the functions f_i. Let $h_A = \{\,(x,y) \in \mathcal{U}_r \mid x^2 - y^2 = 2A\,\}$ be the right branch of a hyperbola centered at the origin that lives in $\mathcal{U}_r$. First we define the upper part of the construction with non-negative y coordinates. For $i = 0, \dots, k-1$, densely choose k points $q_i \in h_A$ such that $y(q_i) \geq 0$ and $\sqrt{2A} = x(q_0) < x(q_1) < \cdots < x(q_{k-1})$. Define further $f_0(x) = 0$ and $f_k(x) = \sqrt{2} - x$. For $0 < i < k$, define $f_i : [0, \sqrt{2}) \to \mathbb{R}$ using

$$f_i(x(q_i)) = y(q_i) \text{ and} \tag{3}$$

$$f_i'(x) = \begin{cases} -1 & \text{for } x \leq x(q_i) \\ 1 - 2\dfrac{f_i(x) - f_{i-1}(x)}{f_{i+1}(x) - f_{i-1}(x)} & \text{for } x > x(q_i). \end{cases} \tag{4}$$

This means, each f_i with $0 < i < k$ has slope -1 in $[0, x(q_i))$, then it intersects h_A at q_i according to Equation (3), and then it has a slope depending on the current values of f_{i-1}, f_i and f_{i+1} according to Equation (4). For the symmetric part with negative y coordinates we define $q_{-i} = (x(q_i), -y(q_i))$ and $f_{-i}(x) = -f_i(x)$ for $0 < i \leq k$. Observe that, for $0 \leq i < k$, we get $q_{-i} \in h_A$ and the f_{-i} adhere to Equation (4). We are now ready to define the point set $P_{\varepsilon,k}$ for $\varepsilon > 0$ and $k \in \mathbb{N}$ as

$$P_{\varepsilon,k} = \{\,(0,0)\,\} \cup \bigcup_{i,j \in \mathbb{Z}} \left\{ \big(j\delta,\ f_i(j\delta)\big) \;\middle|\; -k < i < k, x(q_i) \leq j\delta < \sqrt{2} \right\}.$$

We require that $q_i \in P_{\varepsilon,k}$ for all $-k < i < k$, so we choose ε such that it divides all $x(q_i)$.

In order to be able to choose $P_{\varepsilon,k}$ as shown above, we need that the f_i are well-behaved: They must be defined in $[0, \sqrt{2})$, and should only intersect h_A at q_i. Intuitively, this is true, since the differential equation drives each function f_i to the midpoint of the functions f_{i-1} and f_{i+1}. The proof is given in the following lemma.

▶ **Lemma 31.** *Each function f_i intersects h_A exactly once, namely at q_i. Furthermore, $f_i(x)$ is differentiable for all $i = -k, \dots, k$, and $f_{-k}(x) < \cdots < f_k(x)$ holds for all $x \in [0, \sqrt{2})$.*[4]

We are now able to show Lemma 32: All tiles $t \neq \hat{t}$ have a density of close to $1/2$, unless they are too close to the right corner of $\mathcal{U}_r$, in which case their area is negligible. Afterwards, it only remains to optimize the parameter A, which is done in Theorem 33.

▶ **Lemma 32.** *Let $u, k > 0$ and $\hat{\mathcal{U}} = (\bigcup_{p \in P_{\varepsilon,k}, x(p) \leq \sqrt{2}-u} t_p) \setminus \hat{t}$. Then TilePacking covers $|\hat{\mathcal{U}}|/2 + c_k(\varepsilon)$ area in $\hat{\mathcal{U}}$ for $P_{\varepsilon,k}$ where $\lim_{\varepsilon \to 0} c_k(\varepsilon) = 0$.*

[4] Due to space limitations, the proof can only be found in the arXiv version of this paper (see [11]).

Proof. Consider a point $p \neq (0,0)$, $x(p) \leq \sqrt{2} - u$ that lies on some curve f_i and creates the tile t. Assuming $\varepsilon < u$, there exists another point $p' = (x(p) + \varepsilon, f_j(x(p) + \varepsilon)) \in P_{\varepsilon,k}$. By Lemma 31, we can assume that $f_{i+1}(x) - f_i(x) > \varepsilon$ for all $i = -k, \ldots, k-1$, $x \leq \sqrt{2} - u$. It follows from $|f_i'| \leq 1$ that p' is a lower staircase point of t. (For the same reason there cannot be points further to the right that are lower staircase points.) The tile t is therefore only restricted by the tiles from points with x-coordinate $x(p')$. Therefore, t has exactly two maximal rectangles which TilePacking can choose from (see Figure 14).

TilePacking will choose the larger one of the two maximal rectangles (call them R_1 and R_2). Since there are multiple points with the same x-coordinate as p, any of them can be processed first by TilePacking. This gives rise to areas Z_w,Z_h that may be covered by the tile t or by tiles directly above or below it (see Figure 14). W.l.o.g. we assume that TilePacking covers both Z_w and Z_h when choosing the maximal rectangle for t (since assuming this for all such tiles may only increase the covered area).

Since all f_i are differentiable in $[0, \sqrt{2})$, Taylor's Theorem provides a function $g(x)$ with $\lim_{x \to 0} g(x) = 0$ such that $f_i(x+\varepsilon) = f_i(x) + f_i'(x) \cdot \varepsilon + g(\varepsilon) \cdot \varepsilon$. Denote by w, h the dimensions of the rectangle $X = R_1 \cap R_2$. Then $w = (\varepsilon + f_i(x(p) + \varepsilon) - f_i(x(p)))/\sqrt{2} = (\varepsilon + f_i'(x(p))\varepsilon + g(\varepsilon)\varepsilon)/\sqrt{2} = (1 + f_i'(x(p)) + g(\varepsilon))\varepsilon/\sqrt{2}$ and similarly $h = (1 - f_i'(x(p)) - g(\varepsilon))\varepsilon/\sqrt{2}$.

From $|f_i'| \leq 1$ for all f_i, one can easily see that the rectangles X, Z_w and Z_h have widths and heights in $\mathrm{O}(\varepsilon)$, giving them a total area of $W := X + Z_w + Z_h = \mathrm{O}(\varepsilon^2)$.

The tile t also contains two additional rectangles with an area of $Y_w = w((f_i(x(p)) - f_{i-1}(x(p)))/\sqrt{2} - h)$ and $Y_h = h((f_{i+1}(x(p)) - f_i(x(p)))/\sqrt{2} - w)$. Note that this also holds if $x(q_{i\pm1}) > x(p)$, as we extended $f_{i\mp1}$ with lines of slope ± 1. In this case the two rectangles are restricted by $q_{i\mp1}$'s tile, respectively. Hence, when evaluating the functions at $x(p)$:

$$\begin{aligned}
|Y_w - Y_h| &= |w(f_i - f_{i-1})/\sqrt{2} - h(f_{i+1} - f_i)/\sqrt{2}| \\
&= |(1 + f_i' + g(\varepsilon))(\varepsilon(f_i - f_{i-1}))/2 - (1 - f_i' - g(\varepsilon))(\varepsilon(f_{i+1} - f_i))/2| \\
&= |(f_{i+1}(g(\varepsilon) + f_i' - 1) - f_{i-1}(g(\varepsilon) + f_i' + 1)) + 2f_i| \cdot \varepsilon/2 \\
&= |(g(\varepsilon)(f_{i+1} - f_{i-1}) + f_i'(f_{i+1} - f_{i-1}) - f_{i-1} - f_{i+1} + 2f_i)| \cdot \varepsilon/2 \\
&= |g(\varepsilon)||f_{i+1} - f_{i-1}| \cdot \varepsilon/2 \leq |g(\varepsilon)| \cdot \varepsilon\sqrt{2},
\end{aligned}$$

where the last inequality holds by Lemma 31, which gives us $f_{i+1}(x) - f_{i-1}(x) \leq f_k(x) - f_{-k}(x) \leq 2\sqrt{2}$ for $x \in [0, \sqrt{2})$.

W.l.o.g. assume $Y_w > Y_h$. Then for the tile t with area $|t| = W + Y_w + Y_h$, TilePacking covers at most $W + Y_w \leq W + Y_w/2 + (Y_h + |g(\varepsilon)| \cdot \varepsilon\sqrt{2})/2 = |t|/2 + \mathrm{O}(\varepsilon(|g(\varepsilon)| + \varepsilon))$. As $\hat{\mathcal{U}}$ is the union of such tiles and $|P_{\varepsilon,k}| = \mathrm{O}(k/\varepsilon)$, we have a total coverage of $|\hat{\mathcal{U}}|/2 + \mathrm{O}(k(|g(\varepsilon)| + \varepsilon))$. This immediately gives us the function $c_k(\varepsilon) = \mathrm{O}(k(|g(\varepsilon)| + \varepsilon))$ with $\lim_{\varepsilon \to 0} c_k(\varepsilon) = 0$. ◀

▶ **Theorem 33.** *TilePacking has no better lower bound than* $(1 - e^{-2})/2$.

Proof. We analyze the area ρ covered by TilePacking on $P_{\varepsilon,k}$ for some fixed k and u as ε approaches 0. The bound then follows from letting k go to ∞ and u go to 0.

By Lemma 32, TilePacking covers half of $\hat{\mathcal{U}} = (\bigcup_{p \in P_{\varepsilon,k}, x(p) \leq \sqrt{2} - u} t_p) \setminus \hat{t}$ (plus $c_k(\varepsilon)$ that approaches 0 for $\varepsilon \to 0$) for each $u > 0$. Additionally, at most u^2 area is covered from all tiles at points p with $x(p) > \sqrt{2} - u$. TilePacking covers $A + Q$ area in $\hat{t}$, where an error term Q is introduced because the q_i points only provide an approximation of h_A. Q can easily be bounded by, e.g., $Q \leq \max_i(x(q_i) - x(q_{i-1})) + \max_i(y(q_i) - y(q_{i-1}))$ (the biggest rectangular strip that fits between two consecutive q_i points, in $\mathcal{U}$). (Note that all q_i lie in $P_{\varepsilon,k}$, so no additional error is introduced.) In total, using $E = Q + c_k(\varepsilon) + u^2$,

$$\rho \leq A + |\hat{\mathcal{U}}|/2 + E \leq A + (1 - |\hat{t}|)/2 + E \leq A + (1 - (A + \int_A^1 \frac{A}{x}\,\mathrm{d}x))/2 + E$$
$$\leq (1 + A + A \ln A)/2 + E.$$

Minimizing the last term leads to $\rho \leq (1 - e^{-2})/2 + E$ at $A = e^{-2}$. Q approaches 0 when $k \to \infty$ since the q_i lie densely on h_A. Hence E approaches 0 for large k and small u, ε. ◀

7 Conclusion

We have shown that TilePacking's worst-case coverage lies between 39% and 43.3%. Our lower bound substantially improves over the previous best lower bound of roughly 9.1% [12], while our upper bound is the first non-trivial upper bound for TilePacking. Note that both bounds easily transfer to the (arguably more natural) GreedyPacking algorithm [12].

Our analysis crucially relies on a novel charging scheme and on a new analysis framework. The latter reduces the task of proving good coverage to finding good lower bounds on the tiles' charging ratios (see Section 3). The versatility of this approach shows in the fact that already a comparatively simple and short analysis yields a lower bound of 25%. Moreover, our approach provides structural insights: e.g., it allows us to characterize the exact shape of worst-case tiles as a function of their density (see Figures 8 and 9). We believe that our framework might help to analyze similar algorithms for (variants of) LLARP.

Concerning the remaining gap of size roughly 4 percentage points between our bounds, we believe that both bounds can be improved. For the lower bound, one shortcoming of our analysis is that tiles are analyzed individually, ignoring their local relationships in the unit square. Our lower bound basically predicts that the worst-case instance of TilePacking should consist solely of tiles whose shapes resemble Figure 8, which seems impossible.

Regarding the upper bound, there is still a noticeable gap between the maximal area that is coverable by the crowns (see Figure 4) and the area into which the crowns fall in our upper bound construction (see Figure 14). In particular, our construction uses only one tile (in the origin) with a large charging ratio, while all other tiles can be shown to have a charging ratio of roughly 1. Also note that our upper bound construction might be of interest with respect to the approximation variant of LLARP: an optimal solution should be able to fill most of the unit square, such that our results would imply a corresponding bound on the approximation ratio of TilePacking.

We leave as a major open question to find new algorithms for LLARP that might tackle the 50% conjecture. Note that our upper bound is tailored towards a specific greedy algorithm, so there is reasonable hope that other (possibly also greedy) algorithms might still achieve a coverage of 50%.

References

1 Anna Adamaszek, Parinya Chalermsook, and Andreas Wiese. How to Tame Rectangles: Solving Independent Set and Coloring of Rectangles via Shrinking. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques (APPROX/RANDOM)*, pages 43–60, 2015. doi:10.4230/LIPIcs.APPROX-RANDOM.2015.43.

2 Anna Adamaszek and Andreas Wiese. Approximation Schemes for Maximum Weight Independent Set of Rectangles. In *54th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 400–409, 2013. doi:10.1109/FOCS.2013.50.

3 Anna Adamaszek and Andreas Wiese. A quasi-PTAS for the Two-Dimensional Geometric Knapsack Problem. In *Proceedings of the Twenty-Sixth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1491–1505, 2015. doi:10.1137/1.9781611973730.98.

4 Hugo A. Akitaya, Matthew D. Jones, David Stalfa, and Csaba D. Tóth. Maximum Area Axis-Aligned Square Packings. In *MFCS*, pages 77:1–77:15, 2018.

5 Antonios Antoniadis, Felix Biermeier, Andrés Cristi, Christoph Damerius, Ruben Hoeksma, Dominik Kaaser, Peter Kling, and Lukas Nölke. On the Complexity of Anchored Rectangle Packing. In *27th Annual European Symposium on Algorithms (ESA)*, pages 8:1–8:14, 2019. doi:10.4230/LIPIcs.ESA.2019.8.

6 Kevin Balas, Adrian Dumitrescu, and Csaba D. Tóth. Anchored rectangle and square packings. *Discret. Optim.*, 26:131–162, 2017. doi:10.1016/j.disopt.2017.08.003.

7 Kevin Balas and Csaba D. Tóth. On the number of anchored rectangle packings for a planar point set. *Theor. Comput. Sci.*, 654:143–154, 2016. doi:10.1016/j.tcs.2016.03.007.

8 Vincent Bian. Special Configurations in Anchored Rectangle Packings, 2018. arXiv:1809.01769.

9 Therese C. Biedl, Ahmad Biniaz, Anil Maheshwari, and Saeed Mehrabi. Packing boundary-anchored rectangles and squares. *Comput. Geom.*, 88:101610, 2020. doi:10.1016/j.comgeo.2020.101610.

10 Timothy M. Chan and Sariel Har-Peled. Approximation Algorithms for Maximum Independent Set of Pseudo-Disks. *Discret. Comput. Geom.*, 48(2):373–392, 2012. doi:10.1007/s00454-012-9417-5.

11 Christoph Damerius, Dominik Kaaser, Peter Kling, and Florian Schneider. On greedily packing anchored rectangles. *CoRR*, abs/2102.08181, 2021. arXiv:2102.08181.

12 Adrian Dumitrescu and Csaba D. Tóth. Packing anchored rectangles. *Comb.*, 35(1):39–61, 2015. doi:10.1007/s00493-015-3006-1.

13 Ruben Hoeksma and Matthew Maat. A better lower bound for Lower-Left Anchored Rectangle Packing. *CoRR*, abs/2102.05747, 2021. arXiv:2102.05747.

14 IBM. Ponder This Challenge: Puzzle for June 2004, 2004. URL: https://www.research.ibm.com/haifa/ponderthis/challenges/June2004.html.

15 Klaus Jansen and Malin Rau. Improved approximation for two dimensional strip packing with polynomial bounded width. *Theor. Comput. Sci.*, 789:34–49, 2019. doi:10.1016/j.tcs.2019.04.002.

16 Konstantinos G. Kakoulis and Ioannis G. Tollis. Labeling Algorithms. In Roberto Tamassia, editor, *Handbook on Graph Drawing and Visualization*, pages 489–515. Chapman and Hall/CRC, 2013.

17 Arturo I. Merino and Andreas Wiese. On the Two-Dimensional Knapsack Problem for Convex Polygons. In *47th International Colloquium on Automata, Languages, and Programming, (ICALP)*, pages 84:1–84:16, 2020. doi:10.4230/LIPIcs.ICALP.2020.84.

18 William Thomas Tutte, editor. *Recent Progress in Combinatorics: Proceedings of the 3rd Waterloo Conference on Combinatorics*. Academic Press, 1969.

19 Peter Winkler. Packing Rectangles. *Mathematical Mind-Benders*, pages 133–134, 2007.

20 Peter Winkler. Puzzled: Rectangles Galore. *Commun. ACM*, 53(11):112, 2010. doi:10.1145/1839676.1839700.

21 Peter Winkler. Puzzled: Solutions and Sources. *Commun. ACM*, 53(9):110, 2010. doi:10.1145/1810891.1810917.

Approximately Counting Independent Sets of a Given Size in Bounded-Degree Graphs

Ewan Davies
Department of Computer Science, University of Colorado, Boulder, CO, USA

Will Perkins
Department of Mathematics, Statistics, and Computer Science,
University of Illinois at Chicago, IL, USA

Abstract

We determine the computational complexity of approximately counting and sampling independent sets of a given size in bounded-degree graphs. That is, we identify a critical density $\alpha_c(\Delta)$ and provide (i) for $\alpha < \alpha_c(\Delta)$ randomized polynomial-time algorithms for approximately sampling and counting independent sets of given size at most αn in n-vertex graphs of maximum degree Δ; and (ii) a proof that unless NP=RP, no such algorithms exist for $\alpha > \alpha_c(\Delta)$. The critical density is the occupancy fraction of hard core model on the clique $K_{\Delta+1}$ at the uniqueness threshold on the infinite Δ-regular tree, giving $\alpha_c(\Delta) \sim \frac{e}{1+e}\frac{1}{\Delta}$ as $\Delta \to \infty$.

2012 ACM Subject Classification Theory of computation → Approximation algorithms analysis; Mathematics of computing → Approximation algorithms; Theory of computation → Algorithm design techniques; Theory of computation → Random walks and Markov chains

Keywords and phrases approximate counting, independent sets, Markov chains

Digital Object Identifier 10.4230/LIPIcs.ICALP.2021.62

Category Track A: Algorithms, Complexity and Games

Related Version *Full Version*: https://arxiv.org/abs/2102.04984

Funding *Will Perkins*: supported in part by NSF grants DMS-1847451 and CCF-1934915.

1 Introduction

Counting and sampling independent sets in graphs are fundamental computational problems arising in several fields including algorithms, statistical physics, and combinatorics. Given a graph G, let $\mathcal{I}(G)$ denote the set of independent sets of G. The independence polynomial of G is

$$Z_G(\lambda) = \sum_{I \in \mathcal{I}(G)} \lambda^{|I|} = \sum_{k \geq 0} i_k(G)\lambda^k \,,$$

where $i_k(G)$ is the number of independent sets of size k in G. The independence polynomial also arises as the partition function of the hard-core model from statistical physics.

With G and λ as inputs, exact computation of $Z_G(\lambda)$ is #P-hard [32, 18], but the complexity of approximating $Z_G(\lambda)$ has been a major topic in recent theoretical computer science research. There is a detailed understanding of the complexity of approximating $Z_G(\lambda)$ for the class of graphs of maximum degree Δ, in particular showing that there is a *computational threshold* which coincides with a certain probabilistic phase transition as one varies the value of λ.

The *hard-core model* on G at fugacity λ is the probability distribution on $\mathcal{I}(G)$ defined by

$$\mu_{G,\lambda}(I) = \frac{\lambda^{|I|}}{Z_G(\lambda)} \,.$$

Defined on a lattice like $\mathbb{Z}^d$ (through an appropriate limiting procedure), this is a simple model of a gas (the hard-core lattice gas) and it exhibits an order/disorder phase transition as λ changes. The hard-core model can also be defined on the infinite Δ-regular tree (the *Bethe lattice*). Kelly [23] determined the critical threshold for uniqueness of the infinite volume measure on the tree, namely

$$\lambda_c(\Delta) = \frac{(\Delta-1)^{\Delta-1}}{(\Delta-2)^{\Delta}}\,. \tag{1}$$

This value of λ also marks a computational threshold for the complexity of approximating $Z_G(\lambda)$ on graphs of maximum degree Δ. One can approximate $Z_G(\lambda)$ up to a relative error of ε in time polynomial in n and $1/\varepsilon$ with several different methods, provided G is of maximum degree Δ and $\lambda < \lambda_c(\Delta)$. The first such algorithm is based on correlation decay on trees and is due to Weitz [33], but recently alternative algorithms based on polynomial interpolation [3, 27, 28] and Markov chains [2, 6, 7] for this problem have also been given. Conversely, for $\lambda > \lambda_c(\Delta)$ a result of Sly and Sun [31] and Galanis, Štefankovič, and Vigoda [16] (following Sly [30]) states that unless NP=RP there is no polynomial-time algorithm for approximating $Z_G(\lambda)$ on graphs of maximum degree Δ. Counting and sampling are closely related, and by standard reduction techniques the same computational threshold holds for the problem of approximately sampling independent sets from the hard-core distribution.

The hard-core model is an example of the *grand canonical ensemble* from statistical physics, where one studies physical systems that can freely exchange particles and energy with a reservoir. Closely related is the *canonical ensemble*, where one removes the reservoir and considers a system with a fixed number of particles. In the context of independent sets in graphs, this corresponds to the uniform distribution on independent sets of some fixed size k. Here the number $i_k(G)$ of independent sets of size k in G plays the role of the partition function. In this paper we answer affirmatively the natural question of whether there is a similar complexity phase transition for the problem of approximating $i_k(G)$, and the related problem of sampling independent sets of size k approximately uniformly. Analogous to the critical fugacity in the hard-core model, we identify a critical density $\alpha_c(\Delta)$, and for $\alpha < \alpha_c(\Delta)$ we give a fully polynomial-time randomized approximation scheme (FPRAS, defined below) for counting independent sets of size k in n-vertex graphs of maximum degree Δ, where $0 \le k \le \alpha n$. We also show that unless NP=RP there is no such algorithm for $\alpha > \alpha_c(\Delta)$.

In statistical physics the grand canonical ensemble and the canonical ensemble are known to be equivalent in some respects under certain conditions, and the present authors, Jenssen, and Roberts [12] used this idea to give a tight upper bound on $i_k(G)$ for large k in large Δ-regular graphs G (see also [10] for the case of small k). Here, the main idea in our proofs is also to exploit the equivalence of ensembles. For algorithms at subcritical densities we approximately sample independent sets from the hard-core model and show that with sufficiently high probability we get an independent set of the desired size, distributed approximately uniformly. For hardness at supercritical densities we construct an auxiliary graph G' such that $i_k(G')$ is approximately proportional to $Z_G(\lambda)$ for some $\lambda > \lambda_c(\Delta)$, and hence is hard to approximate. Our counting and sampling algorithms for independent sets of size k permit higher densities than previous algorithms for this problem based on Markov chains [5, 1], and an algorithm implicit in [10] based on the cluster expansion.

A pleasant feature of our methods is the incorporation of several advances from recent research on related topics. From the geometry of polynomials we use a state-of-the-art zero-free region for $Z_G(\lambda)$ due to Peters and Regts [28] and a central limit theorem of

Michelen and Sahasrabudhe [26, 25] (though an older result of Lebowitz, Pittel, Ruelle and Speer [24] would also suffice), and we also apply the very recent development that a natural Markov chain (the Glauber dynamics) for sampling from the hard-core model mixes rapidly at fugacities $\lambda < \lambda_c(\Delta)$ on all graphs of maximum degree at most Δ [1, 7]. Finally, our results also show a connection between these algorithmic and complexity-theoretic problems and extremal combinatorics problems for bounded-degree graphs [9, 12, 10], see also the survey [34].

1.1 Preliminaries

Given an error parameter ε and real numbers z, $\hat{z}$, we say that $\hat{z}$ is a *relative ε-approximation* to z if $e^{-\varepsilon} \leq \hat{z}/z \leq e^{\varepsilon}$. A *fully polynomial-time randomized approximation scheme* or *FPRAS* for a counting problem is a randomized algorithm that with probability at least 3/4 outputs a relative ε-approximation to the solution of the problem in time polynomial in the size of the input and $1/\varepsilon$. If the algorithm is deterministic (i.e. succeeds with probability 1) then it is a *fully polynomial-time approximation scheme* (*FPTAS*). An ε-approximate sampling algorithm for a probability distribution μ outputs a random sample from a distribution $\hat{\mu}$ such that the total variation distance $\|\mu - \hat{\mu}\|_{TV} \leq \varepsilon$, and an *efficient sampling scheme* is, for all $\varepsilon > 0$ an ε-approximate sampling algorithm which runs in time polynomial in the size of the input and $\log(1/\varepsilon)$. Note that approximate sampling schemes whose running times are polynomial in $1/\varepsilon$ or in $\log(1/\varepsilon)$ are common in the literature, but we adopt the stronger definition for this paper. The inputs to our algorithms are graphs, and input size corresponds to the number of vertices of the graph.

An *independent set* in a graph $G = (V, E)$ is a subset $I \subset V$ such that no edge of E is contained in I. The *density* of such an independent set I is $|I|/|V|$, and it will be convenient for us to parametrize independent sets by their density instead of their size. We write $\mathcal{I}(G)$ for the set of all independent sets in G, $\mathcal{I}_k(G)$ for the set of independent sets of size k in G, and $i_k(G) = |\mathcal{I}_k(G)|$ for the number of such sets. Recall the hard-core distribution $\mu_{G,\lambda}$ on $\mathcal{I}(G)$ is given by $\mu_{G,\lambda}(I) = \lambda^{|I|}/Z_G(\lambda)$. We also define the *occupancy fraction* $\alpha_G(\lambda)$ of the hard-core model on G at fugacity λ to be the expected density of a random independent set drawn according to $\mu_{G,\lambda}$. Let $\mathcal{G}_\Delta$ be the set of graphs of maximum degree Δ.

The critical density that we show constitutes a computational threshold for the problems of counting and sampling independent sets of a given size in graphs of maximum degree Δ is

$$\alpha_c(\Delta) = \frac{\lambda_c(\Delta)}{1 + (\Delta+1)\lambda_c(\Delta)} = \frac{(\Delta-1)^{\Delta-1}}{(\Delta-2)^{\Delta} + (\Delta+1)(\Delta-1)^{\Delta-1}},$$

with λ_c the critical fugacity as in (1). This may seem unexpected at first sight, but has a natural interpretation. The threshold is in fact the quantity $\alpha_{K_{\Delta+1}}(\lambda_c(\Delta))$, the occupancy fraction of the clique on $\Delta + 1$ vertices at the critical fugacity $\lambda_c(\Delta)$. This is a natural threshold because the occupancy fraction is a monotone increasing function of λ, and the clique on $\Delta + 1$ vertices has the minimum occupancy fraction over all graphs of maximum degree Δ. Thus, for any $G \in \mathcal{G}_\Delta$, the value of λ which makes $\alpha_G(\lambda) > \alpha_c(\Delta)$ must be greater than $\lambda_c(\Delta)$. Conversely, if $\alpha < \alpha_c(\Delta)$ then for every graph $G \in \mathcal{G}_\Delta$ there is some $\lambda < \lambda_c(\Delta)$ such that $\alpha_G(\lambda) = \alpha$.

1.2 Our results

We are now ready to state our main result.

▶ **Theorem 1.**

(a) *For every $\alpha < \alpha_c(\Delta)$ there is an FPRAS for $i_{\lfloor \alpha n \rfloor}(G)$ and an efficient sampling scheme for the uniform distribution on $\mathcal{I}_{\lfloor \alpha n \rfloor}(G)$ for n-vertex graphs G of maximum degree Δ.*

(b) *Unless* NP=RP*, for every $\alpha \in (\alpha_c(\Delta), 1/2)$ there is no FPRAS for $i_{\lfloor \alpha n \rfloor}(G)$ for n-vertex, Δ-regular graphs G.*

The assumption NP≠RP, which is that polynomial-time algorithms using randomness cannot solve all problems in NP, is standard in computational complexity theory. Indeed, this assumption is used in [30, 31, 16] to show hardness of approximation for $Z_G(\lambda)$ on regular graphs at supercritical fugacities, which we apply directly. The upper bound of $1/2$ on α in b is required since in a regular graph (of degree ≥ 1) there are no independent sets of density greater than $1/2$ and counting those of density $1/2$ amounts to counting connected components in a bipartite graph (which can be done in polynomial time). For graphs of maximum degree Δ there is no such barrier, and in this case our methods can also be used to prove b for $\alpha \in (\alpha_c(\Delta), 1)$.

On the algorithmic side, Bubley and Dyer [5] showed via path coupling that a natural Markov chain for sampling independent sets of size k in n-vertex graphs of maximum degree Δ mixes rapidly when $k < n/(2\Delta + 2)$, and recently this was slightly improved to $k < n/(2\Delta)$ via the method of high-dimensional expanders by Alev and Lau [1] (who also gave an improved bound in terms of the smallest eigenvalue of the adjacency matrix of G). The fast mixing of this Markov chain provides a randomized algorithm for approximate sampling and an FPRAS for approximate counting for this range of k. Implicit in the work of the present authors and Jenssen [10] is an alternative method based on the cluster expansion that yields an FPTAS for $i_k(G)$ when $k < e^{-5}n/(\Delta + 1)$, and although we did not try to optimize the constant it seems unlikely that without significant extension the cluster expansion approach could yield a sharp result. Considering asymptotics as $\Delta \to \infty$, these previous algorithms work for densities up to $(c + o(1))/\Delta$ with the constant c being $1/2$ or $e^{-5} \approx 0.007$ respectively. Here, our algorithms work for densities α satisfying

$$\alpha < \alpha_c(\Delta) = (1 + o(1))\frac{e}{1+e}\frac{1}{\Delta},$$

as $\Delta \to \infty$. The constant $e/(1 + e)$ is approximately 0.731, and our hardness proof shows that this is tight.

Our sampling algorithm is based on searching over possible values of λ until we find one for which the mean size of an independent set from the hard-core model is close to the target k. We then repeatedly sample from the hard-core model until we obtain an independent set of size k and output this independent set. Our approximate counting algorithm is based on a standard reduction of approximate counting to approximate sampling.

The method of sampling from the canonical ensemble by sampling from the grand canonical ensemble and conditioning on obtaining an object the desired size is quite old and appears in the seminal papers of Jerrum and Sinclair [20, 21]. The key technical step in applying this method is to prove a lower bound on the probability of obtaining the desired size; in [20, 21] this is accomplished by using log-concavity of the specific distribution on sizes. Since the hard-core model does not have this property in general, we need a different argument; our new argument is based on the rapid mixing of Glauber dynamics.

Harris and Kolmogorov [19] have recently investigated the general problem of estimating the coefficients of partition functions given access to samples from the corresponding Gibbs distribution. Applying their ideas and results to this problem could likely lead to more

efficient run times for the problem of approximating the entire sequence $\{i_k(G)\}$, $1 \le k \le \alpha n$. Their ideas could also be applied to improve the efficiency of the reduction from counting to sampling, as they describe a more efficient "cooling schedule" for the simulated annealing technique that we use for this type of reduction.

1.3 Triangle-free graphs

As an additional application of our techniques we find an approximate computational threshold for the class of triangle-free graphs.

▶ **Theorem 2.** *For every $\delta > 0$ there is Δ_0 large enough so that the following is true.*

(a) *For $\Delta \ge \Delta_0$ and $\alpha < \frac{1-\delta}{\Delta}$ there is an FPRAS and efficient sampling scheme for $i_{\lfloor \alpha n \rfloor}(G)$ for the class of triangle-free graphs of maximum degree Δ.*

(b) *For $\Delta \ge \Delta_0$ and $\alpha \in \left(\frac{1+\delta}{\Delta}, 1/2\right)$ there is no FPRAS for $i_{\lfloor \alpha n \rfloor}(G)$ for the class of triangle-free graphs of maximum degree Δ.*

The proof of this theorem uses a result on the occupancy fraction of triangle-free graphs from [11], see Section 4.

1.4 Related work

Counting independent sets of a specified size has arisen in various places as a natural fixed-parameter version of counting independent sets, and is equivalent to counting cliques of a specified size in the complement graph. Exact computation of $i_k(G)$ in an n-vertex graph H is trivially possible in time $O(k^2 n^k)$, though improvements can be made via fast matrix multiplication algorithms (see e.g. [15]). Another branch of research concerns the complexity (in both time and number of queries to the graph data structure) of counting and approximately counting cliques. For example, in [14] the authors gave a randomized approximation algorithm for approximating the number of cliques of size k. Results of this kind perform poorly in our setting, which is equivalent to counting cliques in the complement of bounded-degree graphs, because such graphs are very dense. In particular, the main result of [14] has expected running time $\Omega((nk/e)^k)$ in our setting.

With a focus on bounded-degree graphs and connections to statistical physics, our work is closer in spirit to that of Curticapean, Dell, Fomin, Goldberg, and Lapinskas [8]. There, the authors consider the problem of counting independent sets of size k in bipartite graphs from the perspective of parametrized complexity. They give algorithms for exact computation and approximation of $i_k(G)$ in bipartite graphs (of bounded degree and otherwise), including a fixed parameter tractable randomized approximation scheme, though their running times are exponential in k. We note that the complexity of approximately counting the total number of independent sets in bipartite graphs (a problem known as #BIS) is unknown [13].

1.5 Questions and future directions

For the hard-core model, the algorithm of Weitz [33] gives a deterministic approximation algorithm (FPTAS) for $Z_G(\lambda)$ for $\lambda < \lambda_c(\Delta)$. The approach of Barvinok along with results of Patel and Regts and Peters and Regts give another FPTAS for the same range of parameters [3, 27, 28]. Our algorithm for approximating the number of independent sets of a given size uses randomness, but we conjecture that there is a deterministic algorithm that works for the same range of parameters. (The cluster expansion approach of [10] gives an FPTAS but only for smaller values of α).

▶ **Conjecture 3.** *There is an FPTAS for $i_{\lfloor \alpha n \rfloor}(G)$ for $G \in \mathcal{G}_\Delta$ and all $\alpha < \alpha_c(\Delta)$.*

The Markov chain analyzed in [5, 1] is the "down/up" Markov chain: starting from an independent set $I_t \in \mathcal{I}_k(G)$ at step t, pick a uniformly random vertex $v \in I_t$ and a uniformly random vertex $w \in V$. Let $I' = (I_t \setminus v) \cup w$. If $I' \in \mathcal{I}_k(G)$, let $I_{t+1} = I'$; if not, let $I_{t+1} = I_t$.

▶ **Conjecture 4.** *The down/up Markov chain for sampling from $\mathcal{I}_{\lfloor \alpha n \rfloor}(G)$ mixes rapidly for $\alpha < \alpha_c(\Delta)$ and all $G \in \mathcal{G}_\Delta$.*

One of the steps of our proof leads to a natural probabilistic conjecture concerning the hard-core model in bounded degree graphs.

▶ **Conjecture 5.** *Suppose G is a graph on n vertices of maximum degree Δ. Then if $\lambda < \lambda_c(\Delta)$ and $k = \lfloor \mathbb{E}_{G,\lambda}|\mathbf{I}| \rfloor$, we have*

$$\mathbb{P}_{G,\lambda}[|\mathbf{I}| = k] = \Omega(n^{-1/2}),$$

where the implied constant only depends on Δ and λ and the expectation and probability are with respect to the hard-core model on G at fugacity λ.

Lemma 8 below gives the weaker bound $\Omega(n^{-1}\log^{-1} n)$. A stronger conjecture would be that a local central limit theorem for $|\mathbf{I}|$ holds whenever $\lambda < \lambda_c(\Delta)$.

Finally, our proofs of Theorems 1 and 2 show a close connection between the computational threshold for sampling independent sets of a given size in bounded-degree graphs and the extremal combinatorics problem of minimizing the occupancy fraction in the hard-core model over a class of bounded-degree graphs. We expect that a rigorous connection between the two problems can be proved.

2 Algorithms

In this section, we fix $\Delta \geq 3$ and $\alpha < \alpha_c(\Delta)$. We first give an algorithm that, for $G \in \mathcal{G}_\Delta$ on n vertices and $k \leq \alpha n$, returns an ε-approximate uniform sample from $\mathcal{I}_k(G)$ and runs in time polynomial in n and $\log(1/\varepsilon)$; this proves the sampling part of Theorem 1a. We then use this algorithm to approximate $i_k(G)$ using a standard simulated annealing process to prove the approximate counting part of Theorem 1a.

Given $\lambda \geq 0$, let $\mathbf{I}$ be a random independent set from the hard-core model on G at fugacity λ. We will write $\mathbb{P}_{G,\lambda}$ for probabilities over the hard-core measure $\mu_{G,\lambda}$, so e.g. $\mathbb{P}_{G,\lambda}(|\mathbf{I}| = k)$ is the probability that $\mathbf{I}$ is of size exactly k. Often we will suppress the dependence on G.

A key tool that we use for probabilistic analysis and to approximately sample from $\mu_{G,\lambda}$ is the *Glauber dynamics*. This is a Markov chain with state space $\mathcal{I}(G)$ and stationary distribution $\mu_{G,\lambda}$. Though the algorithm of Weitz [33] was the first to give an efficient approximate sampling algorithm for $\mu_{G,\lambda}$ for $\lambda < \lambda_c(\Delta)$ and all $G \in \mathcal{G}_\Delta$, a randomized algorithm with better running time now follows from recent results showing that the Glauber dynamics mix rapidly for this range of parameters [2, 6, 7]. The *mixing time* $T_{\text{mix}}(\mathcal{M}, \varepsilon)$ of a Markov chain $\mathcal{M}$ is the number of steps from the worst-case initial state I_0 for the resulting state to have a distribution within total variation distance ε of the stationary distribution. We will use the following result of Chen, Liu, and Vigoda [7], and the sampling algorithm that it implies.

▶ **Theorem 6** ([7])**.** *Given $\Delta \geq 3$ and $\xi \in (0, \lambda_c(\Delta))$, there exists $C > 0$ such that the following holds. For all $0 \leq \lambda < \lambda_c(\Delta) - \xi$ and graphs $G \in \mathcal{G}_\Delta$ on n vertices, the mixing time $T_{\text{mix}}(\mathcal{M}, \varepsilon)$ of the Glauber dynamics $\mathcal{M}$ for the hard-core model on G with fugacity λ is at most $Cn\log(n/\varepsilon)$. This implies an ε-approximate sampling algorithm for $\mu_{G,\lambda}$ for $G \in \mathcal{G}_\Delta$ that runs in time $O(n \log n \log(n/\varepsilon))$.*

The sampling algorithm follows from the mixing time bound; the extra factor $\log n$ is the cost of implementing one step of Glauber dynamics (which requires reading $O(\log n)$ random bits to sample a vertex uniformly). Note that the implicit constant in the running time depends on how close λ is to $\lambda_c(\Delta)$, but in applications of this theorem we will have $\lambda \leq \lambda_c(\Delta) - \xi$ for some fixed $\xi > 0$, so that the implicit constant depends only on ξ, which in turn depends on α.

2.1 Approximate sampling

The algorithm Sample-k listed in Algorithm 1 uses Theorem 6 and a binary search on values of λ to generate samples from $\mathcal{I}_k(G)$. The algorithm requires access to distributions $\hat{\mu}_\lambda$ which are meant to be approximate versions of the hard-core model on G at fugacity λ. For the algorithm to work, we require that there is some value of λ such that sampling from the hard-core model at fugacity λ is likely to yield a set of size exactly k, and we will establish that this holds for some λ in a set of size $O(n^2)$ and for a suitable definition of "likely". Quantitatively, these results inform the length $C \log n$ of the for loop and the value of N. An intuitive choice for λ would be the unique value that makes k the expected size of an independent set from the hard-core model on G, and in fact it suffices to find a value sufficiently close to this.

Algorithm 1 Sample-k.

input : $\alpha < \alpha_c$; $\varepsilon > 0$; $G \in \mathcal{G}_\Delta$ of size n; integer $k \leq \alpha n$
output : $I \in \mathcal{I}_k(G)$ with distribution within ε total variation distance of the uniform distribution of $\mathcal{I}_k(G)$

1 Let $\lambda_* = \frac{\alpha}{1-\alpha(\Delta+1)}$
2 For $t = 0, \dots, \lfloor 2\lambda_* n^2 \rfloor$, let $\lambda_t = t/(2n^2)$
3 Let $\Lambda_0 = \{\lambda_t : t = 0, \dots, \lfloor 2\lambda_* n^2 \rfloor\}$
4 **for** $i = 1, \dots, C \log n$, **do**
5 Let λ be a median of the set Λ_{i-1}
6 With $N = C' n^2 \log\left(\frac{\log n}{\varepsilon}\right)$, take N independent samples $I_1, \dots, I_N$ from a distribution $\hat{\mu}_\lambda$ on $\mathcal{I}(G)$
7 Let $\kappa = \frac{1}{N} \sum_{j=1}^{N} |I_j|$
8 If $|\kappa - k| \leq 1/4$ and there exists $j \in \{1, \dots, N\}$ so that $|I_j| = k$, then output I_j for the smallest such j and **halt**
9 If $\kappa \leq k$, let $\Lambda_i = \{\lambda' \in \Lambda_{i-1} : \lambda' > \lambda\}$. If instead $\kappa > k$, let $\Lambda_i = \{\lambda' \in \Lambda_{i-1} : \lambda' < \lambda\}$
10 **end**
11 If no independent set of size k has been obtained by the end of the for loop (or if $\Lambda_j = \emptyset$ at any step), use a greedy algorithm and output an arbitrary $I \in \mathcal{I}_k(G)$

▶ **Theorem 7.** *Let C be the constant in Line 4 and N be as in Line 6 of Sample-k (Algorithm 1). If the distributions $\hat{\mu}_\lambda$ are each within total variation distance $\varepsilon/(2CN \log n)$ of $\mu_{G,\lambda}$, the output distribution of Sample-k is within total variation distance ε of the uniform distribution of $\mathcal{I}_k(G)$. The running time of Sample-k is $O(N \log n \cdot T(n, \varepsilon))$ where $T(n, \varepsilon)$ is the running time required to produce a sample from $\hat{\mu}_\lambda$ satisfying the above guarantee.*

The sampling part of Theorem 1 follows immediately from Theorem 7 since by Theorem 6 we can obtain $\varepsilon/(2CN\log n)$-approximate samples from $\mu_{G,\lambda}$ in time $O(n\log n\log(n\log n\cdot N/\varepsilon))$. Thus, the total running time of Sample-k with this guarantee on $\hat{\mu}_\lambda$ is

$$O(N\cdot n\log^2 n\cdot\log(nN/\varepsilon))\leq n^3\log^3 n\cdot\operatorname{polylog}\left(\tfrac{\log n}{\varepsilon}\right).$$

Note that the use of Glauber dynamics in Sample-k could be replaced by any other polynomial-time approximate sampler for $\mu_{G,\lambda}$, such as the algorithm due to Weitz [33] based on the method of correlation decay. We do however use rapid mixing of the Glauber dynamics in Lemma 8 below to prove correctness of our algorithm, by establishing a lower bound on the probability that a set sampled from $\mu_{G,\lambda}$ has size close to its mean. Note that e.g. a resolution of our Conjecture 5 could imply the necessary probabilistic statement without appealing to the Glauber dynamics, however.

Before we prove Theorem 7, we collect a number of preliminary results that we will use. The first is a bound on the probability of getting an independent set of size close to the mean from the hard-core model when $\lambda<\lambda_c(\Delta)$. We use the notation $n\alpha_G(\lambda)$ for the expected size of an independent set from the hard-core model on G at fugacity λ to avoid ambiguities.

▶ **Lemma 8.** *For $\Delta\geq 3$ and $\alpha<\alpha_c(\Delta)$, there is a unique $\lambda_*<\lambda_c(\Delta)$ so that $\alpha_{K_{\Delta+1}}(\lambda_*)=\alpha$, and the following holds. For any $G\in\mathcal{G}_\Delta$ on n vertices and any $1\leq k\leq\alpha n$, there exists an integer $t\in\{0,1,\dots,\lfloor 2\lambda_* n^2\rfloor\}$ so that*

$$\left|n\alpha_G(t/(2n^2))-k\right|\leq 1/2\,. \tag{2}$$

Moreover, if t satisfies (2) then

$$\mu_{G,t/(2n^2)}(\mathcal{I}_k(G))=\Omega\left(\frac{1}{n\log n}\right).$$

To prove this lemma we need several more results. The first is an extremal bound on $\alpha_G(\lambda)$ for $G\in\mathcal{G}_\Delta$. The statement of the theorem follows from a stronger property proved by Cutler and Radcliffe in [9]; see [12] for discussion.

▶ **Theorem 9** ([9])**.** *For all $G\in\mathcal{G}_\Delta$ and all $\lambda\geq 0$,*

$$\alpha_G(\lambda)\geq\alpha_{K_{\Delta+1}}(\lambda)=\frac{\lambda}{1+\lambda(\Delta+1)}\,.$$

We next rely on a zero-free region for $Z_G(\lambda)$ due to Peters and Regts [28], so that we can apply the subsequent central limit theorem.

▶ **Theorem 10** ([28])**.** *Let $\Delta\geq 3$ and $\xi\in(0,\lambda_c(\Delta))$. Then there exists $\delta>0$ such that for every $G\in\mathcal{G}_\Delta$ the polynomial Z_G has no roots in the complex plane that lie within distance δ of the real interval $[0,\lambda_c(\Delta)-\xi)$.*

The *probability generating function* of a discrete random variable $\mathbf{X}$ distributed on the non-negative integers is the polynomial in z given by $f(z)=\sum_{j\geq 0}\mathbb{P}(\mathbf{X}=j)z^j$, and the above result shows that at subcritical fugacity the probability generating function of $|\mathbf{I}|$ has no zeros close to 1 in $\mathbb{C}$. This lets us use the following result of Michelen and Sahasrabudhe [25].

▶ **Theorem 11** ([25])**.** *For $n\geq 1$ let $\mathbf{X}_n$ be a random variable taking values in $\{0,\dots,n\}$ with mean μ_n, standard deviation σ_n, and probability generating function f_n. If f_n has no roots within distance δ_n of 1 in $\mathbb{C}$, and $\sigma_n\delta_n/\log n\to\infty$, then $(X_n-\mu_n)/\sigma_n$ tends to a standard normal in distribution.*

The final tools we need are simple bounds on the variance of the size of an independent set from the hard-core model.

▶ **Lemma 12.** *Let G be a graph on n vertices and let $\mathbf{I}$ be a random independent set drawn from the hard-core model on G at fugacity λ. Then, if the maximum degree of G is at most Δ and $M \geq n/(\Delta+1)$ is the size of a largest independent set in G, we have*

$$\frac{\lambda}{(1+\lambda)^{2+\Delta}} M \leq \operatorname{var}(|\mathbf{I}|) \leq n^2 \frac{\lambda}{1+\lambda}\,.$$

Proof. For the upper bound note that $|\mathbf{I}|$ is the sum of the indicator random variables $\mathbf{X}_v$ that the vertex $v \in V(G)$ is in $\mathbf{I}$. Then because $\mathbb{P}(\mathbf{X}_v = 1) \leq \lambda/(1+\lambda)$ for all v, from the Cauchy–Schwarz inequality in the form $\operatorname{cov}(\mathbf{X}_u, \mathbf{X}_v)^2 \leq \operatorname{var}(\mathbf{X}_u)\operatorname{var}(\mathbf{X}_v)$ we obtain

$$\operatorname{var}(|\mathbf{I}|) = \sum_{u \in V(G)} \sum_{v \in V(G)} \operatorname{cov}(\mathbf{X}_u, \mathbf{X}_v) \leq n^2 \frac{\lambda}{1+\lambda}\,.$$

For the lower bound, let J be some fixed independent set in G of maximum size M. Now write $\mathbf{X} = |\mathbf{I}|$, and let $\mathbf{K} = \mathbf{I} \setminus J$. By the law of total variance,

$$\operatorname{var}(\mathbf{X}) = \mathbb{E}[\operatorname{var}(\mathbf{X}|\mathbf{K})] + \operatorname{var}(\mathbb{E}[\mathbf{X}|\mathbf{K}]) \geq \mathbb{E}[\operatorname{var}(\mathbf{X}|\mathbf{K})]\,.$$

But we have $\mathbf{X} = |\mathbf{K}| + |\mathbf{I} \cap J|$, and conditioned on $\mathbf{K}$ the set $|\mathbf{I} \cap J|$ is distributed according to the hard-core model on $J \setminus N_G(\mathbf{K})$, the subset of J uncovered by $\mathbf{K}$. Since J is independent, this is a sum of at most $|J|$ independent, identically distributed Bernoulli random variables with probability $\lambda/(1+\lambda)$.

Now, writing $\mathbf{U} = |J \setminus N_G(\mathbf{K})|$ for the number variables in the sum we have

$$\operatorname{var}(\mathbf{X}) \geq \mathbb{E}[\operatorname{var}(\mathbf{X}|\mathbf{K})] = \frac{\lambda}{(1+\lambda)^2}\mathbb{E}\mathbf{U}\,.$$

A vertex $u \in J$ is uncovered by $\mathbf{K}$ precisely when $N(u) \cap \mathbf{K} = \emptyset$. Then by successive conditioning and the maximum degree condition, the probability that u is uncovered by $\mathbf{K}$ is at least $(1+\lambda)^{-\Delta}$. This means $\mathbb{E}\mathbf{U} \geq |J|(1+\lambda)^{-\Delta}$ and hence

$$\operatorname{var}(\mathbf{X}) \geq \frac{\lambda}{(1+\lambda)^{2+\Delta}} M\,.$$

The assertion $M \geq n/(\Delta+1)$ follows from the fact that any n-vertex graph of maximum degree Δ contains an independent set of size at least $n/(\Delta+1)$, which is easy to prove by analyzing a greedy algorithm. ◀

Now we are ready to prove Lemma 8.

Proof of Lemma 8. A standard calculation gives

$$\frac{\partial}{\partial\lambda}\alpha_G(\lambda) = \frac{1}{n}\frac{\partial}{\partial\lambda}\frac{\lambda Z_G'(\lambda)}{Z_G(\lambda)} = \frac{1}{n\lambda}\operatorname{var}(|\mathbf{I}|)\,,$$

and so Lemma 12 gives that $0 < \alpha_G'(\lambda) \leq n$ for all $\lambda > 0$.

Next, let $\lambda_* < \lambda_c(\Delta)$ be the solution to the equation $\alpha_{K_{\Delta+1}}(\lambda_*) = \alpha$. This means

$$\lambda_* = \frac{\alpha}{1-\alpha(\Delta+1)}\,,$$

as defined in Sample-k. The fact that $\lambda_* < \lambda_c(\Delta)$ follows from the fact that $\alpha < \alpha_c(\Delta) = \alpha_{K_{\Delta+1}}(\lambda_c(\Delta))$, and that occupancy fractions are strictly increasing. Then using Theorem 9 we have that

$$\alpha_G(\lambda_*) \geq \alpha_{K_{\Delta+1}}(\lambda_*) = \alpha\,, \tag{3}$$

and so there exists $\lambda \in (0, \lambda_*]$ such that $n\alpha_G(\lambda) = k$. Using the upper bound on $\alpha'_G(\lambda)$, we see that as λ increases over an interval of length $1/(2n^2)$, the function $n\alpha_G(\lambda)$ can increase by at most $1/2$. Hence, there is at least one integer $t \in \{1, \ldots, \lfloor 2\lambda_* n^2 \rfloor\}$ such that $|n\alpha_G(t/(2n^2)) - k| \leq 1/2$.

The second statement of Lemma 8 follows from a central limit theorem for $|\mathbf{I}|$ and rapid mixing of the Glauber dynamics. There is a close connection between zeros of the probability generating function of $|\mathbf{I}|$ and the zeros of the partition function itself. The probability generating function of $|\mathbf{I}|$ is

$$f(z) = \sum_{j\geq 0} \mathbb{P}_\lambda(|\mathbf{I}| = j) z^j = \sum_{j\geq 0} \frac{i_j(G)\lambda^j z^j}{Z_G(\lambda)} = \frac{Z_G(\lambda z)}{Z_G(\lambda)}\,.$$

Then for λ such that $Z_G(\lambda) \neq 0$, z is a root of f if and only if $z\lambda$ is a root of $Z_G(\lambda)$. By our assumptions on t, when $\lambda = t/(2n^2)$ Theorem 10 gives the existence of $\delta > 0$ such that for all $G \in \mathcal{G}_\Delta$ there are no complex zeros of f within distance δ/λ of 1. This is because Theorem 10 means that $Z_G(z\lambda) = 0$ implies $|z\lambda - \lambda| \geq \delta$. The condition of Theorem 11 which states that $\sigma_n\delta_n/\log n \to \infty$ is met because $\lambda < \lambda_c(\Delta) \leq 4$ and so

$$\sigma_n\delta_n \geq \sqrt{\frac{\lambda}{(1+\lambda)^{2+\Delta}}\frac{n}{\Delta+1}} \cdot \frac{\delta}{\lambda} \geq \Omega\Big(\sqrt{n/\lambda}\Big) > \omega(\log n)\,.$$

Now, let $\lambda = t/(2n^2)$ and suppose that (2) holds, meaning that k is within $1/2$ of $n\alpha_G(\lambda)$. The standard deviation of the size of a set drawn from $\mu_{G,\lambda}$ is at least a constant, which follows from the lower bound in Lemma 12 of $\Omega(\sqrt{\lambda n})$ and the fact that $\lambda \geq \Omega(1/n)$. To see this, note that $\alpha_G(\lambda) \leq \lambda/(1+\lambda)$ for any graph and non-negative fugacity λ. This holds because when $\mathbf{I}$ is a random independent set from the hard-core model, conditioned on a vertex v having no neighbors in $\mathbf{I}$, $v \in \mathbf{I}$ with probability $\lambda/(1+\lambda)$. If v has a neighbor in $\mathbf{I}$ then $v \notin \mathbf{I}$ with probability 1, and the bound follows. Then using (2), we have

$$n\frac{\lambda}{1+\lambda} \geq n\alpha_G(\lambda) \geq k - 1/2 \geq 1/2\,,$$

and so $\lambda \geq \Omega(1/n)$. We deduce that k is within some constant number $r > 0$ of standard deviations of the mean size $n\alpha_G(\lambda)$. The central limit theorem and standard properties of the normal distribution mean that there are constants $\rho > 0$ (small enough as a function of r) and n_0 such that for all $n \geq n_0$, with probability at least ρ, $|\mathbf{I}|$ is at least r standard deviations below the mean, and similarly with probability at least ρ it is at least r standard deviations above the mean. So we have $\mathbb{P}_{G,\lambda}(|\mathbf{I}| \geq k) \geq \rho$ and $\mathbb{P}_{G,\lambda}(|\mathbf{I}| \leq k) \geq \rho$.

The transition probabilities when we are at state I in the Glauber dynamics are given by the following random experiment. Choose a vertex $v \in V(G)$ uniformly at random and let

$$I' = \begin{cases} I \cup \{v\} & \text{with probability } \lambda/(1+\lambda)\,, \\ I \setminus \{v\} & \text{with probability } 1/(1+\lambda)\,. \end{cases}$$

Now if I' is independent in G move to state I', otherwise stay in state I. This means that the sequence of sizes of set visited must take consecutive integer values. By Theorem 6, there is a constant C'' such that from an arbitrary starting state, in $C''n\log n$ steps the distribution π of the current state is within total variation distance $\rho/2$ of the hard-core model. Then the following statements hold.

- **(i)** Starting from an independent set of size at most k, with probability at least $\rho/2$ the state after $C''n\log n$ steps is an independent set of size at least k.
- **(ii)** Starting from an independent set of size at least k, with probability at least $\rho/2$ the state after $C''n\log n$ steps is an independent set of size at most k.

Consider starting from an initial state distributed according to $\mu_{G,\lambda}$. Then every subsequent state is also distributed according to $\mu_{G,\lambda}$, and the above facts mean that for any sequence of $C''n\log n$ consecutive steps, with probability at least $\rho/2$ we see a state of size exactly k. Recalling that $\lambda = t/(2n^2)$, this immediately implies that

$$\mu_{G,t/(2n^2)}(\mathcal{I}_k(G)) \geq \frac{\rho}{2C''n\log n}\,,$$

as required. ◀

Proof of Theorem 7. We first prove the theorem under the assumption that each $\hat{\mu}_\lambda$ is exactly the hard-core measure $\mu_{G,\lambda}$, taking note of how many times we sample from any $\hat{\mu}_\lambda$.

We say a *failure* occurs at step i in the FOR loop if either of the following occur:

1. $|n\alpha_G(\lambda) - \kappa| > 1/4$.
2. $|n\alpha_G(\lambda) - k| \leq 1/2$ but the algorithm did not output an independent set of size k in step i.

We show that the probability that a failure occurs at any time during the algorithm is at most $\varepsilon/2$. By a union bound, it is enough to show that the probability of either type of failure at a given step i is at most $\frac{\varepsilon}{4C\log n}$.

Consider an arbitrary step i with its value of λ. To bound the quantity $\mathbb{P}(|n\alpha_G(\lambda) - \kappa| > 1/4)$, note that κ is the mean of N independent samples from $\hat{\mu}_\lambda$, which we currently assume to be $\mu_{G,\lambda}$. Then we have $\mathbb{E}\kappa = n\alpha_G(\lambda)$ and Hoeffding's inequality gives $\mathbb{P}(|n\alpha_G(\lambda) - \kappa| > 1/4) \leq 2e^{-N/(8n^2)}$, so for this to be at most $\varepsilon/(4C\log n)$ we need only $N \geq \Omega\big(n^2\log\big(\frac{\log n}{\varepsilon}\big)\big)$.

To bound the probability that the current step involves λ such that $|n\alpha_G(\lambda) - k| \leq 1/2$, but we fail to get a set of size k in the N samples, observe that we have N independent trials for getting a set of size k, and each trial succeeds with probability $p \geq c/(n\log n)$ by Lemma 8. Then the probability we see no successful trials is $\big(1 - \frac{c}{n\log n}\big)^N$, which is at most $\varepsilon/(4C\log n)$ for $N \geq \Omega\big(n\log n\cdot\log\big(\frac{\log n}{\varepsilon}\big)\big)$. Thus, we can take $N = \Theta\big(n^2\log\big(\frac{\log n}{\varepsilon}\big)\big)$, as in line 6 of Sample-k.

Next we show that in the event that no failure occurs during the running of the algorithm, the algorithm outputs an independent set I with distribution within $\varepsilon/2$ total variation distance of the uniform distribution on $\mathcal{I}_k(G)$.

We first observe that if no failure occurs, the algorithm at some point reaches a value of λ so that $|n\alpha_G(\lambda) - k| \leq 1/2$. This is a simple consequence of Lemma 8, which means there exists some t with this property, and the binary search structure of the algorithm. In particular, in each iteration of the FOR loop, at line (e) the size of the set Λ_i being searched goes down by (at least) half. Conditioned on no failures, the search also proceeds in the correct half of λ_i because we search the upper half only when $\kappa < k - 1/4$ and so conditioned on no failure we have $n\alpha_G(\lambda) \leq \kappa + 1/4 < k$ and hence using a larger value of λ must bring $n\alpha_G(\lambda)$ closer to k. The case $\kappa > k + 1/4$ is similar. This means that, conditioned on no failures, the algorithm must reach a value of λ such that $|n\alpha_G(\lambda) - k| \leq 1/4$.

Note that $\mu_{G,\lambda}$ conditioned on getting a set of size exactly k is precisely the uniform distribution on $\mathcal{I}_k(G)$, hence if the algorithm outputs an independent set of size k during the FOR loop, its distribution is exactly uniform distribution on $\mathcal{I}_k(G)$. Thus, under the assumption that each $\hat{\mu}_\lambda$ is precisely $\mu_{G,\lambda}$ we have shown that with probability at least $1 - \varepsilon/2$ no failures occur, and hence a perfectly uniform sample from $\mathcal{I}_k(G)$ is output during the FOR loop.

We do not have access to an efficient exact sampler for $\mu_{G,\lambda}$, so we make do with the approximate sampler from Theorem 6. One interpretation of total variation distance is that when each $\hat{\mu}_\lambda$ has total variation distance at most ξ from $\mu_{G,\lambda}$, there is a coupling between $\hat{\mu}_\lambda$ and $\mu_{G,\lambda}$ such that the probability they disagree is at most ξ. Then to prove Theorem 7 we consider a third failure condition: that during any of the calls to a sampling algorithm for any $\hat{\mu}_\lambda$ the output differs from what would have been given by $\mu_{G,\lambda}$ under this coupling. Since we make at most $CN \log n$ calls to such sampling algorithms, provided $\xi \leq \varepsilon/(2CN \log n)$ the probability of any failure of this kind is at most $\varepsilon/2$. Together with the above proof for samples distributed exactly according to $\mu_{G,\lambda}$ which successfully returns uniform samples from $\mathcal{I}_k(G)$ with probability $1 - \varepsilon/2$, we have now shown the existence of a sampler that with probability $1 - \varepsilon$ returns uniform samples from $\mathcal{I}_k(G)$, and makes at most $CN \log n$ calls to a $\varepsilon/(2CN \log n)$-approximate sampler for $\mu_{G,\lambda}$ (at various values of λ). Interpreting this in terms of total variation distance, this means we have an ε-approximate sampler for the uniform distribution on $\mathcal{I}_k(G)$ with running time $O(N \log n \cdot T(n, \varepsilon))$. ◀

2.2 Approximate counting

Given a graph $G = (V, E)$ on n vertices and $j \geq 0$, let $f_j(G) = (j+1)i_{j+1}(G)/i_j(G)$. This $f_j(G)$ has an interpretation as the *expected free volume* over a uniform random independent set $\mathbf{J} \in \mathcal{I}_j(G)$, that is, $f_j = \mathbb{E}|V \setminus (\mathbf{J} \cup N(\mathbf{J}))|$. This holds because each vertex in $V \setminus (\mathbf{J} \cup N(\mathbf{J}))$ can be added to $\mathbf{J}$ to make an independent set of size $j+1$, and each such set is counted $j+1$ times in this way. Then by a simple telescoping product we have

$$i_k(G) = \prod_{j=0}^{k-1} \frac{f_j(G)}{j+1}\,, \tag{4}$$

and hence if for $0 \leq j \leq k-1$ we can obtain a relative ε/k approximation to f_j in time polynomial in n and $1/\varepsilon$ then we can obtain a relative ε-approximation to $i_k(G)$ in time polynomial in n and $1/\varepsilon$. By the definition of f_j as an expectation over a uniform random independent set of size j, we can use an efficient sampling scheme for this distribution to approximate f_j, which is provided by Theorem 7. That is, by repeatedly sampling independent sets of size j approximately uniformly and recording the free volume we can approximate the expected free volume $f_j(G)$, and hence the corresponding term of the product in (4). Doing this for all $0 \leq j \leq k-1$ thus provides an approximation to $i_k(G)$. This scheme is an example of *simulated annealing*, which can be used as a general technique for obtaining approximation algorithms from approximate sampling algorithms. For more details, see e.g. [22, 29]. Here the integer j is playing the role of inverse temperature, and we approximate $i_k(G)$ by estimating $f_j(G)$ (by sampling from $\mathcal{I}_j(G)$) with the cooling schedule $j = 0, 1, \ldots, k-1$. We expect that a more sophisticated cooling schedule can be used to decrease the running time of our reduction, see for example [19].

Since this annealing process is standard, we sketch a simple version of the method. Suppose that for all $0 \leq j \leq k-1$ we have a randomized algorithm that with probability at least $1 - \delta'$ returns a relative ε/k-approximation $\hat{t}_j$ to $f_j(G)/(j+1)$ in time T'. Then (4)

implies that with probability at least $1 - k\delta$, the product $\hat{\imath}_k = \prod_{j=0}^{k-1} \hat{t}_j$ is a relative ε-approximation to $i_k(G)$, and this takes time kT' to compute. For the FPRAS in Theorem 1, it therefore suffices to design the hypothetical algorithm with $\delta' = 1/(4k)$ and T' polynomial in n and $1/\varepsilon$.

First, suppose that we have access to an exactly uniform sampler for $\mathcal{I}_j(G)$ for $0 \le j \le k-1$, but impose the smaller failure probability bound of $\delta'/2$. Then, for each j, let $\hat{t}_j$ be the sample mean of m computations of $|V \setminus (\mathbf{J} \cup N(\mathbf{J}))|/(j+1)$ where $\mathbf{J}$ is a uniform random independent set of size j. We note that as a random variable $|V \setminus (\mathbf{J} \cup N(\mathbf{J}))|/(j+1)$ has a range of at most $j\Delta/(j+1)$ in a graph of maximum degree Δ because $0 \le |N(\mathbf{J})| \le j\Delta$, and for $j \le k-1$ and

$$k \le \alpha n < \alpha_c(\Delta) n < \frac{e}{1+e}\frac{n}{\Delta},$$

we have

$$\frac{|V \setminus (\mathbf{J} \cup N(\mathbf{J}))|}{j+1} \ge \frac{n - j(\Delta+1)}{j+1} \ge \frac{\Delta}{e} - 1.$$

Let S_j be the mean of m samples of $|V \setminus (\mathbf{J} \cup N(\mathbf{J}))|/(j+1)$. Then, using that for $\varepsilon' \le 1$ it suffices to ensure $|S_j - \mu| \le \varepsilon'\mu/2$ for S_j to be a relative ε'-approximation to μ, by Hoeffding's inequality,

$$m \ge \Omega(\varepsilon^{-2}k^2 \log(1/\delta')) = \Omega(\varepsilon^{-2}k^2 \log k)$$

samples are sufficient to obtain the required approximation accuracy ε' with the required success probability $1 - \delta'/2$. Since we do not have an exact sampler, we use the approximate sampler obtained in this section with total variation distance $\delta'/2$. Using the coupling between the exact and the approximate sampler that we used in the proof of Theorem 7, this suffices to obtain the required sampling accuracy with failure probability at most δ'. Recalling that $k \le n$, it is now simple to check that the running time of the entire annealing scheme is polynomial in n and $1/\varepsilon$. This completes the proof of Theorem 1a.

3 Hardness

To prove hardness we will use the notion of an "approximation-preserving reduction" from [13]. We reduce the problem of approximating the hard-core partition function $Z_G(\lambda)$ on a Δ-regular graph G, which we recall is hard for $\lambda > \lambda_c$ (see [16, 31]), to the problem of approximating $i_k(G')$ for Δ-regular graph G' that can be constructed in time polynomial in the size of G. In particular, we show that it suffices to find an $\varepsilon/2$-approximation to $i_k(G')$ in order to obtain an ε-approximation to $Z_G(\lambda)$.

Let $\mathrm{IS}(\alpha, \Delta)$ be the problem of computing $i_{\lfloor \alpha n \rfloor}(G)$ for a Δ-regular graph G on n vertices. Let $\mathrm{HC}(\lambda, \Delta)$ be the problem of computing $Z_G(\lambda)$ for a Δ-regular graph G.

▶ **Theorem 13.** *For every $\Delta \ge 3$ and $\alpha \in (\alpha_c(\Delta), 1/2)$, there exists $\lambda > \lambda_c(\Delta)$ so that there is an approximation-preserving reduction from $\mathrm{HC}(\lambda, \Delta)$ to $\mathrm{IS}(\alpha, \Delta)$.*

Theorem 13 immediately implies the hardness part of Theorem 1 as the results of [16, 31] show that there is no FPRAS for $\mathrm{HC}(\lambda, \Delta)$ for any $\lambda > \lambda_c(\Delta)$ unless NP=RP.

Proof of Theorem 13. Fix $\Delta \ge 3$, and let $\alpha \in (\alpha_c(\Delta), 1/2)$ be given. We will construct a Δ-regular graph H on n_H vertices such that for some value $\lambda \in (\lambda_c(\Delta), \infty)$ we have $\alpha_H(\lambda) = \alpha$. Our reduction is then as follows: given a Δ-regular graph G on n vertices and

$\varepsilon > 0$, let G' be the disjoint union of G with rH, the graph of r disjoint copies of H, with $r = \lceil C\Delta n^2/\varepsilon \rceil$ for some absolute constant C. Let $N = |V(G')| = n + rn_H$. We will prove that

$$e^{-\varepsilon/2}\frac{i_k(G')}{i_k(rH)} \le Z_G(\lambda) \le e^{\varepsilon/2}\frac{i_k(G')}{i_k(rH)}\,, \tag{5}$$

where $k = \lfloor \alpha N \rfloor$.

Since G' can be constructed and $i_k(rH)$ computed in time polynomial in n, this provides the desired approximation-preserving reduction. What remains is to construct the graph H satisfying $\alpha_H(\lambda) = \alpha$ and then to prove (5).

Constructing H

The graph $H = H_{a,b}$ will consist of the union of a copies of the complete bipartite graph $K_{\Delta,\Delta}$ and b copies of the clique $K_{\Delta+1}$. Clearly H is Δ-regular. Since the occupancy fraction of any graph is a strictly increasing function of λ, and the relevant occupancy fractions satisfy $\alpha_{K_{\Delta+1}}(\lambda_c(\Delta)) = \alpha_c(\Delta)$ and $\lim_{\lambda\to\infty}\alpha_{K_{\Delta,\Delta}}(\lambda) = 1/2$, we see that there exist integers $a, b \ge 0$ (with at least one positive) and $\lambda > \lambda_c(\Delta)$ so that $\alpha_{H_{a,b}}(\lambda) = \alpha$. A given pair (a, b) provides a suitable $H_{a,b}$ when

$$\alpha_{H_{a,b}}(\lambda_c(\Delta)) < \alpha < \lim_{\lambda\to\infty}\alpha_{H_{a,b}}(\lambda) = \frac{a\Delta + b}{2a\Delta + b(\Delta+1)}\,,$$

and hence it can be shown that for all $\Delta \ge 3$ one of the pairs $(0,1)$, $(1,16)$, $(1,6)$, $(1,3)$, $(2,3)$, $(2,1)$, $(1,0)$ suffices for (a,b), and a suitable pair is easy to find efficiently. This provides us with the desired graph H. From here on, fix these values a, b, λ and let $n_H = 2a\Delta + b(\Delta+1)$.

Proving (5)

We now form G' by taking the union of G (a Δ-regular graph on n vertices) and r copies of H. Let $N = n + rn_H$ be the number of vertices of G', and write $k = \lfloor \alpha N \rfloor$. Let rH be the graph consisting of the disjoint union of r copies of H. We can write:

$$i_k(G') = \sum_{j=0}^{n} i_j(G) i_{k-j}(rH) = i_k(rH)\sum_{j=0}^{n} i_j(G)\frac{i_{k-j}(rH)}{i_k(rH)}\,.$$

Now to prove (5) it suffices to show that for $r \ge C\Delta n^2/\varepsilon$ and $0 \le j \le n$, we have

$$e^{-\varepsilon/2}\lambda^j \le \frac{i_{k-j}(rH)}{i_k(rH)} \le e^{\varepsilon/2}\lambda^j\,. \tag{6}$$

We have the exact formula (for any $0 \le j \le k$)

$$i_{k-j}(rH) = \frac{Z_{rH}(\lambda)}{\lambda^{k-j}}\mathbb{P}_{rH,\lambda}(|\mathbf{I}| = k-j)$$

and so

$$\frac{i_{k-j}(rH)}{i_k(rH)} = \lambda^j\frac{\mathbb{P}_{rH,\lambda}(|\mathbf{I}| = k-j)}{\mathbb{P}_{rH,\lambda}(|\mathbf{I}| = k)}\,,$$

where $\mathbb{P}_{rH,\lambda}$ denotes probabilities with respect to an independent set $\mathbf{I}$ drawn according to the hard-core model on rH at fugacity λ. It is then enough to show

$$e^{-\varepsilon/2} \le \frac{\mathbb{P}_{rH,\lambda}(|\mathbf{I}| = k-j)}{\mathbb{P}_{rH,\lambda}(|\mathbf{I}| = k)} \le e^{\varepsilon/2}\,.$$

This will follow from a local central limit theorem (e.g. [17]) since $|\mathbf{I}|$ is the sum of r i.i.d. random variables and the fact that $\mathbb{E}_{rH,\lambda}|\mathbf{I}|$ is close to both k and $k-j$. The following theorem gives us what we need.

▶ **Theorem 14** (Gnedenko [17]). *Let $X_1, \dots, X_r$ be i.i.d. integer valued random variables with mean μ and variance σ^2, and suppose that the support of X_1 includes two consecutive integers. Let $S_r = X_1 + \cdots + X_r$. Then*

$$\mathbb{P}(S_r = k) = \frac{1}{\sqrt{2\pi r}\sigma} \exp\left[-(k-n\mu)^2/(2r\sigma^2)\right] + o(r^{-1/2}),$$

with the error term $o(r^{-1/2})$ uniform in k.

This immediately implies that with μ and σ^2 the mean and standard deviation of the hard-core model on H at fugacity λ,

$$\frac{\mathbb{P}_{rH,\lambda}(|\mathbf{I}| = k-j)}{\mathbb{P}_{rH,\lambda}(|\mathbf{I}| = k)} = \frac{e^{-[j^2 - 2(k-r\mu)j]/(2r\sigma^2)} + o(e^{(k-r\mu)^2/(2r\sigma^2)}/r)}{1 + o(e^{(k-r\mu)^2/(2r\sigma^2)}/r)}.$$

It therefore suffices to show that for large enough r, namely $r \geq C\Delta n^2/\varepsilon$, we can make $[j^2 - 2(k-r\mu)j]/(2r\sigma^2)$ small compared to ε and show that $(k-r\mu)^2/(2r\sigma^2)$ is bounded above by some absolute constant. Note that $\mu = \alpha n_H$, and by Lemma 12 we have for all $\Delta \geq 3$ (and any choices of α, λ, a, b made according to our conditions),

$$\sigma^2 \geq \frac{\lambda}{(1+\lambda)^{2+\Delta}}(a\Delta + b) \geq \frac{\lambda_c(\Delta)}{(1+\lambda_c(\Delta))^{2+\Delta}}(a\Delta + b) \geq \frac{0.00384}{\Delta}.$$

Since $k = \lfloor \alpha N \rfloor = \lfloor \alpha n + r\alpha n_H \rfloor$, we then have $(k - r\mu)^2 \leq \alpha^2 n^2 < n^2$, and hence $\frac{(k-r\mu)^2}{2r\sigma^2} \leq C'\Delta\frac{n^2}{r}$, where C' is an absolute constant. Now since $0 \leq j \leq n$ we also have

$$\left|\frac{j^2 - 2(k-r\mu)j}{2r\sigma^2}\right| \leq C'\Delta\frac{n^2}{r}.$$

This means that provided we take C to be a large enough absolute constant and $r \geq C\Delta n^2/\varepsilon$, we have (5) as required. ◀

4 Triangle-free graphs

In this section we briefly describe the modifications to the proofs in Sections 2 and 3 to yield Theorem 2, an analogue of Theorem 1 for triangle-free graphs.

4.1 Algorithms

We use the following lower bound on the occupancy fraction of triangle-free graphs.

▶ **Theorem 15** ([11]). *For every $\delta > 0$, there is Δ_0 large enough so that for every $\Delta \geq \Delta_0$, and every triangle-free $G \in \mathcal{G}_\Delta$,*

$$\alpha_G(\lambda_c(\Delta) - 1/\Delta^2) \geq \frac{1-\delta}{\Delta}.$$

This statement follows from [11, Theorem 3] and some asymptotic analysis of the bound for $\lambda = \lambda_c(\Delta) - 1/\Delta^2$ as $\Delta \to \infty$. Now the algorithm for Theorem 2 is essentially the same as for Theorem 1, but since we assume the graph G is triangle free we can use a stronger

lower bound on the occupancy fraction than Theorem 9. Let $\delta > 0$ and $\alpha < (1-\delta)/\Delta$ as in Theorem 2. Then Theorem 15 means that for $\Delta \geq \Delta_0$ and any triangle-free graph $G \in \mathcal{G}_\Delta$ we have

$$\alpha_G(\lambda_c(\Delta) - 1/\Delta^2) \geq \frac{1-\delta}{\Delta} > \alpha\,.$$

But occupancy fractions are continuous and strictly increasing, so with $\lambda_* = \lambda_c(\Delta) - 1/\Delta^2$ there exists $\lambda \in (0, \lambda_*]$ such that $k = n\alpha_G(\lambda)$, as in the proof of Lemma 8 but permitting larger α. The analysis of the algorithm can then proceed exactly as in the proofs of Lemma 8 and Theorem 1, completing the proof of Theorem 2a.

4.2 Hardness

The proof of hardness for triangle-free graphs is the same as that for general graphs, but we replace $K_{\Delta+1}$ with a (constant-sized) random regular graph in the construction. Bhatnagar, Sly, and Tetali [4] showed that the local distribution of the hard-core model on the random regular graph converges to that of the unique translation-invariant hard-core measure on the infinite regular tree for a range of λ including $\lambda = \lambda_c(\Delta)$. This means that if K is a random Δ-regular graph on n vertices and α_{T_Δ} denotes the occupancy fraction of the unique translation-invariant hard-core measure on the infinite Δ-regular tree (see [4, 11]) we have with probability $1 - o_n(1)$,

$$\alpha_K(\lambda_c(\Delta)) = \alpha_{T_\Delta}(\lambda_c(\Delta)) + o_n(1) = \frac{1 + o_{n,\Delta}(1)}{\Delta}\,,$$

where $o_n(1) \to 0$ as $n \to \infty$ and $o_{n,\Delta}(1) \to 0$ as both n and Δ tend to infinity. Thus, for fixed $\delta \in (0,1)$, there is $n_0 = n_0(\delta)$ and $\Delta_0 = \Delta_0(\delta)$ such that with probability at least $1-\delta$ a random Δ-regular graph K on n_0 vertices has $\alpha_G(\lambda_c(\Delta)) \leq (1+\delta)/\Delta$. This means that in time bounded by a function of δ an exhaustive search over Δ-regular graphs on n_0 vertices must yield a graph K with the property $\alpha_K(\lambda_c(\Delta)) \leq (1+\delta)/\Delta$. Now we replace $K_{\Delta+1}$ with the random Δ-regular graph K in the proof above, which for $\Delta \geq \Delta_0$ allows us to work with any $\alpha \in ((1+\delta)/\Delta, 1/2)$ by the above argument. To finish the proof, we require that approximating $Z_G(\lambda)$ is hard for Δ-regular *triangle-free* graphs G when $\lambda > \lambda_c$. This follows directly from the proof of Sly and Sun [31], as their gadget which shows hardness for Δ-regular graphs contains no triangles. Thus, we have the following analogue of Theorem 13, where we let $\mathrm{IS}'(\alpha, \Delta)$ be the problem of computing $i_{\lfloor \alpha n \rfloor}(G)$ for a Δ-regular triangle-free graph G on n vertices.

▶ **Theorem 16.** *Given $\delta > 0$ there exists Δ_0 such that the following holds for all $\Delta \geq \Delta_0$. For every $\alpha \in ((1+\delta)/\Delta, 1/2)$, there exists $\lambda > \lambda_c(\Delta)$ so that there is an approximation-preserving reduction from $\mathrm{HC}(\lambda, \Delta)$ to $\mathrm{IS}'(\alpha, \Delta)$.*

This implies Theorem 2b.

References

1 Vedat Levi Alev and Lap Chi Lau. Improved analysis of higher order random walks and applications. In *Proceedings of the 52nd Annual ACM SIGACT Symposium on Theory of Computing*, pages 1198–1211, Chicago IL USA, 2020. ACM. doi:10.1145/3357713.3384317.

2 N. Anari, K. Liu, and S. O. Gharan. Spectral Independence in High-Dimensional Expanders and Applications to the Hardcore Model. In *2020 IEEE 61st Annual Symposium on Foundations of Computer Science (FOCS)*, pages 1319–1330, 2020. doi:10.1109/FOCS46700.2020.00125.

3 Alexander Barvinok. *Combinatorics and Complexity of Partition Functions*, volume 30 of *Algorithms and Combinatorics*. Springer International Publishing, Cham, 2016. doi:10.1007/978-3-319-51829-9.

4 Nayantara Bhatnagar, Allan Sly, and Prasad Tetali. Decay of correlations for the hardcore model on the d-regular random graph. *Electronic Journal of Probability*, 21(0), 2016. doi:10.1214/16-EJP3552.

5 R. Bubley and M. Dyer. Path coupling: A technique for proving rapid mixing in Markov chains. In *Proceedings 38th Annual Symposium on Foundations of Computer Science*, pages 223–231, Miami Beach, FL, USA, 1997. IEEE Comput. Soc. doi:10.1109/SFCS.1997.646111.

6 Z. Chen, K. Liu, and E. Vigoda. Rapid Mixing of Glauber Dynamics up to Uniqueness via Contraction. In *2020 IEEE 61st Annual Symposium on Foundations of Computer Science (FOCS)*, pages 1307–1318, November 2020. doi:10.1109/FOCS46700.2020.00124.

7 Zongchen Chen, Kuikui Liu, and Eric Vigoda. Optimal Mixing of Glauber Dynamics: Entropy Factorization via High-Dimensional Expansion. *CoRR*, 2020. arXiv:2011.02075.

8 Radu Curticapean, Holger Dell, Fedor Fomin, Leslie Ann Goldberg, and John Lapinskas. A fixed-parameter perspective on #BIS. *Algorithmica*, 81(10):3844–3864, 2019.

9 Jonathan Cutler and A.J. Radcliffe. The maximum number of complete subgraphs in a graph with given maximum degree. *Journal of Combinatorial Theory, Series B*, 104:60–71, 2014. doi:10.1016/j.jctb.2013.10.003.

10 Ewan Davies, Matthew Jenssen, and Will Perkins. A proof of the Upper Matching Conjecture for large graphs. *CoRR*, April 2020. arXiv:2004.06695.

11 Ewan Davies, Matthew Jenssen, Will Perkins, and Barnaby Roberts. On the average size of independent sets in triangle-free graphs. *Proc. Amer. Math. Soc.*, 146(1):111–124, 2017. doi:10.1090/proc/13728.

12 Ewan Davies, Matthew Jenssen, Will Perkins, and Barnaby Roberts. Tight bounds on the coefficients of partition functions via stability. *Journal of Combinatorial Theory, Series A*, 160:1–30, November 2018. doi:10.1016/j.jcta.2018.06.005.

13 Martin Dyer, Leslie Ann Goldberg, Catherine Greenhill, and Mark Jerrum. The relative complexity of approximate counting problems. *Algorithmica*, 38(3):471–500, 2004. doi:10.1007/978-3-642-04016-0.

14 Talya Eden, Dana Ron, and C. Seshadhri. On Approximating the Number of k-Cliques in Sublinear Time. *SIAM Journal on Computing*, 49(4):747–771, January 2020. doi:10.1137/18M1176701.

15 Friedrich Eisenbrand and Fabrizio Grandoni. On the complexity of fixed parameter clique and dominating set. *Theoretical Computer Science*, 326(1):57–67, 2004. doi:10.1016/j.tcs.2004.05.009.

16 Andreas Galanis, Daniel Štefankovič, and Eric Vigoda. Inapproximability of the Partition Function for the Antiferromagnetic Ising and Hard-Core Models. *Combinatorics, Probability and Computing*, 25(4):500–559, July 2016. doi:10.1017/S0963548315000401.

17 B. V. Gnedenko. On a local limit theorem of the theory of probability. *Uspehi Matem. Nauk (N. S.)*, 3(3(25)):187–194, 1948.

18 C. Greenhill. The complexity of counting colourings and independent sets in sparse graphs and hypergraphs. *Comput. complex.*, 9(1):52–72, November 2000. doi:10.1007/PL00001601.

19 David G. Harris and Vladimir Kolmogorov. Parameter estimation for Gibbs distributions. *CoRR*, 2021. arXiv:2007.10824.

20 Mark Jerrum and Alistair Sinclair. Approximating the permanent. *SIAM Journal on Computing*, 18(6):1149–1178, 1989.

21 Mark Jerrum and Alistair Sinclair. Polynomial-Time Approximation Algorithms for the Ising Model. *SIAM Journal on Computing*, 22(5):1087–1116, October 1993. doi:10.1137/0222066.

22 Mark Jerrum and Alistair Sinclair. The Markov chain Monte Carlo method: An approach to approximate counting and integration. In *Approximation Algorithms for NP-Hard Problems*. PWS Pub. Co, 1997.

23 F. P. Kelly. Stochastic Models of Computer Communication Systems. *Journal of the Royal Statistical Society. Series B (Methodological)*, 47(3):379–395, 1985.

24 J.L. Lebowitz, B. Pittel, D. Ruelle, and E.R. Speer. Central limit theorems, Lee–Yang zeros, and graph-counting polynomials. *Journal of Combinatorial Theory, Series A*, 141:147–183, July 2016. doi:10.1016/j.jcta.2016.02.009.

25 Marcus Michelen and Julian Sahasrabudhe. Central limit theorems and the geometry of polynomials. *CoRR*, 2019. arXiv:1908.09020.

26 Marcus Michelen and Julian Sahasrabudhe. Central limit theorems from the roots of probability generating functions. *Advances in Mathematics*, 358:106840, December 2019. doi:10.1016/j.aim.2019.106840.

27 Viresh Patel and Guus Regts. Deterministic Polynomial-Time Approximation Algorithms for Partition Functions and Graph Polynomials. *SIAM J. Comput.*, 46(6):1893–1919, January 2017. doi:10.1137/16M1101003.

28 Han Peters and Guus Regts. On a Conjecture of Sokal Concerning Roots of the Independence Polynomial. *Michigan Math. J.*, 68(1):33–55, April 2019. doi:10.1307/mmj/1541667626.

29 Alistair Sinclair and Mark Jerrum. Approximate counting, uniform generation and rapidly mixing Markov chains. *Information and Computation*, 82(1):93–133, July 1989. doi:10.1016/0890-5401(89)90067-9.

30 Allan Sly. Computational Transition at the Uniqueness Threshold. In *2010 IEEE 51st Annual Symposium on Foundations of Computer Science*, pages 287–296, Las Vegas, NV, USA, October 2010. IEEE. doi:10.1109/FOCS.2010.34.

31 Allan Sly and Nike Sun. Counting in two-spin models on d-regular graphs. *Ann. Probab.*, 42(6):2383–2416, November 2014. doi:10.1214/13-AOP888.

32 Leslie G. Valiant. The Complexity of Enumeration and Reliability Problems. *SIAM J. Comput.*, 8(3):410–421, August 1979. doi:10.1137/0208032.

33 Dror Weitz. Counting independent sets up to the tree threshold. In *Proceedings of the Thirty-Eighth Annual ACM Symposium on Theory of Computing - STOC '06*, page 140, Seattle, WA, USA, 2006. ACM Press. doi:10.1145/1132516.1132538.

34 Yufei Zhao. Extremal regular graphs: independent sets and graph homomorphisms. *The American Mathematical Monthly*, 124(9):827–843, 2017. doi:10.4169/amer.math.monthly.124.9.827.

Linear Time Runs Over General Ordered Alphabets

Jonas Ellert
Department of Computer Science, Technical University of Dortmund, Germany

Johannes Fischer
Department of Computer Science, Technical University of Dortmund, Germany

Abstract

A run in a string is a maximal periodic substring. For example, the string `bananatree` contains the runs $\texttt{anana} = (\texttt{an})^{5/2}$ and $\texttt{ee} = \texttt{e}^2$. There are less than n runs in any length-n string, and computing all runs for a string over a linearly-sortable alphabet takes $\mathcal{O}(n)$ time (Bannai et al., SIAM J. Comput. 2017). Kosolobov conjectured that there also exists a linear time runs algorithm for general ordered alphabets (Inf. Process. Lett. 2016). The conjecture was almost proven by Crochemore et al., who presented an $\mathcal{O}(n\alpha(n))$ time algorithm (where $\alpha(n)$ is the extremely slowly growing inverse Ackermann function). We show how to achieve $\mathcal{O}(n)$ time by exploiting combinatorial properties of the Lyndon array, thus proving Kosolobov's conjecture. This also positively answers the at least 29-year-old question whether square-freeness can be tested in linear time over general ordered alphabets (Breslauer, PhD thesis, Columbia University 1992).

2012 ACM Subject Classification Theory of computation → Design and analysis of algorithms

Keywords and phrases String algorithms, Lyndon array, runs, squares, longest common extension, general ordered alphabets, combinatorics on words

Digital Object Identifier 10.4230/LIPIcs.ICALP.2021.63

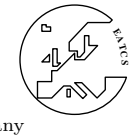
Category Track A: Algorithms, Complexity and Games

Supplementary Material *Software (Source Code)*: `https://github.com/jonas-ellert/linear-time-runs/`; archived at `swh:1:snp:aa7b1dc6293d939b5ec6e554d3102b15a518b7e7`

1 Introduction and Related Work

A run in a string S is a maximal periodic substring. For example, the string $S = \texttt{bananatree}$ contains exactly the runs $\texttt{anana} = (\texttt{an})^{5/2}$ and $\texttt{ee} = \texttt{e}^2$. Identifying such repetitive structures in strings is of great importance for applications like text compression, text indexing and computational biology (for a general overview see [8]). To name just one example, runs in human genes (called maximal tandem repeats) are involved with a number of neurological disorders [5]. In 1999, Kolpakov and Kucherov showed that the maximum number $\rho(n)$ of runs in a length-n string is bounded by $\mathcal{O}(n)$, and provided a word RAM algorithm that outputs all runs in linear time [18]. The algorithm is based on the Lempel-Ziv factorization and only achieves $\mathcal{O}(n)$ time for *linearly-sortable alphabets*, i.e. alphabets that are totally ordered and for which a sequence of σ alphabet symbols can be sorted in $\mathcal{O}(\sigma)$ time. Since then, it has been an open question whether there exists a linear time runs algorithm for *general ordered alphabets*, i.e. totally ordered alphabets for which the order of any two symbols can be determined in constant time. Any such algorithm must not use the Lempel-Ziv factorization, since for general ordered alphabets of size σ it cannot be constructed in $o(n \lg \sigma)$ time [19].

Kolpakov and Kucherov also conjectured that the maximum number of runs is bounded by $\rho(n) < n$, which started a 15 year-long search for tighter upper bounds of $\rho(n)$. Rytter was the first to give an explicit constant with $\rho(n) < 5n$ [25]. After multiple incremental improvements of this bound (e.g. [7, 9, 24]), Bannai et al. [2] finally proved the conjecture by showing $\rho(n) < n$ for arbitrary alphabets, which was subsequently even surpassed for binary texts [12]. (The current best bound for binary alphabets is $\rho(n) < \frac{183}{193}n$ [17].)



On the algorithmic side, Bannai et al. also provided a new linear time algorithm that computes all the runs [2]. While (just like the algorithm by Kolpakov and Kucherov) it only achieves the time bound for linearly-sortable alphabets, it no longer relies on the Lempel-Ziv factorization. Instead, the main effort of the algorithm lies in the computation of $\Theta(n)$ *longest common extensions (LCEs)*; given two indices $i, j \in [1, n]$, their LCE is the length of the longest common prefix of the suffixes $S[i..n]$ and $S[j..n]$. For linearly-sortable alphabets, we can precompute a data structure in $\mathcal{O}(n)$ time that answers arbitrary LCE queries in constant time (see e.g. [11]), thus yielding a linear time runs algorithm. Kosolobov showed that for general ordered alphabets any batch of $\mathcal{O}(n)$ LCEs can be computed in $\mathcal{O}(n \lg^{2/3} n)$ time, and conjectured the existence of a linear time runs algorithm for general ordered alphabets [20]. Gawrychowski et al. improved this result to $\mathcal{O}(n \lg \lg n)$ time [14]. Finally, Crochemore et al. noted that the required LCEs satisfy a special non-crossing property. They showed how to compute $\mathcal{O}(n)$ non-crossing LCEs in $\mathcal{O}(n\alpha(n))$ time, resulting in an $\mathcal{O}(n\alpha(n))$ time algorithm that computes all runs over general ordered alphabets [10] (where α is the inverse Ackermann function).

This is also the asymptotically fastest known algorithm for testing whether a string is square-free. A square is a substring $\alpha\alpha$ for some non-empty word α. A string is square-free if and only if it contains no runs (because every square is contained in a run, and every run contains at least one square). The question whether square-freeness over general ordered alphabets can be tested in linear time dates back at least to Breslauer's PhD dissertation [4, Section 4.4], which was published almost 30 years ago.[1] Testing for square-freeness over general *unordered* alphabets (where only constant time equality testing of symbols is permitted) takes at least $\Omega(n \lg n)$ symbol comparisons [22].

Our Contributions. We show how to compute the LCEs required by the algorithm by Bannai et al. in $\mathcal{O}(n)$ time and space, resulting in the first linear time runs algorithm for general ordered alphabets. Thus we prove Kosolobov's conjecture, and provide the first linear time algorithm to test for square-freeness over general ordered alphabets. Our solution differs from all previous approaches in the sense that it cannot answer a sequence of *arbitrary* non-crossing LCE queries. Instead, our algorithm is specifically designed *exactly for the LCEs required by the runs algorithm*. This allows us to utilize powerful combinatorial properties of the *Lyndon array* (a definition follows in Section 2) that do not generally hold for arbitrary non-crossing LCE sequences.

Even though the main contribution of our work is the improved asymptotic time bound, it is worth mentioning that our algorithm is also very fast in practice. On modern hardware, computing all runs for a text of length 10^7(= 10MB) takes only one second.

A Note on the Model. As mentioned earlier, our algorithm runs in linear time for *general ordered alphabets*, whereas previous algorithms achieve this time bound only when the alphabet is *linearly-sortable*. This is comparable with the distinction between *comparison-based* and *integer* sorting: while in the comparison-model sorting n items requires $\Omega(n \lg n)$ time, integer sorting is faster ($O(n\sqrt{\lg \lg n})$ time [16] and sometimes even linear, e.g. when the word width w satisfies $w = \mathcal{O}(\lg n)$ and one can use radix sort, or when $w \geq (\lg^{2+\epsilon} n)$ [1]). Whereas it is a major open problem whether integer sorting can always be done in linear time, this paper settles a symmetric open problem for the computation of runs.

[1] We thank the reviewer who kindly pointed out to us that this was an open problem.

The remainder of the paper is structured as follows: First, we introduce the basic notation, definitions, and auxiliary lemmas (Section 2). Then, we give a simplified description of the runs algorithm by Bannai et al. and show how the required LCEs relate to the Lyndon array (Section 3). Our linear time algorithm to compute the LCEs is described in Section 4. We discuss additional practical aspects and experimental results in Section 5.

2 Preliminaries

Our analysis is performed in the word RAM model (see e.g. [15]), where we can perform fundamental operations (logical shifts, basic arithmetic operations etc.) on words of size w bits in constant time. For an input of size n we assume $\lceil \log_2 n \rceil \leq w$. We write $[i,j] = [i,j+1) = (i-1,j] = (i-1,j+1)$ with $i,j \in \mathbb{N}$ to denote the set of integers $\{x \mid x \in \mathbb{N} \wedge i \leq x \leq j\}$.

Strings. Let Σ be a finite and totally ordered set. A *string* S of length $|S| = n$ over the *alphabet* Σ is a sequence $S[1] \dots S[n]$ of n *symbols* (also called *characters*) from Σ. The alphabet is called a *general ordered alphabet* if order testing (i.e. evaluating $\sigma_1 < \sigma_2$ for $\sigma_1, \sigma_2 \in \Sigma$) is possible in constant time. For $i,j \in [1,n]$, we use the interval notation $S[i..j] = S[i..j+1) = S(i-1..j] = S(i-1..j+1)$ to denote the *substring* $S[i] \dots S[j]$. If however $i > j$, then $S[i..j]$ denotes the *empty string* ϵ. The substring $S[i..j]$ is called *proper* if $S[i..j] \neq S$. A *prefix* of S is a substring $S[1..j]$ (including $S[1..0] = \epsilon$), while the *suffix* S_i is the substring $S[i..n]$ (including $S_{n+1} = \epsilon$). Given two strings S and T of length n and m respectively, their concatenation is defined as $ST = S[1] \dots S[n]T[1] \dots T[m]$. For any positive integer k, the k-times concatenation of S is denoted by S^k. Let $\ell_{\max} = \min(n,m)$. The *longest common prefix (LCP)* of S and T has length $\textsc{lcp}(S,T) = \max\{\ell \mid \ell \in [0,\ell_{\max}] \wedge S[1..\ell] = T[1..\ell]\}$, while the *longest common suffix* has length $\textsc{lc-suff}(S,T) = \max\{\ell \mid \ell \in [0,\ell_{\max}] \wedge S_{n-\ell+1} = T_{m-\ell+1}\}$. Let $\ell' = \textsc{lcp}(S,T)$. For a string S of length n and indices $i,j \in [1,n]$, we define the *longest common right-extension (R-LCE) and left-extension (L-LCE)* as $\textsc{lce}_r(i,j) = \textsc{lcp}(S_i,S_j)$ and $\textsc{lce}_\ell(i,j) = \textsc{lc-suff}(S[1..i],S[1..j])$ respectively. The total order on Σ induces a *lexicographical order* $\prec$ on the strings over Σ in the usual way. Given three suffixes, we can deduce properties of their R-LCEs from their lexicographical order:

▶ **Lemma 1.** *Let $S_i \prec S_j \prec S_k$ be suffixes of a string, then it holds $\textsc{lce}_r(i,k) \leq \textsc{lce}_r(i,j)$ and $\textsc{lce}_r(i,k) \leq \textsc{lce}_r(j,k)$.*

Proof. Assume $\ell = \textsc{lce}_r(i,j) < \textsc{lce}_r(i,k)$, then $S_i[1..\ell] = S_j[1..\ell] = S_k[1..\ell]$ and $S_j[\ell+1] \neq S_i[\ell+1] = S_k[\ell+1]$. This implies $S_i \prec S_j \Leftrightarrow S_k \prec S_j$, which contradicts $S_i \prec S_j \prec S_k$. The proof of $\textsc{lce}_r(i,k) \leq \textsc{lce}_r(j,k)$ works analogously. ◀

Repetitions and Runs. Let S be a string and let $S[i..j]$ be a non-empty substring. We say that $p \in \mathbb{N}^+$ is a *period* of $S[i..j]$ if and only if $\forall x \in [i,j-p] : S[x] = S[x+p]$. If additionally $(j-i+1) \geq p$, then $S[i..i+p)$ is called *string period* of $S[i..j]$. Furthermore, p is called *shortest period* of $S[i..j]$ if there is no $q \in [1,p)$ that is also a period of $S[i..j]$. Analogously, a string period of $S[i..j]$ is called *shortest string period* if there is no shorter string period of $S[i..j]$. A *run* is a triple $\langle i,j,p \rangle$ such that p is the shortest period of $S[i..j]$, $(j-i+1) \geq 2p$ (i.e. there are at least two consecutive occurrences of the shortest string period $S[i..i+p)$), and neither $\langle i-1,j,p \rangle$ nor $\langle i,j+1,p \rangle$ satisfies these properties (assuming $i > 1$ and $j < n$, respectively).

Lyndon Words and Nearest Smaller Suffixes. For a length-n string S and $i \in [1, n]$, the string $S_i S[1..i)$ is called *cyclic shift* of S, and *non-trivial cyclic shift* if $i > 1$. A *Lyndon word* is a non-empty string that is lexicographically smaller than any of its non-trivial cyclic shifts, i.e. $\forall i \in [2, n] : S \prec S_i S[1..i)$. The Lyndon array of S identifies the longest Lyndon word starting at each position of S.

▶ **Definition 2** (Lyndon Array). *Given a string S of length n, its Lyndon array $\lambda[1, n]$ is defined by $\forall i \in [1, n] : \lambda[i] = \max\{j - i + 1 \mid j \in [i, n] \wedge S[i..j]$ is a Lyndon word$\}$.*

An alternative representation of the Lyndon array is the next-smaller-suffix array.

▶ **Definition 3** (Next Smaller Suffixes). *Given a string S of length n, its* next-smaller-suffix (NSS) array *is defined by $\forall i \in [1, n] : \mathsf{nss}[i] = \min\{j \mid j = n + 1 \vee (j \in (i, n] \wedge S_i \succ S_j)\}$. If $\mathsf{nss}[i] \leq n$, then $S_{\mathsf{nss}[i]}$ is called the* next smaller suffix of S_i.

▶ **Lemma 4** (Lemma 15 [13]). *The longest Lyndon word starting at any position $i \in [1, n]$ of a length-n string S is exactly the substring $S[i..\mathsf{nss}[i])$, i.e. $\forall i \in [1, n] : \lambda[i] = \mathsf{nss}[i] - i$.*

An example of the Lyndon and NSS array is provided in Figure 1a. The NSS edges in the example do not intersect. This property also holds in the general case:

▶ **Lemma 5.** *Let $i \in [1, n]$ and $i' \in [i, \mathsf{nss}[i])$. Then it holds $\mathsf{nss}[i'] \leq \mathsf{nss}[i]$.*

Proof. Due to $i' \in [i, \mathsf{nss}[i])$ and Definition 3 it holds $S_{i'} \succ S_{\mathsf{nss}[i]}$. Assume that the lemma does not hold, then we have $\mathsf{nss}[i] \in (i', \mathsf{nss}[i'])$ and Definition 3 implies $S_{i'} \prec S_{\mathsf{nss}[i]}$. ◀

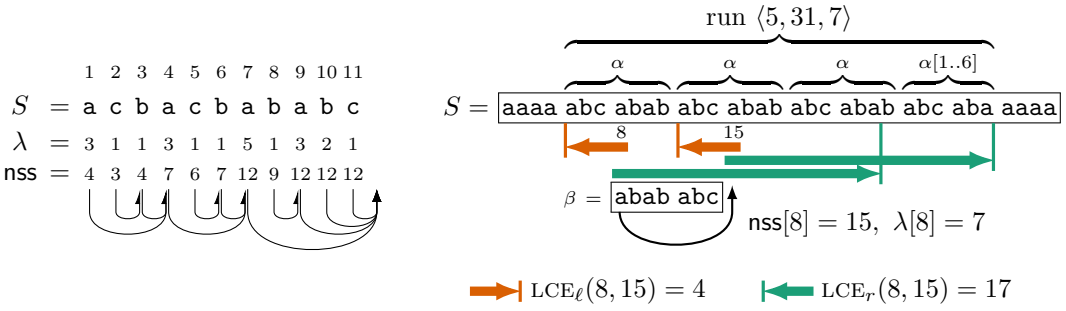
3 The Runs Algorithm Revisited

In this section, we recapitulate the main ideas of the algorithm by Bannai et al. [2], which is the basis of our solution for general ordered alphabets. The key insight is that every run is *rooted* in a longest Lyndon word, allowing us to compute all runs from the Lyndon array.

▶ **Definition 6.** *Let $\langle i, j, p\rangle$ be a run in a string S. We say that $\langle i, j, p\rangle$ is* (lexicographically) decreasing *if and only if $S_i \succ S_{i+p}$. Otherwise, $\langle i, j, p\rangle$ is* (lexicographically) increasing.

▶ **Lemma 7.** *Let $\langle i, j, p\rangle$ be a decreasing run, then there is exactly one index $i_0 \in [i..i + p)$ such that $\lambda[i_0] = p$.*

Proof. Consider any $i_0 \in [i, i+p)$. By the definition of runs, we have $S[i..i_0) = S[i+p..i_0+p)$. Since the run is decreasing it follows $S_i \succ S_{i+p} \iff S[i..i_0)S_{i_0} \succ S[i + p..i_0 + p)S_{i_0+p} \iff S_{i_0} \succ S_{i_0+p}$. This implies $\mathsf{nss}[i_0] \leq i_0 + p$, and due to Lemma 4 also $\lambda[i_0] \leq p$. Next, we show that there is at least one index $i_0 \in [i..i + p)$ such that $S[i_0..i_0 + p)$ is a Lyndon word. Let $\alpha = S[i..i+p)$. Assume that the described index i_0 does not exist, then from $S[i..i+2p) = \alpha\alpha$ follows that no cyclic shift of α is a Lyndon word. Let β be a lexicographically minimal cyclic shift of α, then this shift is not unique (otherwise β would be a Lyndon word), and thus there must be a cyclic shift $\beta_k\beta[1..k) = \beta[1..k)\beta_k$ with $k > 1$. This however implies that β is of the form $\beta = \mu^k$ for some string μ and an integer $k > 1$ (see Lemma 3 in [21]), which contradicts the fact that α is the *shortest* string period of the run. Finally, let $\alpha_k\alpha[1..k)$ with $k \in [1, p]$ be the unique lexicographically smallest cyclic shift of α (and thus a Lyndon word), then it is easy to see that only $i_0 = i + k - 1$ satisfies $\lambda[i_0] = p$. ◀



(a) String $S = \texttt{acbacbababc}$, its Lyndon array λ, and its NSS array nss.

(b) Decreasing run $\langle 5, 31, 7\rangle$ with $S[5..31] = (\texttt{abcabab})^{27/7}$. The run has shortest string period $\alpha = \texttt{abcabab}$, and is rooted in position 8 (with longest Lyndon word $\beta = S[8..15) = \alpha_4\alpha[1..3] = \texttt{abababc}$).

Figure 1 An edge from text position a to text position b indicates $\mathsf{nss}[a] = b$.

▶ **Definition 8** (Root of a Run). *Let $\langle i, j, p\rangle$ be a decreasing run, and let $i_0 \in [i..i + p)$ be the unique index with $\lambda[i_0] = p$ (as described in Lemma 7). We say that $\langle i, j, p\rangle$ is* rooted in i_0.

An example of a decreasing run and its root is provided in Figure 1b. Note that our notion of a root differs from the L-roots introduced by Crochemore et al. [6]. While an L-root is *any* length-p Lyndon word contained in the run, our root is exactly the *leftmost* one.

Given a longest Lyndon word $S[i_0..\mathsf{nss}[i_0])$ of length $p = \mathsf{nss}[i_0] - i_0 = \lambda[i_0]$, it is easy to determine whether i_0 is the root of a decreasing run. We simply try to extend the periodicity as far as possible to both sides by using the LCE functions. For this purpose, we only need to compute $l = \textsc{lce}_\ell(i_0, \mathsf{nss}[i_0])$ and $r = \textsc{lce}_r(i_0, \mathsf{nss}[i_0])$. Let $i = i_0 - l + 1$ and $j = \mathsf{nss}[i_0] + r - 1$, then clearly the substring $S[i..j]$ has smallest period p, and we cannot extend the substring to either side without breaking the periodicity. Thus, if $j - i + 1 \geq 2p$ then $\langle i, j, p\rangle$ is a run. Note that this run is only rooted in i_0 if additionally $i_0 \in [i..i + p)$ (or equivalently $l \leq p$) holds. For the index $i_0 = 8$ in Figure 1b, we have $l = \textsc{lce}_\ell(8, 15) = 4$ and $r = \textsc{lce}_r(8, 15) = 17$. Therefore, the run starts at position $i = 8 - 4 + 1 = 5$ and ends at position $j = 15 + 17 - 1 = 31$. From $l = 4 \leq 7 = p$ follows that 8 is actually the root.

Since each decreasing run is rooted in exactly one index, we can find all decreasing runs by checking for each index whether it is the root of a run. This procedure is outlined in Algorithm 1. First, we compute the NSS array (line 2). Then, we investigate one index $i_0 \in [1, n]$ at a time (line 3), and consider it as the root of a run with period $p = \mathsf{nss}[i_0] - i_0$ (line 4). If the left-extension covers an entire period (i.e. $\textsc{lce}_\ell(i_0, \mathsf{nss}[i_0]) > p$), then we have already investigated the root of the run in an earlier iteration of the for-loop, and no further action is required (line 5). Otherwise, we compute the left and right border of the potential run as described earlier (lines 6–7). If the resulting interval has length at least $2p$, then we have discovered a run that is rooted in i_0 (lines 8–9).

Time and space complexity. The NSS array can be computed in $\mathcal{O}(n)$ time and space for general ordered alphabets [3]. Assume for now that we can answer L-LCE and R-LCE queries in constant time, then clearly the rest of the algorithm also requires $\mathcal{O}(n)$ time and space. The correctness of the algorithm follows from Lemma 7 and the description. We have shown:

Algorithm 1 Compute all decreasing runs.

Input: String S of length n.
Output: Set R of all decreasing runs in S.

```
1: R ← ∅
2: compute array nss
3: for i_0 ∈ [1, n] with nss[i_0] ≠ n + 1 do
4:     p ← nss[i_0] − i_0
5:     if LCE_ℓ(i_0, nss[i_0]) ≤ p then
6:         i ← i_0 − LCE_ℓ(i_0, nss[i_0]) + 1
7:         j ← nss[i_0] + LCE_r(i_0, nss[i_0]) − 1
8:         if j − i + 1 ≥ 2p then
9:             R ← R ∪ {⟨i, j, p⟩}
```

▶ **Lemma 9.** *Let S be a string of length n over a general ordered alphabet, and let* nss *be its NSS array. We can compute all decreasing runs of S in $\mathcal{O}(n) + t(n)$ time and $\mathcal{O}(n) + s(n)$ space, where $t(n)$ and $s(n)$ are the time and space needed to compute $\text{LCE}_\ell(i, \mathsf{nss}[i])$ and $\text{LCE}_r(i, \mathsf{nss}[i])$ for all $i \in [1, n]$ with $\mathsf{nss}[i] \neq n + 1$.*

In order to also find all *increasing* runs, we only need to rerun the algorithm with reversed alphabet order. This way, previously increasing runs become decreasing.

4 Algorithm for Computing the LCEs

In this section, we show how to precompute the LCEs required by Algorithm 1 in linear time and space. Our approach is asymmetric in the sense that we require different algorithms for L-LCEs and R-LCEs (whereas previous approaches usually compute L-LCEs by applying the R-LCE algorithm to the reverse text). However, for both directions we use similar properties of the Lyndon array that are shown in Lemmas 10 and 11 and visualized in Figure 2a.

▶ **Lemma 10.** *Let $i \in [1, n]$ and $j = \mathsf{nss}[i] \neq n + 1$. If $\text{LCE}_r(i, j) \geq (j - i)$, then it holds $\text{LCE}_r(j, j + (j - i)) = \text{LCE}_r(i, j) - (j - i)$ and $\mathsf{nss}[j] = j + (j - i)$.*

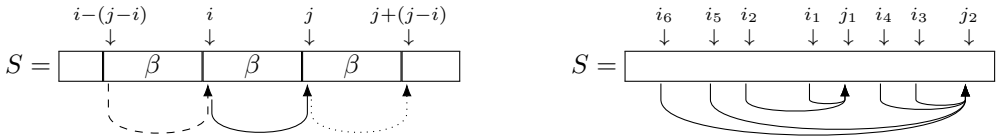
Proof. From $\text{LCE}_r(i, j) \geq (j - i)$ follows $\text{LCE}_r(i, j) = (j - i) + \text{LCE}_r(j, j + (j - i))$, which is equivalent to $\text{LCE}_r(j, j + (j - i)) = \text{LCE}_r(i, j) - (j - i)$. It remains to be shown that $\mathsf{nss}[j] = j + (j - i)$. Due to $\mathsf{nss}[i] = j$ it holds $S_i \succ S_j$. Since $S_i \succ S_j$ and $\text{LCE}_r(i, j) \geq (j - i)$, we have $S_{i+(j-i)} \succ S_{j+(j-i)}$, which implies $\mathsf{nss}[j] \leq j + (j - i)$. Note that $\mathsf{nss}[i] = j$ and Lemma 4 imply that $S[i..j) = S[j..j + (j - i))$ is a Lyndon word. Thus it holds $\lambda[j] \geq (j - i)$, or equivalently $\mathsf{nss}[j] \geq j + (j - i)$. ◀

▶ **Lemma 11.** *Let $i \in [1, n]$ and $j = \mathsf{nss}[i] \neq n + 1$. If $\text{LCE}_\ell(i, j) > (j - i)$, then it holds $\text{LCE}_\ell(i - (j - i), i) = \text{LCE}_\ell(i, j) - (j - i)$ and $\mathsf{nss}[i - (j - i)] = i$.*

Proof. Analogous to Lemma 10. ◀

4.1 Computing the R-LCEs

First, we will briefly describe our general technique for computing LCEs, and our method of showing the linear time bound. Assume for this purpose that we want to compute $\ell = \text{LCE}_r(i, j)$ with $i < j$. It is easy to see that we can determine ℓ by performing $\ell + 1$ individual character comparisons (by simultaneously scanning the suffixes S_i and S_j from left



(a) Lemmas 10 and 11. The dotted edge follows from $\text{LCE}_r(i,j) \geq (j-i)$ (Lemma 10). The dashed edge follows from $\text{LCE}_\ell(i,j) > (j-i)$ (Lemma 11).

(b) Relative order of R-LCE computations from first to last: $\text{LCE}_r(i_1,j_1)$, $\text{LCE}_r(i_2,j_1)$, $\text{LCE}_r(i_3,j_2)$, $\text{LCE}_r(i_4,j_2)$, $\text{LCE}_r(i_5,j_2)$, $\text{LCE}_r(i_6,j_2)$.

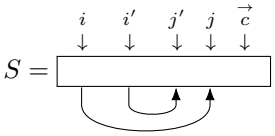
Figure 2 As before, an edge from text position a to text position b indicates $\mathsf{nss}[a] = b$.

to right until we find a mismatch). Whenever we use this naive way of computing an LCE, we *charge* one character comparison to each of the indices from the interval $[j, j+\ell)$. This way, we account for ℓ character comparisons. Since we want to compute $\mathcal{O}(n)$ R-LCE values in $\mathcal{O}(n)$ time, we can afford a constant time overhead (i.e. a constant number of unaccounted character comparisons) for each LCE computation. Thus, there is no need to charge the $(\ell+1)$-th comparison to any index. At the time at which we want to compute ℓ, we may already know some lower bound $k \leq \ell$. In such cases, we simply skip the first k character comparisons and compute $\ell = k + \text{LCE}_r(i+k, j+k)$. This requires $\ell - k + 1$ character comparisons, of which we charge $\ell - k$ to the interval $[j+k..j+\ell)$.

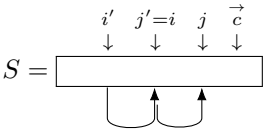
Ultimately, we will show that all R-LCE values $\text{LCE}_r(i,j)$ with $i \in [1,n]$ and $j = \mathsf{nss}[i] \neq n+1$ can be computed in a way such that each text position gets charged at most once, which results in the desired linear time bound. From now on, we refer to i as the *left index* and j as the *right index* of the R-LCE computation. Our algorithm computes the R-LCEs in the following order (a visualization is provided in Figure 2b): We consider the possible right indices $j \in [2,n]$ one at a time and in *increasing* order. For each right index j, we then consider the corresponding left indices i with $\mathsf{nss}[i] = j$ in *decreasing* order (we will see how to efficiently deduce this order from the Lyndon array later).

Assume that we are computing the R-LCEs in the previously described order, and let $\ell = \text{LCE}_r(i,j)$ with $j = \mathsf{nss}[i] \neq n+1$ be the next value that we want to compute. The set of indices that we have already considered as left indices for LCE computations is $I = \{x \mid (\mathsf{nss}[x] < j) \vee ((\mathsf{nss}[x] = j) \wedge (i < x))\}$. For example, when we compute $\text{LCE}_r(i_4, j_2)$ in Figure 2b it holds $\{i_1, i_2, i_3\} \subseteq I$. At this point in time, the rightmost text position that we have already inspected is $\overrightarrow{c} = \max_{x \in I}(\mathsf{nss}[x] + \text{LCE}_r(x, \mathsf{nss}[x]))$ if $I \neq \emptyset$, or $\overrightarrow{c} = 1$ otherwise. Due to the nature of our charging method, we have not charged any indices from the interval $[\overrightarrow{c}, n]$ yet. Thus, in order to show that we can compute all LCEs without charging any index twice, it suffices to show how to compute $\ell = \text{LCE}_r(i,j)$ without charging any index from the interval $[1, \overrightarrow{c})$. If $j \geq \overrightarrow{c}$ then we naively compute ℓ and charge the character comparisons to the interval $[j, j+\ell)$, thus only charging previously uncharged indices. The new value of $\overrightarrow{c}$ is $j + \ell$. If however $j < \overrightarrow{c}$, then the computation of ℓ depends on the previously computed LCEs, which we describe in the following.

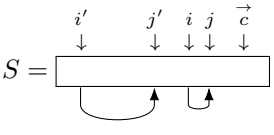
Let $\ell' = \text{LCE}_r(i', j')$ with $j' = \mathsf{nss}[i']$ be the *most recently* computed R-LCE that satisfies $j' + \ell' = \overrightarrow{c}$. Our strategy for computing ℓ depends on the position of i relative to i' and j'. First, note that $i \notin [i', j')$ because otherwise Lemma 5 implies $j \leq j'$, which contradicts our order of computation. This leaves us with three possible cases (as before, a directed edge from text position a to text position b indicates $\mathsf{nss}[a] = b$):



Case R1: $i < i'$
(possibly $j' = j$)



Case R2: $i = j'$

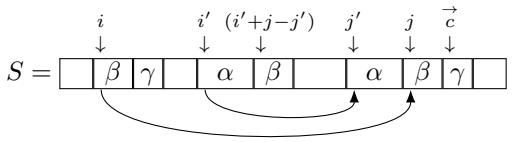


Case R3: $i > j'$

Now we explain the cases in detail. Each case is accompanied by a schematic drawing. We strongly advise the reader to study the drawings alongside the description, since they are essential for an easy understanding of the matter.

Case R1: $i < i'$ (and $j' \leq j < \vec{c}$)**.**

$|\alpha| = j - j'$, $|\beta| = \vec{c} - j$
$\ell' = |\alpha\beta|$, $\ell = |\beta\gamma|$

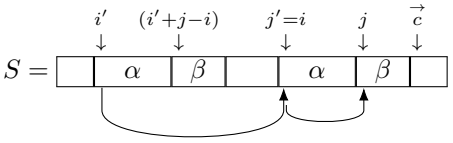


Due to $i < (i' + j - j') < j = \mathsf{nss}[i]$ we have $S_j \prec S_i \prec S_{i'+j-j'}$. From Lemma 1 follows $\vec{c} - j = \text{LCE}_r(i' + j - j', j) \leq \text{LCE}_r(i, j) = \ell$, i.e. both S_i and S_j start with β. Since now we know a lower bound $\vec{c} - j \leq \ell$ of the desired LCE value, we can skip character comparisons during its computation. Later, we will see that the same bound also holds for most of the other cases. Generally, whenever we can show $\vec{c} - j \leq \ell$ we use the following strategy. We compute $\ell = (\vec{c} - j) + \text{LCE}_r(i + (\vec{c} - j), \vec{c})$ using $\ell - (\vec{c} - j) + 1$ character comparisons, of which we charge $\ell - (\vec{c} - j)$ to the interval $[\vec{c}, j + \ell)$. Thus we only charge previously uncharged positions. We continue with $i' \leftarrow i$, $j' \leftarrow j$, $\ell' \leftarrow \ell$, and $\vec{c} \leftarrow j + \ell$.

Case R2: $i = j'$. We divide this case into two subcases.

Case R2a: $\ell' < j' - i'$.

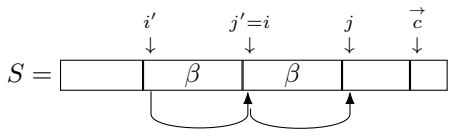
$|\alpha| = j - j'$, $|\beta| = \vec{c} - j$



From $j < \vec{c} \implies j - i < \vec{c} - i = \ell'$ and $\ell' < j' - i'$ follows $i' + j - i < j' = i$. Therefore, $\mathsf{nss}[i'] = i$ and Definition 3 imply $S_i \prec S_{i'+j-1}$. Due to $\mathsf{nss}[i] = j$ we also have $S_j \prec S_i$, such that it holds $S_j \prec S_i \prec S_{i'+j-1}$. It is easy to see that $S_{i'+j-i}$ and S_j share a prefix β of length $\text{LCE}_r(i' + j - i, j) = \vec{c} - j$. In fact, also S_i has prefix β because Lemma 1 implies that $\text{LCE}_r(i' + j - i, j) \leq \text{LCE}_r(i, j) = \ell$. Thus it holds $\vec{c} - j \leq \ell$, which allows us to use the strategy from Case R1.

Case R2b: $\ell' \geq j' - i'$.

$|\beta| = j' - i'$, $\ell = \ell' - |\beta|$

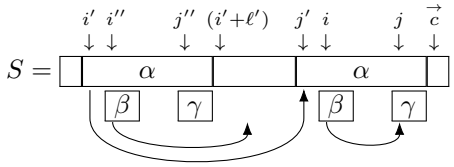


Due to $\ell' \geq j' - i'$, Lemma 10 implies $j = i + (j' - i')$ and $\ell = \ell' - (j' - i')$. Since i', j', and ℓ' are known, we can compute ℓ in constant time without performing any character comparisons. We continue with $i' \leftarrow i$, $j' \leftarrow j$, and $\ell' \leftarrow \ell$ (leaving $\vec{c}$ unchanged).

Case R3: $i > j'$. This is the most complicated case, and it is best explained by dividing it into three subcases. Let $d = j' - i'$, $i'' = i - d$, $j'' = j - d$, and $\ell'' = \text{LCE}_r(i'', j'')$. (In this situation it is implied that $j'' \leq j'$ because otherwise $\ell' = \text{LCE}_r(i', j')$ would not be the *most recently* computed R-LCE that satisfies $j' + \ell' = \overrightarrow{c}$. However, since our proof does not rely on this property, we will not explain it in more detail.)

Case R3a: $\mathsf{nss}[i''] \neq j''$:

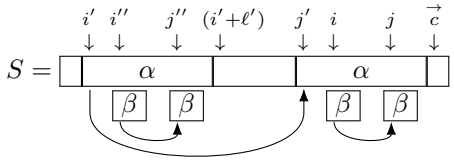
$|\alpha| = \ell'$, $|\beta| = |\gamma| = \overrightarrow{c} - j$
$\ell'' \geq |\beta|$, $\ell \geq |\beta|$



First, note that $S[i'..i' + \ell') = S[j'..\overrightarrow{c})$ implies $S[i..j) = S[i''..j'')$. From $\mathsf{nss}[i] = j$ follows that $S[i..j) = S[i''..j'')$ is a Lyndon word. Thus, due to Lemma 4 and $\mathsf{nss}[i''] \neq j''$ it holds $\mathsf{nss}[i''] > j''$, which implies $S_{i''} \prec S_{j''}$. Let $\beta = S[i''..i'' + \overrightarrow{c} - j) = S[i..i + \overrightarrow{c} - j)$ and let $\gamma = S[j''..i' + \ell') = S[j..\overrightarrow{c})$. From $S_{i''} \prec S_{j''}$ follows $\beta \preceq \gamma$, while $S_i \succ S_j$ implies $\beta \succeq \gamma$. Thus it holds $\beta = \gamma$, and therefore $\text{LCE}_r(i, j) \geq |\gamma| = \overrightarrow{c} - j$. This means that we can use the strategy from Case R1.

Case R3b: $\mathsf{nss}[i''] = j''$ and $(j'' + \ell'') < (i' + \ell')$:

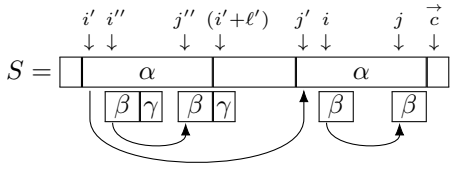
$|\alpha| = \ell'$, $|\beta| = \ell'' = \ell$



Due to $\ell'' = \text{LCE}_r(i'', j'')$, there is a shared prefix $\beta = S[i''..i'' + \ell'') = S[j''..j'' + \ell'')$ between $S_{i''}$ and $S_{j''}$, and the first mismatch between the two suffixes is $S[i'' + \ell''] \neq S[j'' + \ell'']$. Because of $(j'' + \ell'') < (i' + \ell')$, both the shared prefix and the mismatch are contained in $S[i'..i' + \ell')$ (i.e. in the first occurrence of α). If we consider the substring $S[j'..j' + \ell')$ instead (i.e. the second occurrence of α), then S_i and S_j clearly also share the prefix $\beta = S[i..i + \ell'') = S[j..j + \ell'')$, with the first mismatch occurring at $S[i + \ell''] \neq S[j + \ell'']$. Thus it holds $\ell = \ell''$. Due to $\mathsf{nss}[i''] = j''$ and our order of R-LCE computations, we have already computed ℓ''. Therefore, we can simply assign $\ell \leftarrow \ell''$ and continue without changing i', j', ℓ', and $\overrightarrow{c}$.

Case R3c: $\mathsf{nss}[i''] = j''$ and $(j'' + \ell'') \geq (i' + \ell')$:

$|\alpha| = \ell'$, $|\beta| = \overrightarrow{c} - j$, $|\beta\gamma| = \ell''$
$\ell \geq |\beta|$



This situation is similar to Case R3b. There is a shared prefix $\beta = S[i''..i'' + \overrightarrow{c} - j) = S[j''..i' + \ell')$ between the suffixes $S_{i''}$ and $S_{j''}$. They may share an even longer prefix $\beta\gamma$, but only the first $|\beta| = \overrightarrow{c} - j$ symbols of their LCP are contained in $S[i'..i' + \ell')$ (i.e. in the first occurrence of α). If we consider the substring $S[j'..j' + \ell')$ instead (i.e. the second occurrence of α), then S_i and S_j clearly also share at least the prefix $\beta = S[i..i + \overrightarrow{c} - j) = S[j..\overrightarrow{c})$. Thus it holds $\overrightarrow{c} - j \leq \ell$, and we can use the strategy from Case R1.

We have shown how to compute ℓ without charging any index twice. It follows that the total number of character comparisons for all R-LCEs is $\mathcal{O}(n)$.

A Simple Algorithm for R-LCEs. While the detailed differentiation between the six subcases helps to show the correctness of our approach, it can be implemented in a significantly simpler way (see Algorithm 2). At all times, we keep track of j', $\overrightarrow{c}$ and the distance $d = j' - i'$ (line 1). We consider the indices $j \in [2, n]$ in increasing order (line 2). For each index j, we then consider the indices i with $\mathsf{nss}[i] = j$ in decreasing order (line 3). Each time we want to compute an R-LCE value $\ell = \text{LCE}_r(i, j)$, we first check whether Case R3b applies (line 4). If it does, then we simply copy the previously computed R-LCE value $\text{LCE}_r(i-d, j-d)$ (line 5). Otherwise, we either compute the LCE naively (if $j \geq \overrightarrow{c}$), or we have to apply the strategy from Case R1 (since all other cases except for Case R2b use this strategy; in Case R2b it holds $\overrightarrow{c} - j = \ell$, which means that it can also be solved with the strategy from Case R1). If $j \geq \overrightarrow{c}$ then in lines 7–8 we have $k = 0$, and thus we naively compute $\text{LCE}_r(i, j)$ by scanning. If however $j < \overrightarrow{c}$, then we have $k = \overrightarrow{c} - j$, and we skip k character comparisons. In any case, we update the values j', $\overrightarrow{c}$, and d accordingly (line 9).

Algorithm 2 Compute all R-LCEs.

Input: String S of length n and its NSS array nss.
Output: R-LCE value $\text{LCE}_r(i, \mathsf{nss}[i])$ for each index $i \in [1, n]$ with $\mathsf{nss}[i] \neq n + 1$.

1: $j' \leftarrow 0$; $\overrightarrow{c} \leftarrow 1$; $d \leftarrow 0$
2: **for** $j \in [2, n]$ in increasing order **do**
3: **for** i **with** $\mathsf{nss}[i] = j \neq n + 1$ in decreasing order **do**
4: **if** $\left(\begin{array}{l} i, j \in (j', \overrightarrow{c}) \\ \wedge\ \mathsf{nss}[i-d] = j - d \\ \wedge\ j + \text{LCE}_r(i-d, j-d) < \overrightarrow{c} \end{array}\right)$ **then**
5: $\text{LCE}_r(i, j) \leftarrow \text{LCE}_r(i-d, j-d)$ ▷ retrieve LCE in constant time
6: **else**
7: $k \leftarrow \max(\overrightarrow{c}, j) - j$
8: $\text{LCE}_r(i, j) \leftarrow k + \text{NAIVE-SCAN-LCE}_r(i+k, j+k)$
9: $j' \leftarrow j$; $\overrightarrow{c} \leftarrow j + \text{LCE}_r(i, j)$; $d \leftarrow j - i$

The correctness of the algorithm follows from the description of Cases 1–3. Since for each left index i we have to store at most one R-LCE, we can simply maintain the LCEs in a length-n array, where the i-th entry is $\text{LCE}_r(i, \mathsf{nss}[i])$. This way, we use linear space and can access the R-LCE that is required in line 5 in constant time. Apart from the at most n character comparisons that we charge to the indices, we only need a constant number of additional primitive operations per computed R-LCE. The order of iteration can be realized by first generating all $(i, \mathsf{nss}[i])$-pairs, and then using a linear time radix sorter to sort the pairs in increasing order of their second component and decreasing order of their first component. We have shown:

▶ **Lemma 12.** *Given a string of length n and its NSS array nss, we can compute $\text{LCE}_r(i, \mathsf{nss}[i])$ for all indices $i \in [1, n]$ with $\mathsf{nss}[i] \neq n + 1$ in $\mathcal{O}(n)$ time and space.*

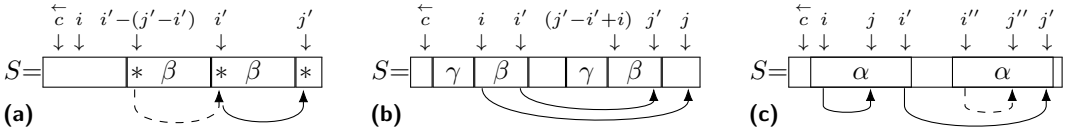


Figure 3 Illustration of the proofs of the three properties in Section 4.2.

4.2 Computing the L-LCEs

Our solution for the L-LCEs is similar to the one for R-LCEs, but differs in subtle details. We generally compute $\ell = \text{LCE}_\ell(i, j)$ by simultaneously scanning the prefixes $S[1..i]$ and $S[1..j]$ from right to left until we find the first mismatch. This takes $\ell + 1$ character comparisons, of which we charge ℓ comparisons to the interval $(i - \ell, i]$. As before, if some lower bound $k \leq \ell$ is known then we skip k character comparisons. In this case, we compute the L-LCE as $\ell = k + \text{LCE}_\ell(i - k, j - k)$, and charge $\ell - k$ comparisons to the interval $(i - \ell, i - k]$.

Again, we will show how to compute all values $\text{LCE}_\ell(i, \mathsf{nss}[i])$ with $i \in [1, n]$ and $\mathsf{nss}[i] \neq n + 1$ such that each index gets charged at most once. In contrast to the more complex R-LCE iteration order, we can simply compute the L-LCE values in *decreasing* order of i. Thus, when we want to compute $\ell = \text{LCE}_\ell(i, j)$ with $j = \mathsf{nss}[i] \neq n + 1$, we have already considered the indices $I = \{x \mid x \in (i, n] \wedge \mathsf{nss}[x] \neq n + 1\}$ as left indices of L-LCE computations. The leftmost text position that we have already inspected so far at this point is $\overleftarrow{c} = \min_{x \in I}(x - \text{LCE}_\ell(x, \mathsf{nss}[x]))$ if $I \neq \emptyset$, or $\overleftarrow{c} = n$ otherwise. Due to our charging method, we have not charged any index from the interval $[1, \overleftarrow{c}]$ yet. Thus, we only have to show how to compute ℓ without charging indices from $(\overleftarrow{c}, n]$. Let $\ell' = \text{LCE}_\ell(i', j')$ be the most recently computed L-LCE that satisfies $i' - \ell' = \overleftarrow{c}$. If $i \leq \overleftarrow{c}$ then we compute ℓ naively and charge the character comparisons to the interval $(i - \ell, i]$ (thus only charging previously uncharged indices). If however $i > \overleftarrow{c}$, then our strategy is more complicated. Before explaining it in detail, we show three important properties that hold in the present situation.

First, we show that $i \geq i' - (j' - i')$. Assume the opposite (as visualized in Figure 3a), then from $\overleftarrow{c} = i' - \ell' < i$ follows $\ell' > j' - i'$. Thus, Lemma 11 implies $\mathsf{nss}[i' - (j' - i')] = i'$ (dashed edge) and $\text{LCE}_\ell(i' - (j' - i'), i') = \ell' - (j' - i')$. Due to our order of computation and $i < i' - (j' - i')$ we must have already computed this L-LCE. However, it holds $i' - (j' - i') - \text{LCE}_\ell(i' - (j' - i'), i') = \overrightarrow{c}$, which contradicts the fact that $\ell' = \text{LCE}_\ell(i', j')$ is the *most recently* computed L-LCE with $i' - \ell' = \overleftarrow{c}$.

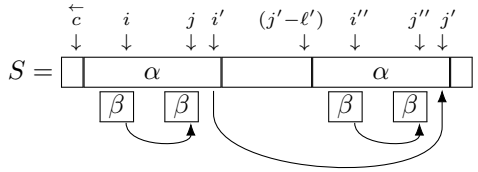
Next, we show that $j \leq i'$. First, note that $j \notin (i', j')$, since due to $i < i'$ we would otherwise contradict Lemma 5. Thus we only have to show $j < j'$. Assume for this purpose that $j \geq j'$ (as visualized in Figure 3b). From $j' - i' + i \in (i, \mathsf{nss}[i])$ and Definition 3 follows $S_i \prec S_{j'-i'+i}$. Because of $\text{LCE}_\ell(i', j') > (i' - i)$ it holds $S[i..i'] = S[j' - i' + i..j'] (= \beta)$. Thus $S_i \prec S_{j'-i'+i}$ implies $S_{i'} \prec S_{j'}$, which contradicts the fact that $\mathsf{nss}[i'] = j'$.

Lastly, let $d = j' - i'$, $i'' = i + d$, and $j'' = j + d$ (as visualized in Figure 3c). Now we show that $\mathsf{nss}[i''] = j''$ (dashed edge in the figure). Because of $\alpha = S(\overleftarrow{c}..i'] = S(j' - \ell'..j']$ it holds $S[i..j) = S[i''..j'')$. From $\mathsf{nss}[i] = j$ and Lemma 4 follows that $S[i''..j'')$ is a Lyndon word, and thus $\mathsf{nss}[i''] \geq j''$. We have already shown that $i \geq i' - (j' - i')$, which implies $i'' \geq i'$. Due to $\mathsf{nss}[i'] = j'$ and $i'' \in [i', j')$ it follows from Lemma 5 that $\mathsf{nss}[i''] \leq j'$. Now assume $\mathsf{nss}[i''] \in (j'', j']$, then $S[i''..\mathsf{nss}[i'']) = S[i..j + (\mathsf{nss}[i''] - j''))$ is a Lyndon word, which contradicts the fact that $S[i..j)$ is the longest Lyndon word starting at position i. Thus, we have ruled out all possible values of $\mathsf{nss}[i'']$ except for j''.

Now we show how to compute ℓ. We keep using the definition of i'' and j'' from the previous paragraph. Furthermore, let $\ell'' = \text{LCE}_\ell(i'', j'')$. There are two possible cases.

Case L1: $(i'' - \ell'') > (j' - \ell')$.

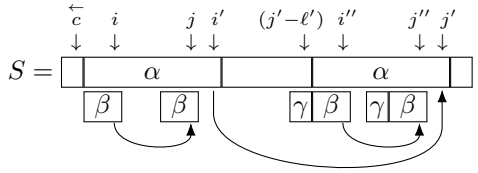
$\ell' = |\alpha|$, $\ell = \ell'' = |\beta|$



Due to $\ell'' = \text{LCE}_\ell(i'', j'')$, the prefixes $S[1..i'']$ and $S[1..j'']$ share the suffix $\beta = S(i'' - \ell''..i''] = S(j'' - \ell''..j'']$, and the first (from the right) mismatch between these prefixes is $S[i'' - \ell''] \neq S[j'' - \ell'']$. Both the shared suffix and the mismatch are contained in $S(j' - \ell'..j']$ (i.e. in the right occurrence of α). If we consider the substring $S(\overleftarrow{c}..i']$ instead (i.e. the left occurrence of α), then $S[1..i]$ and $S[1..j]$ clearly also share the suffix $\beta = S(i - \ell''..i] = S(j - \ell''..j]$, with the first mismatch occurring at $S[i - \ell''] \neq S[j'' - \ell]$. Thus it holds $\ell = \ell''$. Due to $\mathsf{nss}[i''] = j''$ and our order of L-LCE computations, we have already computed ℓ''. Therefore, we can simply assign $\ell \leftarrow \ell''$ and continue without changing i', j', ℓ', and $\overleftarrow{c}$.
(Note that possibly $i'' \neq i' \wedge j'' = j'$. We provide a sketch in Figure 4a.)

Case L2: $(i'' - \ell'') \leq (j' - \ell')$.

$\ell' = |\alpha|$, $\ell'' = |\beta\gamma|$, $\ell \geq |\beta|$



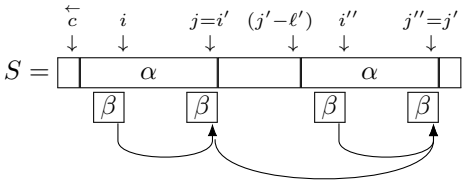
This situation is similar to Case L1. There is a shared suffix $\beta = S(j' - \ell'..i''] = S(j'' - (i - \overleftarrow{c})..j'']$ between the prefixes $S[1..i'']$ and $S[1..j'']$. They may share an even longer suffix $\gamma\beta$, but only the rightmost $|\beta| = i' - \overleftarrow{c}$ symbols of this suffix are contained in $S(j' - \ell'..j']$ (i.e. in the right occurrence of α). If we consider the substring $S(\overleftarrow{c}..i']$ instead (i.e. the left occurrence of α), then $S[1..i]$ and $S[1..j]$ clearly also share the suffix $\beta = S(\overleftarrow{c}..i] = S(j - (i - \overleftarrow{c})..j]$. Thus it holds $i - \overleftarrow{c} \leq \ell$, and we can skip the first $i - \overleftarrow{c}$ character comparisons by computing the LCE as $\ell = (i - \overleftarrow{c}) + \text{LCE}_\ell(\overleftarrow{c}, j + \overleftarrow{c} - i)$. We charge $\ell - (i - \overleftarrow{c})$ character comparisons to the previously uncharged interval $(i - \ell, \overleftarrow{c}]$, and continue with $i' \leftarrow i$, $j' \leftarrow j$, $\ell' \leftarrow \ell$, and $\overleftarrow{c} \leftarrow i - \ell$.
(Note that possibly $i'' \neq i' \wedge j'' = j'$ or even $i'' = i' \wedge j'' = j'$. We provide schematic drawings in Figures 4b and 4c.)

We have shown how to compute ℓ without charging any index twice. It follows that the total number of character comparisons for all LCEs is $\mathcal{O}(n)$. For completeness, we outline a simple implementation of our approach in Algorithm 3. Lines 4–5 correspond to Case L1. If $i \leq \overleftarrow{c}$, then lines 7–9 compute the LCE naively. Otherwise, they correspond to Case L2.

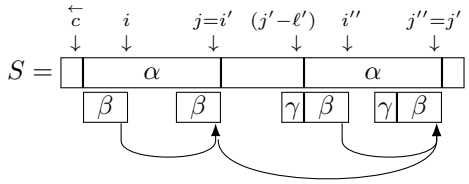
▶ **Lemma 13.** *Given a string of length n and its NSS array* nss*, we can compute* $\text{LCE}_\ell(i, \mathsf{nss}[i])$ *for all indices $i \in [1, n]$ with* $\mathsf{nss}[i] \neq n + 1$ *in $\mathcal{O}(n)$ time and space.*

▶ **Corollary 14.** *Given a string of length n over a general ordered alphabet, we can find all runs in the string in $\mathcal{O}(n)$ time and space.*

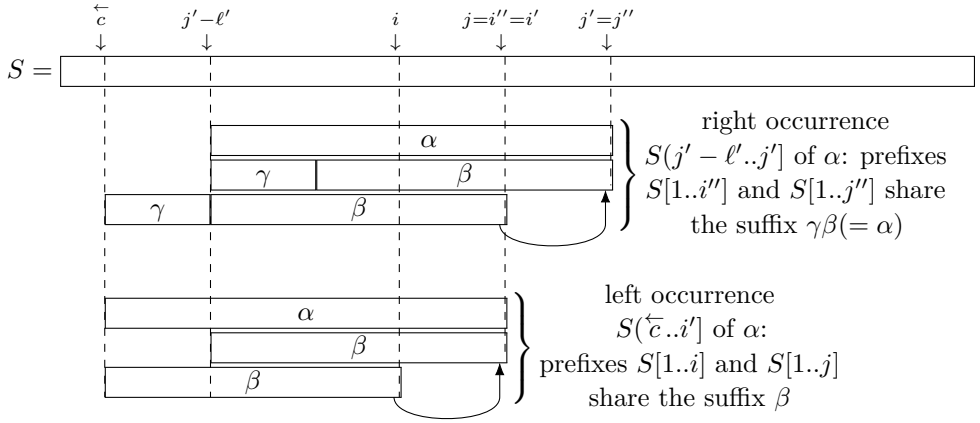
Proof. Computing the increasing runs takes $\mathcal{O}(n)$ time and space due to Lemmas 9, 12, and 13. For decreasing runs, we only have to reverse the order of the alphabet and rerun the algorithm. ◀



(a) Case L1 with $i'' \neq i'$ and $j'' = j'$.



(b) Case L2 with $i'' \neq i'$ and $j'' = j'$.



(c) Case L2 with $i'' = i'$ and $j'' = j'$.

Figure 4 Additional drawings for Cases L1 and L2.

Algorithm 3 Compute all L-LCEs.

Input: String S of length n and its NSS array nss.
Output: L-LCE value $\text{LCE}_\ell(i, \mathsf{nss}[i])$ for each index $i \in [1, n]$ with $\mathsf{nss}[i] \neq n + 1$.

$i' \leftarrow 0$; $\overleftarrow{c} \leftarrow n$; $d \leftarrow 0$
for $i \in [1, n]$ **with** $\mathsf{nss}[i] \neq n + 1$ in decreasing order **do**
  $j \leftarrow \mathsf{nss}[i]$
  if $i \in (\overleftarrow{c}, i') \wedge i - \text{LCE}_\ell(i + d, j + d) > \overleftarrow{c}$ **then**
    $\text{LCE}_\ell(i, j) \leftarrow \text{LCE}_\ell(i + d, j + d)$ ▷ retrieve LCE in constant time
  else
    $k \leftarrow i - \min(\overleftarrow{c}, i)$
    $\text{LCE}_\ell(i, j) \leftarrow k + \text{NAIVE-SCAN-LCE}_\ell(i - k, j - k)$
    $i' \leftarrow i$; $\overleftarrow{c} \leftarrow i - \text{LCE}_\ell(i, j)$; $d \leftarrow j - i$

Table 1 Throughput achieved by our runs algorithm using an AMD EPYC 7452 processor. We repeated each experiment five times and use the median throughput as the final result (the minimum and maximum throughputs were almost identical to the median). All numbers are truncated to one decimal place.

Text n in MiB	t_{49} [23] 1077 MiB	sources 201 MiB	pitches 53 MiB	proteins 1024 MiB	dna 385 MiB	english 1024 MiB	xml 282 MiB	ecoli 107 MiB	cere 439 MiB	fib41 255 MiB	rs.13 206 MiB	tm29 256 MiB
runs/100n	94.4	4.7	11.7	7.0	25.3	2.4	3.4	24.4	23.6	76.3	92.7	83.3
MiB/s	15.0	11.4	11.0	10.9	8.8	10.5	12.8	9.0	9.2	15.4	15.1	15.6

5 Practical Implementation

We implemented our algorithm for the runs computation in C++17 and evaluated it by computing all runs on texts from the natural, real repetitive, and artificial repetitive text collections of the Pizza-Chili corpus[2]. Additionally, we used the binary run-rich strings proposed by Matsubara et al. [23] as input. Table 1 shows the throughput that we achieve, i.e. the number of input bytes (or equivalently input symbols) that we process per second. On the string `tm29` we achieve the highest throughput of 15.6 MiB/s. The lowest throughput of 8.8 MiB/s occurs on the text `dna`. Generally, we perform better for run-rich strings.

Lastly, it is noteworthy that our new method of LCE computation leads to a remarkably simple implementation of the runs algorithm. In fact, the entire implementation *including the computation of the NSS array* needs only 250 lines of code. We achieve this by interleaving the computation of the R-LCEs with the computation of the NSS array, which also improves the practical performance. For technical details we refer to the source code, which is publicly available on GitHub[3].

6 Conclusion and Open Questions

We have shown the first linear time algorithm for computing all runs over a general ordered alphabet. The algorithm is also very fast in practice and remarkably easy to implement. It is an open question whether our techniques could be used for the computation of runs on tries, where the best known algorithms require super-linear time even for linearly-sortable alphabets (see e.g. [26]).

References

1 Arne Andersson, Torben Hagerup, Stefan Nilsson, and Rajeev Raman. Sorting in linear time? *Journal of Computer and System Sciences*, 57(1):74–93, 1998. doi:10.1006/jcss.1998.1580.

2 Hideo Bannai, Tomohiro I, Shunsuke Inenaga, Yuto Nakashima, Masayuki Takeda, and Kazuya Tsuruta. The "runs" theorem. *SIAM Journal on Computing*, 46(5):1501–1514, 2017. doi:10.1137/15M1011032.

3 Philip Bille, Jonas Ellert, Johannes Fischer, Inge Li Gørtz, Florian Kurpicz, J. Ian Munro, and Eva Rotenberg. Space efficient construction of Lyndon arrays in linear time. In *Proceedings of the 47th International Colloquium on Automata, Languages, and Programming (ICALP 2020)*, pages 14:1–14:18, Saarbrücken, Germany, July 2020. doi:10.4230/LIPIcs.ICALP.2020.14.

[2] http://pizzachili.dcc.uchile.cl/texts.html, http://pizzachili.dcc.uchile.cl/repcorpus.html

[3] https://github.com/jonas-ellert/linear-time-runs/

4 Dany Breslauer. *Efficient String Algorithmics*. PhD thesis, Columbia University, New York, USA, 1992. URL: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.34.9146.

5 Helen Budworth and Cynthia T. McMurray. *A Brief History of Triplet Repeat Diseases*, volume 1010 of *Methods in Molecular Biology*, pages 3–17. Springer, 2013. doi:10.1007/978-1-62703-411-1_1.

6 M. Crochemore, C.S. Iliopoulos, M. Kubica, J. Radoszewski, W. Rytter, and T. Waleń. Extracting powers and periods in a word from its runs structure. *Theoretical Computer Science*, 521:29–41, 2014. doi:10.1016/j.tcs.2013.11.018.

7 Maxime Crochemore and Lucian Ilie. Maximal repetitions in strings. *Journal of Computer and System Sciences*, 74(5):796–807, 2008. doi:10.1016/j.jcss.2007.09.003.

8 Maxime Crochemore, Lucian Ilie, and Wojciech Rytter. Repetitions in strings: Algorithms and combinatorics. *Theoretical Computer Science*, 410(50):5227–5235, 2009. doi:10.1016/j.tcs.2009.08.024.

9 Maxime Crochemore, Lucian Ilie, and Liviu Tinta. The "runs" conjecture. *Theoretical Computer Science*, 412(27):2931–2941, 2011. doi:10.1016/j.tcs.2010.06.019.

10 Maxime Crochemore, Costas S. Iliopoulos, Tomasz Kociumaka, Ritu Kundu, Solon P. Pissis, Jakub Radoszewski, Wojciech Rytter, and Tomasz Waleń. Near-optimal computation of runs over general alphabet via non-crossing lce queries. In *Proceedings of the 23rd International Symposium on String Processing and Information Retrieval (SPIRE 2016)*, pages 22–34, Beppu, Japan, October 2016. doi:10.1007/978-3-319-46049-9_3.

11 Johannes Fischer and Volker Heun. Theoretical and practical improvements on the RMQ-problem, with applications to LCA and LCE. In *Proceedings of the 17th Annual Symposium on Combinatorial Pattern Matching (CPM 2006)*, pages 36–48, Barcelona, Spain, 2006. doi:10.1007/11780441_5.

12 Johannes Fischer, Stepan Holub, Tomohiro I, and Moshe Lewenstein. Beyond the runs theorem. In Costas S. Iliopoulos, Simon J. Puglisi, and Emine Yilmaz, editors, *String Processing and Information Retrieval - 22nd International Symposium, SPIRE 2015, London, UK, September 1-4, 2015, Proceedings*, volume 9309 of *Lecture Notes in Computer Science*, pages 277–286. Springer, 2015. doi:10.1007/978-3-319-23826-5_27.

13 Frantisek Franek, A. S. M. Sohidull Islam, Mohammad Sohel Rahman, and William F. Smyth. Algorithms to compute the Lyndon array. In *Proceedings of the Prague Stringology Conference 2016 (PSC 2016)*, pages 172–184, Prague, Czech Republic, 2016. URL: http://www.stringology.org/event/2016/p15.html.

14 Pawel Gawrychowski, Tomasz Kociumaka, Wojciech Rytter, and Tomasz Walen. Faster longest common extension queries in strings over general alphabets. In *Proceedings of the 27th Annual Symposium on Combinatorial Pattern Matching (CPM 2016)*, pages 5:1–5:13, Tel Aviv, Israel, 2016. doi:10.4230/LIPIcs.CPM.2016.5.

15 Torben Hagerup. Sorting and searching on the word RAM. In *Proceedings of the 15th Annual Symposium on Theoretical Aspects of Computer Science (STACS 98)*, pages 366–398, Paris, France, February 1998. doi:10.1007/BFb0028575.

16 Yijie Han and M. Thorup. Integer sorting in $\mathcal{O}(n\sqrt{\log\log n})$ expected time and linear space. In *Proceedings of the 43rd Annual IEEE Symposium on Foundations of Computer Science (FOCS 2002)*, pages 135–144, Vancouver, Canada, 2002. doi:10.1109/SFCS.2002.1181890.

17 Stepan Holub. Prefix frequency of lost positions. *Theor. Comput. Sci.*, 684:43–52, 2017. doi:10.1016/j.tcs.2017.01.026.

18 R. Kolpakov and G. Kucherov. Finding maximal repetitions in a word in linear time. In *Proceedings of the 40th Annual Symposium on Foundations of Computer Science (FOCS 1999)*, pages 596–604, New York, NY, USA, 1999. doi:10.1109/SFFCS.1999.814634.

19 Dmitry Kosolobov. Lempel-Ziv factorization may be harder than computing all runs. In *Proceedings of the 32nd International Symposium on Theoretical Aspects of Computer Science (STACS 2015)*, pages 582–593, Munich, Germany, 2015. doi:10.4230/LIPIcs.STACS.2015.582.

20 Dmitry Kosolobov. Computing runs on a general alphabet. *Information Processing Letters*, 116(3):241–244, 2016. doi:10.1016/j.ipl.2015.11.016.

21 R. C. Lyndon and M. P. Schützenberger. The equation $a^m = b^n c^p$ in a free group. *Michigan Mathematical Journal*, 9(4):289–298, 1962. doi:10.1307/mmj/1028998766.

22 Michael G Main and Richard J Lorentz. An o(n log n) algorithm for finding all repetitions in a string. *Journal of Algorithms*, 5(3):422–432, 1984. doi:10.1016/0196-6774(84)90021-X.

23 Wataru Matsubara, Kazuhiko Kusano, Hideo Bannai, and Ayumi Shinohara. A series of run-rich strings. In Adrian Horia Dediu, Armand Mihai Ionescu, and Carlos Martín-Vide, editors, *Proceedings of the 3rd International Conference on Language and Automata Theory and Applications (LATA 2009)*, pages 578–587, Tarragona, Spain, 2009. doi:10.1007/978-3-642-00982-2_49.

24 Simon J. Puglisi, Jamie Simpson, and W.F. Smyth. How many runs can a string contain? *Theoretical Computer Science*, 401(1):165–171, 2008. doi:10.1016/j.tcs.2008.04.020.

25 Wojciech Rytter. The number of runs in a string: Improved analysis of the linear upper bound. In *Proceedings of the 24th Annual Symposium on Theoretical Aspects of Computer Science (STACS 2006)*, pages 184–195, Marseille, France, 2006. doi:10.1007/11672142_14.

26 Ryo Sugahara, Yuto Nakashima, Shunsuke Inenaga, Hideo Bannai, and Masayuki Takeda. Computing runs on a trie. In *Proceedings of the 30th Annual Symposium on Combinatorial Pattern Matching (CPM 2019)*, volume 128, pages 23:1–23:11, Pisa, Italy, June 2019. doi:10.4230/LIPIcs.CPM.2019.23.

Decremental APSP in Unweighted Digraphs Versus an Adaptive Adversary

Jacob Evald
BARC, University of Copenhagen, Denmark

Viktor Fredslund-Hansen
BARC, University of Copenhagen, Denmark

Maximilian Probst Gutenberg
ETH Zurich, Switzerland

Christian Wulff-Nilsen
BARC, University of Copenhagen, Denmark

Abstract

Given an unweighted digraph $G = (V, E)$, undergoing a sequence of edge deletions, with $m = |E|, n = |V|$, we consider the problem of maintaining all-pairs shortest paths (APSP).

Whilst this problem has been studied in a long line of research [ACM'81, FOCS'99, FOCS'01, STOC'02, STOC'03, SWAT'04, STOC'13] and the problem of $(1+\epsilon)$-approximate, weighted APSP was solved to near-optimal update time $\tilde{O}(mn)$ by Bernstein [STOC'13], the problem has mainly been studied in the context of an *oblivious* adversary which fixes the update sequence before the algorithm is started. In this paper, we make significant progress on the problem for an adaptive adversary which can perform updates based on answers to previous queries:

- We first present a *deterministic* data structure that maintains the *exact* distances with total update time $\tilde{O}(n^3)$[1].
- We also present a *deterministic* data structure that maintains $(1+\epsilon)$-approximate distance estimates with total update time $\tilde{O}(\sqrt{m}n^2/\epsilon)$ which for sparse graphs is $\tilde{O}(n^{2+1/2}/\epsilon)$.
- Finally, we present a randomized $(1+\epsilon)$-approximate data structure which works against an adaptive adversary; its total update time is $\tilde{O}(m^{2/3}n^{5/3} + n^{8/3}/(m^{1/3}\epsilon^2))$ which for sparse graphs is $\tilde{O}(n^{2+1/3}/\epsilon^2)$.

Our exact data structure matches the total update time of the best *randomized* data structure by Baswana et al. [STOC'02] and maintains the distance matrix in near-optimal time. Our approximate data structures improve upon the best data structures against an adaptive adversary which have $\tilde{O}(mn^2)$ total update time [JACM'81, STOC'03].

2012 ACM Subject Classification Theory of computation → Data structures design and analysis; Theory of computation → Shortest paths; Theory of computation → Dynamic graph algorithms

Keywords and phrases Dynamic Graph Algorithm, Data Structure, Shortest Paths

Digital Object Identifier 10.4230/LIPIcs.ICALP.2021.64

Category Track A: Algorithms, Complexity and Games

Related Version *Extended Version*: `arXiv:2010.00937` [5]

Funding *Jacob Evald*: The author is supported by the Starting Grant 7027-00050B from the Independent Research Fund Denmark under the Sapere Aude research career programme.
Viktor Fredslund-Hansen: The author is supported by the Starting Grant 7027-00050B from the Independent Research Fund Denmark under the Sapere Aude research career programme.
Maximilian Probst Gutenberg: Supported by Rasmus Kyng's Start-up Grant at ETH. Work done while at BARC, supported by Thorup's Investigator Grant No. 16582.

[1] We use $\tilde{O}$-notation to hide logarithmic factors.

Christian Wulff-Nilsen: The author is supported by the Starting Grant 7027-00050B from the Independent Research Fund Denmark under the Sapere Aude research career programme.

Acknowledgements We thank anonymous reviewers for their comments and remarks that helped improve the presentation of the paper.

1 Introduction

Shortest paths is a classical algorithmic problem dating back to the 1950s. The two main variants are the all-pairs shortest paths (APSP) problem and the single-source shortest paths (SSSP) problem, both of which have been extensively studied in various models, including the partially and fully-dynamic setting.

A dynamic graph algorithm is an algorithm that maintains information about a graph that is subject to updates such as insertions and deletions of edges or vertices. Such a graph can model real-world networks that change over time, such as road networks where traffic changes and roads are blocked from time to time. We say that a dynamic graph problem is *decremental* if it only allows deletions, *incremental* if it only allows insertions and *fully-dynamic* if it allows both. Incremental and decremental graphs are referred to as being *partially-dynamic*. A dynamic graph algorithm aims to efficiently process a sequence of online updates interspersed with queries about some property of the underlying dynamic graph.

1.1 Problem Definition

In this paper, we consider the *decremental all-pairs shortest-paths* problem where the goal is to efficiently maintain shortest path distances between all pairs of vertices in a decremental directed graph $G = (V, E)$. We shall restrict our attention to the case where G is unweighted. Letting m denote the initial number of edges and $n = |V|$, we want a data-structure which for any $u, v \in V$ supports the following operations:

- $\textsc{Dist}(u, v)$: reports the distance $d_G(u, v)$ from u to v in the current version of G,
- $\textsc{Delete}(u, v)$: deletes an edge (u, v) from E.

We furthermore consider the problem also in its relaxed version where we only aim to maintain *approximate* distance estimates which can then be queried. We denote by $\tilde{d}_G(u, v)$ a distance estimate for the distance from u to v and we say that an APSP algorithm has an *approximation ratio* (or *stretch*) of $t > 1$ if for any $u, v \in V$, we have that $d_G(u, v) \leq \tilde{d}_G(u, v) \leq t \cdot d_G(u, v)$. This paper will be concerned with both the exact and the $(1 + \epsilon)$-approximate version of the problem.

Another focus of this article is the *adversarial model*; the adversarial model defines the model under which the sequence of updates and queries are assumed to be made by an *adversary*. We say that a performance guarantee of an algorithm works against an *oblivious* adversary if the adversary must define the sequence of updates before the algorithm starts for the guarantee to hold. Thus the sequence of updates is independent of any random bits used by the algorithm. This is opposed to algorithms that work against an *adaptive* adversary, where the adversary is allowed to create the update sequence "on the go", e.g. based on answers to previous queries made to the data structure. Depending on the data structure, these choices may not be independent of the random choices made, which may result in the data structure performing poorly. One key advantage of a data structure that works against an adaptive adversary is that it can be used inside an algorithm as a black box, regardless of whether that algorithm adapts its updates to answers to queries. We point out that *deterministic* data structures always work against an adaptive adversary.

The performance of a *partially-dynamic* algorithm is usually measured in terms of the *total update time*. That is, the accumulated time it takes to process all updates (edge deletions). The *query time*, on the other hand, is the time to answer a single distance query. A natural goal is to minimize the total update time while keeping the stretch and query time small. Since all the structures presented in this paper explicitly maintain a distance matrix, the query time is constant.

1.2 Prior Work

The naive approach to dynamic APSP is to recompute the shortest path distances after each update using the best static algorithm. The query time is then constant and the time for a single update is $\tilde{O}(mn)$ for APSP and $\tilde{O}(m)$ for SSSP. At the other end of the spectrum one could achieve optimal update time by simply updating the input graph and only running an SSSP algorithm whenever a query is processed. Running a static algorithm each time, however, fails to reuse any information between updates whatsoever and gives a high query time, motivating more efficient dynamic approaches that do this.

In 1981, Even and Shiloach [6] gave a deterministic data-structure for maintaining a shortest path tree to given depth d in an undirected, unweighted decremental graph in total time $O(md)$. Henzinger and King [7] and King [10] later adapted this to directed graphs with integer weights. Running their structure for each vertex solves the decremental all-pairs shortest paths problem in $O(mn^2W)$ time, where edge weights are integers in $[1, W]$.

Henzinger and King were the first to improve upon this bound, giving an algorithm with total update time $\tilde{O}(mn^{2.5}\sqrt{W})$ [10] which is an improvement for $W = \omega(n)$. Demetrescu and Italiano [4] improved this data structure slightly and showed that the restriction to integral edge weights can be removed. Finally, the same authors [3] presented a data structure with total update time $\tilde{O}(mn^2)$ which is the state of the art for any data structure against an adaptive adversary up to today. In fact, their algorithm can be extended to a fully-dynamic algorithm with $\tilde{O}(n^2)$ amortized update time and which can handle vertex updates[2]. We also point out that this data structure was later simplified and generalized by Thorup [11].

Around the same time Baswana, Hariharan, and Sen [1] gave an *oblivious* Monte-Carlo construction with total update time $\tilde{O}(n^3)$ for *unweighted* graphs. Further, they showed that their data structure could be adapted to give an $(1+\epsilon)$-approximate APSP algorithm for *weighted* graphs with total update time of $\tilde{O}(\sqrt{m}n^2/\varepsilon)$. In the exact setting, the oblivious adversary assumption is only required when paths are to be reported rather than just shortest path distances which are unique. Finally, Bernstein presented a $(1+\epsilon)$-approximate algorithm with total running time $\tilde{O}(mn\log(W)/\epsilon)$ by using a clever approach of shortcutting paths [2]. Whilst his algorithm achieves near-optimal running time, again, the algorithm has to assume an oblivious adversary.

More recently, Karczmarz and Łącki [9] gave a deterministic $(1+\epsilon)$-approximate APSP algorithm for decremental graphs that runs in total time $\tilde{O}(n^3\log(W)/\epsilon)$. They also presented the first non-trivial algorithm for incremental graphs [8] achieving total update time $\tilde{O}(mn^{4/3}\log(W)/\epsilon)$.

We refer the reader to the full version of the paper [5] for a more comprehensive treatment of related work which also includes algorithms for *undirected* graphs and algorithms with larger stretch.

[2] In this case, vertex updates refers to insertions or deletions of vertices with up to $n-1$ incident edges.

1.3 Our Contributions

In this paper, we present three new data structures for the all-pairs shortest paths problem. Our first theorem gives a *deterministic* data structure for the exact variant of the problem with near-optimal $\tilde{O}(n^3)$ total update time. It also matches the best *randomized* algorithm by Baswana et al. [1] and constitutes a significant improvement over the previous best deterministic bound of $\tilde{O}(mn^2)$ which is obtained by running an ES-tree [6] from every source or by the data structure Italiano et al. [3] (that also works in weighted graphs) and improves over all but the sparsest graph densities.

Our exact data structure is near-optimal as there is an $\Omega(n^3)$ lower bound on the total update time of any decremental data structure that explicitly maintains the distance matrix. The lower bound follows by considering an initial undirected, unweighted graph consisting of a simple path $v_0, v_1, v_2, \ldots, v_{n-1}$ plus additional edges (v_i, v_{i+2}) for each even $i \leq n-3$. Deleting these additional edges in any order creates $\Omega(n^3)$ distance matrix changes in total.

▶ **Theorem 1.** *Let G be an unweighted directed graph with n vertices and initially m edges. Then there exists a deterministic data structure which maintains all-pairs shortest path distances in G undergoing an online sequence of edge deletions using a total time of $O(n^3 \log^3 n)$. The $n \times n$ distance matrix is explicitly maintained so that at any point, a shortest path distance query can be answered in constant time. The data structure can report a shortest path between any query pair in time proportional to the length of the path.*

Our second result is concerned with maintaining $(1+\epsilon)$-approximate all-pairs shortest path distances. This constitutes the first deterministic data structure that solves the problem in subcubic time with small approximation error (except for graphs that are not extremely dense).

▶ **Theorem 2.** *Let G be an unweighted directed graph with n vertices and initially m edges. Then given $\epsilon > 0$, there exists a deterministic data structure that maintains all-pairs $(1+\epsilon)$-approximate shortest path distances in G undergoing an online sequence of edge deletions using a total time of $O(\sqrt{m}n^2 \log^2(n)/\epsilon)$. At any point, a $(1+\epsilon)$-approximate shortest path distance query can be answered in constant time and a $(1+\epsilon)$-approximate shortest path between the query pair can be reported in time proportional to the length of the path.*

Our third result gives a data structure achieving a better time bound. While we use randomization to achieve the improved time bound, our algorithm again works against an adaptive adversary.

▶ **Theorem 3.** *Let G be an unweighted directed graph with n vertices and initially m edges. Then given any $\epsilon > 0$, there exists a Las Vegas data structure that maintains all-pairs $(1+\epsilon)$-approximate shortest path distances in G under an online sequence of edge deletions using a total expected time of $\tilde{O}(m^{2/3}n^{5/3}/\epsilon + n^{8/3}/(m^{1/3}\epsilon^2))$. This bound holds w.h.p. and the data structure works against an adaptive adversary. At any point, a $(1+\epsilon)$-approximate shortest path distance query can be answered in constant time.*

We summarize our results as well as previous state-of-the-art results in Table 1.

1.4 Overview

Our overall approach for the deterministic data structures is similar to that of Baswana et al. [1] but with a key difference that allows us to avoid using a randomized hitting set and instead rely on deterministic separators. The idea of the construction by Baswana et

Table 1 Our results and previous state-of-the-art results for decremental APSP.

Time	Approximation	Adversary/ Deterministic	Reference
$O(mn^2)$	exact	deterministic	[6, 3]
$\tilde{O}(n^3)$	exact	deterministic	**New Result**
$\tilde{O}(n^3)$	exact	adaptive	[1]
$\tilde{O}(\sqrt{m}n^2/\epsilon)$	$(1+\epsilon)$	deterministic	**New Result**
$\tilde{O}(m^{2/3}n^{5/3}/\epsilon + n^{8/3}/(m^{1/3}\epsilon^2))$	$(1+\epsilon)$	adaptive	**New Result**
$\tilde{O}(\sqrt{m}n^2/\epsilon)$	$(1+\epsilon)$	oblivious	[1]
$\tilde{O}(nm)$	$(1+\epsilon)$	oblivious	[2]

al. relies on a well-known result which says that if we sample a subset H_i^ρ of the vertices of size $\tilde{O}(n/\rho^i)$ (where ρ is some constant strictly larger than 1), each with uniform probability, then, w.h.p. we "hit" each shortest-path of length $[\rho^i, \rho^{i+1})$ between any pair of vertices in any version of the graph G.

Phrased differently, given vertices $u, v \in V$, we have that if a shortest path from u to v is of length $\ell \in [\rho^i, \rho^{i+1})$, then there is some vertex $w \in H_i^\rho$, such that the concatenation of a shortest path from u to w and a shortest path from w to v is of length ℓ. For each such w, we say w is a *witness* for the tuple (u, v) for distance ℓ.

Now for each $u, v \in V$, if the initial distance from u, v was $\ell \in [\rho^i, \rho^{i+1})$, we can check H_i^ρ to find a witness w. If the length of the path from u to w to v is increased, we can continue our scanning of H_i^ρ to see whether another witness exists. If there is no witness $w \in H_i^\rho$ left at some stage, we know that there is no path of length ℓ left in G w.h.p. and increase our guess by setting $\ell \mapsto \ell + 1$.

Sampling initially a hitting set H_i^ρ for every $i \in [0, \log_\rho n]$, we can find the "right" hitting set for each distance ℓ. Observe now that for each tuple $(u, v) \in V^2$, we have to scan a hitting set of size $\tilde{O}(n/\rho^i)$ for $\rho^{i+1} - \rho^i \sim \rho^{i+1}$ levels before the hitting set index i is increased which only occurs $O(\log n)$ times, thus we only spend time $\tilde{O}(n)$ for each vertex tuple (u, v). Thus, the total running time of the searches for witnesses can be bound by $\tilde{O}(n^3)$.

The Deterministic Exact Data Structure. Our construction is similar in the sense that we maintain witnesses for each distance scale $[\rho^i, \rho^{i+1})$ for every $i \in [0, \log_\rho n]$ such that each distance ℓ is in one such distance scale. The key difference is that instead of using a randomized *global* hitting set H_i^ρ for a distance scale $[\rho^i, \rho^{i+1})$, our construction relies on deterministically maintaining a small local vertex separator $S_i(u)$ for every vertex $u \in V$ of size $\tilde{O}(n/\rho^i)$ separating all shortest paths starting in u with a distance in $[\rho^i, \rho^{i+1})$.

More precisely, for each distance scale $[\rho^i, \rho^{i+1})$ and vertex $u \in V$, we maintain a separator $S_i(u)$ that satisfies the invariant that every shortest path from u to a vertex v at distance at least ρ^i is intersected by a vertex in $S_i(u)$. If this invariant is violated after an adversarial update, then we find such a vertex v and need to add additional vertices to $S_i(u)$ during the time step. The challenge is to take these additional separator vertices such that the total size of $S_i(u)$ is not increased beyond $\tilde{O}(n/\rho^i)$. The separator procedure makes use of sparse layers of BFS trees and here is where we rely on the assumption that the graph is unweighted. We defer the details of the separator procedure to a later section and continue our discussion of the APSP data structure.

Since we need to detect whether vertices have distance less than ρ^i from u or not in G, we further have to use a bottom-up approach to compute distances after an edge deletion, i.e. we start with the smallest possible distance range and update all distances in this range and then update larger distances using the information already computed. This issue did not arise in Baswana et al. [1] but can be handled by a careful approach. The distances computed for one distance scale include all distances to and from witnesses for the next larger distance scale.

It is now easy to see that the scanning for witnesses can be implemented in the same time as in the analysis sketched above by scanning the list of local separator vertices which serve as witnesses instead of the hitting set. Further, we can maintain local vertex separators using careful arguments in total time $\tilde{O}(mn)$ giving our result in Theorem 1.

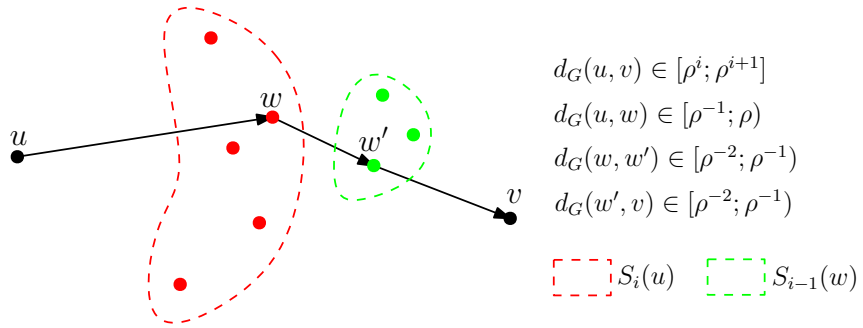


Figure 1 Illustration of separators and path "hierarchy". Here $u \rightsquigarrow v$ goes through a witness w, and $w \rightsquigarrow v$ goes through w'. If the length of the path $w' \rightsquigarrow v$ is increased by Δ, the distance estimates of all 2-hop-paths that use $w' \rightsquigarrow v$ as a sub-path are increased by that amount. In this case, the estimate for $w \rightsquigarrow w' \rightsquigarrow v$ is increased and is propagated to the next level where subsequently the estimate for $u \rightsquigarrow w \rightsquigarrow v$ is possibly increased.

The Deterministic Approximate Data Structure. In order to improve the running time for sparse graphs, we can further focus on only considering distances that are roughly at a $(1+\epsilon)$-multiplicative factor from each other. More concretely, instead of increasing the expected distance from ℓ to $\ell+1$ when we cannot find a witness for some path from u to v for distance ℓ, we can increase the next expected distance level ℓ' to $\sim (1+\epsilon)\ell$ and consider every vertex w a witness if there is a path $u \rightsquigarrow w \rightsquigarrow v$ of length at most ℓ'. Thus, we handle fewer distances and can thereby reduce the time to maintain distances that are at least d in total time $\tilde{O}(n^3/d + mn)$. Again, a careful approach is necessary to ensure that approximations do not add up over distance scales.

This is no faster than the data structure for exact distances when d is small so in order to get Theorem 2, we use the $O(mnd)$ data structure of Even and Shiloach [6] to maintain distances up to d. Picking d such that $mnd = n^3/d$ gives the result of Theorem 2 (the term $\tilde{O}(mn)$ vanishes since it is subsumed by the two other terms, also we assumed $\epsilon > 0$ to be a constant to simplify the presentation).

Maintaining Separators. We now sketch how to deterministically maintain a "small" local separator for a vertex $s \in V$ with some useful invariants.

Let S be the local separator for s. The first invariant that will be useful is that any vertex $t \in V$ that is reachable from s in $G \setminus S$, is "close" to s or roughly within distance d. As edges are deleted from G, the distances from s to such vertices t may increase. If a vertex

t moves too far away from s, the invariant is re-established by growing BFS trees in parallel, one layer at a time, from s in $G \setminus S$ and from t in the graph obtained from $G \setminus S$ by reversing the orientations of all edges. The search halts when a layer (corresponding to the leaves of the BFS tree at the current iteration) that is "thin" is found, and its vertices are added to S; vertices that are on the opposite side of the separator than s are cut off as they must all be too far away from s. Here, "thin" refers to a BFS layer such that the number of vertices added to the separator is only a factor $\tilde{O}(1/d)$ times the number of vertices cut off. It is well known that such a layer exists (cfr. Lemma 5 for the details). Summing up, it follows that $|S| = \tilde{O}(n/d)$ at all times. By marking vertices as they are searched (according to the side of the BFS layer on which they are found), the vertices that are "cut off" from s by the augmented separator will never be searched again, and the cost of searching the edges of either side of the search can be charged to sum of the degree of these vertices, for a total update time of $O(m)$.

For our randomized data structure, we need an additional property that essentially allows us to take a snapshot of the current separator and use it in later updates rather than having to repeatedly update the separator. This will be key to getting an improved randomized time bound. Details can be found in Lemma 6 which states our separator result.

The Randomized Approximate Data Structure. The randomized approximate data structure of Theorem 3 follows the same overall approach but is technically more involved. Instead of keeping track of all 2-hop paths $u \rightsquigarrow s \rightsquigarrow v$[3] for every $s \in S_i(u)$, the randomized data structure samples a subset of these by picking each vertex of $S_i(u)$ independently with some probability p. It only keeps track of approximate shortest path distances going through this subset rather than the full set $S_i(u)$. This will speed up the above since the subset of the separator we need to scan is smaller by a factor p. However, this approach fails once no short 2-hop path intersects the sampled subset. At this point, w.h.p. there should only be short 2-hop paths through $O(\log n/p)$ vertices of $S_i(u)$ so also in this case, the subset can be kept small. However, scanning linearly through $S_i(u)$ to find this small subset will take $\tilde{O}(n/d)$ time and happen over all pairs (u, v).

Our solution is roughly the following. Suppose no sampled vertex certifies an approximate short path from u to v. Then v scans linearly through $S_i(u)$ to find the small size $O(\log n/p)$ subset $S'_i(u)$. Consider the set W of vertices w such that $d_G(w, v)$ is small compared to d, i.e., $d_G(w, v) \leq \epsilon d$ for some small constant $\epsilon > 0$. Then we show that the small subset $S'_i(u)$ found for v can also be used for each vertex $w \in W$. The intuition is that for any vertex $s \in S_i(u) \setminus S'_i(u)$, the approximate shortest path distance from u to w through s must be large since otherwise we get a short path $u \rightsquigarrow s \rightsquigarrow w \rightsquigarrow v$ from u to v through s, contradicting that $s \notin S'_i(u)$.

It follows that if $|W|$ is large, the $\tilde{O}(n/d)$ cost of scanning $S_i(u)$ can be distributed among a large number of vertices of W. Dealing with the case where $|W|$ is small is more technical so we omit it here.

The way we deal with an adaptive adversary is roughly as follows. Consider a deterministic data structure that behaves like the randomized data structure above, except that it maintains 2-hop paths $u \rightsquigarrow s \rightsquigarrow v$ for all $S_i(u)$ rather than only through a sampled subset. The slack from the approximation allows us to round up all "short" approximate distances to the same value. Hence, as long as the randomized data structure has short 2-hop paths, it maintains exactly the same approximate distances as the deterministic structure and hence the approximate distances output to the adversary is independent of the random bits used.

[3] Note that such a path may have more than one intermediate vertex, but it is useful to think of it as a path of two weighted edges/hops (u, s) and (s, v) since this is what is maintained by the data structure.

2 Definitions and Notation

In the following, let $G = (V, E)$ be a directed unweighted graph. The graph $G_{\mathbf{rev}}$ is obtained from G by reversing the orientation of each edge. For any two vertices $u, v \in V$, we denote by $u \rightsquigarrow v$ a shortest path from u to v in G and let $d_G(u, v)$ denote the length of such a path. We extend this notation to sets so that, e.g., $d_G(u, V') = \min\{d_G(u, v) | v \in V'\}$ for $V' \subseteq V$.

We define a BFS-layer to mean the set of nodes at some fixed distance from some v in G. An *in-tree* in G is a BFS tree in $G_{\mathbf{rev}}$.

We will need notation to refer to dynamically changing data at specific points in time. Consider a sequence of updates to some object X where each update takes place at a time step $t \in \mathbb{N}$. We denote by $X^{(t)}$ the object just after update t. Here, X could be a graph, a shortest path distance, etc.

For handling small distances, we rely on the data-structure of Even and Shiloach [6], the properties of which we will state in the following lemma:

▶ **Lemma 4** ([6]). *Given a directed unweighted graph G undergoing a sequence of edge deletions, a source vertex $s \in V$, and $d > 0$, a shortest path tree in G rooted at s can be maintained up to distance d in total time $O(md)$. The structure requires $O(m)$ space and can be constructed in time $O(m + n)$.*

3 Maintaining Separators

Lemma 6 below provides a key tool used in all of our data structures. It gives an efficient data structure that maintains a growing separator set S of small size in a decremental graph G. To prove it, we need the following well-known result.

▶ **Lemma 5.** *Given a directed unweighted n-vertex graph $G = (V, E)$, given $d_1, d_2 \in \mathbb{N}_0$ with $d_2 - d_1 + 1 \geq \lg n$, and given vertices $u, v \in V$ with $d_G(u, v) \geq d_2$, a BFS tree in G with root u contains a layer $L \subseteq V$ with $d_1 \leq d_G(u, L) \leq d_2$ and $|L| \leq |L_-| \lg n/(d_2 - d_1 + 1)$ where $L_- = \{w \in V | d_G(u, w) < d_G(u, L)\}$ is the union of layers closer to u than L.*

Proof. Denote by L_i the ith layer of the BFS tree from u. For each i, let $L_{<i} = \cup_{j<i} L_j$. Let $q = (d_2 - d_1 + 1)/\lg n$. Assume for contradiction that L does not exist. Then for $i = d_1, \ldots, d_2$, $|L_i| > |L_{<i}|/q$ so $|L_{<i+1}| = |L_i| + |L_{<i}| > (1 + 1/q)|L_{<i}|$. Since $q \geq 1$, we have $(1 + 1/q)^q \geq 2$ so

$$|L_{<d_2+1}| > (1 + 1/q)^{d_2-d_1+1} |L_{<d_1}| \geq 2^{(d_2-d_1+1)/q} = n,$$

contradicting that there are only n vertices in G. ◀

We now state and prove Lemma 6. It gives an efficient data structure that maintains a growing separator set S of small size in a decremental graph G with the following guarantees. Let s be a fixed vertex and let d be some given threshold distance. Then at every time step, vertices reachable from s in $G \setminus S$ are of distance slightly less than d from s in G. Conversely, for vertices v not reachable from s in $G \setminus S$, we have $d_G(s, v) = \Omega(d)$; furthermore, if $d_G(s, v)$ is larger than d by some small constant factor then any shortest path $s \rightsquigarrow v$ in G can be decomposed into $s \rightsquigarrow w \rightsquigarrow v$ such that $w \in S$, $d_G(s, w) \leq d$, and $d_G(w, v) \leq d$. In fact, the lemma states that w can be chosen in S^{t_0} where t_0 is the first time step in which $d_G(s, v)$ became (slightly) larger than d; note that this is a stronger statement since S is growing over time.

▶ **Lemma 6.** *Let $G = (V, E)$ be an n-vertex unweighted digraph undergoing a sequence of edge deletions, let $s \in V$ be a source, and let $d \in \mathbb{N}$ with $d > 33 \lg n$. Let $\mathcal{O}$ be a data structure that maintains for each $v \in V$ a distance estimate $\tilde{d}(s, v) \geq d_G(s, v)$ such that if $d_G(s, v) \leq d$ then $\tilde{d}(s, v) \leq \frac{4}{3} d_G(s, v)$. Whenever an estimate $\tilde{d}(s, v)$ grows to a value of at least $\frac{32}{33}d$, $\mathcal{O}$ outputs v. Then there is a data structure $\mathcal{S}$ with access to $\mathcal{O}$ which maintains a growing set $S \subseteq V$ such that for each $v \in V$,*

1. *if v is reachable from s in $G \setminus S$ then $d_G(s, v) < \frac{32}{33}d$ and otherwise $d_G(s, v) > \frac{2}{3}d$,*
2. *if t_0 is a time step in which $d < d_G^{(t_0)}(s, v) \leq \frac{34}{33}d$ then for every time step $t_1 \geq t_0$ in which $d_G^{(t_1)}(s, v) \leq \frac{34}{33}d$, any shortest s-to-v path P in $G^{(t_1)}$ intersects $S^{(t_0)}$ and for the first such intersection vertex w along P, $d_G^{(t_1)}(s, w) \leq d$, and $d_G^{(t_1)}(w, v) \leq d$.*

At any time, $|S| = O(n \log n/d)$ and $\mathcal{S}$ has total update time $O(m)$, excluding the time spent by $\mathcal{O}$.

Proof. Let $\epsilon = \frac{1}{33}$. For each $v \in V$, let $\hat{d}(v)$ be obtained from the degree of v in the initial graph G by rounding up to the nearest multiple of $\Delta = \lceil m/n \rceil$. In the description of $\mathcal{S}$ below, processing one edge takes at most one unit of time.

Data structure $\mathcal{S}$ initializes $S = \emptyset$ and unmarks all vertices of V. Whenever $\mathcal{O}$ outputs an unmarked vertex v (marked output vertices are ignored), $\mathcal{S}$ runs a modified BFS from s in $G_S = G \setminus S$ which for each vertex w spends $\hat{d}(w)$ time to process its outgoing edges; this can always be achieved by busy-waiting at w if needed. In parallel, $\mathcal{S}$ runs a similar modified BFS from v in $G'_S = (G \setminus S)_{\mathbf{rev}}$[4]. The search from s halts if a layer L_s is found such that $\frac{2}{3}d < d_{G_S}(s, L_s) \leq (\frac{2}{3} + \epsilon)d$ and $|L_s| = O((x \log n)/d)$ (for a suitable hidden constant to be specified) where x is the number of vertices visited by the search, excluding L_s. Similarly, the search from v halts if a layer L_v is found such that $d_{G'_S}(v, L_v) < \epsilon d$ and $|L_v| = O((y \log n)/d)$ (again, for a suitable hidden constant) where y is the number of vertices visited by the search excluding L_v. Let L be the first of the two layers found. $\mathcal{S}$ halts both searches when L is found. Then L is added to S and all vertices visited by the search from v in G'_S are marked. The hidden constants are chosen such that the existence of L follows from Lemma 5; this lemma applies since by assumption, $\epsilon d > \lg n$.

Observe that when $\mathcal{O}$ outputs v, we have $d_G(s, v) \geq (1 - \epsilon)d/(4/3) = (\frac{2}{3} + 2\epsilon)d$ as otherwise we have $d_G(s, v) \leq d$ and hence $\tilde{d}(s, v) \leq \frac{4}{3} d_G(s, v) < (1 - \epsilon)d = \frac{32}{33}d$. This shows the existence of L_s and L_v and that no vertex or edge is visited by both searches. We have $d_{G_S}(s, v) \geq d_G(s, v) \geq (\frac{2}{3} + 2\epsilon)d$ and $d_{G_S}(s, L_s) > \frac{2}{3}d$ and for every $w \in L_v$,

$$d_{G_S}(s, w) \geq d_{G_S}(s, v) - d_{G_S}(w, v) \geq \left(\frac{2}{3} + 2\epsilon\right) d - d_{G'_S}(v, w) = \left(\frac{2}{3} + 2\epsilon\right) d - d_{G'_S}(v, L_v)$$
$$> \left(\frac{2}{3} + \epsilon\right) d,$$

implying that $d_{G_S}(s, L_v) > (\frac{2}{3} + \epsilon)d$. Hence $d_{G_S}(s, L) \geq \min\{d_{G_S}(s, L_s), d_{G_S}(s, L_v)\} > \frac{2}{3}d$.

Showing part 1. Let $v \in V$ and consider any point during the sequence of updates. Assume first that v is reachable from s in G_S. Then $\mathcal{O}$ has not yet output v. For suppose otherwise. If v was unmarked when it was output by $\mathcal{O}$, the above procedure would separate v from s with S, making v unreachable from s in G_S. Conversely, if v was marked, v would already be unreachable from s in G_S since a vertex is only marked when it is separated from s in G_S. In both cases, we have a contradiction. It follows that $\mathcal{O}$ did not output v so $d_G(s, v) \leq \tilde{d}(s, v) < \frac{32}{33}d$, as desired.

[4] By "parallel", we mean that $\mathcal{S}$ alternates between spending one unit of time in one search, then one unit of time in the other search, and so on.

Now, assume that v is not reachable from s in G_S. We may assume that there is a shortest path P from s to v in G since otherwise $d_G(s,v) = \infty > \frac{2}{3}d$. Let w be the first vertex of S along P. It suffices to show that for the prefix P' of P from s to w, $|P'| > \frac{2}{3}d$. At some earlier point in time, the procedure added w to S; just prior to this, P' was contained in G_S so from the above, $|P'| > \frac{2}{3}d$, as desired.

Showing part 2. Let $t_0 \leq t_1$ satisfy the second part of the lemma. Since $d_G^{(t_0)}(s,v) > d$ by assumption, the first part of the lemma implies that v is not reachable from s in $G_S^{(t_0)}$ and hence v is also not reachable from s in $G_S^{(t_1)}$.

Let P be a shortest path from s to v in $G^{(t_1)}$. From what we have just shown, P must intersect $S^{(t_0)}$. Let w be the first vertex of $S^{(t_0)}$ along P. Then clearly, $d_G^{(t_1)}(s,v) = d_G^{(t_1)}(s,w) + d_G^{(t_1)}(w,v)$. Since the vertex w' preceding w on P is reachable from s in $G_S^{(t_0)}$, the first part of the lemma implies that $d_G^{(t_0)}(s,w) \leq d_G^{(t_0)}(s,w') + 1 < \frac{32}{33}d + 1$ and $d_G^{(t_0)}(s,w) > \frac{2}{3}d$. The latter implies that $d_G^{(t_1)}(w,v) = d_G^{(t_1)}(s,v) - d_G^{(t_1)}(s,w) \leq \frac{34}{33}d - d_G^{(t_0)}(s,w) < \frac{34}{33}d - \frac{2}{3}d < d$, showing one of the two inequalities in the second part of the lemma.

We show the other inequality by contradiction so assume that $d_G^{(t_1)}(s,w) > d$. Then $d_G^{(t_1)}(s,w) \geq d+1$ so by the above $d_G(s,w)$ would have increased by more than $d + 1 - (\frac{32}{33}d + 1) = \frac{1}{33}d$ from time step t_0 to t_1. Combining this with $d_G^{(t_1)}(s,v) = d_G^{(t_1)}(s,w) + d_G^{(t_1)}(w,v)$, $d_G^{(t_0)}(w,v) \leq d_G^{(t_1)}(w,v)$, and the triangle inequality, we get

$$d_G^{(t_1)}(s,v) - d_G^{(t_0)}(s,v) \geq d_G^{(t_1)}(s,w) + d_G^{(t_1)}(w,v) - (d_G^{(t_0)}(s,w) + d_G^{(t_0)}(w,v)) > \frac{1}{33}d$$

This contradicts the assumption $d < d_G^{(t_0)}(s,v) \leq d_G^{(t_1)}(s,v) \leq \frac{34}{33}d$. We conclude that $d_G^{(t_1)}(s,w) \leq d$ and $d_G^{(t_1)}(w,v) \leq d$ which shows the second part of the lemma.

Bounding $|S|$ and running time. To bound, $|S|$, consider the two parallel searches from s and from v, respectively, in some update. As argued earlier, there cannot be an edge or vertex visited by both searches. Let X resp. Y be the set of vertices visited by the BFS from s resp. v, excluding L_s resp. L_v and let $x = |X|$ and $y = |Y|$.

Assume first that $L = L_s$. Then all vertices in $Y \cup L_v$ become unreachable in G_S once L has been added to S. For each $w \in V$, $\hat{d}(w)/\Delta \geq 1$. Since each BFS spends $\hat{d}(w)$ time to process edges incident to w and since the two searches run in parallel, we have

$$|L| = O((x \log n)/d) = O\left(\frac{\log n}{d} \sum_{w \in X} \frac{\hat{d}(w)}{\Delta}\right) = O\left(\frac{\log n}{d} \sum_{w \in Y \cup L_v} \frac{\hat{d}(w)}{\Delta}\right)$$

Now, assume that $L = L_v$. Then all vertices of $Y \cup L_v$ become unreachable in G_S once L has been added to S so again,

$$|L| = O((y \log n/d) = O\left(\frac{\log n}{d} \sum_{w \in Y \cup L_v} \frac{\hat{d}(w)}{\Delta}\right)$$

In both cases, the cost $|L|$ of adding L to S can be paid for by charging each vertex w no longer reachable from s in G_S a cost of $O(\frac{\log n}{d}\hat{d}(w)/\Delta)$. Since a vertex is only charged once during the course of the algorithm, we get that for the final separator S (and hence for each intermediate separator),

$$|S| = O\left(\frac{\log n}{d}\sum_{w\in V}\frac{\hat{d}(w)}{\Delta}\right) = O\left(\frac{\log n}{d}\sum_{w\in V}\frac{d(w)+\Delta}{\Delta}\right) = O\left(\frac{\log n(m+n\lceil m/n\rceil)}{d\lceil m/n\rceil}\right)$$
$$= O\left(\frac{n\log n}{d}\right)$$

where the last bound follows since we may assume that all vertices are initially reachable from s in G, implying $m \geq n-1$ and hence $\lceil m/n\rceil = \Theta(m/n)$. This shows the desired bound on $|S|$.

The running time cost of any two parallel searches can be charged to the total degree of the vertices that get marked since they all become unreachable in G_S (this is the vertex set Y above). Since a marked vertex is never visited again by a BFS search, the total running time of parallel searches over all updates is $O(m)$, as desired. ◀

The lemma is somewhat technical and its full strength is only needed for the randomized data structure. For the deterministic data structures, the second part of the lemma will only be applied to the current time step $t_1 = t_0$ so it can be simplified to:

2. if $d < d_G(s,v) \leq \frac{34}{33}d$ then any shortest s-to-v path P in G intersects S and for the first such intersection vertex w along P, $d_G(s,w) \leq d$, and $d_G(w,v) \leq d$.

4 Deterministic Decremental APSP

In this section, we present our deterministic data structures for the exact resp. $(1+\epsilon)$-approximate decremental APSP problem and show Theorems 1 and 2. In the following, let $G = (V,E)$ denote the decremental graph.

4.1 Exact distances

Let $\rho = \frac{34}{33}$ and $D_i = \rho^i$ for $i = 0,\ldots,\lfloor \log_\rho n\rfloor$. For each i and each $u \in V$, we give a data structure $\mathcal{D}_i(u)$ which for any query vertex v maintains a value $\tilde{d}_i(u,v) \geq d_G(u,v)$ with equality if $d_G(u,v) \in (D_i, D_{i+1}]$. In each update, these data structures will be updated in order of increasing i.

Handling all-pairs shortest path distances up to at most $33\lg n$ can be done in $O(mn\log n)$ time using the data structure of Even and Shiloach so we only consider i such that $D_i > 33\lg n$. This allows us to apply Lemma 6. Consider such an i and assume that we already have data structures for all values smaller than i.

Data structure $\mathcal{D}_i(u)$ maintains a separator set $S_i(u)$ using an instance $\mathcal{S}_i(u)$ of the data structure of Lemma 6 with $s = u$ and $d = D_i$; inductively, we have an exact data structure for distances smaller than d and this data structure plays the role of $\mathcal{O}$ with $\tilde{d} = d_G$. At the beginning of each update, $\mathcal{S}_i(u)$ updates $S_i(u)$. Then for each v, if $\mathcal{O}$ reports that $\tilde{d}(u,v)$ has increased from a value of at most D_i to a value strictly greater than D_i, $\mathcal{D}_i(u)$ initializes $S_i(u,v)$ to be the current separator set $S_i(u)$; $\mathcal{D}_i(u)$ then initializes a priority queue $Q_i(u,v)$ where elements are all $s \in S_i(u,v)$ with corresponding keys $\tilde{d}_{i-1}(u,s) + \tilde{d}_{i-1}(s,v)$. During updates, whenever $\mathcal{D}_{i-1}(u)$ resp. $\mathcal{D}_{i-1}(s)$ reports that $\tilde{d}_{i-1}(u,s)$ resp. $\tilde{d}_{i-1}(s,v)$ increases, the key value of s in $Q_i(u,v)$ increases by the same amount. Note that after initialization, $S_i(u,v)$ remains fixed and so does $Q_i(u,v)$ (except for key value changes).

For each vertex v, $\mathcal{D}_i(u)$ maintains $\tilde{d}_i(u,v)$ as the min key value in $Q_i(u,v)$ after $Q_i(u,v)$ has been initialized; prior to this, $\tilde{d}_i(u,v) = \infty$. This completes the description of each $\mathcal{D}_i(u)$.

The overall data structure $\mathcal{D}$ maintains a priority queue $Q(u,v)$ for each vertex pair (u,v) with an element for each i of key value $\tilde{d}_i(u,v)$. For i in increasing order, $\mathcal{D}$ updates $\mathcal{D}_i(u)$ for each u. Whenever a data structure $\mathcal{D}_i(u)$ increases a value $\tilde{d}_i(u,v)$, the corresponding key in $Q(u,v)$ is increased accordingly. On a query (u,v), $\mathcal{D}$ reports the min key value in $Q(u,v)$.

Correctness. We prove that for each i, each time step t_1, and each vertex pair (u,v), $\tilde{d}_i(u,v) \geq d_G(u,v)$ with equality if $d_G(u,v) \in (D_i, D_{i+1}]$. The inequality is clear since every estimate corresponds to the length of some path in the current graph. The equality part is shown by induction on i.

The base cases where $D_i < 33 \lg n$ are clear so pick i with $D_i \geq 33 \lg n$ and $d_G^{(t_1)}(u,v) \in (D_i, D_{i+1}]$ and assume that correctness holds for all vertex pairs and time steps for $i-1$. Let $t_0 \leq t_1$ be the first time step such that $d_G^{(t_0)}(u,v) \in (D_i, D_{i+1}]$. Note that $S_i(u,v) = S_i(u)^{(t_0)}$. The induction hypothesis and the second part of Lemma 6 combined with the observation that no key value in $Q_i(u,v)$ is below $d_G(u,v)$, it follows that the min key value in $Q_i(u,v)$ equals $d_G^{(t_1)}(u,v)$. This shows correctness.

Running time. Consider an $i \in \{0, \ldots, \lfloor \log_\rho n \rfloor\}$ with $D_i \geq 33 \lg n$ and a vertex $u \in V$. We will show that maintaining $\mathcal{D}_i(u)$ takes $O(n^2 \log^2 n)$ time using a standard binary heap. Total time over all i and u will thus be $O(n^3 \log^3 n)$. This dominates the $O(n^3 \log^2 n)$ time to maintain priority queues $Q(u,v)$ and the $O(mn \log n)$ time for the data structure of Even and Shiloach for small values of i.

Maintaining $S_i(u)$ takes a total of $O(m)$ time by Lemma 6. The total number of elements in priority queues $Q_i(u,v)$ over all $v \in S_i(u,v)$ is $O(n^2 \log n / D_i)$, again by Lemma 6. The number of increase-key operations for a single priority queue element s of $Q_i(u,v)$ is $O(D_i)$ which takes a total of $O(D_i \log n)$ time. Over all elements of priority queues $Q_i(u,v)$, this is $O(n^2 \log^2 n)$.

Reporting paths. It is easy to extend our data structure to efficiently answer queries for shortest paths (rather than only shortest path distances) between any vertex pair (u,v). Associated with the min element of $Q(u,v)$ is a vertex s such that for the associated index i, $\tilde{d}_i(u,v) = d_G(u,v) = d_G(u,s) + d_G(s,v)$, $\tilde{d}_{i-1}(u,s) = d_G(u,s)$, and $\tilde{d}_{i-1}(s,v) = d_G(s,v)$. Hence, by recursively querying for pairs (u,s) and (s,v), a shortest u-to-v path in G is reported in time proportional to its length.

4.2 Approximate distances

Let $\epsilon > 0$ be given. We now present our deterministic data structure for the $(1+\epsilon)$-approximate variant of the problem.

The data structure is quite similar to the one for the exact variant so we only describe the changes needed. For $i > 0$ and $u \in V$, we describe data structure $\mathcal{D}_i(u)$ and assume that we have data structures for values less than i. As before, we only consider i with $D_i \geq 33 \lg n$.

Let $\epsilon' > 0$ be a value depending on ϵ such that $(1+\epsilon')^c = \rho$ for some $c \in \mathbb{N}$; we will specify ϵ' later. For $j = 0, \ldots, c = \log_{1+\epsilon'} \rho$, let $d_{i,j} = D_i(1+\epsilon')^j$. This partitions each interval $(D_i, D_{i+1}]$ into c sub-intervals $(D_i(1+\epsilon')^j, D_i(1+\epsilon')^{j+1}]$ for $j = 0, \ldots, c-1$.

$\mathcal{D}_i(u)$ maintains $S_i(u)$ as in the exact version. For each $v \in V$, $\mathcal{D}_i(u)$ maintains an initially empty set $S_i(u,v)$. Once $\mathcal{D}_{i-1}(u)$ reports that $\tilde{d}_{i-1}(u,v)$ increased from a value of at most $D_i(1+\epsilon')^i$ to a value strictly greater than $D_i(1+\epsilon')^i$, $\mathcal{D}_i(u)$ sets $S_i(u,v)$ equal to the current set $S_i(u)$.

For each $j = 0, \ldots, c-1$, a data structure $\mathcal{D}_{i,j}(u)$ maintains the following set for each vertex v:

$$Q_{i,j}(u,v) = \left\{ s \in S_i(u,v) \mid \tilde{d}_{i-1}(u,s) + \tilde{d}_{i-1}(s,v) \leq (1+\epsilon')^i d_{i,j} \right\}.$$

$Q_{i,j}(u,v)$ is maintained by $\mathcal{D}_{i,j}(u)$ as a queue in which every $s \in Q_{i,j}(u,v)$ has key $\tilde{d}_{i-1}(u,s) + \tilde{d}_{i-1}(s,v)$ and where s is removed from $Q_{i,j}(u,v)$ (or increased to ∞) when this value exceeds $(1+\epsilon')^i d_{i,j}$.

For each vertex v, define $\tilde{d}_{i,j}(u,v) = (1+\epsilon')^i d_{i,j}$ if $Q_{i,j}(u,v)$ contains at least one element and otherwise $\tilde{d}_{i,j}(u,v) = \infty$.

Data structure $\mathcal{D}_i(u)$ maintains a min-priority queue $Q_i(u,v)$ for each vertex v with an element of key value $\tilde{d}_{i,j}(u,v)$ for each j. On query v, it outputs $\tilde{d}_i(u,v) = \min\{k, \tilde{d}_{i-1}(u,v)\}$ where k is the min-key value of this queue, i.e., $\tilde{d}_i(u,v) = \min\{\tilde{d}_{i-1}(u,v), \min_j \tilde{d}_{i,j}(u,v)\}$.

The overall data structure $\mathcal{D}$ works in the same manner as for the exact data structure.

Correctness. Consider any point during the sequence of edge deletions. We will show that for suitable choice of ϵ', the estimate $\tilde{d}(u,v)$ that $\mathcal{D}$ outputs satisfies $d_G(u,v) \leq \tilde{d}(u,v) \leq (1+\epsilon) d_G(u,v)$ for every vertex pair (u,v).

We first show that $d_G(u,v) \leq \tilde{d}(u,v)$. It suffices to prove by induction on $i \geq 0$ that $d_G(u,v) \leq \tilde{d}_i(u,v)$. The proof holds for small i such that $D_i < 33 \lg n$ since then we use the data structure of Even and Shiloach, implying $\tilde{d}_i(u,v) = d_G(u,v)$. Now, consider an i such that $D_i \geq 33 \lg n$ and assume that the claim holds for smaller values than i. We have $\tilde{d}_i(u,v) = \min\{\tilde{d}_{i-1}(u,v), \min_j \tilde{d}_{i,j}(u,v)\}$ and $\tilde{d}_{i,j}(u,v) \geq (1+\epsilon')^i d_{i,j} \geq \tilde{d}_{i-1}(u,s) + \tilde{d}_{i-1}(s,v)$ for each $s \in Q_{i,j}(u,v)$. Additionally, if $Q_{i,j}(v) = \emptyset$ then $\tilde{d}_{i,j}(u,v) = \infty$. The induction hypothesis now implies $\tilde{d}_i(u,v) \geq d_G(u,v)$, showing the induction step. Thus, $d_G(u,v) \leq \tilde{d}(u,v)$.

To show that $\tilde{d}(u,v) \leq (1+\epsilon) d_G(u,v)$, we prove by induction on $i \geq 0$ that during all updates and for all vertex pairs (u,v), if $d_G(u,v) \in (0, D_{i+1}]$ then $\tilde{d}_i(u,v) \leq (1+\epsilon')^i d_G(u,v)$. If we can show this then picking $\epsilon' \leq \ln(1+\epsilon)/(\lfloor \log_\rho n \rfloor)$ gives $\tilde{d}(u,v) \leq (1+\epsilon')^{\lfloor \log_\rho n \rfloor} d_G(u,v) \leq e^{\epsilon' \lfloor \log_\rho n \rfloor} d_G(u,v) \leq (1+\epsilon) d_G(u,v)$ for every vertex pair (u,v).

We only need to consider i with $D_i \geq 33 \lg n$ since otherwise, we use the data structure of Even and Shiloach. Assume inductively that the claim holds for values less than i.

Let t_1 be the current time step and consider a vertex pair (u,v) with $d_G^{(t_1)}(u,v) \in (0, D_{i+1}]$. By the induction hypothesis, we may assume that $d_G^{(t_1)}(u,v) \in (D_i, D_{i+1}]$. We may further assume that $\tilde{d}_{i-1}^{(t_1)}(u,v) > D_i(1+\epsilon')^i$ since otherwise,

$$\tilde{d}_i^{(t_1)}(u,v) \leq \tilde{d}_{i-1}^{(t_1)}(u,v) \leq D_i(1+\epsilon')^i < (1+\epsilon')^i d_G^{(t_1)}(u,v).$$

Let $t_0 \leq t_1$ be the first time step where $\tilde{d}_{i-1}^{(t_0)}(u,v) > D_i(1+\epsilon')^i$. We must have $d_G^{(t_0)}(u,v) > D_i$ since otherwise, the induction hypothesis would imply $\tilde{d}_{i-1}^{(t_0)}(u,v) \leq d_G^{(t_0)}(u,v)(1+\epsilon')^{i-1} \leq D_i(1+\epsilon')^{i-1}$, contradicting the choice of t_0. Since also $d_G^{(t_0)}(u,v) \leq d_G^{(t_1)}(u,v) \leq D_{i+1}$, Lemma 6 implies that there is a vertex $s \in S_i^{(t_0)}(u) = S_i^{(t_0)}(u,v) = S_i^{(t_1)}(u,v)$ such that $d_G^{(t_1)}(u,v) = d_G^{(t_1)}(u,s) + d_G^{(t_1)}(s,v)$, $d_G^{(t_1)}(u,s) \leq D_i$, and $d_G^{(t_1)}(s,v) \leq D_i$.

Pick j such that $d_G^{(t_1)}(u,v) \in (d_{i,j}, d_{i,j+1}]$. By the induction hypothesis,

$$\tilde{d}_{i-1}^{(t_1)}(u,s) + \tilde{d}_{i-1}^{(t_1)}(s,v) \leq (1+\epsilon')^{i-1} d_G^{(t_1)}(u,v) \leq (1+\epsilon')^{i-1} d_{i,j+1} = (1+\epsilon')^i d_{i,j}.$$

Hence, $Q_{i,j}(u,v)$ is non-empty at time step t_1 so $\tilde{d}_i^{(t_1)}(u,v) \leq \tilde{d}_{i,j}^{(t_1)}(u,v) = (1+\epsilon')^i d_{i,j} \leq (1+\epsilon')^i d_G^{(t_1)}(u,v)$. This shows the induction step.

Running time. The analysis is similar to the one for exact distances. Pick an $i \in \{0, \ldots, \lfloor \log_\rho n \rfloor\}$ with $D_i \geq 33 \lg n$. The total time to maintain $S_i(u)$ over all u is $O(mn)$.

Observe that each approximate distance $\tilde{d}_{i-1}(u_1, u_2)$ is of the form $(1+\epsilon')^{i'} d_{i',j}$ for $i' \leq i-1$. Since each element s in a queue $Q_{i,j}(u,v)$ has key value $\tilde{d}_{i-1}(u,s) + \tilde{d}_{i-1}(s,v)$, it follows that the number of increase-key operations applied to s in $Q_{i,j}(u,v)$ is $O(\log_{1+\epsilon'} D_i) = O(\log D_i/\epsilon') = O(\log n/\epsilon')$. For our purpose, a simplified queue $Q_{i,j}(u,v)$ suffices which keeps a counter of the number of elements of key value at most $(1+\epsilon')^i d_{i,j}$; this follows since the min key value is at most $(1+\epsilon')^i d_{i,j}$ if and only if the counter is strictly greater than 0. Every queue operation for $Q_{i,j}(u,v)$ can then be supported in $O(1)$ time. The number of elements in $Q_{i,j}(u,v)$ over all u, v, and j is $O(cn^3 \log n/D_i) = O(n^3 \log n/(D_i\epsilon'))$ by Lemma 6. This gives a total time bound of $O(mn + n^3 \log^2 n/(D_i(\epsilon')^2))$. This dominates the time spent on maintaining priority queues $Q_i(u,v)$.

Recall from above that $\epsilon' \leq \ln(1+\epsilon)/(\lfloor \log_\rho n \rfloor)$. The only additional constraint on ϵ' is that $(1+\epsilon')^c = \rho$ for some $c \in \mathbb{N}$. This can be achieved with $\epsilon' = \Theta(\ln(1+\epsilon)/(\lfloor \log_\rho n \rfloor))$. Hence, we get a time bound of $O(mn + n^3 \log^4 n/(D_i\epsilon^2))$.

Note that this bound is no better than the exact data structure for small D_i. We thus consider a hybrid data structure that only applies our data structure when D_i is above some distance threshold d and otherwise applies the data structure of Even and Shiloach which takes a total of $O(mnd)$ time. Summing over all $D_i > d$ and applying a geometric sums argument, the total time for our hybrid data structure is

$$O(mnd + \sum_{i:D_i>d} n^3 \log^4 n/(D_i\epsilon^2))) = O(mnd + n^3 \log^4 n/(d\epsilon^2)))$$

Setting $d = n \log^2 n/(\epsilon\sqrt{m})$ gives Theorem 2. Showing the bound for reporting approximate shortest paths in the theorem is done in the same way as in Section 4.1.

5 Randomized Decremental APSP

In this section, we provide a high-level overview of the randomized $(1+\epsilon)$-approximate data structure and analysis to achieve the result presented in Theorem 3. Building on this overview we will then prove the theorem.

Let us start by focusing on maintaining approximate distances close to the value D_i from a single vertex u and for now we assume an oblivious adversary.

Maintaining a sampled separator subset. Instead of maintaining each separator $S_{i,j}(u,v)$ (with associated with priority queue $Q_{i,j}(u,v)$) as the full vertex separator $S_i(u)$, we obtain a speed-up by only maintaining a sampled subset of $S_i(u)$. As long as this sampled subset certifies that there is a short two-hop path from u to v, the data structure proceeds as in the previous section. When this is no longer the case, there might still be a short two-hop path from u to v through a non-sampled vertex s in the full separator set $S_i(u)$. However, since there are no more sampled candidates, the expected number of vertices of $S_i(u)$ that provide a short two-hop path is small and we can update $S_{i,j}(u,v)$ to be this small subset. It follows that in expectation, $S_{i,j}(u,v)$ can be kept small at all times, which is needed to give a speed-up.

A speed-up using shallow in-trees. The problem with the data structure sketched above is that the entire set $S_i(u)$ had to be scanned in order to update $S_{i,j}(u,v)$ which means that the data structure will not be faster than our deterministic structure from the previous section.

To deal with this, consider the following modification. The set $S_{i,j}(u,v)$ is updated as before by scanning over the entire set $S_i(u)$. Now, an in-tree $T(v)$ is grown from v of radius at most $\epsilon' D_i$. Each vertex v' in $T(v)$ then inherits the set of v, i.e., $S_{i,j}(u,v')$ is updated to the set $S_{i,j}(u,v)$ and this update is fast since $S_{i,j}(u,v)$ is small in expectation. This works since v is a proxy for v' in the sense that a short two-hop path from u to v' via $S_{i,j}(u,v)$ can be extended with a short suffix from $T(v)$, giving a short two-hop path from u to v via $S_{i,j}(u,v)$ (as $T(v)$ is an in-tree of small radius). Now, the time spent on the single scan of $S_i(u)$ can be distributed among all vertices of $T(v)$ and the number of such vertices must be at least $\epsilon' D_i + 1$ (if not, v would be within distance $\epsilon' D_i$ from u).

Unfortunately, the time analysis for the above procedure breaks down if the in-trees grown during the sequence of updates overlap too much. We now sketch how to deal with this. Mark vertices of each in-tree grown so far. When the BFS procedure grows a new in-tree $T(v)$, this procedure is modified by having it backtrack at previously marked vertices which thus become leaves of $T(v)$; this set of marked leaves will be referred to as L in the detailed description below.

Case 1, dealing with a large in-tree. If the number of unmarked vertices visited in $T(v)$ is greater than $\epsilon' D_i$, the above procedure and analysis can be applied; this is referred to as Case 1 in the detailed description below.

Case 2, dealing with a small in-tree. Otherwise, we are in Case 2; here we recall that $T(v)$ has small radius and observe that the only way to enter $T(v)$ from $G \setminus T(v)$ is through L. Hence, for every vertex s in the union $\cup_{v' \in L} S_{i,j}(u,v')$, there is a good two-hop path from u to v through s. But since we know that there is only a small number of such vertices left (in expectation), this union must be small. Furthermore, the union must contain a good separator for every vertex in $T(v)$ (again because $T(v)$ has small radius and because $T(v)$ must be entered through L) and we thus have an efficient way to update $S_{i,j}(u,w)$ for all $w \in T(v)$.

Handling an adaptive adversary. Above we assumed an oblivious adversary. When the adversary is adaptive, we need to be more careful since the approximate distances reported might reveal information about which vertices have been sampled. To deal with this, we round up every two-hop distance on a given distance scale to the same upper bound value (this will only increase the weight of each two-hop path by a small factor so that the output to a query will still be $(1+\epsilon)$-approximate). Hence, the rounded up approximate weight of a two-hop path $u \rightsquigarrow s \rightsquigarrow v$ is the same for every choice of "good" separator vertex s regardless of whether it was sampled or not. It follows that our randomized structure outputs the same distance estimates as a slower deterministic algorithm that maintains the full separator sets. Hence, our randomized algorithm works against an adaptive adversary, as desired.

5.1 The data structure

We now make the the overview formal. First, redefine $\rho = \frac{34-\frac{1}{2}}{33} = \frac{67}{66}$ and pick ϵ' such that $(1+\epsilon')^c = \rho$ for some $c \in \mathbb{N}$ and such that $\rho(1+\epsilon') \leq \frac{34}{33}$. For each u and i such that $D_i \geq 33 \lg n$, a separator $S_i(u)$ is maintained with a data structure $\mathcal{S}_i(u)$ as in Section 4.

We extend the range of index j by 1 so that $j \in \{0, \ldots, c+1\}$. Each structure $\mathcal{D}_{i,j}(u)$ maintains a growing set $M_{i,j}(u)$ of marked vertices; this set is initially empty. In the following, let $U_{i,j}(u) = V \setminus M_{i,j}(u)$ denote the set of unmarked vertices and let $G_{U_{i,j}(u)}$ denote the graph with vertex set V and containing the edges of G having at least one unmarked endpoint.

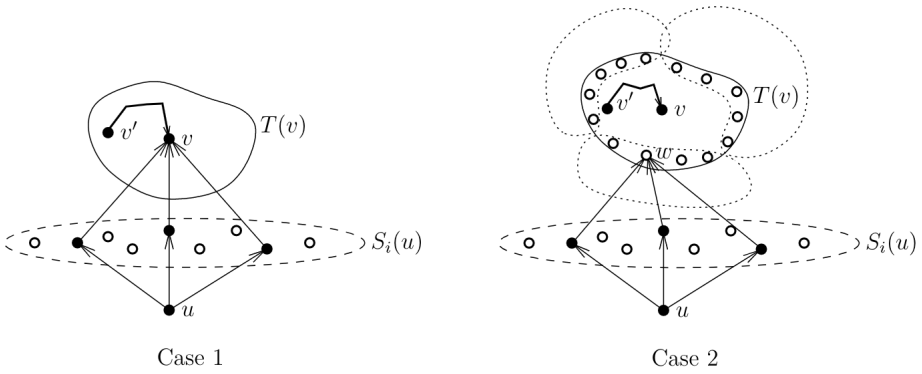


Figure 2 The two cases in the description of the randomized algorithm. Case 1: black vertices of $S_i(u)$ form the subset of vertices s with $\tilde{d}_{i-1}(u,s)+\tilde{d}_{i-1}(s,v) \leq d_{i,j}(1+\epsilon')^{2i}$. For each $v' \in V(T(v))$, $Q_{i,j}(u,v')$ is set to be this subset (with key values adjusted). 2-hop paths from u to v through the subset are shown. Case 2: Vertices of L are shown in white inside $T(v)$. Dotted regions are trees touching $T(v)$. For a $w \in L$, black vertices of $S_i(u)$ form the set $Q_{i,j}(u,w)$ and 2-hop paths from u to w through this set are shown. Q is the union of these sets over all $w \in L$. For each $v' \in V(T(v)) \setminus L$, $Q_{i,j}(u,v')$ is a subset of Q.

In each update, $\mathcal{D}_{i,j}(u)$ maintains $S_{i,j}(u,v)$ and $Q_{i,j}(u,v)$ for $v \in V$ in the following way.

For each $v \in V$ and every vertex s added to $S_i(u)$ in the current update, s is added to $S_{i,j}(u,v)$ with some probability p to be fixed later. Note that only vertices v for which s is actually added to $S_{i,j}(u,v)$ need to be processed. In the full version of the paper [5], we employ a different sampling scheme that avoids having to flip a coin for every vertex $v \in V$ in every update.

For vertices v such that $v \in M_{i,j}(u)$ or such that both $v \in U_{i,j}(u)$ and $\tilde{d}_{i-1}(u,v) \leq D_i(1+\epsilon')^{2i}$, no further processing is done.

Now, assume that $v \in U_{i,j}(u)$ and that $\tilde{d}_{i-1}(u,v) > D_i(1+\epsilon')^{2i}$. If this inequality did not hold in the previous update, each (sampled) vertex of $S_{i,j}(u,v)$ is added to a new min-queue $Q_{i,j}(u,v)$ with key values as in the previous section. Conversely, if the inequality did hold in the previous update, each new (sampled) vertex added to $S_{i,j}(u,v)$ in the current update is added to $Q_{i,j}(u,v)$.

If the min key value of $Q_{i,j}(u,v)$ is greater than $d_{i,j}(1+\epsilon')^{2i}$, $\mathcal{D}_{i,j}(u)$ grows an in-tree $T(v)$ from v in $G_{U_{i,j}(u)}$ up to radius $\epsilon' D_i$.

There are now two cases (see Figure 2): $|V(T(v)) \setminus M_{i,j}(u)| > \epsilon' D_i$ and $|V(T(v)) \setminus M_{i,j}(u)| \leq \epsilon' D_i$.

Case 1: If $|V(T(v)) \setminus M_{i,j}(u)| > \epsilon' D_i$ then $\mathcal{D}_{i,j}(u)$ scans once over $S_i(u)$ to find the subset of vertices $s \in S_i(u)$ for which $\tilde{d}_{i-1}(u,s) + \tilde{d}_{i-1}(s,v) \leq d_{i,j}(1+\epsilon')^{2i}$. For each $v' \in V(T(v))$, $Q_{i,j}(u,v')$ is set to contain exactly this subset of vertices s but with key value $\tilde{d}_{i-1}(u,s) + \tilde{d}_{i-1}(s,v')$.

Case 2: If $|V(T(v)) \setminus M_{i,j}(u)| \leq \epsilon' D_i$ then let $L = V(T(v)) \cap M_{i,j}(u)$ and let $Q = \cup_{v' \in L} Q_{i,j}(u,v')$. For each $v' \in V(T(v)) \setminus L$, $\mathcal{D}_{i,j}(u,v)$ sets $Q_{i,j}(u,v')$ to contain the elements $s \in Q$ with $\tilde{d}_{i-1}(u,s) + \tilde{d}_{i-1}(s,v) \leq d_{i,j}(1+\epsilon')^{2i}$; their key values are $\tilde{d}_{i-1}(u,s) + \tilde{d}_{i-1}(s,v')$.

In both cases, $\mathcal{D}_{i,j}(u)$ then marks all vertices of $T(v)$, i.e., $M_{i,j}(u) \leftarrow M_{i,j}(u) \cup V(T(v))$.

Approximate distances $\tilde{d}_{i,j}(u,v)$ are maintained by $\mathcal{D}_{i,j}(u)$ in a way similar to that in Section 4.2: $\tilde{d}_{i,j}(u,v) = (1+\epsilon')^{2i} d_{i,j}$ if the min key value of $Q_{i,j}(u,v)$ is at most $(1+\epsilon')^{2i} d_{i,j}$ and otherwise $\tilde{d}_{i,j}(u,v) = \infty$.

Data structures $\mathcal{D}_i(u)$ as well as the overall data structure $\mathcal{D}$ work exactly as in Section 4.2.

5.2 Correctness

Consider any point during the sequence of edge deletions. We will show that for suitable choice of ϵ', we have $d_G(u,v) \leq \tilde{d}(u,v) \leq (1+\epsilon) d_G(u,v)$.

We do this by proving that during all updates and for all vertex pairs (u,v), if $d_G(u,v) \in (0, D_{i+1}(1+\epsilon')]$ then $\tilde{d}_i(u,v) \leq (1+\epsilon')^{2i} d_G(u,v)$. By picking $\epsilon' = \ln(1+\epsilon)/(2\lfloor \log_\rho n \rfloor)$, this will give $d_G(u,v) \leq \tilde{d}_G(u,v) \leq (1+\epsilon')^{2\lfloor \log_\rho n \rfloor} d_G(u,v) \leq e^{2\lfloor \log_\rho n \rfloor \epsilon'} d_G(u,v) \leq (1+\epsilon) d_G(u,v)$, as desired.

The proof is by induction on i. The claim is clear for i with $D_i < 33 \lg n$ since then we use the data structure of Even and Shiloach. Now, consider an i with $D_i \geq 33 \lg n$ and assume that the claim holds for values less than i. By the induction hypothesis, we only need to consider pairs (u,v) with $d_G(u,v) \in (D_i(1+\epsilon'), D_{i+1}(1+\epsilon')]$, i.e., $d_G(u,v) \in (d_{i,j}, d_{i,j+1}]$ with $j > 0$.

We first show the following invariant for marked vertices that holds prior to each update over the entire sequence of updates:

▶ **Invariant 7.** *At the end of each update, for every $w \in M_{i,j}(u)$ with $d_G(u,w) \in (d_{i,j}, d_{i,j+1}]$, each shortest u-to-w path in G intersects a vertex $s \in Q_{i,j}(u,w)$ such that $d_G(u,s) \leq D_i$ and $d_G(s,w) \leq D_i$.*

Proof. The invariant is shown by induction on the rank of w in the order in which vertices are marked. Note that this is a proof by induction inside a step of the main proof by induction on i; in addition to the induction hypothesis stated above, we may thus assume that the invariant holds for values less than i. Additionally, for the current value of i, we may assume by induction that the invariant holds for vertices of lower rank than w.

Let t_1 be a time step with $w \in M_{i,j}(u)^{(t_1)}$ and $d_G^{(t_1)}(u,w) \in (d_{i,j}, d_{i,j+1}]$, let $t_0 \leq t_1$ be the time step in which w was marked, and let r be the vertex from which an in-tree $T(r) \ni w$ was grown in time step t_0. Let P be a shortest u-to-w path in $G^{(t_1)}$.

We must have $\tilde{d}_{i-1}^{(t_0)}(u,r) > D_i(1+\epsilon')^{2i}$ since otherwise, no processing would be done for r in time step t_0, contradicting that $T(r)$ is grown in that time step. We also have $d_G^{(t_0)}(u,r) > D_i(1+\epsilon')$ since otherwise the induction hypothesis would give the contradiction $D_i(1+\epsilon') \geq d_G^{(t_0)}(u,r) \geq \tilde{d}_{i-1}^{(t_0)}(u,r)/(1+\epsilon')^{2(i-1)} > D_i(1+\epsilon')^{2i-2(i-1)} = D_i(1+\epsilon')^2$.

By the triangle inequality and the fact that $w \in T(r)$ and $T(r)$ has radius at most $\epsilon' D_i$, we get $d_G^{(t_0)}(u,w) \geq d_G^{(t_0)}(u,r) - d_G^{(t_0)}(w,r) > D_i(1+\epsilon') - \epsilon' D_i = D_i$. Hence, $D_i < d_G^{(t_0)}(u,w) \leq d_G^{(t_1)}(u,w) \leq d_{i,j+1} \leq \frac{34}{33} D_i$ so by Lemma 6, P intersects $S_i^{(t_0)}(u)$ and for the first such intersection vertex s along P, $d_G^{(t_0)}(u,s) \leq d_G^{(t_1)}(u,s) \leq D_i$ and $d_G^{(t_0)}(s,w) \leq d_G^{(t_1)}(s,w) \leq D_i$. We consider the two cases in the description of $\mathcal{D}_{i,j}(u)$ (see Figure 3):

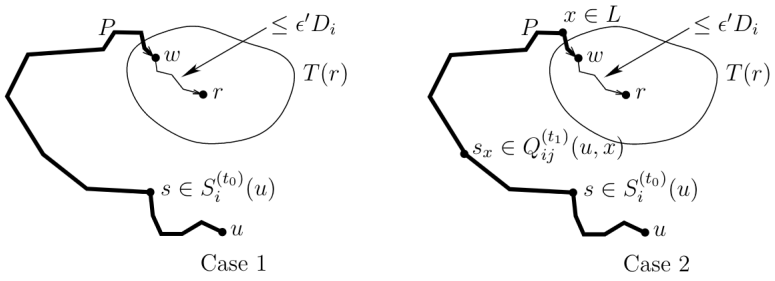


Figure 3 The two cases in the proof of Invariant 7. The path P is marked in bold.

Case 1. It suffices to show that $s \in Q_{i,j}^{(t_1)}(u, w)$. We have $d_G^{(t_0)}(s, r) \leq d_G^{(t_0)}(s, w) + d_G^{(t_0)}(w, r) \leq (1 + \epsilon')D_i$. By the induction hypothesis,

$$\begin{aligned}
\tilde{d}_{i-1}^{(t_0)}(u, s) + \tilde{d}_{i-1}^{(t_0)}(s, r) &\leq (1 + \epsilon')^{2(i-1)} d_G^{(t_0)}(u, r) \\
&\leq (1 + \epsilon')^{2i-2}(d_G^{(t_0)}(u, w) + \epsilon' D_i) \\
&\leq (1 + \epsilon')^{2i-2}(d_G^{(t_1)}(u, w) + \epsilon' D_i) \\
&\leq (1 + \epsilon')^{2i-2}(d_{i,j+1} + \epsilon' d_{i,j+1}) \\
&= (1 + \epsilon')^{2i} d_{i,j},
\end{aligned}$$

so $s \in Q_{i,j}^{(t_0)}(u, w) = Q_{i,j}^{(t_1)}(u, w)$, showing maintenance of the invariant.

Case 2. We first show that P must intersect the set L formed when growing $T(r)$ in time step t_0. Since we are in Case 2, every leaf of $T(r)$ either belongs to L or has no ingoing edges from vertices not in $T(r)$; otherwise, $T(r)$ would contain more than $\epsilon' D_i$ vertices since it is grown up to radius $\epsilon' D_i$. Hence, the only way that P could not intersect L would be if P were fully contained in $T(r)$. But this is not possible since then $T(r)$ would contain at least $|P| + 1 \geq d_{i,j} + 1 > D_i \geq \epsilon' D_i$ unmarked vertices at the beginning of time step t_0, contradicting that we are in Case 2.

Thus, P intersects L and we have $w \notin L$ since w was an unmarked vertex of $T(r)$ when growing this tree. Let x be the last vertex of P belonging to L. Since x was marked earlier than w, the induction hypothesis implies that the subpath of P from u to x intersects $Q_{i,j}^{(t_1)}(u, x) = Q_{i,j}^{(t_0)}(u, x)$ in a vertex s_x such that $d_G^{(t_0)}(u, s_x) \leq d_G^{(t_1)}(u, s_x) \leq D_i$ and $d_G^{(t_0)}(s_x, x) \leq d_G^{(t_1)}(s_x, x) \leq D_i$. The latter implies $d_G^{(t_0)}(s_x, r) \leq (1 + \epsilon')D_i$. By the induction hypothesis, $\tilde{d}_{i-1}^{(t_0)}(u, s_x) + \tilde{d}_{i-1}^{(t_0)}(s_x, r) \leq (1 + \epsilon')^{2(i-1)}(d_G^{(t_0)}(u, s_x) + d_G^{(t_0)}(s_x, r)) = (1 + \epsilon')^{2(i-1)} d_G^{(t_0)}(u, r)$ which by the same calculations as in Case 1 is at most $(1 + \epsilon')^{2i} d_{i,j}$. Inspecting the execution of $\mathcal{D}_{i,j}(u)$ in Case 2, it follows that $s_x \in Q_{i,j}^{(t_0)}(u, w) = Q_{i,j}^{(t_1)}(u, w)$. We have $s_x \in Q_{i,j}^{(t_0)}(u, x) \subseteq S_i^{(t_0)}(u)$. Since s is the first vertex of $S_i^{(t_0)}(u)$ along P, P can thus be decomposed into $u \rightsquigarrow s \rightsquigarrow s_x \rightsquigarrow x \rightsquigarrow w$ and we get $d_G^{(t_1)}(u, s_x) \leq D_i$ (as shown above) and $d_G^{(t_1)}(s_x, w) \leq d_G^{(t_1)}(s, w) \leq D_i$. This shows maintenance of the invariant with s_x in place of s. ◀

Now, we continue with our proof by induction on i. Consider any vertex pair (u, v) at the end of an update with $d_G(u, v) \in (d_{i,j}, d_{i,j+1}]$ and $j > 0$. If $v \notin M_{i,j}(u)$ and $\tilde{d}_{i-1}(u, v) \leq (1 + \epsilon')^{2i} D_i$ then $d_G(u, v) \leq \tilde{d}_{i,j}(u, v) \leq (1 + \epsilon')^{2i} D_i < (1 + \epsilon')^{2i} d_G(u, v)$, as desired.

Now assume that $v \notin M_{i,j}(u)$ and $\tilde{d}_{i-1}(u,v) > (1+\epsilon')^{2i}D_i$. Since v was not marked in the current update, the min key value of $Q_{i,j}(u,v)$ at the end of the update is at most $d_{i,j}(1+\epsilon')^{2i}$ so $d_G(u,v) \leq \tilde{d}_{i,j}(u,v) \leq (1+\epsilon')^{2i}d_{i,j} < (1+\epsilon')^{2i}d_G(u,v)$, as desired.

Finally assume that $v \in M_{i,j}(u)$. By Invariant 7, there is an $s \in Q_{i,j}(u,v)$ such that $d_G(u,v) = d_G(u,s) + d_G(s,v)$, $d_G(u,s) \leq D_i$, and $d_G(s,v) \leq D_i$. By the induction hypothesis, $d_G(u,v) \leq \tilde{d}_i(u,v) \leq \tilde{d}_{i-1}(u,s) + \tilde{d}_{i-1}(s,v) \leq (1+\epsilon')^{2(i-1)}d_G(u,v)$, as desired. This completes the inductive proof and correctness follows.

5.3 Running time

Maintaining separators $S_i(u)$ over all u and i takes $O(mn\log_\rho n) = O(mn\log n)$ time by Lemma 6. For the remaining time analysis, we focus on a single data structure $\mathcal{D}_{i,j}(u)$. It is useful in the following to regard this structure as handling an adversarial sequence of updates consisting of changes to approximate distances maintained by structures $\mathcal{D}_{i'}(v)$ for $i' < i$ and $v \in V$. We will give an expected time bound for $\mathcal{D}_{i,j}(u)$ and we shall rely on the following key lemma; the proof can be found in the full version of the paper [5].

▶ **Lemma 8.** *Let $r \in V$. If at some point in the sequence of updates, $\mathcal{D}_{i,j}(u)$ grows an in-tree from r then at the end of that update, the expected number of vertices $s \in S_i(u)$ satisfying $\tilde{d}_{i-1}(u,s) + \tilde{d}_{i-1}(s,r) \leq D_i(1+\epsilon')^{2i}$ is $O(\ln n/p)$. This bound holds against an adaptive adversary.*

▶ **Corollary 9.** *When a vertex v is marked, $E[|Q_{i,j}(u,v)|] = O(\ln n/p)$ and this bound holds against an adaptive adversary.*

Proof. Consider the update in which v is marked and let r be the root of the in-tree $T(r)$ containing v. If $|T(r)| \geq \epsilon' D_i$ then $Q_{i,j}(u,v) = Q_{i,j}(u,r) \subseteq S_i(u)$ and all $s \in Q_{i,j}(u,r)$ satisfy the inequality of Lemma 8. In the case where $|T(r)| < \epsilon' D_i$, let Q be as defined in the description of the data structure. Then vertices $s \in Q \subseteq S_i(u)$ are only added to $Q_{i,j}(u,v)$ if they satisfy the inequality of Lemma 8. The corollary now follows. ◀

Now, we can bound the time spent by $\mathcal{D}_{i,j}(u)$. The total time spent on growing in-trees is $O(m)$ since every edge (w_1,w_2) visited must have $w_1 \notin M_{i,j}(u)$ at the beginning of the BFS search and $w_1 \in M_{i,j}(u)$ immediately afterwards and a vertex can never be unmarked. This also bounds the time spent on marking vertices.

The total expected number of sampled vertices added to $Q_{i,j}(u,v)$ prior to v being marked is at most xp where x is the size of the set $S_i(u)$ after the final update. By Lemma 6, $x = O(n\log n/D_i)$. By Corollary 9, the expected size of $Q_{i,j}(u,v)$ after v is marked is $O(\ln n/p)$. Using the same argument as in the running time analysis of Section 4.2, the number of increase-key operations applied to a single element of $Q_{i,j}(u,v)$ is $O(\log n/\epsilon')$. Hence, the total expected time spent on operations on $Q_{i,j}(u,v)$ is $O((n\log n \cdot p/D_i + \log n/p)\log^3 n/\epsilon)$.

Whenever $\mathcal{D}_{i,j}(u)$ grows an in-tree $T(r)$ with $|V(T(r)) \setminus M_{i,j}(u,v)| > \epsilon' D_i$, scanning $S_i(u)$ takes $O(n\log n/D_i)$ time by Lemma 6. Since all vertices of $V(T(r)) \setminus M_{i,j}(u,v)$ are marked just after $T(r)$ is grown and since vertices are never unmarked, the number of such trees over the course of the updates is at most $n/(\epsilon' D_i)$ so the total time for all these scans is $O(n^2\log^2 n/(\epsilon D_i^2))$.

Whenever $\mathcal{D}_{i,j}(u)$ grows an in-tree $T(r)$ with $|V(T(r)) \setminus M_{i,j}(u,v)| \leq \epsilon' D_i$, the set $\cup_{x\in L}Q_{i,j}(u,x)$ needs to be computed. Note that for each $x \in L$, $E[|Q_{i,j}(u,x)|] = O(\log n/p)$ by Corollary 9. At least one edge (y,x) ingoing to x belongs to $T(r)$ and this edge is not part of any later grown in-tree since x is marked immediately after $T(r)$ is grown. We charge a cost of $O(\log n/p)$ to (y,x) for computing $Q_{i,j}(u,x)$. Over all $x \in L$, this pays for computing $\cup_{x\in L}Q_{i,j}(u,x)$ and we get a total expected time bound for this part of $O(m\log n/p)$.

Summing the above over all u, v, i, and j, we get a total expected time bound for our data structure of

$$\tilde{O}(mn/\epsilon + \sum_i \sum_j (n^3 \cdot p/(D_i\epsilon) + n^2/(p\epsilon) + n^3/(\epsilon D_i^2) + mn/p).$$

Since this bound is only fast for sufficiently large i, we pick a distance threshold d and apply the algorithm of Even and Shiloach for distances of at most d and our data structure for distances above d. By a geometric sums argument, our hybrid algorithm has a expected total time bound of

$$\begin{aligned}&\tilde{O}(mnd + mn/\epsilon + n^3 \cdot p/(d\epsilon^2) + n^2/(p\epsilon^2) + n^3/(\epsilon^2 d^2) + mn/(p\epsilon))\\ &= \tilde{O}(mnd + n^3 \cdot p/(d\epsilon^2) + n^2/(p\epsilon^2) + n^3/(d^2\epsilon^2) + mn/(p\epsilon))\end{aligned}$$

Setting the second and fifth terms equal to each other, we get $p = \tilde{\Theta}(\sqrt{m\epsilon d}/n)$ and the time bound simplifies to

$$\tilde{O}(mnd + \sqrt{m}n^2/(\sqrt{d}\epsilon^{3/2}) + n^3/(\sqrt{md}\epsilon^{5/2}) + n^3/(d^2\epsilon^2)).$$

We balance the first two terms by setting $d = \tilde{\Theta}(n^{2/3}/(m^{1/3}\epsilon))$ and we get a time bound of

$$\tilde{O}(m^{2/3}n^{5/3}n/\epsilon + n^{8/3}/(m^{1/3}\epsilon^2)),$$

which shows the time bound of Theorem 3.

References

1. Surender Baswana, Ramesh Hariharan, and Sandeep Sen. Improved decremental algorithms for maintaining transitive closure and all-pairs shortest paths. In *Proceedings of the thiry-fourth annual ACM symposium on Theory of computing*, pages 117–123. ACM, 2002.
2. Aaron Bernstein. Maintaining shortest paths under deletions in weighted directed graphs. *SIAM Journal on Computing*, 45(2):548–574, 2016.
3. Camil Demetrescu and Giuseppe F Italiano. A new approach to dynamic all pairs shortest paths. *Journal of the ACM (JACM)*, 51(6):968–992, 2004.
4. Camil Demetrescu and Giuseppe F Italiano. Fully dynamic all pairs shortest paths with real edge weights. *Journal of Computer and System Sciences*, 72(5):813–837, 2006.
5. Jacob Evald, Viktor Fredslund-Hansen, Maximilian Probst Gutenberg, and Christian Wulff-Nilsen. Decremental APSP in directed graphs versus an adaptive adversary. *CoRR*, abs/2010.00937, 2020. `arXiv:2010.00937`.
6. Shimon Even and Yossi Shiloach. An on-line edge-deletion problem. *Journal of the ACM (JACM)*, 28(1):1–4, 1981.
7. M.R. Henzinger and Valerie King. Fully dynamic biconnectivity and transitive closure. In *Proceedings of IEEE 36th Annual Foundations of Computer Science*, pages 664–672. IEEE Comput. Soc. Press, 1995. `doi:10.1109/SFCS.1995.492668`.
8. Adam Karczmarz and Jakub Lacki. Reliable hubs for partially-dynamic all-pairs shortest paths in directed graphs. In *27th Annual European Symposium on Algorithms (ESA 2019)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2019.
9. Adam Karczmarz and Jakub Łącki. Simple label-correcting algorithms for partially dynamic approximate shortest paths in directed graphs. In *Symposium on Simplicity in Algorithms*, pages 106–120. SIAM, 2020.
10. Valerie King. Fully dynamic algorithms for maintaining all-pairs shortest paths and transitive closure in digraphs. In *Foundations of Computer Science, 1999. 40th Annual Symposium on*, pages 81–89. IEEE, 1999.
11. Mikkel Thorup. Fully-dynamic all-pairs shortest paths: Faster and allowing negative cycles. In *Scandinavian Workshop on Algorithm Theory*, pages 384–396. Springer, 2004.

On the Approximability of Multistage Min-Sum Set Cover

Dimitris Fotakis
National Technical University of Athens, Greece

Panagiotis Kostopanagiotis
National Technical University of Athens, Greece

Vasileios Nakos
Saarland University, Saarland Informatics Campus, Saarbrücken, Germany
Max Planck Institute for Informatics, Saarland Informatics Campus, Saarbrücken, Germany

Georgios Piliouras
Singapore University of Technology and Design, Singapore

Stratis Skoulakis
Singapore University of Technology and Design, Singapore

Abstract

We investigate the polynomial-time approximability of the multistage version of Min-Sum Set Cover (Mult-MSSC), a natural and intriguing generalization of the classical List Update problem. In Mult-MSSC, we maintain a sequence of permutations $(\pi^0, \pi^1, \ldots, \pi^T)$ on n elements, based on a sequence of requests $\mathcal{R} = (R^1, \ldots, R^T)$. We aim to minimize the total cost of updating π^{t-1} to π^t, quantified by the Kendall tau distance $\mathrm{d_{KT}}(\pi^{t-1}, \pi^t)$, plus the total cost of covering each request R^t with the current permutation π^t, quantified by the position of the first element of R^t in π^t.

Using a reduction from Set Cover, we show that Mult-MSSC does not admit an $O(1)$-approximation, unless P = NP, and that any $o(\log n)$ (resp. $o(r)$) approximation to Mult-MSSC implies a sublogarithmic (resp. $o(r)$) approximation to Set Cover (resp. where each element appears at most r times). Our main technical contribution is to show that Mult-MSSC can be approximated in polynomial-time within a factor of $O(\log^2 n)$ in general instances, by randomized rounding, and within a factor of $O(r^2)$, if all requests have cardinality at most r, by deterministic rounding.

2012 ACM Subject Classification Theory of computation → Approximation algorithms analysis

Keywords and phrases Approximation Algorithms, Multistage Min-Sum Set Cover, Multistage Optimization Problems

Digital Object Identifier 10.4230/LIPIcs.ICALP.2021.65

Category Track A: Algorithms, Complexity and Games

Funding *Dimitris Fotakis*: Supported by the Hellenic Foundation for Research and Innovation (H.F.R.I.) under the "First Call for H.F.R.I. Research Projects to support Faculty members and Researchers and the procurement of high-cost research equipment grant", project BALSAM, HFRI-FM17-1424.
Vasileios Nakos: Supported by the project TIPEA that has received funding from the European Research Council (ERC) under the European Unions Horizon 2020 research and innovation programme (grant agreement No. 850979).
Georgios Piliouras: Supported by NRF2019-NRF-ANR095 ALIAS grant, grant PIE-SGP-AI-2018-01, NRF 2018 Fellowship NRF-NRFF2018-07, AME Programmatic Fund (Grant No. A20H6b0151) from the Agency for Science, Technology and Research (A*STAR) and AI Singapore grant AISG2-RP-2020-016.
Stratis Skoulakis: Supported by NRF 2018 Fellowship NRF-NRFF2018-07.

1 Introduction

In *Multistage Min-Sum Set Cover* (Mult-MSSC), we are given a universe U on n elements, a sequence of requests $\mathcal{R} = (R_1, \ldots, R_T)$, with $R_t \subseteq U$, and an initial permutation π^0 of the elements of U. We aim to maintain a sequence of permutations $(\pi^0, \pi^1, \ldots, \pi^T)$ of U, so as to minimize the total cost of updating (or moving from) π^{t-1} to π^t in each time step plus the total cost of covering each request R_t with the current permutation π^t. The cost of moving from π^{t-1} to π^t is the number of inverted element pairs between π^{t-1} and π^t, i.e., the Kendall Tau distance $\mathrm{d_{KT}}(\pi^{t-1}, \pi^t)$. The cost $\pi^t(R_t)$ of covering a request R_t with a permutation π^t is the position of the first element of R_t in π^t, i.e., $\pi^t(R_t) = \min\{i \,|\, \pi^t(i) \in R_t\}$. Thus, given $\mathcal{R} = (R_1, \ldots, R_T)$, we aim to minimize $\sum_{t=1}^{T} \left(\mathrm{d_{KT}}(\pi^{t-1}, \pi^t) + \pi^t(R_t)\right)$.

The Mult-MSSC problem is a natural generalization of the (offline version of the) classical *List Update* problem [26], where $|R_t| = 1$ for all requests $R_t \in \mathcal{R}$. The offline version of List Update is NP-hard [2], while it is known that any 5/4-approximation has to resort to *paid exchanges*, where an element different from the requested one is moved forward to the list [24, 28]. Mult-MSSC was introduced in [17] as the multistage extension of Min-Sum Set Cover (MSSC) [15], where we aim to compute a single static permutation π that minimizes the total covering cost $\sum_{t=1}^{T} \pi(R_t)$. [17] presented a (simple polynomial-time) online algorithm for Mult-MSSC with competitive ratio between $\Omega(r\sqrt{n})$ and $O(r^{3/2}\sqrt{n})$ for r-bounded instances, where all requests have cardinality at most r, and posed the polynomial-time approximability of Mult-MSSC as an interesting open question. Mult-MSSC is also related to recently studied time-evolving (a.k.a. multistage or dynamic) optimization problems (e.g., multistage matroid, spanning set and perfect matching maintenance [19], time-evolving Facility Location [14, 3]), where we aim to maintain a sequence of near-optimal feasible solutions to a combinatorial optimization problem, in response to time-evolving underlying costs, without changing too much the solution from one step to the next.

Motivation. Mult-MSSC is motivated by applications, such as web search, news, online shopping, paper bidding, etc., where items are presented to the users sequentially. Then, the item ranking is of paramount importance, because user attention is usually restricted to the first few items in the sequence (see e.g., [27, 13, 16, 10]). If a user does not spot an item fitting her interests there, she either leaves the service (in case of news or online shopping, see e.g., the empirical evidence in [12]) or settles on a suboptimal action (in case of paper bidding, see e.g., [11]). To mitigate such situations and increase user retention, modern online services highly optimize item rankings based on user scrolling and click patterns. Each user t is represented by her set of preferred items (or item categories) R_t . The goal of the service provider is to continually maintain an item ranking π^t, so that the current user t finds one of her favorite items at a relatively high position in π^t. Continual ranking update is dictated by the fact that users with different characteristics and preferences tend to use the online service during the course of the day (e.g., elderly people in the morning, middle-aged people in the evening, young people at the night – similar patterns apply for people from different countries and timezones). Moreover, different user categories react in nonuniform ways to different trends (in e.g., news, fashion, sports, scientific topics). For consistency and stability, however, the ranking should change neither too much nor too frequently. Mult-MSSC makes the (somewhat simplifying) assumptions that the service provider has a relatively accurate knowledge of user preferences and their arrival order, and that its total cost is proportional to how deep in π^t the current user t should reach, before she finds one of her favorite items, and to how much the ranking changes from one user to the next.

From a theoretical viewpoint, Mult-MSSC was used in [17] as a natural benchmark for studying the dynamic competitive ratio of Online Min-Sum Set Cover, where the algorithm updates its permutation online, without any knowledge of future requests. As in Mult-MSSC, the objective is to minimize the total moving plus the total covering cost.

Contribution and Techniques. In this work, we initiate a study of the polynomial-time approximability of Mult-MSSC. Using a reduction from Set Cover, we show (Theorem 7) that Mult-MSSC does not admit a $c \log n$-approximation, for some absolute constant c, unless P = NP. Moreover our reduction establishes that an $o(r)$-approximation for r-bounded instances of Mult-MSSC implies an $o(r)$-approximation for Set Cover, in case each element appears in at most r requests.

Our main technical contribution is to show that Mult-MSSC can be approximated in polynomial-time within a factor of $O(\log^2 n)$ in general instances, by randomized rounding (Theorem 10), and within a factor of $O(r^2)$ in r-bounded instances, by deterministic rounding (Theorem 11).

For both results, we consider a restricted version of Mult-MSSC, inspired by the Move-to-Front (MTF) algorithm for List Update, where in each time step t, we can only move a single element of R_t from its position in π^{t-1} to the first position of π^t. Since such a permutation π^t coves R_t with unit cost, we now aim to select the element of each R_t moved to front of π^t, so as to minimize the total moving cost $\sum_{t=1}^{T} \mathrm{d_{KT}}(\pi^{t-1}, \pi^t)$. It is not hard to see that the optimal cost of serving $\mathcal{R}$ under the restricted Move-to-Front version of Mult-MSSC is within a factor of 4 from the optimal cost under the original, more general, definition of Mult-MSSC.

Hence, approximating Mult-MSSC boils down to determining which element of R_t should become the top element of π^t. To this end, we relax permutations to doubly stochastic matrices and consider a Linear Programming relaxation of the restricted Move-to-Front version of Mult-MSSC, which we call *Fractional-MTF* (see Definition 8). Given the optimal solution of the aforementioned linear program, which is a sequence of doubly stochastic matrices $(A^0, A^1, \ldots, A^T)$, with A^0 corresponding to the initial permutation π^0, our main technical challenge is to round each doubly stochastic matrix A^t to a permutation π^t such that (i) there is an element of R_t at one of the few top positions of π^t; and (ii) the total moving cost $\sum_{t=1}^{T} \mathrm{d_{KT}}(\pi^{t-1}, \pi^t)$ of the rounded solution is comparable to the total moving cost $\sum_{t=1}^{T} \mathrm{d_{FR}}(A^{t-1}, A^t)$ of the optimal solution of Fractional-MTF, where $\mathrm{d_{FR}}$ is a notion of distance equivalent to Spearman's footrule distance on permutations (see Definition 4).

Working towards a randomized rounding approach, we first observe that rounding each doubly stochastic matrix independently may result in a permutation sequence with total moving cost significantly larger than that of Fractional-MTF (see also the discussion after Lemma 9). In Theorem 10, we show that a dependent randomized rounding with logarithmic scaling of entries (Algorithm 1), similar in spirit with the randomized rounding approach [8, 25] for Generalized Min-Sum Set Cover, results in an approximation ratio of $O(\log^2 n)$. Interestingly, Algorithm 1 without the logarithmic scaling results in a permutation sequence with the expected moving cost within a factor of 4 from the optimal moving cost of Fractional-MTF. However, we lose a logarithmic factor in the approximation ratio, because we need to scale up the entries of each doubly stochastic matrix A^t, so as to ensure that some element of R_t appears in the few top positions of π^t with sufficiently large probability. The other logarithmic factor is lost because there could be a logarithmic number of elements allocated to the same position of the resulting permutation by the randomized rounding.

Our deterministic rounding of Algorithm 2 for r-bounded request sequences is motivated by the deterministic rounding for Set Cover and Vertex Cover. We observe that in the optimal solution of Fractional-MTF, in each time step t, there is some element $e \in R_t$ with $A^t_{e1} \geq 1/r$ (i.e., e occupies a fraction of at least $1/r$ of the first position in the "fractional permutation" A^t). Algorithm 2 simply moves any such element to the front of π^t. The most challenging part of the analysis is to establish that for any optimal solution $(A^0, A^1, \ldots, A^T)$ of Fractional-MTF with respect to an r-bounded request sequence, there exists a sequence of doubly stochastic matrices $(A^0, \hat{A}^1, \ldots, \hat{A}^T)$ with the entries of each $\hat{A}^t$ being multiples of $1/r$, such that (i) the moving cost of $(A^0, \hat{A}^1, \ldots, \hat{A}^T)$ is bounded from above by the optimal cost of Fractional-MTF; and (ii) each matrix $\hat{A}^t$ contains in the first position the element that Algorithm 2 keeps in the first position at round t, with mass at least $1/r$. Then we show (Lemma 20) that for any sequence of doubly stochastic matrices $(A^0, \hat{A}^1, \ldots, \hat{A}^T)$ satisfying the above properties, the moving cost of Algorithm 2 is at most the moving cost of the doubly stochastic matrices, $\sum_{t=1}^{T} \mathrm{d_{FR}}(\hat{A}^t, \hat{A}^{t-1})$. The latter is done through the use of an appropriate potential function based on an extension of the Kendall-Tau distance to doubly stochastic matrix with entries being multiples of $1/r$.

A potentially interesting insight is that the technical reason for the quadratic dependence of our approximation ratios on $\log n$ and r is conceptually similar to the reason for the (best possible) approximation ratio of $4 = 2 \cdot 2$ in [15] (see the discussion after Theorem 10). Hence, we conjecture that any $o(\log^2 n)$ (resp. $o(r^2)$) approximation to Mult-MSSC must imply a sublogarithmic (resp. $o(r)$) approximation to Set Cover.

Other Related Work. The MSSC problem generalizes various NP-hard problems, such as Min-Sum Vertex Cover and Min-Sum Coloring and it is well-studied. Feige, Lovasz and Tetali [15] proved that the greedy algorithm, which picks in each position the element that covers the most uncovered requests, is a 4-approximation (that was also implicit in [9]) and that no $(4 - \varepsilon)$-approximation is possible, unless P = NP. In Generalized MSSC (a.k.a. *Multiple Intents Re-ranking*), there is a covering requirement $K(R_t)$ for each request R_t and the cost of covering a request R_t is the position of the $K(R_t)$-th element of R_t in the (static) permutation π. The MSSC problem is the special case where $K(R_t) = 1$ for all requests R_t. Another notable special case of Generalized MSSC is the Min-Latency Set Cover problem [20], which corresponds to the other extreme case where $K(R_t) = |R_t|$ for all requests R_t. Generalized MSSC was first studied by Azar et al. [5], who presented a $O(\log r)$-approximation; later $O(1)$-approximation algorithms were obtained [8, 25, 23, 6].

Further generalizations of Generalized MSSC have been considered, such as the Submodular Ranking problem, studied in [4], which generalizes both Set Cover and MSSC, and the Min-Latency Submodular Cover, studied by Im et al. [22]. We refer to [22, 21] for a detailed discussion on the connections between these problems and their applications.

The online version of MSSC, which generalizes the famous List Update problem, was studied in [17]. They proved that its static deterministic competitive ratio is $\Theta(r)$ and presented a natural memoryless algorithm, called *Move-all-Equally*, with static competitive ratio in $\Omega(r^2)$ and $2^{O(\sqrt{\log n \cdot \log r})}$ and dynamic competitive ratio in $\Omega(r\sqrt{n})$ and $O(r^{3/2}\sqrt{n})$-competitive. Subsequently, [18] considered MSSC from the viewpoint of online learning. Through dimensionality reduction from permutations to doubly stochastic matrices, they obtained randomized (resp. deterministic) polynomial-time online learning algorithms with $O(1)$-regret for Generalized MSSC (resp. $O(r)$-regret for MSSC).

2 Preliminaries and Basic Definitions

The set of elements e is denoted by U with $|U| = n$. A permutation of the elements is denoted by π where π_i denotes the element lying at position i (for $1 \leq i \leq n$) and $\mathrm{Pos}(e, \pi)$ denotes the position of the element $e \in U$ in permutation π.

▶ **Definition 1** (Kendall-Tau Distance). *Given the permutations π^A, π^B, a pair of elements (e, e') is inverted if and only if $\mathrm{Pos}(e, \pi^A) > \mathrm{Pos}(e', \pi^A)$ and $\mathrm{Pos}(e, \pi^B) < \mathrm{Pos}(e', \pi^B)$ or vice versa. The Kendall-Tau distance between the permutations π^A, π^B, denoted by $\mathrm{d_{KT}}(\pi^A, \pi^B)$, is the number of inverted pairs.*

▶ **Definition 2** (Spearman' Footrule Distance). *The FootRule distance between the permutations π^A, π^B is defined as $\mathrm{d_{FR}}(\pi^A, \pi^B) = \sum_{e \in U} |\mathrm{Pos}(e, \pi^A) - \mathrm{Pos}(e, \pi^B)|$.*

The Kendall-Tau distance and FootRule distance are approximately equivalent, $\mathrm{d_{KT}}(\pi^A, \pi^B) \leq \mathrm{d_{FR}}(\pi^A, \pi^B) \leq 2 \cdot \mathrm{d_{KT}}(\pi^A, \pi^B)$. Moreover both of them satisfy the triangle inequality.

▶ **Definition 3.** *An $n \times n$ matrix with positive entries (rows stand for the elements and columns for the positions) is called stochastic if $\sum_{i=1}^{n} A_{ei} = 1$ for all $e \in U$ and doubly stochastic if (additionally) $\sum_{e \in U} A_{ei} = 1$ for all $1 \leq i \leq n$.*

A permutation of the elements π can be equivalent represented by a 0-1 doubly stochastic matrix A, where $A_{ei} = 1$ if element e lies at position i and 0 otherwise. When clear from context, we use the notion of permutation and (0-1) doubly stochastic matrix interchangeably.

The notion of FootRule distance can be naturally extended to stochastic matrices. Given two doubly stochastic matrices A, B consider the min-cost transportation problem, transforming row A_e to the row B_e where the cost of transporting a unit of mass between column i and column j equals $|i - j|$. Formally for each row e, define a complete bipartite graph where on the left part lie the entries (e, i) for $1 \leq i \leq n$ and on the right part the entries (e, j) for $1 \leq j \leq n$. The mass transported from entry (e, i) to entry (e, j) (denoted as f_{ij}^e) costs $f_{ij}^e \cdot |i - j|$ and the total mass *leaving* (e, i) equals A_{ei} and the total mass *arriving* at (e, j) equals B_{ej}.

▶ **Definition 4.** *The FootRule distance between two stochastic matrices A, B, denoted by $\mathrm{d_{FR}}(A, B)$, is the optimal value of the following linear program,*

$$
\begin{aligned}
\textit{min} \quad & \sum_{e \in U} \sum_{i=1}^{n} \sum_{j=1}^{n} |i - j| \cdot f_{ij}^e \\
\textit{s.t} \quad & \sum_{i=1}^{n} f_{ij}^e = B_{ej} \qquad \textit{for all } e \in U \textit{ and } j = 1, \ldots, n \\
& \sum_{j=1}^{n} f_{ij}^e = A_{ei} \qquad \textit{for all } e \in U \textit{ and } i = 1, \ldots, n \\
& f_{ij}^e \geq 0 \qquad\qquad \textit{for all } e \in U \textit{ and } i, j = 1, \ldots, n
\end{aligned}
$$

▶ **Example 5.** Let the stochastic matrices $A = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$, $B = \begin{pmatrix} 1/3 & 1/3 & 1/3 \\ 1/2 & 1/2 & 0 \\ 1/4 & 0 & 3/4 \end{pmatrix}$. The FootRule distance $\mathrm{d_{FR}}(A, B) = \underbrace{(0 \cdot 1/3 + 1 \cdot 1/3 + 2 \cdot 1/3)}_{\text{first row}} + \underbrace{(1 \cdot 1/2 + 0 \cdot 1/2 + 1 \cdot 0)}_{\text{second row}} + \underbrace{(2 \cdot 1/4 + 1 \cdot 0 + 0 \cdot 3/4)}_{\text{third row}} = 2$.

Up next we present the formal definition of Multistage Min-Sum Set Cover.

▶ **Definition 6** (Multistage Min-Sum Set Cover)**.** *Given a universe of elements U, a sequence of requests $R_1, \ldots, R_T \subseteq U$ and an initial permutation of the elements π^0. The goal is to select a sequence of permutation $\pi^1, \ldots, \pi^T$ that minimizes*

$$\sum_{t=1}^{T} \pi^t(R_t) + \sum_{t=1}^{T} \mathrm{d_{KT}}(\pi^t, \pi^{t-1})$$

where $\pi^t(R_t)$ is the position of the first element of R_t that we encounter in π^t, $\pi^t(R_t) = \min\{1 \leq i \leq n : \pi^t_i \in R_t\}$.

We refer to $\sum_{t=1}^{T} \pi^t(R_t)$ as **covering cost** and to $\sum_{t=1}^{T} \mathrm{d_{KT}}(\pi^t, \pi^{t-1})$ as **moving cost**. We denote with π^t_{Opt} the permutation of the optimal solution of Mult-MSSC at round t, with o_t the element that the optimal solution uses to cover the request R_t (the element of R_t appearing first in π^t_{Opt}), and with $\mathrm{OPT}_{\text{Mult-MSSC}}$ the cost of the optimal solution. Finally we call an instance of Mult-MSSC *r-bounded* in case the cardinality of the requests is bounded by r, $|R_t| \leq r$.

3 Approximation Algorithms for Dynamic Min-Sum Set Cover

There exists an approximation-preserving reduction from Set − Cover to Mult-MSSC that provides us with the following inapproximability results.

▶ **Theorem 7.**

- *There is no $c \cdot \log n$-approximation algorithm for* Mult-MSSC *(for a sufficiently small constant c) unless* P = NP.
- *For r-bounded sequences, there is no $o(r)$-approximation algorithm for* Mult-MSSC*, unless there is a $o(r)$-approximation algorithm for* Set − Cover *with each element being covered by at most r sets.*

The proof of Theorem 7 is fairly simple, given an instance of Set − Cover we construct an instance of Mult-MSSC in which the initial permutation π^0 contains in the first positions some dummy elements (they do not appear in any of the requests) and in the last positions the sets of the Set − Cover (we consider an element of Mult-MSSC for each set of Set − Cover). Finally each request for Mult-MSSC is associated with an element of the Set − Cover and contains the *elements in* Mult-MSSC*/ sets in* Set − Cover containing it.

Proof. Let the equivalent definition of Set − Cover in which we are given a universe of element $E = \{1, \ldots, n\}$ and sets $S_1, S_2, \ldots, S_m \subseteq E$ and we are asked to select the minimum number of elements covering all the sets (an element e covers set S_i if $e \in S_i$).

Consider the instance of Mult-MSSC with the elements $U = \{1, \ldots, n\} \cup \{d_1, \ldots, d_{n^2m}\}$. The elements $\{d_1, \ldots, d_{n^2m}\}$ are dummy in the sense that they appear in none of the requests R_t. Let the initial permutation π_0 contain in the first n^2m positions the dummy elements and in the last n positions the elements $\{1, \ldots, n\}$, $\pi_0 = [d_1, \ldots, d_{n^2m}, 1, \ldots, n]$ and the request sequence of Mult-MSSC be $S_1, S_2, \ldots, S_m$.

Let a c-approximation algorithm for Mult-MSSC producing the permutation $\pi_1, \ldots, \pi_m$ the cost of which is denoted by Alg. Let also CoverAlg denote the set composed by the element that the c-approximation algorithm uses to cover the requests, CoverAlg = {the element of S_t appearing first in π_t}. Then,

$$\mathrm{Alg} \geq n^2 m \cdot |\mathrm{CoverAlg}|.$$

Now consider the following solution for Mult-MSSC constructed by the optimal solution for Set − Cover. This solution initially moves the elements of the optimal covering set $\mathrm{OPT}_{\mathrm{SetCover}}$ to the first positions and then never changes the permutation. Clearly the cost of this solution is upper bounded by

$$\text{Set} - \text{Cover}_{\text{Mult-MSSC}} \leq \underbrace{|\mathrm{OPT}_{\mathrm{SetCover}}| \cdot (n^2 m + n)}_{\text{moving cost}} + \underbrace{m \cdot |\mathrm{OPT}_{\mathrm{SetCover}}|}_{\text{covering cost}}.$$

In case $\mathrm{Alg} \leq c \cdot \text{Set} - \text{Cover}_{\text{Mult-MSSC}}$, we directly get that $|\mathrm{CoverAlg}| \leq 3c \cdot |\mathrm{OPT}_{\mathrm{SetCover}}|$. There is no polynomial-time approximation algorithm for Set−Cover with approximation ratio better than $\log m$. The latter holds even for instance of Set − Cover for which $m = \mathrm{poly}(n)$ [1] where $\mathrm{poly}(\cdot)$ is a polynomial with degree bounded by a universal constant. Since the number of elements $|U|$, in the constructed instance of Mult-MSSC is $n^2 m$, any $c \cdot \log |U|$-approximation for Mult-MSSC (for c sufficiently small) implies an approximation algorithm for Set − Cover with approximation ratio less than $\log n$. In case there exists an $c = o(r)$-approximation algorithm for Mult-MSSC for requests sequences $R_1, \ldots, R_T$ where $|R_t| \leq r$, we obtain an $o(r)$-approximation for algorithm for Set − Cover for sets with cardinality bounded by r. In the standard form of Set − Cover this is translated into the fact that each element belongs in at most r sets. ◀

Both the $O(\log^2 n)$-approximation algorithm (for requests of general cardinality) and the $O(r^2)$-approximation algorithm for r-bounded requests, that we subsequently present, are based on rounding a linear program called *Fractional Move To Front*. The latter is the linear program relaxation of *Move To Front*, a problem closely related to Multistage Min-Sum Set Cover. MTF asks for a sequence of permutations $\pi^1, \ldots, \pi^T$ such as at each round t, an element of R_t lies on the first position of π^t and $\sum_{t=1}^{T} \mathrm{d_{FR}}(\pi^t, \pi^{t-1})$ is minimized.

▶ **Definition 8.** *Given a sequence of requests $R_1, \ldots, R_T \subseteq U$ and an initial permutation of the elements π^0, consider the following linear program, called* Fractional − MTF*,*

$$\begin{aligned}
\textit{min} \quad & \sum_{t=1}^{T} \mathrm{d_{FR}}(A^t, A^{t-1}) \\
\textit{s.t} \quad & \sum_{i=1}^{n} A^t_{ei} = 1 \qquad \textit{for all } e \in U \textit{ and } t = 1, \ldots, T \\
& \sum_{e \in U} A^t_{ei} = 1 \qquad \textit{for all } i = 1, \ldots, n \textit{ and } t = 1, \ldots, T \\
& \sum_{e \in R_t} A^t_{e1} = 1 \qquad \textit{for all } t = 1, \ldots, T \\
& A^0 = \pi^0 \\
& A^t_{ei} \geq 0 \qquad \textit{for all } e \in U,\ i = 1, \ldots, n \textit{ and } t = 1, \ldots, T
\end{aligned}$$

where $\mathrm{d_{FR}}(\cdot, \cdot)$ is the FootRule distance of Definition 4.

There is an elegant argument (appeared in previous works, e.g., [17]) showing that the optimal solution of MTF is at most $4 \cdot \mathrm{OPT}_{\text{Mult-MSSC}}$. In Lemma 9 we provide the argument and establish that Fractional − MoveToFront is a 4-approximate relaxation of Mult-MSSC.

▶ **Lemma 9.** $\sum_{t=1}^{T} \mathrm{d_{FR}}(A^t, A^{t-1}) \leq 4 \cdot \mathrm{OPT}_{\text{Mult-MSSC}}$ *where $A^1, \ldots, A^t$ is the optimal solution of* Fractional − MTF*.*

Proof of Lemma 9. Let o_t the element of R_t appearing first in the permutation π^t_{Opt}. Consider the sequence of permutation $\pi^0, \pi^1, \ldots, \pi^T$ constructed by moving at each round t, the element o_t to the first position of the permutation. Notice that $\pi^0, \pi^1, \ldots, \pi^T$ is a feasible solution for both MoveToFront and Fractional $-$ MTF. The first key step towards the proof of Lemma 9 is that

$$\mathrm{d}_{\mathrm{KT}}(\pi^t, \pi^{t-1}) + \mathrm{d}_{\mathrm{KT}}(\pi^t, \pi^t_{\text{Opt}}) - \mathrm{d}_{\mathrm{KT}}(\pi^{t-1}, \pi^t_{\text{Opt}}) \leq 2 \cdot \pi^t_{\text{Opt}}(R_t)$$

To understand the above inequality, let k_t be the position of o_t in permutation π^{t-1}. Out of the $k_t - 1$ elements on the right of o_t in permutation π^{t-1}, let $Left_t$ ($Right_t$) denote the elements that are on the left (right) of o_t in permutation π^{t-1}_{Opt}. It is not hard to see that $\pi^t_{\text{Opt}}(R_t) \geq |Left_t|$, $\mathrm{d}_{\mathrm{KT}}(\pi^t, \pi^{t-1}) = |Left_t| + |Right_t|$ and $\mathrm{d}_{\mathrm{KT}}(\pi^t, \pi^t_{\text{Opt}}) - \mathrm{d}_{\mathrm{KT}}(\pi^{t-1}, \pi^t_{\text{Opt}}) = |Left_t| - |Right_t|$. Using the fact that $\mathrm{d}_{\mathrm{KT}}(\pi^t, \pi^t_{\text{Opt}}) - \mathrm{d}_{\mathrm{KT}}(\pi^{t-1}, \pi^t_{\text{Opt}}) \leq \mathrm{d}_{\mathrm{KT}}(\pi^t_{\text{Opt}}, \pi^{t-1}_{\text{Opt}})$ and the previous inequality we get,

$$\mathrm{d}_{\mathrm{KT}}(\pi^t, \pi^{t-1}) + \mathrm{d}_{\mathrm{KT}}(\pi^t, \pi^t_{\text{Opt}}) - \mathrm{d}_{\mathrm{KT}}(\pi^{t-1}, \pi^{t-1}_{\text{Opt}}) \leq 2 \cdot \pi^t_{\text{Opt}}(R_t) + \mathrm{d}_{\mathrm{KT}}(\pi^t_{\text{Opt}}, \pi^{t-1}_{\text{Opt}})$$

and by a telescopic sum we get $\sum_{t=1}^{T} \mathrm{d}_{\mathrm{KT}}(\pi^t, \pi^{t-1}) \leq 2 \cdot \mathrm{OPT}_{\text{Mult-MSSC}}$. The proof follows by the fact that $\mathrm{d}_{\mathrm{FR}}(\pi^t, \pi^{t-1}) \leq 2 \cdot \mathrm{d}_{\mathrm{KT}}(\pi^t, \pi^{t-1})$. ◀

As already mentioned, our main technical contribution is the design of *rounding schemes* converting the optimal solution, $A^1, \ldots, A^T$, of Fractional $-$ MTF into a sequence of permutations $\pi^1, \ldots, \pi^T$. This is done so as to bound the moving cost of our algorithms by the moving cost $\sum_{t=1}^{T} \mathrm{d}_{\mathrm{FR}}(A^t, A^{t-1})$. We then separately bound the covering cost, $\sum_{t=1}^{T} \pi^t(R_t)$ by showing that always an element of R_t lies on the first positions of π^t.

The main technical challenge in the design of our rounding schemes is ensure to that the moving cost of our solutions $\sum_{t=1}^{T} \mathrm{d}_{\mathrm{KT}}(\pi^t, \pi^{t-1})$ is approximately bounded by the moving cost $\sum_{t=1}^{T} \mathrm{d}_{\mathrm{FR}}(A^t, A^{t-1})$. Despite the fact that the connection between doubly stochastic matrices and permutations is quite well-studied and there are various rounding schemes converting doubly stochastic matrices to probability distributions on permutations (such as the Birkhoff–von Neumann decomposition or the schemes of [8, 25, 6, 17]), using such schemes in a *black-box* manner does not provide any kind of positive results for Mult-MSSC. For example consider the case where $A^1 = \cdots = A^T$ and thus $\sum_{t=1}^{T} \mathrm{d}_{\mathrm{FR}}(A^t, A^{t-1}) = \mathrm{d}_{\mathrm{FR}}(A^1, A^0)$. In case a randomized rounding scheme is applied *independently to each* A^t, there always exists a positive probability that $\pi^t \neq \pi^{t-1}$ and thus the moving cost will far exceed $\mathrm{d}_{\mathrm{FR}}(A^1, A^0)$ as T grows. The latter reveals the need for *coupled rounding schemes* that convert the overall sequence of matrices $A^1, \ldots, A^T$ to a sequence of permutations $\pi^1, \ldots, \pi^T$. Such a rounding scheme is presented in Algorithm 1 and constitutes the back-bone of our approximation algorithm for requests of general cardinality.

The rounding scheme described in Algorithm 1, imposes correlation between the different time-steps by simply requiring that each element e selects α_e once and for all and by breaking ties lexicographically (any consistent tie-breaking rule would also work). In Lemma 12 of Section 4, we show that no matter the sequence of doubly stochastic matrices, the rounding scheme of Algorithm 1 produces a sequence of permutations with overall moving cost at most $4 \log^2 n$ the moving cost of the matrix-sequence[1] and thus establishes that the overall moving cost of Algorithm 1 is bounded by $4 \log^2 n \cdot \mathrm{OPT}_{\text{Mult-MSSC}}$. The $\log n$ multiplication

[1] By omitting the $\log n$-multiplication step of Step 7, one could establish that the moving cost of the produced permutations is at most 4 times the moving cost of the matrix-sequence, however omitting the $\log n$ multiplication could lead in prohibitively high covering cost.

Algorithm 1 A Randomized Algorithm for Mult-MSSC.

Input: A sequence of requests $R_1, \ldots, R_T$ and an initial permutation of the elmenents π^0.
Output: A sequence of permutations $\pi^1, \ldots, \pi^T$.

1: Find the optimal solution $A^0 = \pi^0, A^1, \ldots, A^T$ for Fractional $-$ MTF.
2: **for** each element $e \in U$ **do**
3: Select α_e uniformly at random in $[0, 1]$.
4: **end for**
5: **for** $t = 1 \ldots T$ **do**
6: **for** all elements $e \in U$ **do**
7: $I_e^t := \text{argmin}_{1 \leq i \leq n} \{\log n \cdot \sum_{s=1}^{i} A_{es}^t \geq \alpha_e\}$.
8: **end for**
9: $\pi^t :=$ sort elements according to I_e^t with ties being broken lexicographically.
10: **end for**

in Step 7 serves as a *probability amplifier* ensuring that at least one element of R_t lies in the relatively first positions of π^t and permits us to approximately bound the covering cost $\sum_{t=1}^{T} \mathbb{E}[\pi^t(R_t)]$ by the covering cost of the optimal solution for Mult-MSSC, $\sum_{t=1}^{T} \pi_{\text{Opt}}^t(R_t)$.

▶ **Theorem 10.** *Algorithm 1 is a $O(\log^2 n)$-approximation algorithm for* Mult-MSSC.

Despite the fact that in Step 7 of Algorithm 1, we multiply the entries of A^t with $\log n$ the overall guarantee is $O(\log^2 n)$. At a first glance the latter seems quite strange but admits a rather natural explanation. For most of the positions i, the probability that an element e admits index $I_e^t = i$ is roughly $\log n \cdot A_{ei}^t$, but due to the fact each index $j \leq i$ is on expectation selected by $\log n$ other elements, the expected position of e in the produced permutation is roughly $\log^2 n$ times the expected value of $\text{argmin}_{1 \leq i \leq n} \{\sum_{s=1}^{i} A_{es}^t \geq \alpha_e\}$. This phenomenon relates with the elegant fitting argument given in [15] to prove that the greedy algorithm is 4-approximation for the original *Min-Sum Set Cover* (which is tight unless P = NP). The latter makes us conjecture that the tight inapproximability bound for Mult-MSSC is $\Omega(\log^2 n)$ for requests of general cardinality.

Motivated by the r-approximation LP-based algorithm for instances of Set $-$ Cover in which elements belong in at most r sets, we examine whether the $O(\log^2 n)$ for Mult-MSSC can be ameliorated in case of r-bounded request sequences. Interestingly, the simple *greedy rounding* scheme (described[2] in Algorithm 2) provides such a $O(r^2)$-approximation algorithm.

Algorithm 2 A Greedy-Rounding Algorithm for Mult-MSSC for r-Bounded Sequences.

Input: A request sequence $R_1, \ldots, R_T$ with $|R_t| \leq r$ and an initial permutation π^0.
Output: A sequence of permutations $\pi^1, \ldots, \pi^T$.

1: Find the optimal solution $A^0 = \pi^0, A^1, \ldots, A^T$ for Fractional $-$ MTF.
2: **for** $t = 1 \ldots T$ **do**
3: $\pi^t :=$ in π^{t-1}, move to the first position an element $e \in R_t$ such that $A_{e1}^t \geq 1/r$
4: **end for**

[2] Step 3 of Algorithm 2 is well-defined since $|R_t| \leq r$ and $\sum_{e \in R_t} A_{e1}^t = 1$.

The $O(r^2)$-approximation guarantee of Algorithm 3 is formally stated and proven in Theorem 11. The main technical challenge is that we cannot directly compare the moving cost of Algorithm 2 with $\sum_{t=1}^{T} \mathrm{d_{FR}}(A^t, A^{t-1})$ and thus we deploy a two-step detour.

In the first step (Lemma 19), we prove the existence of a sequence of doubly stochastic matrices $\hat{A}^0 = \pi^0, \hat{A}^1, \ldots, \hat{A}^T$ for which each $\hat{A}^t$ satisfies that **(i)** its entries of are **multiples of $1/r$**, **(ii)** $\hat{A}^t_{e_t 1} \geq 1/r$ where e_t is the element that Algorithm 2 moves to the first position at round t, and **(iii)** the sequence $\hat{A}^0 = \pi^0, \hat{A}^1, \ldots, \hat{A}^T$ admits moving cost at most $\sum_{t=1}^{T} \mathrm{d_{FR}}(A^t, A^{t-1})$. In order to establish the existence of such a sequence, we construct an appropriate linear program (see Definition 18) based on the elements that Algorithm 2 moves to the first position at each round and prove that it admits an optimal solution with values being multiples of $1/r$. To do the latter, we relate the linear program of Definition 18 with a fractional version of the k-Paging [7] problem and based on the optimal eviction policy (*evict the page appearing the furthest in the future*), we design an algorithm producing optimal solutions for the LP with values being multiple of $1/r$.

In the second step (Lemma 20), we show that for any sequence $\hat{A}^0 = \pi^0, \hat{A}^1, \ldots, \hat{A}^T$ satisfying properties **(i)** and **(ii)**, the moving cost of Algorithm 2 is at most $O(r^2) \cdot \sum_{t=1}^{T} \mathrm{d_{FR}}(\hat{A}^t, \hat{A}^{t-1})$. The latter is achieved through the use of an appropriate *potential function* based on a generalization of Kendall-Tau distance to doubly stochastic matrices with entries being multiples of $1/r$ (see Definition 23).

▶ **Theorem 11.** *Algorithm 2 is a $O(r^2)$-approximation algorithm for* Mult-MSSC.

In Section 4 and 5 we provide the basic steps and ideas in the proof of Theorem 10 and 11 respectively.

4 Proof of Theorem 10

The basic step towards the proof of Theorem 10 is Lemma 12, establishing the fact that once two doubly stochastic matrices are given as input to the randomized rounding of Algorithm 1, the expected distance of the produced permutations is approximately bounded by the distance of the respective doubly stochastic matrices.

▶ **Lemma 12.** *Let the doubly stochastic matrices A, B given as input to the rounding scheme of Algorithm 1. Then for the produced permutations π^A, π^B, $\mathbb{E}\left[\mathrm{d_{KT}}(\pi^A, \pi^B)\right] \leq 4 \log^2 n \cdot \mathrm{d_{FR}}(A, B)$.*

Before exhibiting the proof of Lemma 12 we introduce the notion of *neighboring matrices*.

▶ **Definition 13.** *(Neighboring stochastic matrices) The stochastic matrices A, B are neighboring if and only if they differ in exactly two entries lying on the same row and on consecutive columns.*

▶ **Example 14.** Let $A = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$, $B = \begin{pmatrix} 1/2 & 1/2 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$ and $C = \begin{pmatrix} 1 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix}$. The pair of matrices (A, B) and (A, C) are neighboring while (B, C) are not.

Any doubly stochastic matrix A can be converted to another doubly stochastic matrix B through an intermediate sequence of neighboring stochastic matrices all of which are *almost doubly stochastic* and their overall moving cost equals $\mathrm{d_{FR}}(A, B)$.

▷ Claim 15. Given the doubly stochastic matrices A, B, there exists a finite sequence of stochastic matrices, $A^0, \ldots, A^T$ such that

1. $A^0 = A$ and $A^T = B$.
2. A^t and A^{t-1} are neighboring.
3. the column-sum is bounded by 2, $\sum_{e \in U} A^t_{ei} \leq 2$ for all $1 \leq i \leq n$.
4. $\sum_{t=1}^{T} \mathrm{d_{FR}}(A^t, A^{t-1}) = \mathrm{d_{FR}}(A, B)$.

Proof Sketch of Claim 15. Let f^e_{ij} denotes the optimal solution of the linear program of Definition 4 defining the FootRule distance $\mathrm{d_{FR}}(A, B)$. In case $A \neq B$, there exist elements e_1, e_2 and indices $i < j$ such that $f^{e_1}_{i\ell(i)} > 0$ and $f^{e_2}_{j\ell(j)} > 0$ with $\ell(i) >= j$ and $\ell(j) <= i$.
Let $\epsilon = \min(f^{e_1}_{i\ell(i)}, f^{e_2}_{j\ell(j)})$ and consider the sequence of the $|i-j|$ matrices produced by moving ϵ amount of mass in row e_1 from column i to column j. Then consider the sequence of the $|i-j|$ matrices produced by moving ϵ amount of mass in the row e_2 from column j to column i.
In the overall sequence of $2|i-j|$ stochastic matrices, two consecutive matrices are *neighboring*. Furthermore the column-sum of the matrices does not exceed $1 + \epsilon \leq 2$ and the final matrix A' of the sequence is doubly stochastic. Moreover by the fact that $t(i) \geq j$ and $t(j) \leq i$ we get that the overall moving cost of the sequence equals $\mathrm{d_{FR}}(A, A')$ and that $\mathrm{d_{FR}}(A, B) = \mathrm{d_{FR}}(A, A') + \mathrm{d_{FR}}(A', B)$. Applying the same argument inductively, until we reach matrix B, proves Claim 15. ◁

▶ **Example 16.** Let the doubly stochastic matrices $A = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$, $B = \begin{pmatrix} 0 & 0 & 1 \\ 1/2 & 1/2 & 0 \\ 1/2 & 1/2 & 0 \end{pmatrix}$. A can be converted to B with the following sequence neighboring stochastic matrices, $\begin{pmatrix} 0 & 1 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}, \begin{pmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}, \begin{pmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ 0 & 1 & 0 \end{pmatrix}, \begin{pmatrix} 0 & 0 & 1 \\ 1/2 & 1/2 & 0 \\ 0 & 1 & 0 \end{pmatrix}, \begin{pmatrix} 0 & 0 & 1 \\ 1/2 & 1/2 & 0 \\ 1/2 & 1/2 & 0 \end{pmatrix}$. Notice that the above sequence satisfies all the 4 requirements of Claim 15.

The notion of neighboring matrices is rather helpful since Lemma 12 admits a fairly simple proof in case A, B are neighboring stochastic matrices (notice that the rounding scheme of Algorithm 1 is well-defined even for stochastic matrices). The latter is formally stated and proven in Lemma 17 and is the main technical claim of the section.

▶ **Lemma 17.** *Let π^A, π^B the permutations produced by the rounding scheme of Algorithm 1 (given as input) the stochastic matrices A, B that **i)** are neighboring **ii)** their column-sum is bounded by 2, then $\mathbb{E}[\mathrm{d_{KT}}(\pi^A, \pi^B)] \leq 4 \log^2 n \cdot \mathrm{d_{FR}}(A, B)$*

Proof of Lemma 17. Since A, B are neighboring there exists exactly two consecutive entries for which A, B differ, denoted as (e^*, i^*) and $(e^*, i^* + 1)$. Let $\epsilon := A_{e^* i^*} - B_{e^* i^*}$, by the Definition 4 of FootRule distance, we get that $\mathrm{d_{FR}}(A, B) = |\epsilon|$. Without loss of generality we consider $\epsilon > 0$ (the case $\epsilon < 0$ symmetrically follows). We also denote with O_i the set of elements $O_i := \{e \neq e^\star$ such that $I^A_e = i\}$ and with I^A_e, I^B_e the indices in Step 6 of Algorithm 1.
Since A, B are neighboring, the e-th row of A and the e-th row of B are identical for all $e \neq e^\star$. As a result, $I^A_e = I^B_e$ for all $e \neq e^\star$. Furthermore the neighboring property implies that even for e^*, $\sum_{s=1}^{i} A_{e^\star s} = \sum_{s=1}^{i} B_{e^\star s}$ for all $i \neq i^\star$ and thus $\Pr\left[I^A_{e^\star} = i \wedge I^B_{e^\star} = j\right] = 0$ for $(i, j) \neq (i^\star, i^\star + 1)$. Now notice that

$$\Pr\left[I^A_{e^\star}=i^\star, I^B_{e^\star}=i^\star+1\right] \le \Pr\left[\log n\cdot\sum_{s=1}^{i^\star} B_{e^\star s} \le \alpha_e \le \log n\cdot\sum_{s=1}^{i^\star} A_{e^\star s}\right]$$
$$\le \log n\cdot(A_{e^\star i^\star} - B_{e^\star i^\star}) = \log n\cdot\epsilon$$

Notice also that in case $I^A_{e^\star}=I^B_{e^\star}$, $\mathrm{d_{KT}}(\pi_A,\pi_B)=0$. This is due to the fact that in such a case $I^A_e=I^B_e$ for all $e\in U$ and the fact that ties are broken lexicographically. As a result,

$$\begin{aligned}\mathbb{E}\left[\mathrm{d_{KT}}(\pi_A,\pi_B)\right] &= \Pr\left[I^A_{e^\star}\neq I^B_{e^\star}\right]\cdot\mathbb{E}\left[\mathrm{d_{KT}}(\pi_A,\pi_B)|\ I^A_{e^\star}\neq I^B_{e^\star}\right]\\ &= \Pr[I^A_{e^\star}=i^\star, I^B_{e^\star}=i^\star+1]\cdot\mathbb{E}\left[\mathrm{d_{KT}}(\pi_A,\pi_B)|\ I^A_{e^\star}=i^\star, I^B_{e^\star}=i^\star+1\right]\\ &\le \epsilon\log n\cdot(\mathbb{E}\left[|O_{i^\star}|\right]+\mathbb{E}\left[|O_{i^\star+1}|\right])\end{aligned}$$

where the last inequality follows by the fact that once $I^A_{e^*}=i^*$ and $I^B_{e^*}=i^*+1$, the element e^* can move at most by $|O_{i^*}|+|O_{i^*+1}|$ positions and the fact that $I^A_{e^*}, I^B_{e^*}$ and $|O_{i^*}|, |O_{i^*+1}|$ are independent random variables.
We complete the proof we providing a bound on $\mathbb{E}\left[|O_i|\right]$. Notice that for $e\in U/\{e^*\}$,

$$\Pr[e\in O_i] \le \Pr\left[\log n\sum_{s=1}^{i-1} A_{es} \le \alpha_e \le \log n\sum_{s=1}^{i} A_{es}\right] \le \log n\cdot A_{ei}$$

which implies that $\mathbb{E}\left[|O_i|\right]\le\log n\sum_{e\neq e^\star} A_{ei}\le 2\log n$. Finally we overall get,

$$\mathbb{E}\left[\mathrm{d_{KT}}(\pi_A,\pi_B)\right] \le 4\log^2 n\cdot\mathrm{d_{FR}}(A,B)$$ ◀

The proof of Lemma 12 easily follows by Claim 15 and Lemma 17.

Proof of Lemma 12. Given the doubly stochastic matrices A, B, let the sequence $A = A^0, A^1, \ldots, A^T = B$ of neighboring stochastic matrices ensured by Claim 15. Now let $\pi^0, \pi^1, \ldots, \pi^T$ the sequence of permutations that the randomized rounding of Algorithm 1 produces given as input the sequence $A = A^0, A^1, \ldots, A^T = B$. Notice that,

$$\mathbb{E}\left[\mathrm{d_{KT}}(\pi^A,\pi^B)\right] \le \sum_{t=1}^{t}\mathbb{E}\left[\mathrm{d_{KT}}(\pi^t,\pi^{t-1})\right] \le 4\log^2 n\cdot\sum_{t=1}^{T}\mathrm{d_{FR}}(A^t,A^{t-1}) = 4\log^2 n\cdot\mathrm{d_{FR}}(A,B)$$

where the first inequality follows by the triangle inequality, the second by Lemma 17 and the last equality by Case 4 of Claim 15. ◀

We conclude the section with the proof of Theorem 10.

Proof of Theorem 10. By Lemma 12 and Lemma 9,

$$\sum_{t=1}^{T}\mathbb{E}\left[\mathrm{d_{KT}}(\pi^t,\pi^{t-1})\right] \le 4\log^2 n\cdot\sum_{t=1}^{T}\mathrm{d_{FR}}(A^t,A^{t-1}) \le 4\log^2 n\cdot\mathrm{OPT}_{\text{Mult-MSSC}}$$

Up next we bound the expected covering cost $\sum_{t=1}^{T}\mathbb{E}\left[\pi^t(R_t)\right]$. Notice that since $\sum_{e\in R_t} A^t_{e1} = 1$, the only elements that can have index $I^t_e=1$ are the elements $e\in R_t$. As a result, in case there exists some e at round t with $I^t_e=1$ then $\pi^t(R_t)=1$.

$$\begin{aligned}\mathbb{E}\left[\pi^t(R_t)\right] &\le 1+n\cdot\Pr\left[I^t_e>1 \text{ for all } e\in R_t\right]\\ &\le 1+n\cdot\Pi_{e\in R_t}\left(1-\log n\cdot A^t_{e1}\right)\\ &\le 1+n\cdot e^{-\log n\cdot\sum_{e\in R_t} A^t_{e1}} = 2\cdot\pi^t_{\mathrm{Opt}}(R_t)\end{aligned}$$

where the last inequality follows due to the fact that $\sum_{e\in R_t} A^t_{e1}=1$ and $\pi^t_{\mathrm{Opt}}(R_t)\ge 1$. ◀

5 Proof of Theorem 11

In this section we present the basic steps towards the proof of Theorem 11. We remind that $|R_t| \leq r$ and we denote with e_t the element that Algorithm 2 moves in the fist position at round t. As already mentioned, the proof is structured in two different steps.

1. We prove the existence of a sequence of doubly stochastic matrices $\hat{A}^0 = \pi^0, \hat{A}^1, \ldots, \hat{A}^T$ such that **(i)** the entries of each $\hat{A}^t$ are multiples of $1/r$, **(ii)** each $\hat{A}^t$ admits $1/r$ mass for element e_t in first position ($\hat{A}^t_{e_t 1} \geq 1/r$) and **(iii)** $\sum_{t=1}^{T} \mathrm{d_{FR}}(\hat{A}^t, \hat{A}^{t-1}) \leq \sum_{t=1}^{T} \mathrm{d_{FR}}(A^t, A^{t-1})$.
2. We use properties **(i)** and **(ii)** to prove that the moving cost of Algorithm 2 is roughly upper bounded by $\Theta(r^2) \cdot \sum_{t=1}^{T} \mathrm{d_{FR}}(\hat{A}^t, \hat{A}^{t-1})$.

We start with the construction of the sequence $\hat{A}^0 = \pi^0, \hat{A}^1, \ldots, \hat{A}^T$.

▶ **Definition 18.** *For the sequence of elements $e_1, \ldots, e_T \in U$ (the elements that Algorithm 2 moves to the fist position at each round), consider the following linear program,*

$$
\begin{aligned}
\textit{min} \quad & \sum_{t=1}^{T} \mathrm{d_{FR}}(\hat{A}^t, \hat{A}^{t-1}) & \\
\textit{s.t} \quad & \sum_{i=1}^{n} \hat{A}^t_{ei} = 1 & \textit{for all } e \in U \textit{ and } t = 1, \ldots, T \\
& \sum_{e \in U} \hat{A}^t_{ei} = 1 & \textit{for all } i = 1, \ldots, n \textit{ and } t = 1, \ldots, T \\
& \hat{A}^t_{e_t 1} \geq 1/r & \textit{for all } t = 1, \ldots, T \\
& \hat{A}^0 = \pi^0 & \\
& \hat{A}^t_{ei} \geq 0 & \textit{for all } e \in U,\ i = 1, \ldots, n \textit{ and } t = 1, \ldots, T
\end{aligned}
$$

The sequence $\hat{A}^0 = \pi^0, \ldots, \hat{A}^T$ is defined as the optimal solution of the LP in Definition 18 with the entries of each $\hat{A}^t$ being **multiples of $1/r$**. The existence of such an optimal solution is established in Lemma 19.

▶ **Lemma 19.** *There exists an optimal solution $\hat{A} = \pi^0, \hat{A}^1, \ldots, \hat{A}^T$ for the linear program of Definition 19 such that entries of each $\hat{A}^t$ are multiples of $1/r$.*

The proof of Lemma 19 is one of the main technical contributions of this work. Due to lack of space its proof is deferred to the full version of the paper. We remark that the *semi-integrality property*, that Lemma 19 states, is not due to the properties of the LP's polytope and in fact there are simple instances in which the optimal extreme points do not satisfy it. We establish Lemma 19 via the design of an optimal algorithm for the LP of Definition 18 (Algorithm 3) that always produces solutions with entries being multiples of $1/r$. Up next we describe in brief the idea behind Algorithm 3.

Given the matrix $\hat{A}^{t-1}$, Algorithm 3 construct $\hat{A}^t$ as follows. At first it moves $1/r$ mass from the left-most entry (e_t, j) with $\hat{A}^{t-1}_{e_t j} \geq 1/r$ to the entry $(e_t, 1)$. At this point the third constraint of the LP in Definition 18 is satisfied but the column-stochasticity constraints are violated (the first column admits mass $1 + 1/r$ and the j-th column admits mass $1 - 1/r$). Algorithm 3 inductively restores column-stochasticity from left to right. At step i, all the columns on the left of i are restored and the violations concern the column i and j (i's mass is $1 + 1/r$ and j's mass is $1 - 1/r$). Now Algorithm 3 must move a total of $1/r$ mass from column i to column $i + 1$. In case there exists an element e with total amount of mass greater than $2/r$, Algorithm 2 moves the $1/r$ mass from the entry (e, i) to the entry $(e, i + 1)$. The

reason is that even if $e = e_{t'}$ at some future round t', the third constraint only requires $1/r$ mass. In case there is no such element, Algorithm 3 moves the $1/r$ mass from the element appearing the furthest in the sequence $\{e_t, \ldots, e_T\}$. The latter is in accordance with the optimal eviction policy for k − Paging which at each round evicts the page appearing furthest in the future [7]. The optimality of Algorithm 3 is established in Lemma 21 and the fact that produced solution admits values being $1/r$ is inductively established.

To this end, we can show that all of the desired properties of the sequence $\hat{A} = \pi^0, \hat{A}^1, \ldots, \hat{A}^T$ are satisfied. Property **(i)** is established by Lemma 19. Property **(ii)** is enforced by the constraint $\hat{A}^t_{e_t 1} \geq 1/r$. Now for Property **(iii)**, notice that by the definition of Algorithm 2, $A^t_{e_t 1} \geq 1/r$. As a result, the sequence $A^0 = \pi^0, A^1, \ldots, A^T$ is feasible for the linear program of Definition 18 and thus $\sum_{t=1}^{T} \mathrm{d_{FR}}(\hat{A}^t, \hat{A}^{t-1}) \leq \sum_{t=1}^{T} \mathrm{d_{FR}}(A^t, A^{t-1})$.

▶ **Lemma 20.** *Let $\pi^0, \pi^1, \ldots, \pi^T$ the permutations produced by Algorithm 2 and $e_1, \ldots, e_T$ the elements that Algorithm 2 moves to the first position at each round t. For any sequence of doubly stochastic matrices $\hat{A}^0 = \pi^0, \hat{A}^1, \ldots, \hat{A}^T$ for which Property **(i)** and Property **(ii)** are satisfied, $\sum_{t=1}^{T} \mathrm{d_{KT}}(\pi^t, \pi^{t-1}) \leq 2r^2 \cdot \sum_{t=1}^{T} \mathrm{d_{FR}}(\hat{A}^t, \hat{A}^{t-1}) + r \cdot T$.*

The proof of Theorem 11 directly follows by Lemma 19 and 20. In Section 5.2 we present the basic steps for of Lemma 19.

5.1 Proof of Lemma 19

We prove the existence of an optimal solution $\hat{A}^0 = \pi^0, \hat{A}^1, \ldots, \hat{A}^T$ for the linear program of Definition 18 for which the entries of each matrix $\hat{A}^t$ are multiples of $1/r$ though the design of an optimal greedy algorithm illustrated in Algorithm 3.

The fact that Algorithm 3 produces a solution with entries that multiples of $1/r$ easily follows. Algorithm 3 starts with an integral doubly stochastic matrices ($\hat{A}^0 = \pi^0$) and always moves $1/r$ mass from entry to entry. The optimality of Algorithm 3 is established in Lemma 21 the proof of which is presented in the next section since it is quite technically complicated. However the basic idea of the algorithms is very intuitive, once $\hat{A}^{t-1}_{e_t} = 0$ Algorithm 3 moves $1/r$ mass of e_t from its leftmost position (with mass greaer than $1/r$), denoted as Pos of Step 5. At this point of time, Algorithm 3 has violated the column-stochasticity constraints, $1 + 1/r$ for the first column and $1 - 1/r$ for the Pos-th column and Algorithm 3 must move at total of $1/r$ mass from the first position to next positions until $1/r$ mass reaches the Pos position and column-stochasticity is restored (Step 8). Once Algorithm 3 detects an element with aggregated mass (until position j) $\geq 2/r$, it can safely move $1/r$ of each mass to position $j + 1$ since even if this element appears at some point in the future only $1/r$ is necessary to satisfy the constraint $A^t_{e_t 1} \geq 1/r$ and thus the rest is redundant (Step 11). In case such an element does not exist, Algorithm 3 moves the (useful) $1/r$ mass of the element appearing the furthest in the remaining sequence $\{e_t, \ldots, e_T\}$, which is exactly the same optimal *eviction policy* that the well-studied k − Paging suggests.

▶ **Lemma 21.** *Algorithm 3 produces an optimal solution $\hat{A}^0 = \pi^0, \hat{A}^1, \ldots, \hat{A}^T$ for the linear program of Definition 18 while the entries of each $\hat{A}^t$ are multiples of $1/r$.*

5.2 Proof of Lemma 20

In order to prove Lemma 20, we make use of an appropriate potential function that can be viewed as an extension of the Kendall-Tau distance (see Definition 1) to doubly stochastic matrices with entries being multiples of $1/r$.

Algorithm 3 An Optimal Greedy Algorithm for the LP of Definition 18.

Input: The initial permutation π^0 and the sequence of elements $e_1, \ldots, e_T \in U$
Output: An optimal solution of a linear program of Definition 18 where the entries of $\hat{A}^t$ are multiples of $1/r$.

Initially $\hat{A}^0 \leftarrow \pi_0$
for all rounds $t = 1$ to T **do**
  $\hat{A}^t \leftarrow \hat{A}^{t-1}$
  if $\hat{A}^t_{e_t 1} < 1/r$ **then**
    //Move $1/r$ mass of e_t to the first position
      $\text{Pos} \leftarrow \text{argmin}_{1 \le i \le n}\{A^t_{ei} \ge 1/r\}$
      $\hat{A}^t_{e1} \leftarrow \hat{A}^t_{e1} + 1/r, \hat{A}^t_{e\text{Pos}} \leftarrow \hat{A}^t_{e\text{Pos}} - 1/r$
    //Restore the column-stochasticity constraints from left to right
    for $j = 1$ to $\text{Pos} - 1$ **do**
      if there exists $e \in U$ with $\sum_{s=1}^{j} \hat{A}^t_{es} \ge 2/r$ and $\hat{A}^t_{es} \ge 1/r$ **then**
        //Move $1/r$ of its (redundant) mass to the next position
          $\hat{A}^t_{ej} \leftarrow \hat{A}^t_{ej} - 1/r$, $\hat{A}^t_{ej} \leftarrow \hat{A}^t_{ej} + 1/r$
      else
        //Move the $1/r$ mass, of the element appearing furthest in the future, to the next position
          $e^\star \in U \leftarrow$ the element with $\hat{A}^t_{e^\star j} = 1/r$ furthest in $\{e_{t+1}, \ldots, e_T\}$
          $\hat{A}^t_{e^\star j} \leftarrow \hat{A}^t_{e^\star j} - 1/r$, $\hat{A}^t_{e^\star j} \leftarrow \hat{A}^t_{e^\star j} + 1/r$
      end if
    end for
  end if
end for
return $\hat{A}_1, \ldots, \hat{A}_T$

▶ **Definition 22** (***r***-Index). *The r-index of an element $e \in U$ in the doubly stochastic matrix A, $I^A_e := \text{argmin}\{1 \le i \le n : \sum_{s=1}^{i} A_{es} \ge 1/r\}$*

▶ **Definition 23** (Fractional Kendall-Tau Distance). *Given the doubly stochastic matrices A, B, a pair of elements $(e, e') \in U \times U$ is inverted if and only if one of the following condition holds,*

1. $I^A_e > I^A_{e'}$ *and* $I^B_e < I^B_{e'}$.
2. $I^A_e < I^A_{e'}$ *and* $I^B_e > I^B_{e'}$.
3. $I^A_e = I^A_{e'}$ *and* $I^B_e \neq I^B_{e'}$.
4. $I^A_e \neq I^A_{e'}$ *and* $I^B_e = I^B_{e'}$.

The fractional Kendall-Tau distance between two doubly stochastic matrices A, B, denoted as $\text{d}_{\text{KT}}(A, B)$, is the number of inverted pairs of elements.

Notice that in case of $0-1$ doubly stochastic matrices the Fractional Kendall-Tau distance of Definition 23 coincides with the Kendall-Tau distance of Definition 1.

▷ Claim 24. Fractional Kendall-Tau Distance satisfies the triangle inequality, $\text{d}_{\text{KT}}(A, B) \le \text{d}_{\text{KT}}(A, C) + \text{d}_{\text{KT}}(C, B)$.

Proof of Claim 24. Let $X^{AB}_{ee'} = 1$ if (e, e') is inverted pair for the matrices A, B and 0 otherwise (respectively for $X^{AC}_{ee'}, X^{BC}_{ee'}$). By a short case study one can show that once $X^{AB}_{ee'} = 1$ then $X^{AC}_{ee'} + X^{BC}_{ee'} \ge 1$ which directly implies Claim 24. ◁

In the case of doubly stochastic matrices with their entries being multiples of $1/r$, Fractional Kendall-Tau distance relates to FootRule distance of Definition 4.

▶ **Lemma 25.** *Let the doubly stochastic matrices A, B with entries that are multiples of $1/r$. Then $\mathrm{d_{KT}}(A, B) \leq 2r^2 \cdot \mathrm{d_{FR}}(A, B)$.*

Proof of Lemma 25. We construct a doubly stochastic matrix A' for which the following properties hold,

1. The entries of A' are multiples of $\frac{1}{r}$.
2. $\mathrm{d_{FR}}(A, B) = \mathrm{d_{FR}}(A, A') + \mathrm{d_{FR}}(A', B)$.
3. $\mathrm{d_{KT}}(A, A') \leq 2r^2 \cdot \mathrm{d_{FR}}(A, A')$.

Once the above properties are established, Lemma 25 follows by repeating the same construction until matrix B is reached and by using the fact that the *fractional Kendall-Tau distance* of Definition 23 satisfies the triangle inequality.
Before proceeding with the construction of A', we present the following corollary that follows by an easy exchange argument.

▶ **Corollary 26.** *Let the stochastic matrices A, B with entries multiples of $1/r$, the values f_{ij}^e of the optimal solution in the linear program of Definition 4 (the min-cost transportation problem defining the FootRule distance $\mathrm{d_{FR}}(A, B)$) are multiples of $1/r$.*

In order to construct the matrix A' satisfying the Properties 1-3, we consider three different classes of the entries (e, i). In particular, we call an entry (e, i).

1. *right* if and only if $f_{ij}^e > 0$ for some $j > i$.
2. *left* if and only if $f_{ij}^e > 0$ for some $j < i$.
3. *neutral* if and only if $f_{ij}^e = 0$ for all $j \neq i$.

Note that the above classes do not form a partition of the entries since an entry (e, i) can be both *left* and *right* at the same time.

▶ **Corollary 27.** *Given two doubly stochastic matrices $A \neq B$, there exist entries (e, i) and (e', j) such that*

1. *$j > i$*
2. *the entry (e, i) is right*
3. *the entry (e', j) is left*
4. *the entry (α, ℓ) is neutral for all $\alpha \in U$ and $\ell \in \{i + 1, j - 1\}$*

We construct the matrix A' from matrix A as follows. Consider two entries (e, i) and (e', j) with the properties that Corollary 27 illustrates. The doubly stochastic matrix A' is constructed by moving $1/r$ mass from entry (e, i) to entry (e, j) and by moving $1/r$ mass from entry (e', j) to entry (e', i). More formally,

$$A'_{\alpha\ell} = \begin{cases} A_{\alpha\ell} - \frac{1}{r} & \text{if } (\alpha, \ell) = (e, i) \\ A_{\alpha\ell} - \frac{1}{r} & \text{if } (\alpha, \ell) = (e', j) \\ A_{\alpha k} + \frac{1}{r} & \text{if } (\alpha, \ell) = (e', i) \\ A_{\alpha\ell} + \frac{1}{r} & \text{if } (\alpha, \ell) = (e, j) \\ A_{\alpha\ell} & \text{otherwise} \end{cases}$$

Up next we establish the fact that $\mathrm{d_{FR}}(A, B) = \mathrm{d_{FR}}(A, A') + \mathrm{d_{FR}}(A', B)$.

▷ Claim 28. $\mathrm{d_{FR}}(A', A) = 2|j-i|/r$ and $\mathrm{d_{FR}}(A', B) = \mathrm{d_{FR}}(A, B) - 2|j-i|/r$.

Proof. The fact that $\mathrm{d_{FR}}(A', A) = 2|j-i|/r$ is trivial. We thus focus on showing that $\mathrm{d_{FR}}(A', B) = \mathrm{d_{FR}}(A, B) - 2|j-i|/r$.
Since (e, i) is *right*, there exists an index $\ell(i) > i$ such that $f^e_{i\ell(i)} > 0$. Moreover $f^e_{i\ell(i)} \geq 1/r$ since $f^e_{i\ell(i)}$ is multiple of $1/r$. Notice that $\ell(i) \neq \ell$ for $\ell \in \{i+1, j-1\}$ since all the entries (α, ℓ) are *neutral* (otherwise $\sum_{\alpha \in U} B_{\alpha\ell} > 1$). As a result, transfering $1/r$ mass from entry (e, i) to entry (e, j) decreases the FootRule distance between A and B by $1/r \cdot |i-j|$ since the *final destination* of the $1/r$ mass is the entry $(e, \ell(i))$ that is on the right of entry (e, j), $\ell(i) \geq j$. The claim follows by applying the exact same argument for (e', j). ◁

We now establish the last property that is $\mathrm{d_{KT}}(A, A') \leq 2r^2 \cdot \mathrm{d_{FR}}(A, A')$.

▷ Claim 29. $\mathrm{d_{KT}}(A', B) \leq 4r \cdot |i-j|$

Proof. Notice that apart from e, e', the r-index of each element is the same in both A and A' ($I^A_\alpha = I^{A'}_\alpha$ for all $\alpha \in U \setminus \{e, e'\}$). As a result, by Definition 23, we get that the only inverted pairs can be of the form (e, α) or (e', α).
In case $I^A_e \leq i-1$ then $I^A_e = I^{A'}_e$ and there is no inverted pair of the form (e, α). In case $I^A_e = i$ then $i \leq I^{A'}_e \leq j$ and any element α with $I^A_\alpha = I^{A'}_\alpha \in \{1, i-1\} \cup \{j+1, n\}$ cannot form an inverted pair with e. As a result, a pair (e, α) can be inverted only if $i \leq I^A_\alpha = I^{A'}_\alpha \leq j$. Since the entries of A are multiples of $1/r$ and A is doubly stochastic, there are at most r positive entries at each column of A. As a result, there are at most $r \cdot (j-i+1)$ inverted pairs of the form (e, α). With the symmetric argument one can show that there are at most $r \cdot |j-i+1|$ of the form (e', α). Overall there are at most $2r \cdot |j-i+1|$ inverted pairs between A and A' that are less than $4r \cdot |j-i|$ since $j > i$. ◁

◀

We conclude the section with Lemma 30. Then Lemma 20 follows by Lemma 30 and 25.

▶ **Lemma 30.** *Let $\pi^0, \pi^1, \ldots, \pi^T$ the permutations produced by Algorithm 2 and $e_1, \ldots, e_T$ the elements that Algorithm 2 moves to the first position at each round t. For any sequence of doubly stochastic matrices $B^0 = \pi^0, B^1, \ldots, B^T$ with $B^t_{e_t 1} \geq 1/r$,*

$$\sum_{t=1}^{T} \mathrm{d_{KT}}(\pi^t, \pi^{t-1}) \leq \sum_{t=1}^{T} \mathrm{d_{KT}}(B^t, B^{t-1}) + r \cdot T$$

The proof of Lemma 30 is based on the following two inequalities, $\mathrm{d_{KT}}(\pi^t, \pi^{t-1}) + \mathrm{d_{KT}}(\pi^t, B^t) - \mathrm{d_{KT}}(\pi^{t-1}, B^t) \leq r$ and $\mathrm{d_{KT}}(\pi^{t-1}, B^t) - \mathrm{d_{KT}}(\pi^{t-1}, B^{t-1}) \leq \mathrm{d_{KT}}(B^t, B^{t-1})$. The second inequality follows by the triangle inequality established in Claim 24. The first follows by the fact that $I^{B^t}_{e_t} = 1$ and the definition of Fractional Kendall-Tau distance.

Proof of Lemma 30. Since $B^t_{e_t} \geq 1/r$, the r-index of element e_t in matrix B^t is 1, $I^{B^t}_{e_t} = 1$. We first show that,

$$\mathrm{d_{KT}}\left(\pi^t, \pi^{t-1}\right) + \mathrm{d_{KT}}\left(\pi^t, B^t\right) - \mathrm{d_{KT}}\left(\pi^{t-1}, B^t\right) \leq r$$

To simplify notation let k_t the position of e_t in π^{t-1}. Notice that $\mathrm{d_{KT}}\left(\pi^t, \pi^{t-1}\right) = k_t - 1$. Out of the $k_t - 1$ elements lying on the left of e_t in π^{t-1} there are most $r-1$ elements α with $I^{B^t}_\alpha = 1$ (these elements must admit $B^t_{\alpha 1} \geq 1/r$). The rest of the $k_t - 1$ elements admit r-index $I^{B^t}_\alpha \geq 2$ and thus form inverted pairs with e_t when considering π^{t-1} and B^t.

When e_t moves to the first positions (permutation π^t) these inverted pairs are deactivated ($I_{e_t}^{B^t} = 1$) and new inverted pairs are created between e_t and α with $I_\alpha^{B^t} = 1$, but these new inverted pairs are at most r (for any element α with $I_\alpha^{B^t}$, $B_\alpha^t \geq 1/r$). Also notice no additional inverted pairs (e, α) (with $e \neq e_t$) are created since the order between all the other elements is the same in π^t and π^{t-1}. Overall,

$$\underbrace{\mathrm{d_{KT}}\left(\pi^t, \pi^{t-1}\right)}_{k_t - 1} + \underbrace{\mathrm{d_{KT}}\left(\pi^t, B^t\right) - \mathrm{d_{KT}}\left(\pi^{t-1}, B^t\right)}_{\leq -k_t + 1 + r} \leq r$$

Combining the above inequality with $\mathrm{d_{KT}}\left(\pi^{t-1}, B^t\right) - \mathrm{d_{KT}}\left(\pi^{t-1}, B^{t-1}\right) \leq \mathrm{d_{KT}}\left(B^t, B^{t-1}\right)$ which follows from the triangle inequality we get,

$$\mathrm{d_{KT}}\left(\pi^t, \pi^{t-1}\right) + \mathrm{d_{KT}}\left(\pi^t, B^t\right) - \mathrm{d_{KT}}\left(\pi^{t-1}, B^{t-1}\right) \leq \mathrm{d_{KT}}\left(A^t, B^{t-1}\right) + r.$$

Finally a telescopic sum gives $\sum_{t=1}^{T} \mathrm{d_{KT}}\left(\pi^t, \pi^{t-1}\right) \leq \sum_{t=1}^{T} \mathrm{d_{KT}}\left(B^t, B^{t-1}\right) + r \cdot T + \mathrm{d_{KT}}(\pi^0, B^0) - \mathrm{d_{KT}}(\pi^T, B^T)$ where $\mathrm{d_{KT}}(\pi^0, B^0) = 0$. ◀

6 Concluding Remarks

In this work we examine the polynomial-time approximability of Multistage Min-Sum Set Cover. We present $\Omega(\log n)$ and $\Omega(r)$ inapproximability results for general and r-bounded request sequences, while we respectively provide $O(\log^2 n)$ and $O(r^2)$ polynomial-time approximation algorithms. Closing this gap is an interesting question that our work leaves open. Another interesting research direction concerns the competitive ratio in the online version of Dynamic Min-Sum Set Cover. [18] provides an $\Omega(r)$ lower bound and a $\Theta\left(r^{3/2}\sqrt{n}\right)$-competitive online algorithm for r-bounded sequences. Designing online algorithms for a relaxation of the problem (such as the Fractional $-$ MTF) and using the rounding schemes that this work suggests may be a fruitful approach towards closing this gap.

References

1 Noga Alon, Dana Moshkovitz, and Shmuel Safra. Algorithmic construction of sets for k-restrictions. *ACM Transactions on Algorithms (TALG)*, 2(2):153–177, 2006.

2 Christoph Ambühl. Offline list update is NP-hard. In *Algorithms - ESA 2000, 8th Annual European Symposium, Proceedings*, volume 1879 of *Lecture Notes in Computer Science*, pages 42–51. Springer, 2000.

3 Hyung-Chan An, Ashkan Norouzi-Fard, and Ola Svensson. Dynamic facility location via exponential clocks. *ACM Trans. Algorithms*, 13(2):21:1–21:20, 2017.

4 Yossi Azar and Iftah Gamzu. Ranking with submodular valuations. In *SODA*, pages 1070–1079, 2011. doi:10.1137/1.9781611973082.81.

5 Yossi Azar, Iftah Gamzu, and Xiaoxin Yin. Multiple intents re-ranking. In *STOC*, pages 669–678, 2009. doi:10.1145/1536414.1536505.

6 Nikhil Bansal, Jatin Batra, Majid Farhadi, and Prasad Tetali. Improved approximations for min sum vertex cover and generalized min sum set cover. In *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms, SODA 2021*, pages 998–1005. SIAM, 2021.

7 Nikhil Bansal, Niv Buchbinder, and Joseph Naor. A primal-dual randomized algorithm for weighted paging. *J. ACM*, 59(4):19:1–19:24, 2012. doi:10.1145/2339123.2339126.

8 Nikhil Bansal, Anupam Gupta, and Ravishankar Krishnaswamy. A constant factor approximation algorithm for generalized min-sum set cover. In *SODA*, pages 1539–1545, 2010. doi:10.1137/1.9781611973075.125.

9 Amotz Bar-Noy, Mihir Bellare, Magnús M. Halldórsson, Hadas Shachnai, and Tami Tamir. On chromatic sums and distributed resource allocation. *Inf. Comput.*, 140(2):183–202, 1998. doi:10.1006/inco.1997.2677.

10 Omer Ben-Porat and Moshe Tennenholtz. A game-theoretic approach to recommendation systems with strategic content providers. In *Annual Conference on Neural Information Processing Systems 2018*, NeurIPS 2018, 2018.

11 Guillaume Cabanac and Thomas Preuss. Capitalizing on order effects in the bids of peer-reviewed conferences to secure reviews by expert referees. *J. Am. Soc. Inf. Sci. Technol.*, 64(2):405–415, 2013. doi:10.1002/asi.22747.

12 Mahsa Derakhshan, Negin Golrezaei, Vahideh Manshadi, and Vahab Mirrokni. Product ranking on online platforms. In *Proc. of the 21st ACM Conference on Economics and Computation (EC 2015)*. ACM, 2020. URL: https://ssrn.com/abstract=3130378.

13 Cynthia Dwork, Ravi Kumar, Moni Naor, and D. Sivakumar. Rank aggregation methods for the web. In *Proceedings of the 10th International Conference on World Wide Web*, WWW '01, page 613–622, New York, NY, USA, 2001. Association for Computing Machinery.

14 David Eisenstat, Claire Mathieu, and Nicolas Schabanel. Facility location in evolving metrics. In *Automata, Languages, and Programming - 41st International Colloquium, ICALP 2014, Proceedings, Part II*, volume 8573 of *Lecture Notes in Computer Science*, pages 459–470. Springer, 2014.

15 Uriel Feige, László Lovász, and Prasad Tetali. Approximating min sum set cover. *Algorithmica*, 40(4):219–234, 2004. doi:10.1007/s00453-004-1110-5.

16 Tanner Fiez, Nihar Shah, and Lillian Ratliff. A super* algorithm to determine orderings of items to show users. In *Conference on Uncertainty in Artificial Intelligence*, 2020.

17 Dimitris Fotakis, Loukas Kavouras, Grigorios Koumoutsos, Stratis Skoulakis, and Manolis Vardas. The online min-sum set cover problem. In *Proc. of the 47th International Colloquium on Automata, Languages and Programming (ICALP 2020)*, LIPIcs, 2020.

18 Dimitris Fotakis, Thanasis Lianeas, Georgios Piliouras, and Stratis Skoulakis. Efficient online learning of optimal rankings: Dimensionality reduction via gradient descent. In *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020*, 2020.

19 Anupam Gupta, Kunal Talwar, and Udi Wieder. Changing bases: Multistage optimization for matroids and matchings. In *Automata, Languages, and Programming - 41st International Colloquium, ICALP 2014, Proceedings, Part I*, volume 8572 of *Lecture Notes in Computer Science*, pages 563–575. Springer, 2014.

20 Refael Hassin and Asaf Levin. An approximation algorithm for the minimum latency set cover problem. In *ESA*, pages 726–733, 2005. doi:10.1007/11561071_64.

21 Sungjin Im. Min-sum set cover and its generalizations. In *Encyclopedia of Algorithms*, pages 1331–1334. Springer, 2016.

22 Sungjin Im, Viswanath Nagarajan, and Ruben van der Zwaan. Minimum latency submodular cover. *ACM Trans. Algorithms*, 13(1):13:1–13:28, 2016. doi:10.1145/2987751.

23 Sungjin Im, Maxim Sviridenko, and Ruben van der Zwaan. Preemptive and non-preemptive generalized min sum set cover. *Math. Program.*, 145(1-2):377–401, 2014. doi:10.1007/s10107-013-0651-2.

24 Alejandro López-Ortiz, Marc P. Renault, and Adi Rosén. Paid exchanges are worth the price. *Theoretical Computer Science*, 824-825:1–10, 2020.

25 Martin Skutella and David P. Williamson. A note on the generalized min-sum set cover problem. *Oper. Res. Lett.*, 39(6):433–436, 2011. doi:10.1016/j.orl.2011.08.002.

26 Daniel Dominic Sleator and Robert Endre Tarjan. Self-adjusting binary search trees. *J. ACM*, 32(3):652–686, 1985. doi:10.1145/3828.3835.

27 Matthew J. Streeter, Daniel Golovin, and Andreas Krause. Online learning of assignments. In *Advances in Neural Information Processing Systems 22: 23rd Annual Conference on Neural Information Processing Systems 2009*, pages 1794–1802. Curran Associates, Inc., 2009.

28 Erez Timnat. The list update problem, 2016. Master Thesis, Technion- Israel Institute of Technology.

A Spectral Independence View on Hard Spheres via Block Dynamics

Tobias Friedrich
Hasso Plattner Institute, University of Potsdam, Germany

Andreas Göbel
Hasso Plattner Institute, University of Potsdam, Germany

Martin S. Krejca
Sorbonne University, CNRS, LIP6, Paris, France

Marcus Pappik
Hasso Plattner Institute, University of Potsdam, Germany

Abstract

The hard-sphere model is one of the most extensively studied models in statistical physics. It describes the continuous distribution of spherical particles, governed by hard-core interactions. An important quantity of this model is the normalizing factor of this distribution, called the *partition function*. We propose a Markov chain Monte Carlo algorithm for approximating the grand-canonical partition function of the hard-sphere model in d dimensions. Up to a fugacity of $\lambda < \mathrm{e}/2^d$, the runtime of our algorithm is polynomial in the volume of the system. This covers the entire known real-valued regime for the uniqueness of the Gibbs measure.

Key to our approach is to define a discretization that closely approximates the partition function of the continuous model. This results in a discrete hard-core instance that is exponential in the size of the initial hard-sphere model. Our approximation bound follows directly from the correlation decay threshold of an infinite regular tree with degree equal to the maximum degree of our discretization. To cope with the exponential blow-up of the discrete instance we use clique dynamics, a Markov chain that was recently introduced in the setting of abstract polymer models. We prove rapid mixing of clique dynamics up to the tree threshold of the univariate hard-core model. This is achieved by relating clique dynamics to block dynamics and adapting the spectral expansion method, which was recently used to bound the mixing time of Glauber dynamics within the same parameter regime.

2012 ACM Subject Classification Theory of computation → Random walks and Markov chains

Keywords and phrases Hard-sphere Model, Markov Chain, Partition Function, Gibbs Distribution, Approximate Counting, Spectral Independence

Digital Object Identifier 10.4230/LIPIcs.ICALP.2021.66

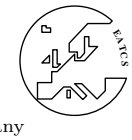
Category Track A: Algorithms, Complexity and Games

Related Version *Full Version*: `https://arxiv.org/abs/2102.07443`

Funding *Martin S. Krejca*: This work was supported by the Paris Île-de-France Region.

1 Introduction

Statistical physics models particle systems as probability distributions. One of the most fundamental and mathematically challenging models in this area is the hard-sphere model, which plays a central role in understanding the thermodynamic properties of monoatomic gases and liquids [7, 29]. It is a continuous model that studies the distribution and macroscopic behavior of indistinguishable spherical particles, assuming only hard-core interactions, i.e., no two particles can occupy the same space.



We focus on computational properties of the *grand-canonical ensemble* of the hard-sphere model in a finite d-dimensional cubic region $\mathbb{V} = [0, \ell)^d$ in the Euclidean space. In the grand-canonical ensemble, the system can exchange particles with its surrounding based on a fugacity parameter λ, which is inverse to the temperature of the system. For the rest of the paper, we make the common assumption that the system is normalized such that the particles have unit volume. That means we fix their radii to $r = (1/v_d)^{1/d}$, where v_d is the volume of a unit sphere in d dimensions.

A simple probabilistic interpretation of the distribution of particles in the grand-canonical ensemble is that centers of points that are drawn from a Poisson point process on $\mathbb{V}$ with intensity λ, conditioned on the event that no two particles overlap (i.e., every pair of centers has distance at least $2r$). The resulting distribution over particle configurations in $\mathbb{V}$ is called the *Gibbs distribution* of the model. An important quantity in such models is the so called partition function $Z(\mathbb{V}, \lambda)$, which can be seen as the normalizing constant of the Gibbs distribution. Formally, it is defined as

$$Z(\mathbb{V}, \lambda) = 1 + \sum_{k \in \mathbb{N}_{>0}} \frac{\lambda^k}{k!} \int_{\mathbb{V}^k} D\Big(x^{(1)}, \dots, x^{(k)}\Big) \, \mathrm{d}\nu^{d \times k},$$

where

$$D\Big(x^{(1)}, \dots, x^{(k)}\Big) = \begin{cases} 1 \text{ if } d\big(x^{(i)}, x^{(j)}\big) \geq 2r \text{ for all } i, j \in [k] \text{ with } i \neq j \\ 0 \text{ otherwise} \end{cases}$$

and $\nu^{d \times k}$ is the Lebesgue measure on $\mathbb{R}^{d \times k}$. Commonly, two computational task are associated with the grand-canonical hard-sphere model: (1) approximating its partition function $Z(\mathbb{V}, \lambda)$, and (2) approximately sampling from the Gibbs distribution.

Studying computational aspects of the hard-sphere model carries a historical weight, as in the seminal work of Metropolis [41], the Monte Carlo method was introduced to investigate a two-dimensional hard-sphere model. Approximate-sampling Markov chain approaches have been mainly focused on the canonical ensemble of the model, that is, the system does not exchange particles with its surrounding and thus, the distribution is defined over a fixed number of spheres [31, 36, 34]. Considering the grand canonical ensemble, exact sampling algorithms have appeared in the literature for the two-dimensional model without asymptotic runtime guarantees [37, 38, 43]. A result that is more aligned with theoretical computer science was given in [28], where the authors introduced an exact sampling algorithm for the grand-canonical hard-sphere model in d-dimensions. Their algorithm is based on partial rejection sampling with a runtime linear in the volume of the system $|\mathbb{V}|$ when assuming a continuous computational model and access to a sampler from a continuous Poisson point process. Their approach is guaranteed to apply for $\lambda < 2^{-(d+1/2)}$.

Besides such sampling results, there is an ongoing effort to improve the known fugacity regime where the Gibbs measure is unique and correlations decay exponentially fast [22, 14, 32, 42]. Note that for many discrete spin systems, such as the hard-core model, correlation decay is closely related to the applicability of different methods for efficient approximation of the partition function [50, 24, 54]. Recently, the correlation decay bounds for the hard-sphere model were improved in [32] to $\lambda < 2/2^d$, using probabilistic arguments, and in [42] to $\lambda < \mathrm{e}/2^d$, based on an analytic approach. A common feature of [32] and [42] is that they translated tools originally developed in theoretical computer science for investigating the discrete hard-core model to the continuous domain.

Our work is in line with the computational view on the hard-sphere model but more algorithmic in nature. We investigate the range of the fugacity λ for which an approximation of $Z(\mathbb{V}, \lambda)$ can be obtained efficiently in terms of the volume of the system $|\mathbb{V}|$, assuming a discrete computational model. Our main result is that for all $\lambda < \mathrm{e}/2^d$ there is a randomized algorithm for ε-approximating the partition function in time polynomial in $|\mathbb{V}|$ and $1/\varepsilon$.

▶ **Theorem 1.** *Let $(\mathbb{V}, \lambda)$ be an instance of the continuous hard-sphere model with $\mathbb{V} = [0, \ell)^d$. If there is a $\delta \in (0, 1]$ such that*

$$\lambda \leq (1 - \delta)\frac{\mathrm{e}}{2^d},$$

then for each $\varepsilon \in (0, 1]$ there is a randomized ε-approximation of $Z(\mathbb{V}, \lambda)$ computable in time polynomial in $|\mathbb{V}|^{1/\delta^2}$ and $\frac{1}{\varepsilon}$.

Note that this bound on λ precisely coincides with the best known bound for the uniqueness of the Gibbs measure in the thermodynamic limit, recently established in [42]. For many discrete spin systems, such as the hard-core model or general anti-ferromagnetic 2-state spin systems, the region of efficient approximation of the partition function is closely related to uniqueness of the Gibbs measure. More precisely, it can be shown that the partition function of every graph of maximum degree Δ can be approximated efficiently if the corresponding Gibbs distribution on an infinite Δ regular tree is unique [39, 53]. A detailed discussion for the discrete hard-core model can be found in the next subsection. In a sense, Theorem 1 can be seen as the algorithmic counterpart of the recent uniqueness result for the continuous hard-sphere model. This answers an open question, asked in [42].

The way we prove our result is quite contrary to [32] and [42]. Instead of translating discrete tools from computer science into the continuous domain, we rather discretize the hard-sphere model. By this, existing algorithmic and probabilistic techniques for discrete models become available, and we avoid continuous analysis.

Our applied discretization scheme is fairly intuitive and results in an instance of the discrete hard-core model. This model has been extensively studied in the computer science community. However, as this hard-core instance is exponential in the size of the continuous system $|\mathbb{V}|$, existing approaches for approximating its partition function, such a Markov chain Monte Carlo methods based on Glauber dynamics, are not feasible. We overcome this problem by applying a Markov chain Monte Carlo approach based on clique dynamics, which were introduced in [23] in the setting of abstract polymer models. Previously known conditions for the rapid mixing of clique dynamics were developed for the multivariate version of the hard-core model. Due to this generality, those conditions do not result in the desired bound in our univariate setting. Instead we relate those clique dynamics to another Markov chain, called *block dynamics*. We then prove the desired mixing time for the block dynamics by adapting a recently introduced technique for bounding the mixing time of Markov chains, based on local spectral expansion [2]. Together with a known self-reducibility scheme for clique dynamics, this results in the desired approximation algorithm.

Note that we aim for a rigorous algorithmic result for approximating the partition function of the continuous hard-sphere model. To be in line with commonly used discrete computational models, our Markov chain Monte Carlo algorithm does not assume access to a continuous sampler but instead samples approximately from a discretized version of the Gibbs distribution. Note that sampling from the continuous hard-sphere partition function cannot be done using a discrete computation model as this would involve infinite float pointer precision. For practical matters, our discretization of the Gibbs distribution can be seen as an approximation of the original continuous Gibbs measure. However, a rigorous comparison between both distributions based on total variation distance is not applicable, due to the fact that one is discrete whereas the other is continuous in nature.

Assuming access to a continuous sampler, we believe that our approach can be used to obtain an approximation of the Gibbs distribution of the continuous model within the same fugacity regime, by adding small perturbations to the drawn sphere centers. This would be in line with the relation between the mixing time of continuous heat-bath dynamics and strong spatial mixing, pointed out in [32], combined with the uniqueness bound from [42].

In Sections 1.1–1.3 we discuss our contributions in more detail and explain how they relate to the existing literature. Finally in Section 1.4 we discuss possible extensions and future work. All technical details, statements and proofs are presented the full version of the paper.

1.1 Discretization and hard-core model

Our discretization scheme expresses the hard-sphere partition function as the partition function of an instance of the (univariate) hard-core model. An instance of the hard-core model is a tuple (G, λ) where $G = (V, E)$ is an undirected graph and $\lambda \in \mathbb{R}_{>0}$. Its partition function is defined as

$$Z(G, \lambda) \coloneqq \sum_{I \in \mathcal{I}(G)} \lambda^{|I|},$$

where $\mathcal{I}(G)$ denotes the independent sets of G. A common way to obtain an approximation for the partition function is by applying a Markov chain Monte Carlo algorithm. This involves sampling from the Gibbs distribution $\mu^{(G,\lambda)}$ of (G, λ), which is a probability distribution on $\mathcal{I}(G)$ that assigns each independent set $I \in \mathcal{I}(G)$ the probability

$$\mu^{(G,\lambda)}(I) = \frac{\lambda^{|I|}}{Z(G, \lambda)}.$$

Conditions for efficient approximation of the hard-core partition function have been studied extensively in the theoretical computer science community. Due to hardness results in [50] and [24], it is known that for general graphs of maximum degree $\Delta \in \{3\} \cup \mathbb{N}_{>5}$ there is a critical parameter value $\lambda_c(\Delta) = (\Delta - 1)^{\Delta-1}/(\Delta - 2)^{\Delta}$, such that there is no FPRAS for the partition function of (G, λ) for $\lambda > \lambda_c(\Delta)$, unless RP = NP. On the other hand, in [54] it was proven that there is a deterministic algorithm for approximating the partition function of (G, λ) for $\lambda < \lambda_c(\Delta)$ that runs in time $|V|^{O(\Delta)}$. The critical value $\lambda_c(\Delta)$ is especially interesting, as it precisely coincides with the upper bound on λ for which the hard-core model on an infinite Δ-regular tree exhibits strong spatial mixing and a unique Gibbs distribution [54]. For this reason, it is also referred to as the *tree threshold*. This relation between computational hardness and phase transition in statistical physics is one of the most celebrated results in the area. Both, the hardness results [25, 3] and the approximation algorithms [46, 30] were later generalized for complex λ.

Note that the computational hardness above the tree threshold $\lambda_c(\Delta)$ for general graphs of maximum degree Δ applies to both, randomized and deterministic algorithms. However, in the randomized setting, Markov chain Monte Carlo methods are known to improve the runtime of the algorithm introduced in [54]. Those approaches use the vertex-wise self-reducibility of the hard-core model to construct a randomized approximation of the partition function based on an approximate sampler for the Gibbs distribution. Commonly, a Markov chain on the state space $\mathcal{I}(G)$, called *Glauber dynamics*, is used to construct the sampling scheme. At each step, a vertex $v \in V$ is chosen uniformly at random. With probability $\lambda/(1 + \lambda)$ the chain tries to add v to the current independent set and otherwise it tries to remove it. The resulting Markov chain is ergodic and reversible with respect

to the Gibbs distribution, meaning that it eventually converges to $\mu^{(G,\lambda)}$. A sequence of results has shown that for all $\Delta \geq 3$ there is a family of graphs with maximum degree Δ, such that the Glauber dynamics are torpidly mixing for $\lambda > \lambda_c(\Delta)$, even without additional complexity-theoretical assumptions [17, 27, 45]. Whether the Glauber dynamics are rapidly mixing for the entire regime $\lambda < \lambda_c(\Delta)$ remained a long-standing open problem, until recently the picture was completed [2]. By relating spectral expansion properties of certain random walks on simplicial complexes to the Glauber dynamics, it was shown that the mixing time is polynomial in $|V|$ below the tree threshold. The mixing time was recently further improved in [12] for a broader class of spin systems by combining simplicial complexes with entropy factorization and using the modified log-Sobolev inequality.

By mapping the hard-sphere model to an instance of the hard-core model we can make use of the existing results about approximation and sampling below the tree threshold. Roughly, our discretization scheme restricts the positions of sphere centers to an integer grid, while scaling the radii of spheres and the fugacity appropriately. For a hard-sphere instance $(\mathbb{V}, \lambda)$ with $\mathbb{V} = [0, \ell)^d$ the hard-core representation for resolution $\rho \in \mathbb{R}_{\geq 1}$ is a hard-core instance (G_ρ, λ_ρ) with $G_\rho = (V_\rho, E_\rho)$. Each vertex $v \in V_\rho$ represents a grid point in the finite integer lattice of side length $\rho\ell$. Two distinct vertices in V_ρ are connected by an edge in E_ρ if the Euclidean distance of the corresponding grid points is less than $2\rho r$. Furthermore, we set $\lambda_\rho = \lambda/\rho^d$. We provide the following result on the rate of convergence of $Z(G_\rho, \lambda_\rho)$ to the hard-sphere partition function $Z(\mathbb{V}, \lambda)$ in terms of ρ.

▶ **Lemma 2.** *Let $(\mathbb{V}, \lambda)$ be an instance of the continuous hard-sphere model in d dimensions. For each resolution $\rho \geq 2\sqrt{d}$ it holds that*

$$1 - \rho^{-1}\mathrm{e}^{\Theta(|\mathbb{V}|\ln|\mathbb{V}|)} \leq \frac{Z(\mathbb{V}, \lambda)}{Z(G_\rho, \lambda_\rho)} \leq 1 + \rho^{-1}\mathrm{e}^{\Theta(|\mathbb{V}|\ln|\mathbb{V}|)}.$$

Although proving this rate of convergence involves some detailed geometric arguments, there is an intuitive reason why the partition functions converge eventually as $\rho \to \infty$. Increasing the resolution ρ also linearly increases the side length of the grid and the minimum distance that sphere centers can have. This is equivalent to putting a grid into $\mathbb{V}$ with increasing granularity but fixing the radii of spheres instead. However, only scaling the granularity of this grid increases the number of possible configurations by roughly ρ^d, which would cause the partition function of the hard-core model to diverge as $\rho \to \infty$. To compensate for this, we scale the weight of each vertex in the hard-core model by the inverse of this factor.

Using this discretization approach, the fugacity bound from Theorem 1 results from simply considering Δ_ρ, the maximum degree of G_ρ and comparing λ_ρ with the corresponding tree threshold $\lambda_c(\Delta_\rho)$. Recall that we assume $r = (1/v_d)^{1/d}$. A simple geometric argument shows that Δ_ρ is roughly upper bounded by $2^d\rho^d$ for sufficiently large ρ. Now, observe that

$$\lambda_\rho = \frac{\lambda}{\rho^d} < \lambda_c\big(2^d\rho^d\big),$$

for $\lambda < \rho^d\lambda_c\big(2^d\rho^d\big)$. This follows from the fact that $\rho^d\lambda_c\big(2^d\rho^d\big)$ converges to $\mathrm{e}/2^d$ from above for $\rho \to \infty$. Thus, the approximation bound from Theorem 1 and the uniqueness bound in [42] coincide with the regime of λ, for which λ_ρ is below the tree threshold $\lambda_c(\Delta_\rho)$ in the limit $\rho \to \infty$.

The arguments above show that for a sufficiently high resolution ρ the partition function of the hard-sphere model $Z(\mathbb{V}, \lambda)$ is well approximated by the partition function of our discretization (G_ρ, λ_ρ) and that (G_ρ, λ_ρ) is below the tree threshold for $\lambda < \mathrm{e}/2^d$. However,

this does not immediately imply an approximation algorithm within the desired runtime bounds. Based on Lemma 2, we still need to choose ρ exponentially large in the volume $|\mathbb{V}|$. Note that the number of vertices in G_ρ is roughly $|V_\rho| \in \Theta\big(\rho^d|\mathbb{V}|\big)$. Even without explicitly constructing the graph, this causes problems, as the best bound for the mixing time of the Glauber dynamics is polynomial in $|V_\rho|$ and thus exponential in $|\mathbb{V}|$. Intuitively, the reason for this mixing time is that the Glauber dynamics only change one vertex at each step. Assuming that each vertex should be updated at least once to remove correlations with the initial state, any mixing time that is sublinear in the number of vertices is unlikely. We circumvent this problem by applying dynamics that update multiple vertices at each step but still allow each step to be computed efficiently without constructing the graph explicitly.

1.2 Block and clique dynamics

Most of the results that we discuss from now on apply to the multivariate version of the hard-core model, that is, each vertex $v \in V$ has its own weight λ_v. For a given graph $G = (V, E)$ we denote the set of such vertex weights by $\boldsymbol{\lambda} = \{\lambda_v\}_{v\in V}$ and write $(G, \boldsymbol{\lambda})$ for the resulting multivariate hard-core instance. In the multivariate setting, the contribution of an independent set $I \in \mathcal{I}(G)$ to the partition function is defined as the product of its vertex weights (i.e., $\prod_{v\in I} \lambda_v$), where the contribution of the empty set is fixed to 1. Similar to the univariate hard-core model, the Gibbs distribution assigns a probability to each independent set proportionally to its contribution to the partition function.

As we discussed before, the main problem with approximating the partition function of our discretization (G_ρ, λ_ρ) is that the required graph G_ρ is exponential in the volume of the original continuous system $|\mathbb{V}|$. As the Glauber dynamics Markov chain only updates a single vertex at each step, the resulting mixing time is usually polynomial in the size of the graph, which is not feasible in our case. Various extensions to Glauber dynamics for updating multiple vertices in each step have been proposed in the literature, two of which we discuss in the following.

Clique dynamics

Recently, in [23] a Markov chain, called *clique dynamics*, was introduced in order to efficiently sample from the Gibbs distribution of abstract polymer models. Note that this is similar to our algorithmic problem, as abstract polymer models resemble multivariate hard-core instances. For a given graph $G = (V, E)$, we call a set $\Lambda = \{\Lambda_i\}_{i\in[m]} \subseteq 2^V$ a clique cover of size m if and only if its union covers all vertices V and each $\Lambda_i \in \Lambda$ induces a clique in G. For an instance of the multivariate hard-core model $(G, \boldsymbol{\lambda})$ and a given clique cover Λ of G with size m the clique dynamics Markov chain $\mathcal{C}(G, \boldsymbol{\lambda}, \Lambda)$ is defined as follows. First, a clique $\Lambda_i \in \Lambda$ for $i \in [m]$ is chosen uniformly at random. Let us write $G[\Lambda_i]$ for the subgraph, induced by Λ_i, and $\boldsymbol{\lambda}[\Lambda_i] = \{\lambda_v\}_{v\in\Lambda_i}$ for the corresponding set of vertex weights. Next, an independent set from $\mathcal{I}(G[\Lambda_i])$ is chosen according to the Gibbs distribution $\mu^{(G[\Lambda_i],\boldsymbol{\lambda}[\Lambda_i])}$. Note that, as the vertices Λ_i form a clique, such an independent set is either the empty set or contains a single vertex from $v \in \Lambda_i$. If the empty set is drawn, all vertices from Λ_i are removed from the current independent set. Otherwise, if a single vertex $v \in \Lambda_i$ is drawn, the chain tries to add v to the current independent set.

Using a coupling argument, it was proven in [23] that the so-called *clique dynamics condition* implies that for any clique cover of size m the clique dynamics are mixing in time polynomial in m and $Z_{\max}$, where $Z_{\max} = \max_{i\in[m]}\{Z(G[\Lambda_i], \boldsymbol{\lambda}[\Lambda_i])\}$ denotes the maximum partition function of a clique in Λ. This is important for the application to polymer models, as

they are usually used to model partition functions of other spin systems, which often results in a multivariate hard-core model of exponential size [33, 9, 35, 10, 6, 8, 26]. As discussed in [23], those instances tend to have polynomial size clique covers that arise naturally from the original spin system. In such cases, the mixing time of clique dynamics is still polynomial in the size of original spin system, as long as the clique dynamics condition is satisfied.

This is very similar to our discretization (G_ρ, λ_ρ). To see this, set $a = \frac{2\rho}{\sqrt{d}}r$ and divide the d-dimensional integer lattice of side length $\rho\ell$ into cubic regions of side length a. Every pair of integer points within such a cubic region has Euclidean distance less than $2\rho r$, meaning that the corresponding vertices in G_ρ are adjacent. Thus, each such cubic region forms a clique, resulting in a clique cover of size $(\rho\ell/a)^d \in \mathrm{O}(|\mathbb{V}|)$. This means, there is always a clique cover with size linear in $|\mathbb{V}|$ and independent of the resolution ρ. By showing that, for the univariate hard-core model, the mixing time of clique dynamics is polynomial in the size of the clique cover for all $\lambda_\rho < \lambda_c(\Delta_\rho)$, we obtain a Markov chain with mixing time polynomial in $|\mathbb{V}|$ independent of the resolution ρ. Unfortunately, the clique dynamics condition does not hold for the entire regime up to the tree threshold in the univariate hard-core model. We overcome this problem by proving a new condition for rapid mixing of clique dynamics based on a comparison with block dynamics.

Block dynamics

Block dynamics are a very natural generalization of Glauber dynamics to arbitrary sets of vertices. For a given graph $G = (V, E)$, we call a set $\Lambda = \{\Lambda_i\}_{i\in[m]} \subseteq 2^V$ a block cover of size m if and only if its union covers all vertices V. We refer to the elements of Λ as blocks. Note that the clique cover discussed before is a special case of a block cover, where all blocks are restricted to be cliques. At each step, the block dynamics Markov chain $\mathcal{B}(G, \lambda, \Lambda)$ chooses a block $\Lambda_i \in \Lambda$ uniformly at random. Then, the current independent set is updated on Λ_i based on the projection of the Gibbs distribution onto Λ_i and conditioned on the current independent set outside Λ_i.

In fact, block dynamics are defined for a much more general class of spin systems than the hard-core model. However, due to the fact that each step of the Markov chain involves sampling from a conditional Gibbs distribution, block dynamics are rarely used as an algorithmic tool on its own. Instead, they are usually used to deduce rapid mixing of other dynamics.

For spin systems on lattice graphs, close connections between the mixing time of block dynamics and Glauber dynamics are known [40]. Such connections were for example applied to improve the mixing time of Glauber dynamics of the Monomer Dimer model on torus graphs [51]. Moreover, block dynamics were used to improve conditions for rapid mixing of Glauber dynamics on specific graph classes, such as proper colorings [16, 18, 19, 44] or the hard-core model [18, 44] in sparse random graphs. A very general result for the mixing time of block dynamics was achieved in [4], who proved that for all spin systems on a finite subgraph of the d-dimensional integer lattice the mixing time of block dynamics is polynomial in the number of blocks if the spin system exhibits strong spatial mixing. This result was later generalized in [5] for the Ising model on arbitrary graphs. Very recently, block dynamics based random equally-sized blocks where used in [12] to prove entropy factorization and improve the mixing time of Glauber dynamics for a variety of discrete spin systems up to the tree threshold.

Although our discretization works by restricting sphere positions to the integer lattice, the resulting graph is rather different from the lattice. Thus, results like those in [4] do not apply to our setting. However, on the other hand, we do not need to prove rapid mixing for arbitrary block covers. Instead, in order to obtain rapid mixing for clique dynamics, it is sufficient to establish this result for cases where all blocks are cliques.

Applying block dynamics directly would involve sampling from a conditional Gibbs distribution within each clique. Due to the exponential size of the cliques in our discretization, this would impose additional algorithmic challenges. Instead, similar to the previous literature, we rather use block dynamics as a tool for proving rapid mixing of another Markov chain, namely clique dynamics.

Improved mixing condition for clique dynamics via block dynamics

We analyze the mixing time of clique dynamics for a given clique cover by relating it to the mixing time of block dynamics, using the cliques as blocks. This is done by investigating a notion of pairwise influence between vertices that has also been used to establish rapid mixing of Glauber dynamics up to the tree threshold [2]. Let $\mathbb{P}_G[w]$ denote the probability of the event that a vertex $w \in V$ is in an independent set drawn from $\mu^{(G,\boldsymbol{\lambda})}$. Further, let $\mathbb{P}_G[\overline{w}]$ denote the probability that w is not in an independent set. We extend this abuse of notation to conditional probabilities, so $\mathbb{P}_G[\cdot \mid \overline{w}]$ for example denotes the probability of some event conditioned on w not being in an independent set. For a pair of vertices $v, w \in V$ the influence $\Psi_G(v, w)$ of v on w is defined as

$$\Psi_G(v,w) = \begin{cases} 0 & \text{if } v = w, \\ \mathbb{P}_G[w \mid v] - \mathbb{P}_G[w \mid \overline{v}] & \text{otherwise.} \end{cases}$$

The following condition in terms of pairwise influence is central to our analysis.

▶ **Condition 3.** *Let $(G, \boldsymbol{\lambda})$ be an instance of the multivariate hard-core model. There is a constant $C \in \mathbb{R}_{>0}$ and a function $q\colon V \to \mathbb{R}_{>0}$ such that for all $S \subseteq V$ and $r \in S$ it holds that*

$$\sum_{v \in S} |\Psi_G(r,v)| q(v) \leq C q(r).$$

Note that this condition appeared before in [13], where it was used for bounding the mixing time of Glauber dynamics for anti-ferromagnetic spin systems. Given Condition 3, we obtain the following result for the mixing time of block dynamics based on a clique cover.

▶ **Theorem 4.** *Let $(G, \boldsymbol{\lambda})$ be an instance of the multivariate hard-core model that satisfies Condition 3. Let Λ be a clique cover for G of size m, and let $Z_{\max} = \max_{i \in [m]} \{Z(G[\Lambda_i], \boldsymbol{\lambda}[\Lambda_i])\}$. The mixing time of the block dynamics $\mathcal{B}(G, \boldsymbol{\lambda}, \Lambda)$, starting from $\emptyset \in \mathcal{I}(G)$, is bounded by*

$$\tau_{\mathcal{B}}^{(\emptyset)}(\varepsilon) \leq m^{\mathrm{O}((2+C)C)} Z_{\max}^{\mathrm{O}((2+C)C)} \ln\left(\frac{1}{\varepsilon}\right).$$

Using a bound for the sum of absolute pairwise influences that was recently established in [13], it follows that the univariate hard-core model satisfies Condition 3 up to the tree threshold. As a result, we know that the mixing time of block dynamics is polynomial in m and $Z_{\max}$ for any clique cover of size m. To the best of our knowledge, this is the first result for the mixing time of block dynamics for the univariate hard-core model on general graphs that holds in this parameter range.

As we aim to apply clique dynamics to avoid sampling from the conditional Gibbs distribution in each step, we still need to prove that Theorem 4 also holds in terms of clique dynamics. To this end, we apply a Markov chain comparison argument from [15] to prove that using clique dynamics instead of block dynamics for the same clique cover Λ increases the mixing time by at most a factor $2Z_{\max}$. The following corollary, which is central for proving Theorem 1, follows immediately.

▶ **Corollary 5.** *Let (G, λ) be an instance of the univariate hard-core model such that the degree of G is bounded by Δ. Let Λ be a given clique cover of size m with $Z_{\max} = \max_{i\in[m]}\{Z(G[\Lambda_i], \lambda)\}$. Denote by $\mathcal{C} = \mathcal{C}(G, \lambda, \Lambda)$ the corresponding clique dynamics. If there is some $\delta \in \mathbb{R}_{>0}$ such that $\lambda \leq (1-\delta)\lambda_c(\Delta)$ then the mixing time of the clique dynamics $\mathcal{C}$, starting from $\emptyset \in \mathcal{I}(G)$, is bounded by*

$$\tau_{\mathcal{C}}^{(\emptyset)}(\varepsilon) \leq m^{\mathrm{O}\left(1/\delta^2\right)} Z_{\max}^{\mathrm{O}\left(1/\delta^2\right)} \ln\left(\frac{1}{\varepsilon}\right).$$

A side journey: comparison to multivariate conditions

In fact, Corollary 5 is sufficient for our application to the hard-sphere model. However, we also aim to set Condition 3 in the context of other conditions for rapid mixing of clique dynamics for the multivariate hard-core model. Note that such a rapid mixing result for clique dynamics caries over to Glauber dynamics by taking each vertex as a separate clique of size 1.

To this end, we compare Condition 3 to a strict version of the clique dynamics condition, originally introduced in [23] in the setting of clique dynamics for abstract polymer models. It turns out that this strict version of the clique dynamics condition directly implies Condition 3. This is especially interesting, as the clique dynamics condition was initially introduced as a local condition (only considering the neighborhood of each vertex) and is based on a coupling argument. However, we show that it can as well be understood as a sufficient condition for the global decay of pairwise influence with increasing distance between vertices.

Formally, we say that the strict clique dynamics condition is satisfied for an instance of the multivariate hard-core model $(G, \boldsymbol{\lambda})$ if there is a function $f\colon V \to \mathbb{R}_{>0}$ and a constant $\alpha \in (0, 1)$ such that for all $v \in V$ it holds that

$$\sum_{w\in N(v)} \frac{\lambda_w}{1+\lambda_w} f(w) \leq (1-\alpha) f(v),$$

where $N(v)$ is the neighborhood of v in G. This is a strict version of the clique dynamics condition in that the original clique dynamics condition would correspond to the case $\alpha = 0$ (i.e., the strict clique dynamics condition requires some strictly positive slack α).

The result of our comparison is summarized in the following statement.

▶ **Lemma 6.** *Let $(G, \boldsymbol{\lambda})$ be an instance of the multivariate hard-core model. If $(G, \boldsymbol{\lambda})$ satisfies the strict clique dynamics condition for a function f and a constant α, then it also satisfies Condition 3 for $q = f$ and $C = \frac{1}{\alpha}$.*

Lemma 6 is proven by translating the calculation of pairwise influences to the self-avoiding walk tree of the graph, based on a result in [13], and applying a recursive argument on this tree.

Despite being an interesting relationship between local coupling arguments and global pairwise influence, Lemma 6 also implies that, from an algorithmic perspective, Theorem 4 can be used to produce similar results as those obtained in [23] for abstract polymer models. Further, note that for the univariate model, using pairwise influence yields strictly better results than any coupling approach in the literature. This raises the question if a refined argument based on pairwise influences can be used in the multivariate setting to improve on the clique dynamics condition, leading to better approximation results on abstract polymer models.

1.3 Analyzing spectral expansion

As core technique for obtaining Theorem 4, we adapt an approach for bounding the mixing time that was recently used to prove rapid mixing of Glauber dynamics for the entire regime below the tree threshold for several applications, such as the hard-core model [2], general two-state spin systems [13], and proper colorings [11, 21]. The idea is to map the desired distribution to a weighted simplicial complex.

A simplicial complex X over a groundset U is a set family $X \subseteq 2^U$ such that for each $\tau \in X$ every subset of τ is also in X. We call the elements $\tau \in X$ the faces of X and refer to its cardinality $|\tau|$ as dimensionality.

For a univariate hard-core instance (G, λ), the authors of [2] construct a simplicial complex over a ground set U that contains two elements $x_v, x_{\overline{v}} \in U$ for each vertex $v \in V$. For every independent set $I \in \mathcal{I}(G)$, a face $\tau_I \in X$ is introduced such that $x_v \in \tau_I$ if $v \in I$ and $x_{\overline{v}} \in \tau_I$ otherwise. The simplicial complex is completed by taking the downward closure of these faces. Note that by construction all maximum faces of the resulting complex are $|V|$-dimensional and there is a one-to-one correspondence between the maximum faces and the independent sets in $\mathcal{I}(G)$. By assigning each maximum face $\tau_I \in X$ an appropriate weight, the Glauber dynamics can be represented as a random walk on those maximum faces, which is sometimes referred to as the two-step walk or down-up walk. Using a local-to-global theorem [1], the mixing time of this two-step walk can then be bounded based on certain local expansion properties of the simplicial complex X. It is then proved that such local expansion properties are well captured by the largest eigenvalue of the pairwise influence matrix Ψ_G, which is a $|V| \times |V|$ matrix that contains the pairwise influence $\Psi_G(v, w)$ for all $v, w \in V$. Finally, by bounding those influences a bound on this largest eigenvalue of Ψ_G is obtained. This analysis was later refined and generalized in [13] to general two-state spin systems.

This method was independently extended in [11] and [21] to the non-Boolean domain by applying it to the Glauber dynamics for proper colorings. The main differences to the Boolean domain are that elements of the simplicial complex now represent combinations of a vertex and a color. Furthermore, the bound on the local spectral expansion was obtained by using a different influence matrix, which captures the effect of selecting a certain color for one vertex on the distribution of colors for another vertex.

Although we are dealing with the hard-core model, which is Boolean in nature, the way that we model block dynamics is mainly inspired by the existing work on proper colorings [11]. Assume we have an instance of the multivariate hard-core model $(G, \boldsymbol{\lambda})$ and let Λ be a clique cover for G of size m such that every pair of distinct cliques is vertex-disjoint (i.e., Λ is a partition of G into cliques). We construct a simplicial complex X based on a ground set U that contains one element $x_v \in U$ for each vertex $v \in V$ and one additional element $\emptyset_i$ for each clique $\Lambda_i \in \Lambda$. We introduce a face $\tau_I \in X$ for each independent set $I \in \mathcal{I}(G)$ such that for every $\Lambda_i \in \Lambda$ we have $\emptyset_i \in \tau_I$ if $\Lambda_i \cap I = \emptyset$ and $x_v \in \tau_I$ if $\Lambda_i \cap I = \{v\}$ for some $v \in \Lambda_i$. The simplicial complex is completed by taking the downward closure of these faces. All maximum faces of the resulting complex are m-dimensional and there is a bijection between the maximum faces and the independent sets of G. Furthermore, there is a natural partitioning $\{U_i\}_{i \in [m]}$ of the ground set U, each partition U_i corresponding to a clique Λ_i, such that every maximum face in X contains exactly one element from each partition U_i.

By weighting each maximum face of X by the contribution of the corresponding independent set to the partition function, the block dynamics based on Λ are equivalent to the two-step walk on X. Thus, in order to bound the mixing time of the block dynamics, it is sufficient to study the local expansion properties of X. To this end, we adapt the influence matrix used for proper colorings in [11]. For $x \in U$, let $\mathbb{P}_G[x]$ denote the probability that

$x \in \tau_I$ for an independent set $I \in \mathcal{I}(G)$ drawn from $\mu^{(G,\boldsymbol{\lambda})}$ and corresponding maximum face $\tau_I \in X$. Similarly as for defining pairwise influences, we extend this notation to conditional probabilities. The clique influence matrix $\Phi_{G,\Lambda}$ contains an entry $\Phi_{G,\Lambda}(x,y)$ for each $x, y \in U$ with

$$\Phi_{G,\Lambda}(x,y) = \begin{cases} 0 & \text{if } x,y \in U_i \text{ for some } i \in [m], \\ \mathbb{P}_G[y \mid x] - \mathbb{P}_G[y] & \text{otherwise.} \end{cases}$$

By using similar linear-algebraic arguments as in [11] we prove that the maximum eigenvalue of $\Phi_{G,\Lambda}$ can be used to upper bound the local spectral expansion of X. To obtain Theorem 4 it is then sufficient to relate Condition 3 to the maximum eigenvalue of $\Phi_{G,\Lambda}$. The following lemma establishes this connection.

▶ **Lemma 7.** *Let $(G,\boldsymbol{\lambda})$ be an instance of the multivariate hard-core model that satisfies Condition 3 for a function q and a constant C. For every $S \subseteq V$ and every disjoint clique cover Λ of $G[S]$ it holds that the largest eigenvalue of $\Phi_{G[S],\Lambda}$ is at most $(2+C)C$.*

Note that our simplicial-complex representation is only given under the assumption that the cliques in the clique cover Λ are pairwise disjoint. Indeed, this is a necessary requirement to map the block dynamics to the two-step walk such that the local-global-theorem from [1] can be applied. Thus, Lemma 7 only helps to prove Theorem 4 for disjoint clique covers. However, we relax this requirement by proving that for every clique cover Λ a disjoint clique cover K can be constructed such that the block dynamics $\mathcal{B}(G,\boldsymbol{\lambda},\Lambda)$ and $\mathcal{B}(G,\boldsymbol{\lambda},K)$ have asymptotically the same mixing time. By this comparison argument, we extend Theorem 4 to arbitrary clique covers.

We are aware that, in the case of Glauber dynamics, more recent techniques for combining simplical complex representations with entropy factorization as proposed in [12] yield superior mixing time results. However, in case of the hard-core model, this approach comes with an additional multiplicative factor of $\Delta^{\mathrm{O}(\Delta^2)}$ in the mixing time (see section 8 of [12]). Although negligible for bounded degree graphs, this would be too much for our application, as the degree of our discretization gets exponentially large in the continuous volume $|\mathbb{V}|$ of the system. Thus, directly relating local spectral expansion with the spectral gap of block dynamics is more suitable in our case. We leave as an open question, whether a modification of the approach in [12] can be applied to further improve our mixing time result.

1.4 Outlook

We obtain the fugacity bound from Theorem 1 based on the tree threshold $\lambda_c(\Delta)$ of the hard-core model. An obvious question is whether there are any structural properties of our discretization that can be used to improve this bound. Similar results are known for specific graph classes, such as the 2-dimensional square lattice [48, 52, 54]. In [42] the authors discuss that a generalization of the connective constant to the continuous Euclidean space might be applicable to improve their uniqueness result for the hard-sphere model. A comparable algorithmic result was already established for the discrete hard-core model in [49]. However, any such improvement for our discretization would require the connective constant of G_ρ to be at least by a constant factor small than its maximum degree Δ_ρ. Unfortunately, due to a result in [47], this is not the case. Although this does not necessarily imply that a similar concept does not work in continuous space, it gives a strong evidence that a more specialized tool instead of the connective constant might be required.

A different direction for future work is to see which other quantities and properties of the model are preserved under discretization. This would especially include the thermodynamic pressure and its analyticity. As a matter of fact, non-analytic points of the pressure along the

positive real axis of fugacity in the thermodynamic limit are known to mark phase transitions in infinite volume systems (see for example [42]). One way to approach this could be to prove a relation between zero-freeness of the continuous and the discretized partition function in a complex neighborhood of the real axis by extending our convergence result to the complex domain. Along this line, insights could be gained in how far properties like correlation decay and phase transitions (or their absence) are preserved under sufficiently fine discretization.

From a purely technical point of view, it is interesting to see if our result on the mixing time of block dynamics in Theorem 4 also holds without the requirement of using cliques as blocks. Especially: is the mixing time for block dynamics for the univariate hard-core model polynomial in the number of blocks for any block cover? Most of our techniques that we use for clique covers, such as modeling the distribution as a simplicial complex and relating its local spectral expansion to the clique influence matrix, can be generalized in a straightforward way for different choices of blocks. However, the main difficulty is to relate generalized versions of the clique influence matrix to pairwise influences, as we do in Lemma 7. One way to circumvent this might be to not rely on pairwise influences at all but to rather investigate the influence matrix directly, for example, via different computational-tree methods.

Finally, it would be interesting to see if approaches like ours can be extended to other continuous models from statistical physics (see for example coarse-graining [20]). We believe that the variety of tools that are already established for discrete spin systems are useful in this setting to establish rigorous computational results for different continuous models. We emphasize that clique and block dynamics are a useful computational tool to handle the exponential blow-ups caused by discretization.

References

1 Vedat Levi Alev and Lap Chi Lau. Improved analysis of higher order random walks and applications. In *Proc. of STOC'20*, pages 1198–1211, 2020. doi:10.1145/3357713.3384317.

2 Nima Anari, Kuikui Liu, and Shayan Oveis Gharan. Spectral independence in high-dimensional expanders and applications to the hardcore model. *CoRR*, abs/2001.00303, 2020. arXiv:2001.00303.

3 Ivona Bezáková, Andreas Galanis, Leslie Ann Goldberg, and Daniel Štefankovič. Inapproximability of the independent set polynomial in the complex plane. In *Proc. of STOC'18*, page 1234–1240, 2018. doi:10.1145/3188745.3188788.

4 Antonio Blanca, Pietro Caputo, Alistair Sinclair, and Eric Vigoda. Spatial mixing and nonlocal Markov chains. *Random Structures & Algorithms*, 55(3):584–614, 2019. doi:10.1002/rsa.20844.

5 Antonio Blanca, Zongchen Chen, and Eric Vigoda. Swendsen-Wang dynamics for general graphs in the tree uniqueness region. *Random Structures & Algorithms*, 56(2):373–400, 2020. doi:10.1002/rsa.20858.

6 Christian Borgs, Jennifer T. Chayes, Tyler Helmuth, Will Perkins, and Prasad Tetali. Efficient sampling and counting algorithms for the Potts model on $\mathbb{Z}^d$ at all temperatures. In *Proc. of STOC'20*, pages 738–751, 2020. doi:10.1145/3357713.3384271.

7 Tomáš Boublik, Ivo Nezbeda, and Karel Hlavaty. *Statistical thermodynamics of simple liquids and their mixtures*. Fundamental Studies in Engineering. Elsevier, 1980.

8 Sarah Cannon and Will Perkins. Counting independent sets in unbalanced bipartite graphs. In *Proc. of SODA'20*, pages 1456–1466, 2020. doi:10.1137/1.9781611975994.88.

9 Katrin Casel, Philipp Fischbeck, Tobias Friedrich, Andreas Göbel, and J. A. Gregor Lagodzinski. Zeros and approximations of holant polynomials on the complex plane. *CoRR*, abs/1905.03194, 2019. arXiv:1905.03194.

10 Zongchen Chen, Andreas Galanis, Leslie Ann Goldberg, Will Perkins, James Stewart, and Eric Vigoda. Fast algorithms at low temperatures via Markov chains. In *Proc. of APPROX-/RANDOM'19*, pages 41:1–41:14, 2019. doi:10.4230/LIPIcs.APPROX-RANDOM.2019.41.

11 Zongchen Chen, Andreas Galanis, Daniel Stefankovic, and Eric Vigoda. Rapid mixing for colorings via spectral independence. *CoRR*, abs/2007.08058, 2020. arXiv:2007.08058.

12 Zongchen Chen, Kuikui Liu, and Eric Vigoda. Optimal mixing of Glauber dynamics: Entropy factorization via high-dimensional expansion. *CoRR*, abs/2011.02075, 2020. arXiv:2011.02075.

13 Zongchen Chen, Kuikui Liu, and Eric Vigoda. Rapid mixing of Glauber dynamics up to uniqueness via contraction. *CoRR*, abs/2004.09083, 2020. arXiv:2004.09083.

14 Hofer-Temmel Christoph et al. Disagreement percolation for the hard-sphere model. *Electronic Journal of Probability*, 24, 2019.

15 Persi Diaconis and Laurent Saloff-Coste. Comparison theorems for reversible Markov chains. *The Annals of Applied Probability*, pages 696–730, 1993.

16 Martin Dyer, Abraham D. Flaxman, Alan M. Frieze, and Eric Vigoda. Randomly coloring sparse random graphs with fewer colors than the maximum degree. *Random Structures & Algorithms*, 29(4):450–465, 2006. doi:10.1002/rsa.20129.

17 Martin Dyer, Alan Frieze, and Mark Jerrum. On counting independent sets in sparse graphs. *SIAM Journal on Computing*, 31(5):1527–1541, 2002. doi:10.1137/S0097539701383844.

18 Charilaos Efthymiou. MCMC sampling colourings and independent sets of $g(n, d/n)$ near uniqueness threshold. In *Proc. of SODA'14*, page 305–316, 2014. doi:10.1137/1.9781611973402.22.

19 Charilaos Efthymiou, Thomas P. Hayes, Daniel Štefankovič, and Eric Vigoda. Sampling random colorings of sparse random graphs. In *Proc. of SODA'18*, page 1759–1771, 2018. doi:10.1137/1.9781611975031.115.

20 Pep Espanol. Statistical mechanics of coarse-graining. In *Novel Methods in Soft Matter Simulations*, pages 69–115. Springer, 2004. doi:10.1007/978-3-540-39895-0_3.

21 Weiming Feng, Heng Guo, Yitong Yin, and Chihao Zhang. Rapid mixing from spectral independence beyond the Boolean domain. *CoRR*, abs/2007.08091, 2020. arXiv:2007.08091.

22 Roberto Fernández, Aldo Procacci, and Benedetto Scoppola. The analyticity region of the hard sphere gas. Improved bounds. *Journal of Statistical Physics*, 128(5):1139–1143, 2007. doi:10.1007/s10955-007-9352-7.

23 Tobias Friedrich, Andreas Göbel, Martin S. Krejca, and Marcus Pappik. Polymer dynamics via cliques: New conditions for approximations. *CoRR*, abs/2007.08293, 2020. arXiv:2007.08293.

24 Andreas Galanis, Qi Ge, Daniel Stefankovic, Eric Vigoda, and Linji Yang. Improved inapproximability results for counting independent sets in the hard-core model. *Random Structures & Algorithms*, 45(1):78–110, 2014. doi:10.1002/rsa.20479.

25 Andreas Galanis, Leslie Ann Goldberg, and Daniel Stefankovic. Inapproximability of the independent set polynomial below the Shearer threshold. *CoRR*, abs/1612.05832, 2016. arXiv:1612.05832.

26 Andreas Galanis, Leslie Ann Goldberg, and James Stewart. Fast algorithms for general spin systems on bipartite expanders. In *Proc. of MFCS'20*, 2020. To appear. arXiv:2004.13442.

27 David Galvin and Prasad Tetali. Slow mixing of Glauber dynamics for the hard-core model on regular bipartite graphs. *Random Structures & Algorithms*, 28(4):427–443, 2006. doi:10.1002/rsa.20094.

28 Heng Guo and Mark Jerrum. Perfect simulation of the hard disks model by partial rejection sampling. *CoRR*, abs/1801.07342, 2018. arXiv:1801.07342v2.

29 Jean-Pierre Hansen and Ian R. McDonald. *Theory of Simple Liquids*. Academic Press, fourth edition edition, 2013. doi:10.1016/B978-0-12-387032-2.00013-1.

30 Nicholas J.A. Harvey, Piyush Srivastava, and Jan Vondrák. Computing the independence polynomial: From the tree threshold down to the roots. In *Proc. of SODA'18*, pages 1557–1576, 2018. doi:10.1137/1.9781611975031.102.

31 Thomas P. Hayes and Cristopher Moore. Lower bounds on the critical density in the hard disk model via optimized metrics. *CoRR*, abs/1407.1930, 2014. arXiv:abs/1407.1930.

32 Tyler Helmuth, Will Perkins, and Samantha Petti. Correlation decay for hard spheres via Markov chains. *CoRR*, abs/2001.05323, 2020. arXiv:2001.05323.

33 Tyler Helmuth, Will Perkins, and Guus Regts. Algorithmic Pirogov–Sinai theory. In *Proc. of STOC'19*, pages 1009–1020, 2019. doi:10.1145/3313276.3316305.

34 Matthew Jenssen, Felix Joos, and Will Perkins. On the hard sphere model and sphere packings in high dimensions. *Forum of Mathematics, Sigma*, 7:e1, 2019. doi:10.1017/fms.2018.25.

35 Matthew Jenssen, Peter Keevash, and Will Perkins. Algorithms for #bis-hard problems on expander graphs. In *Proc. of SODA'19*, pages 2235–2247, 2019. doi:10.1137/1.9781611975482.135.

36 Ravi Kannan, Michael W. Mahoney, and Ravi Montenegro. Rapid mixing of several Markov chains for a hard-core model. In *Proc. of ISAAC'03*, pages 663–675, 2003. doi:10.1007/978-3-540-24587-2_68.

37 Wilfrid S. Kendall. Perfect simulation for the area-interaction point process. In *Probability Towards 2000*, pages 218–234. Springer, 1998. doi:10.1007/978-1-4612-2224-8_13.

38 Wilfrid S. Kendall and Jesper Møller. Perfect simulation using dominating processes on ordered spaces, with application to locally stable point processes. *Advances in Applied Probability*, 32(3):844–865, 2000. doi:10.1239/aap/1013540247.

39 Liang Li, Pinyan Lu, and Yitong Yin. Correlation decay up to uniqueness in spin systems. In *Proc. of SODA'13*, pages 67–84, 2013. doi:10.1137/1.9781611973105.5.

40 Fabio Martinelli. Lectures on Glauber dynamics for discrete spin models. In *Lectures on probability theory and statistics*, pages 93–191. Springer, 1999. doi:10.1007/978-3-540-48115-7_2.

41 Nicholas Metropolis, Arianna W. Rosenbluth, Marshall N. Rosenbluth, Augusta H. Teller, and Edward Teller. Equation of state calculations by fast computing machines. *The Journal of Chemical Physics*, 21(6):1087–1092, 1953. doi:10.1063/1.1699114.

42 Marcus Michelen and Will Perkins. Analyticity for classical gasses via recursion. *CoRR*, abs/2008.00972, 2020. arXiv:2008.00972.

43 Sarat Babu Moka, Sandeep Juneja, and Michel R. H. Mandjes. Analysis of perfect sampling methods for hard-sphere models. *SIGMETRICS Performance Evaluation Review*, 45(3):69–75, 2018. doi:10.1145/3199524.3199536.

44 Elchanan Mossel and Allan Sly. Gibbs rapidly samples colorings of $g(n, d/n)$. *Probability Theory and Related Fields*, 148(1-2):37–69, 2010. doi:10.1007/s00440-009-0222-x.

45 Elchanan Mossel, Dror Weitz, and Nicholas Wormald. On the hardness of sampling independent sets beyond the tree threshold. *Probability Theory and Related Fields*, 143(3-4):401–439, 2009. doi:10.1007/s00440-007-0131-9.

46 Viresh Patel and Guus Regts. Deterministic polynomial-time approximation algorithms for partition functions and graph polynomials. *SIAM Journal on Computing*, 46(6):1893–1919, 2017. doi:10.1137/16M1101003.

47 Mathew D Penrose. Self-avoiding walks and trees in spread-out lattices. *Journal of Statistical Physics*, 77(1-2):3–15, 1994. doi:10.1007/BF02186829.

48 Ricardo Restrepo, Jinwoo Shin, Prasad Tetali, Eric Vigoda, and Linji Yang. Improved mixing condition on the grid for counting and sampling independent sets. *Probability Theory and Related Fields*, 156(1-2):75–99, 2013. doi:10.1007/s00440-012-0421-8.

49 Alistair Sinclair, Piyush Srivastava, Daniel Štefankovič, and Yitong Yin. Spatial mixing and the connective constant: Optimal bounds. *Probability Theory and Related Fields*, 168(1-2):153–197, 2017. doi:10.1007/s00440-016-0708-2.

50 Allan Sly. Computational transition at the uniqueness threshold. In *Proc. of FOCS'10*, pages 287–296, 2010. doi:10.1109/FOCS.2010.34.

51 J. van den Berg and R. Brouwer. Random sampling for the monomer–dimer model on a lattice. *Journal of Mathematical Physics*, 41(3):1585–1597, 2000. doi:10.1063/1.533198.

52 Juan C Vera, Eric Vigoda, and Linji Yang. Improved bounds on the phase transition for the hard-core model in 2-dimensions. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques*, pages 699–713. Springer, 2013. doi:10.1007/978-3-642-40328-6_48.

53 Dror Weitz. Combinatorial criteria for uniqueness of Gibbs measures. *Random Structures & Algorithms*, 27(4):445–475, 2005.

54 Dror Weitz. Counting independent sets up to the tree threshold. In *Proc. of STOC'06*, pages 140–149, 2006. doi:10.1145/1132516.1132538.

Constant-Factor Approximation to Deadline TSP and Related Problems in (Almost) Quasi-Polytime

Zachary Friggstad
Department of Computer Science, University of Alberta, Edmonton, Canada

Chaitanya Swamy
Department of Combinatorics and Optimization, University of Waterloo, Canada

Abstract

We investigate a genre of vehicle-routing problems (VRPs), that we call *max-reward* VRP*s*, wherein nodes located in a metric space have associated rewards that depend on their visiting times, and we seek a path that earns maximum reward. A prominent problem in this genre is *deadline TSP*, where nodes have *deadlines* and we seek a path that visits all nodes by their deadlines and earns maximum reward. Our main result is a *constant-factor approximation* for deadline TSP running in time $O\left(n^{O(\log(n\Delta))}\right)$ in metric spaces with integer distances at most Δ. This is the *first* improvement over the approximation factor of $O(\log n)$ due to Bansal et al. [2] in over 15 years (but is achieved in super-polynomial time). Our result provides the first concrete indication that $\log n$ is unlikely to be a real inapproximability barrier for deadline TSP, and raises the exciting possibility that deadline TSP might admit a polytime constant-factor approximation.

At a high level, we obtain our result by carefully guessing an appropriate sequence of $O(\log(n\Delta))$ nodes appearing on the optimal path, and finding suitable paths between any two consecutive guessed nodes. We argue that the problem of finding a path between two consecutive guessed nodes can be relaxed to an instance of a special case of deadline TSP called *point-to-point (*P2P*) orienteering*. *Any* approximation algorithm for P2P orienteering can then be utilized in conjunction with either a greedy approach, or an LP-rounding approach, to find a good set of paths overall between every pair of guessed nodes. While concatenating these paths does not immediately yield a feasible solution, we argue that it can be covered by a constant number of feasible solutions. Overall our result therefore provides a *novel reduction* showing that any α-approximation for P2P orienteering can be leveraged to obtain an $O(\alpha)$-approximation for deadline TSP in $O\left(n^{O(\log n\Delta)}\right)$ time.

Our results extend to yield the same guarantees (in approximation ratio and running time) for a substantial generalization of deadline TSP, where the reward obtained by a client is given by an arbitrary non-increasing function (specified by a value oracle) of its visiting time. Finally, we discuss applications of our results to variants of deadline TSP, including settings where both end-nodes are specified, nodes have release dates, and orienteering with time windows.

2012 ACM Subject Classification Theory of computation → Approximation algorithms analysis; Mathematics of computing → Discrete optimization

Keywords and phrases Approximation algorithms, Vehicle routing problems, Deadline TSP, Orienteering

Digital Object Identifier 10.4230/LIPIcs.ICALP.2021.67

Category Track A: Algorithms, Complexity and Games

Funding *Zachary Friggstad*: Supported by the Canada Research Chairs program and an NSERC Discovery grant.
Chaitanya Swamy: Supported in part by NSERC grant 327620-09 and an NSERC Discovery Accelerator Supplement Award.

1 Introduction

Vehicle-routing problems (VRPs) constitute a rich class of optimization problems that find a variety of applications and have been extensively studied in the Operations Research and Computer Science literature (see, e.g., [27]) Broadly speaking, vehicle-routing problems can be divided into two categories: one, where we have a *fixed* set of nodes or clients that need to be visited, and we seek the most effective route(s) for visiting these clients (e.g., TSP-style problems [9, 25, 29, 26], minimum-latency problems [5, 23], VRPs with distance bounds [19, 22] and regret bounds [13]); and the other, where, due to resource constraints, we need to *select* which set of clients to visit *and* plan suitable routes for these clients. We investigate a prominent class of VRPs that fall into the second category, wherein nodes have associated rewards that depend on their visiting times, and we seek a path that earns maximum reward. We call this genre of problems *max-reward* VRP*s*, and they constitute a well-studied class of VRPs (see, e.g., [15, 3, 2, 8, 6, 14]).

We consider a fundamental problem in this genre, called the *deadline TSP* problem. In deadline TSP, we are given a (symmetric) metric space $(\{r\} \cup V, c)$ with r being a distinguished starting root node, and each node $v \in V$ has a certain deadline $D_v \geq 0$ and reward $\pi_v \geq 0$. We seek a path starting at r that maximizes the total reward of the nodes on the path that are visited by their deadlines. Since we are in a metric space, by shortcutting, we may assume that all nodes on the path are visited by their deadlines. So equivalently, we seek a maximum-reward r-rooted path that visits all nodes by their deadlines. A simpler problem in the max-reward VRP genre is the *point-to-point (*P2P*) orienteering* problem, wherein we are also given an end-node t, and we seek an r-t path of at most a given length B that collects maximum reward. This is a special case of deadline TSP, which can be seen by setting the deadline of each node v to $B - c_{vt}$.

Max-reward VRPs tend to be more complicated problems than the first category of "fixed-node-set" VRPs mentioned above because of the added combinatorial aspect of selecting which nodes to visit, which is interlinked with the routing decisions, and we have much less of an understanding of max-reward VRPs compared to fixed-node-set VRPs. A constant-factor approximation is known only for P2P orienteering in undirected graphs, which is one of the most rudimentary max-reward VRPs. For other, more-sophisticated, max-reward VRPs – deadline TSP, submodular orienteering, directed orienteering – only logarithmic or polylogarithmic (or worse) approximation factors are known. Furthermore, even for undirected orienteering, the current-best approximation ratio has remained stagnant at $(2 + \epsilon)$ [6] (and fresh LP-based insights were obtained only recently [14]), whereas for s-t path TSP (the corresponding fixed-node-set problem), a steady stream of work [1, 24, 28, 29] has exploited LP-based insights (and other ideas) to improve the approximation ratio to 1.5 [29]. The contrast is even more evident in asymmetric metrics: while $O(1)$-approximation algorithms are now known for asymmetric TSP (ATSP) [26] and s-t path ATSP [17], the best-known guarantee for directed orienteering is an $O(\alpha \log |V|)$-approximation using an LP-relative α-approximation for ATSP [21], which explicitly shows the degradation when moving to the max-reward-VRP version.

Our results. Our main contribution is to provide the *first constant-factor approximation* guarantee for deadline TSP (Theorem 3.1); notably, we obtain a relatively small approximation ratio of $(7.63 + \epsilon)$. We may assume by scaling that all distances in our metric space are integers. Our algorithm runs in time $O\big(n^{O(\log n\Delta)}\big)$, where n is the number of points, and Δ is the diameter of the (scaled) metric space. In particular, for graphical metrics, we obtain a

quasi-polytime (i.e., $O\big(n^{O(\log n)}\big)$-time) constant-factor approximation. Our guarantee yields the *first* improvement over the (polytime) $O(\log n)$ approximation factor obtained by Bansal et al. [2] in over 15 years, even for graphical metrics, but is achieved in super-polynomial time. Prior to our work, it was unclear whether $\log n$ is a real inapproximability barrier for deadline TSP. Our result provides the first concrete indication that this is not the case,[1] and raises the enticing possibility that deadline TSP might admit a polynomial-time constant-factor approximation.

As noted above, constant-factor approximation algorithms are known for P2P orienteering [2, 6, 14], which is a special case of deadline TSP. Our chief technical contribution lies in providing a novel *reduction* showing that *an α-approximation algorithm for* P2P *orienteering can be utilized to obtain an $O(\alpha)$-approximation algorithm for deadline TSP* in $O\big(n^{O(\log n\Delta)}\big)$ time (see Theorem 3.1).

We obtain the *same approximation guarantees* for a substantial generalization of deadline TSP, wherein each node v has a *non-increasing reward function* $\pi_v : \mathbb{R}_+ \mapsto \mathbb{R}_+$, with $\pi_v(x)$ giving the reward of node v if v is visited at time x, and, as before, the goal is to find a path that collects maximum reward. We call this problem *monotone-reward TSP* (Section 4.1). Notice that this problem also captures *discounted-reward TSP*, considered by Blum et al. [3], which is the special case where $\pi_v(x) = \pi_v \cdot \gamma^{-x}$, where $\gamma > 1$ is a discount factor. Our results here only require *value-oracle* access to π_v – an oracle that on input x returns $\pi_v(x)$ – and follow from two distinct approaches (see Theorem 4.1): (a) we show that we can reduce monotone-reward TSP in polytime to deadline TSP with a $(1-\epsilon)$-factor loss; and (b) our algorithm and analysis for deadline TSP readily extend to yield the same guarantees for monotone-reward TSP.

Interestingly, our reduction from orienteering to {deadline, monotone-reward} TSP also applies to *asymmetric metrics* (see Remark 3.8). The recent result of Svensson et al. [26] establishing an $O(1)$-integrality gap for the ATSP LP-relaxation implies (due to the work of [21]) an $O(\log n)$-approximation for directed orienteering; thus our reduction yields an $O(\log n)$-approximation for {deadline, monotone-reward} TSP in asymmetric metrics in $O\big(n^{O(\log n\Delta)}\big)$ time. While this does not improve upon the quasi-polytime $O(\log n)$-approximation for asymmetric deadline TSP that follows from the work of Chekuri and Pál [8], our reduction does show that improved (e.g., constant-factor) approximations to directed orienteering will *directly* translate to analogous improvements for directed deadline TSP in $O\big(n^{O(\log n\Delta)}\big)$ time.

In Section 4.2, we show that our results yield $O(1)$-approximation for some other variants of {deadline, monotone-reward} TSP, namely, settings where: (a) the path must start *and* end at some given nodes; and (b) nodes have release dates but no deadlines, or more generally have a non-decreasing reward function, and we seek a length-bounded path gathering maximum reward (and we are allowed to wait at nodes while traversing a path). We also consider *orienteering with time windows*, wherein we have a time window for each node and we collect its reward if it is visited within its time window. By combining our result for deadline TSP with certain results of Chekuri et al. [6], we obtain an $O\big(\log \frac{L_{\max}}{L_{\min}}\big)$-approximation for orienteering with time windows in $O\big(n^{\log n\Delta}\big)$ time, where $L_{\max}$ and $L_{\min}$ are the lengths of the longest and shortest time windows with non-zero length. This improves upon the approximation guarantee of [6] when the optimum value is large; in particular, if $L_{\max} = O(L_{\min})$, we obtain an $O(1)$-approximation, whereas [6] obtain an $O(\log(\text{optimal value}))$-approximation, but in polynomial time.

[1] More precisely, if $NP \not\subseteq DTIME\,[O\big(n^{O(\log n)}\big)]$, then obtaining an $O(1)$-approximation in graphical metrics cannot be *NP*-hard. Also, any reduction showing *NP*-hardness of an $O(1)$-approximation in general metrics must involve exponentially large distances.

Our techniques. We briefly discuss the techniques that we utilize to obtain our result for deadline TSP (see also "Overview and intuition" in Section 3). We begin by guessing a suitable set of $O(\log n\Delta)$ nodes, which are a subsequence of the nodes encountered along an optimal path P^*. This guessing step requires some care, in light of the fact that we are dealing with *hard* deadlines – i.e., we need to satisfy deadlines *exactly* (and not just approximately) – which rules out certain standard approaches. For instance, a natural attempt would be to view nodes of P^* as being grouped into geometric buckets based on their visiting times and/or deadlines and guess the boundary nodes of the buckets; since nodes in a bucket involve roughly the same visiting time and/or deadline, it is tempting to solve a P2P orienteering instance with the boundary nodes as end-points. But this is too coarse an idea that is incompatible with hard deadlines. Indeed, since one cannot merge similar deadlines/visiting times, any such approach faces the issue that there could be $\Omega(n)$ distinct visiting times and deadlines to consider. Not surprisingly, [7], who take such an approach make the strong assumption that there are only $O(1)$ distinct deadlines. Alternatively, other works on deadline TSP [2, 6] are based on extracting a subset of P^* with a simpler deadline and visiting-time structure, but at the expense of an $O(\log n)$-factor loss in objective.

We seek to avoid both the above bottlenecks, but, as alluded to above, honing in on the correct choice of guessed nodes requires some insight and a more refined approach. At a high level, the sequence of nodes we guess $v_0 := r, v_1, \ldots, v_{\log n\Delta}$ has the property that the length of the v_i-v_{i+1} portion of P^*, which we denote by $P^*_{v_iv_{i+1}}$, is about $c_{v_iv_{i+1}} + \gamma^i$, where γ is some constant (we use $\gamma = 1.5$); equivalently, we say that the v_i-relative regret-length of $P^*_{v_iv_{i+1}}$ is about γ^i. We still obtain P2P-orienteering solutions (with the above distance bound) between consecutive v_i, such that they cumulatively collect enough reward (but they may not have disjoint node-sets). Concatenating these yields a path P that may violate some deadlines. A key insight is that if we were to shortcut past the v_{i-2}-v_i portion of P, then each client visited between v_i and v_{i+1} after shortcutting, *now has its deadline satisfied*: we "save" roughly $\gamma^{i-2} + \gamma^{i-1} \geq \gamma^i$ distance in this shortcutting process, which is enough to guarantee that clients visited between v_i and v_{i+1} in our algorithm have their deadlines satisfied exactly. Thus, we can cover P using a constant number of deadline-TSP solutions. Our eventual algorithm is slightly more involved as we have to account for large leaps in distances as well (thereby requiring us to guess an additional $\log n\Delta$ nodes).

Finally, we consider two approaches for obtaining the P2P-orienteering solutions. The first approach is based on viewing the problem as an instance of the *maximum coverage problem with group budgets* [7], for which a simple greedy algorithm yields a good approximation. The second is a more-sophisticated LP-based approach that yields a somewhat better approximation guarantee. We write an LP to find paths between guessed nodes, and extract from the LP solution a distribution over P2P-orienteering solutions between consecutive guesses. Randomly picking one such path for each v_i will ensure each client is covered with probability proportional to the extent that the LP covers it. While recent LP-based insights [14] for orienteering allow us to obtain a compact LP to obtain such distributions, one can also work with a configuration-style LP that directly encodes the distribution requirement. *Any* P2P-orienteering approximation algorithm can then be used to approximately separate the dual of this LP, and hence yield the desired distributions.

Our $O\big(n^{\log n\Delta}\big)$ running time stems from the need for enumerating $O(\log n\Delta)$ nodes. There is however some hope that this enumeration step and its resulting run-time blowup can be circumvented. At a high level, the search space of our algorithm can be represented by a directed layered graph, whose vertices in layer i encode the various choices for the v_i-v_{i+1} portion of P^*, and arcs encode compatible choices. While we use brute-force enumeration

to find a suitable path in this digraph, one can envisage other means for finding this path, e.g., dynamic programming or linear programming. An important and useful fact to note is that we only have $O(\log n\Delta)$ layers, so that, for $\text{poly}(n, \Delta)$ running time (hence, polytime for graphical metrics), one can afford to take time exponential in the number of layers. (We note that such savings were achieved in the context of another VRP, namely the directed latency problem: Nagarajan and Ravi [20] gave an $O\big(n^{\log n}\big)$-time, $O(\log n)$-approximation based on guessing $O(\log n)$ nodes, and subsequently Friggstad et al. [12] obtained the same approximation in polytime by showing, in essence (roughly speaking), that this guesswork can be eliminated by formulating a suitable LP to provide the guessed nodes.)

Related work. We limit ourselves to a discussion of the work that is most relevant to the max-reward VRP problems we consider; we refer the reader to [27] for more information on vehicle routing problems in general. As mentioned earlier, the current-best approximation factor for deadline TSP is $O(\log n)$ due to Bansal et al. [2]. They also give a bicriteria $O(\log 1/\epsilon)$-approximation that violates deadlines by at most a $(1+\epsilon)$-factor; with integer deadlines this yields an $O(\log D_{\max})$-approximation (where $D_{\max} := \max_v D_v$). Since we obtain $O\big(n^{O(\log n\Delta)}\big)$ running time, it is also relevant to compare our result with the recursive greedy approach of Chekuri and Pál [8]. This is a versatile approach that yields logarithmic approximation guarantees for various problems (including deadline TSP) in quasi-polytime. However, this approach seems hard-pressed to yield anything *better* than logarithmic guarantees (even for deadline TSP); in particular, the $\Omega(\log n)$-inapproximability result for submodular orienteering[2] suggests that one cannot improve their approach to obtain $o(\log n)$ approximation guarantees.

The special case of deadline TSP with uniform node-deadlines is called *rooted orienteering*. Blum et al. [3] obtained the first $O(1)$-approximation algorithm for this problem. Their ideas were refined by [2, 6] to obtain the current-best $(2+\epsilon)$-approximation, which also applies to P2P orienteering [6]. Recently, Friggstad and Swamy [14] developed a different LP-based approach for orienteering, which also yields (slightly inferior) $O(1)$-approximations for these problems. We utilize some of their insights in our work.

Various (other) generalizations of orienteering and deadline TSP have also been studied. Bansal et al. [2] give an $O(\log^2 n)$-approximation for orienteering with time windows, and an $O(\log D_{\max})$ approximation with integer release dates and deadlines. They make the informal remark that a deadline-TSP approximation that relies on an α-approximation for P2P orienteering will translate to an α^2-approximation for orienteering with time windows. However, this comment seems to be in the *specific* context of their approach, and it is unclear if it can be applied with our framework to obtain an $O(1)$-approximation for the time-windows problem in $O\big(n^{O(\log n\Delta)}\big)$; such a result would be quite interesting. Chekuri et al. [6] show that an α-approximation for P2P orienteering yields an $O\big(\alpha \max\{\log \mathsf{opt}, \log \frac{L_{\max}}{L_{\min}}\}\big)$-approximation for orienteering with time windows, where opt is the optimal value, and $L_{\max}$ and $L_{\min}$ are the lengths of the longest and shortest time windows with non-zero length.

Blum et al. [3] considered the special case of monotone-reward TSP called discounted-reward TSP (wherein the reward of node v at time t is $\pi_v\gamma^{-t}$), and devised a 6.753-approximation algorithm. This factor was slightly improved to 5.195 by [10]. Chekuri

[2] Submodular orienteering captures "group orienteering", wherein we are given (disjoint) groups of vertices, and the reward of a path is the number of groups it hits. Using set-cover ideas, one can show that an α-approximation for group orienteering yields an $O(\alpha \log n)$-approximation for the group Steiner tree problem. The $\Omega(\log^2 n)$-inapproximability result for group Steiner tree [16] thus translates to an $\Omega(\log n)$-inapproximability result for submodular orienteering.

and Pál [8] show that their recursive-greedy approach yields a quasi-polytime $O(\log n)$-approximation for a generalization of monotone-reward TSP where the node reward is an *arbitrary* function of time. They obtain the same guarantee for a further generalization of orienteering (even in asymmetric metrics) that they introduce, called *submodular orienteering with time windows*, where the reward of a path is given by a monotone submodular function of the set of nodes visited within their time windows.

In *asymmetric metrics*, Chekuri et al. [6] give an $O(\log^2 \mathsf{opt})$-approximation for P2P orienteering; this also yields guarantees for the time-windows problem via their aforementioned reduction (which also applies to asymmetric metrics). The current-best approximation for directed orienteering is $O(\log n)$, which follows by combining the $O(1)$-integrality gap for the ATSP LP [26] with a result of [21] showing that an LP-relative α-approximation for ATSP yields an $O(\alpha \log n)$-approximation for directed orienteering.

Finally, there is a wealth of literature on fixed-node-set VRPs; we refer the reader to some of the most recent work on these problems [25, 29, 26] for further pointers.

2 Preliminaries and notation

All the problems we consider involve an underlying complete graph $G = (\{r\} \cup V, E)$, where r is a distinguished root node, and metric edge costs $\{c_{uv}\}$. By scaling, we may assume that c_{uv} is an integer for all $u, v \in V \cup \{r\}$. Let $\Delta := \max_{u,v \in V \cup \{r\}} c_{uv}$ be the diameter of the scaled metric space. For a set S of edges, we often use $c(S)$ to denote $\sum_{e \in S} c_e$. We often use V' to denote $\{r\} \cup V$. Let $n = |V'| = |V| + 1$. We call a path P in G rooted if it begins at r. We always think of the nodes on a rooted path P as being ordered in increasing order of their distance along P from r. For any path P and nodes $u, v \in P$, let P_{uv} denote the u-v subpath of P. For a rooted path P and node $v \in P$, the visiting time of v is the distance from r to v along P, which we denote by $c_P(v) := c(P_{rv})$. To avoid excessive notation, we will view a path P sometimes as its edge-set, and sometimes as its node-set; the meaning will be clear from the context.

In *deadline TSP*, each node $v \in V$ has a deadline $D_v \geq 0$ and reward $\pi_v \geq 0$. For notational convenience, set $D_r = \pi_r = 0$. The goal is to find a simple rooted path P such that $c_P(v) \leq D_v$ for all $v \in P$ that maximizes $\sum_{v \in P} \pi_v$. We may assume that $D_v \geq c_{rv}$ for all $v \in V$, as otherwise v can never be visited by a feasible solution and we can simply delete v from our metric space.

Monotone-reward TSP is a substantial generalization of deadline TSP, wherein each node v has a *non-increasing reward function* $\pi_v : \mathbb{R}_+ \mapsto \mathbb{R}_+$. For notational convenience, set $\pi_r(x) = 0$ for all x. The goal is to find a simple rooted path P that maximizes $\sum_{v \in P} \pi_v(c_P(v))$. We assume that each $\pi_v(.)$ is specified via a value oracle, that on input x returns $\pi_v(x)$.

Regret distances. For any $u \in V \cup \{r\}$, and any ordered pair $v, w \in V \cup \{r\}$, define the *regret distance* of (v, w) with respect to u to be $c^{\mathsf{reg}}_u(v, w) := c_{uv} + c_{vw} - c_{uw}$. The regret distances $\{c^{\mathsf{reg}}_u(v, w)\}_{v,w \in V \cup \{r\}}$ form an *asymmetric* metric. The regret-length of a path P with respect to its start node is called the excess of P in [3, 2, 6]. A simple but key insight that we will repeatedly use is that if P is a rooted path, and u, v are nodes on P where u comes before v (recall that nodes on P are ordered by increasing $c_P(\cdot)$), then replacing P_{uv} by the edge uv reduces the length of the path by exactly $c^{\mathsf{reg}}_u(P_{uv}) = c(P_{uv}) - c_{uv}$.

Point-to-point (P2P) orienteering. In P2P orienteering, we have a start node s, *end node* t, and a length bound B, and we seek an s-t path P with $c(P) \leq B$ that maximizes $\sum_{v \in P} \pi_v$. We will often need to restrict the path to only visit nodes from a certain subset $N \subseteq V'$

(where $\{s,t\} \subseteq N$); we refer to this as P2P orienteering with node set N. It will often be convenient to cast the length-bound B, as a regret-bound of $B - c_{st}$ on the regret of P with respect to s, i.e., $c_s^{\mathsf{reg}}(P)$.

Observe that P2P orienteering can be cast as a special case of deadline TSP (with root node s) by setting $D_v = \max\{0, B - c_{vt}\}$ for all $v \in V$: if w is the end-node of a feasible solution to this deadline-TSP instance, then we can always append the edge wt to this path and stay within the length bound of $B = D_t$.

Recently, Friggstad and Swamy [14] devised an LP-based 6-approximation algorithm for P2P orienteering by formulating a polynomial-size LP for the problem and devising an LP-rounding algorithm (see Section 5). Their algorithm yields a distribution of P2P-orienteering solutions that visits each node with probability proportional to the extent it is visited in the LP solution. Our LP-rounding based algorithm (see subroutine LP-Round) for deadline TSP makes use of the latter type of distributional guarantee. In fact, *any* approximation algorithm for P2P orienteering can be used to obtain such a distribution – this follows from [4, 18] – and this leads to our $(7.63+\epsilon)$-approximation algorithm for deadline TSP.

3 Constant-factor approximation for deadline TSP

We now describe our constant-factor approximation algorithm for deadline TSP that runs in time $O\big(n^{O(\log n\Delta)}\big)$. Let P^* be an optimal path, and $\mathsf{opt} = \pi(P^*)$ be the optimal value. We prove the following result.

▶ **Theorem 3.1.** *Algorithm 1 runs in $O\big(n^{O(\log n\Delta)}\big)$ time. Let $\mathcal{A}$ be an α-approximation algorithm for* P2P *orienteering (where $\alpha \geq 1$).*

- **(a)** *Using subroutine* Greedy *in step D1.2 (with algorithm $\mathcal{A}$), Algorithm 1 returns a deadline-TSP solution whose reward is at least $\frac{1}{3(\alpha+1)} \cdot \mathsf{opt}$.*
- **(b)** *Using subroutine* LP-Round *in step D1.2 (with algorithm $\mathcal{A}$), Algorithm 1 returns a deadline-TSP solution with a slightly better expected reward of at least $\frac{1}{3/(1-e^{-1/\alpha})} \cdot \mathsf{opt} \geq \frac{1}{3(\alpha+1)} \cdot \mathsf{opt}$. This guarantee can be derandomized.*

Taking $\alpha = (2+\epsilon)$ above [6], we obtain a $(9+\epsilon)$-approximation using Greedy, and an improved $(7.63+\epsilon)$-approximation using the LP-approach in LP-Round.

Overview and intuition. We first give an overview of the algorithm and convey the underlying intuition; the detailed description appears below as Algorithm 1. Let $\gamma = 1.5$. Note that $\gamma^2 \leq \gamma + 1$.

Let $R^* = c_r^{\mathsf{reg}}(P^*)$ be the regret of P^* with respect to the root r. Set $u_0 := r$. Suppose we "guess" (i.e., enumerate all possible choices for) a sequence of nodes $w_0, u_1, w_1, \ldots$ occurring on P^* (in this order), which are defined as follows. Given u_i for $i \geq 0$, we define w_i, u_{i+1} as follows. Let u_{i+1} be the first node on P^* after u_i such that the regret of $P^*_{u_i u_{i+1}}$ with respect to u_i is at least γ^i. So we have $c_{u_i}^{\mathsf{reg}}(P_{u_i u_{i+1}}) \geq \gamma^i$ and $c_{u_i}^{\mathsf{reg}}(P^*_{u_i v}) < \gamma^i$ for all nodes v prior to u_{i+1} on $P^*_{u_i u_{i+1}}$; since all distances are integers, this implies that $c_{u_i}^{\mathsf{reg}}(P^*_{u_i v}) \leq \lceil \gamma^i \rceil - 1$ for all nodes v prior to u_{i+1} on $P^*_{u_i u_{i+1}}$. Define w_i to be the predecessor of u_{i+1}. If there is no such node u_{i+1}, then u_{i+1} is undefined, and *define* w_i to be the end-node of P^*. Let k be the largest index such that u_k is well defined. Observe that $k = O(\log R^*)$ since for every $i = 0, \ldots, k-1$, we have that $\gamma^i \leq c_{u_i}^{\mathsf{reg}}(P^*_{u_i u_{i+1}}) \leq c_r^{\mathsf{reg}}(P^*_{u_i u_{i+1}})$, and $c_r^{\mathsf{reg}}(P^*) = R^*$. This leads to the $O\big(n^{O(\log n\Delta)}\big)$ running time, since we need to consider $n^{O(\log R^*)}$ guesses for $w_0, \ldots, u_k, w_k$, and $R^* \leq c(P^*) \leq n\Delta$.

We first observe that we can obtain the following lower bound on the deadlines of nodes in $P^*_{u_i w_i}$. The proof of the following lemma is deferred to the analysis.

▶ **Lemma 3.2.** *Consider any index $i = 0, \ldots, k$. The visiting time of a node $v \in P^*_{u_i w_i}$, and hence its deadline D_v, is at least* $\mathsf{lb}_{i,v} := \sum_{j=0}^{i-1} \max\{c_{u_j w_j} + c_{w_j u_{j+1}}, c_{u_j u_{j+1}} + \gamma^j\} + c_{u_i v}$.

Lemma 3.2 implies that the P2P-orienteering instance with node-set $N^i := \{v \in V' : D_v \geq \mathsf{lb}_{i,v}\}$, start node u_i, end node w_i, and regret-bound $\lceil \gamma^i \rceil - 1$ (with respect to u_i), has optimal value at least $\pi(P^*_{u_i w_i})$. (Note that $N^0 = V'$.) Suppose that we are able to find paths $Q^0, Q^1, \ldots, Q^k$, such that:

- **(i)** for every $i = 0, \ldots, k$, we have that Q^i is a u_i-w_i path, visits only nodes of N^i, and $c^{\mathsf{reg}}_{u_i}(Q^i) \leq \lceil \gamma^i \rceil - 1$; and
- **(ii)** $\pi\big(Q^0 \cup \ldots \cup Q^k\big) \geq \rho \cdot \pi(P^*)$, where $0 < \rho \leq 1$ is some constant.

We show that we can use these paths to obtain a deadline-TSP solution of value $\rho \cdot \pi(P^*)/3$; this yields an $O(1)$-approximation for deadline TSP.

Let Z be the path obtained by concatenating all (the nodes of) $Q^0, \ldots, Q^k$. How "far" is Z from being a feasible solution? Assume that the Q^is are node-disjoint (which we can always ensure by shortcutting past all occurrences of a node other than its first occurrence). Consider a node $v \in Q^i$. We can upper bound the visiting time of v by

$$\begin{cases} \sum_{j=0}^{i-1} (c_{u_j w_j} + \lceil \gamma^j \rceil - 1 + c_{w_j u_{j+1}}) & \text{if } v = u_i; \\ \sum_{j=0}^{i-1} (c_{u_j w_j} + \lceil \gamma^j \rceil - 1 + c_{w_j u_{j+1}}) + c_{u_i v} + \lceil \gamma^i \rceil - 1 & \text{otherwise,} \end{cases}$$

where in the latter case, $c_{u_i v} + \lceil \gamma^i \rceil - 1$ upper bounds the time taken to go from u_i to v along Z (as $\lceil \gamma^i \rceil - 1 \geq c^{\mathsf{reg}}_{u_i}(Q^i) \geq c^{\mathsf{reg}}_{u_i}(Q^i_{u_i v})$). For $v \in Q^0$, this shows that its visiting time is at most $c_{u_0 v} = \mathsf{lb}_{0,v}$, which is at most D_v. Nodes in $Q^1 \cup \ldots \cup Q^k$ may however be visited after their deadlines.

The chief insight is that if we replace the u_j-u_{j+1} portion of Z, which currently consists of the node-sequence Q^j, by the direct edge $u_j u_{j+1}$, then we incur a γ^j-savings in the (above upper bound for) visiting times of nodes on Q^i for $i > j$: the term $c_{u_j w_j} + \lceil \gamma^j \rceil - 1 + c_{w_j u_{j+1}}$ in the above upper bound gets replaced by $c_{u_j u_{j+1}}$, and $c^{\mathsf{reg}}_{u_j}(P^*_{u_j w_j}) \leq \lceil \gamma^j \rceil - 1$ implies that

$$c_{u_j w_j} + \lceil \gamma^j \rceil - 1 + c_{w_j u_{j+1}} \geq c(P^*_{u_j w_j}) + c_{w_j u_{j+1}} = c(P^*_{u_j u_{j+1}}) = c^{\mathsf{reg}}_{u_j}(P^*_{u_j u_{j+1}}) + c_{u_j u_{j+1}} \geq \gamma^j + c_{u_j u_{j+1}}.$$

Moreover, since $\gamma^2 \leq \gamma + 1$, this implies that if we "sync up" with P^* at w_{j-1} by visiting w_{j-1} by time $\mathsf{lb}_{j-1,w_{j-1}}$, then deleting the u_j-u_{j+2} portion – i.e., going directly to u_{j+2} from w_{j-1} – ensures that: (a) every $v \in Q^{j+2}$ is visited by its deadline (and, in fact, by $\mathsf{lb}_{j+2,v}$), and (b) we remain in sync with P^* at w_{j+2}. Note also that Z is in sync with P^* at nodes u_0, w_0. Finally, for nodes in Q^1, it suffices to replace the u_0-u_1 portion of Z by the direct edge $u_0 u_1$ in order to visit these nodes by their deadlines, and have w_1 sync up with P^*.

The upshot of these insights is that: (1) for any $\ell \in \{0, 1, 2\}$, the path $Z^{(\ell)}$ given by the node-sequence $r, \{Q^j\}_{0 \leq j \leq k : j = \ell \bmod 3}$ (where r is possibly a duplicated node) is feasible (Lemma 3.3); and (2) together these paths cover all the nodes of Z. Hence, the best of these 3 paths collects reward at least $\pi\big(Q^0 \cup \ldots \cup Q^k\big)/3 \geq \frac{\rho}{3} \cdot \pi(P^*)$.

Finally, we discuss two approaches for finding the $Q^0, \ldots, Q^k$ paths. The first approach is based on observing that the problem of finding these paths to maximize $\pi\big(Q^0 \cup \ldots \cup Q^k\big)$ is an instance of the *maximum coverage problem with group budgets* considered by [7]. Thus, a simple greedy approach (subroutine Greedy) works, where we repeatedly find $Q^0, Q^1, \ldots$ in that order, and to find Q^i, we use a P2P-orienteering α-approximation algorithm with the subset of N^i that has not been covered by Q^j for $j < i$ (or equivalently, we zero out

the rewards of nodes in $Q^0 \cup \ldots \cup Q^{i-1}$). Chekuri and Kumar [7] show that this yields a collection of paths that together obtain reward at least $\mathsf{opt}/(\alpha+1)$; we include the analysis for completeness (Lemma 3.4).

The second approach is a more sophisticated LP-based approach (subroutine LP-Round) that yields a better guarantee. We write a *configuration LP* (Ap-P) to find the Q^i paths. We use variables x^i_v to denote the extent to which v lies in $P^*_{u_i w_i}$, for $i \geq 0$. Let $\mathcal{P}^i := \mathcal{P}^i(u_i, w_i, N^i, \lceil \gamma^i \rceil - 1)$ be the collection of simple u_i-w_i paths of length at most $c_{u_i w_i} + \lceil \gamma^i \rceil - 1$ that visit only nodes of N^i. (Note that if u_i or w_i is not in N^i, then $\mathcal{P}^i = \emptyset$. Also, if $u_i = w_i$ and lies in N^i, then $\mathcal{P}^i$ consists of only the trivial singleton path $\{u_i\}$.) We also have variables $\{z^i_P\}_{P \in \mathcal{P}^i}$, where z^i_P denotes that we choose path $P \in \mathcal{P}^i$. While we cannot solve this LP optimally, we show (see Lemma 3.6) that an α-approximation algorithm for P2P orienteering can be used to obtain a solution $(\overline{x}, \overline{z})$ of objective value at least $OPT_{\text{Ap-P}}$, where each node v is covered by the paths from $\mathcal{P}^i$ to an extent of at least $\overline{x}^i_v/\alpha$. We choose $Q^i \in \mathcal{P}^i$ by sampling from the $\{\overline{z}^i_P\}_{\in \mathcal{P}^i}$ distribution. It is not hard to argue then that $\mathrm{E}\left[\pi\left(\bigcup_{i=2}^k Q^i\right)\right] \geq \left(1 - e^{-1/\alpha}\right) OPT_{\text{Ap-P}}$ (Lemma 3.7), which yields the guarantee stated in Theorem 3.1 (b). We now describe the algorithm in detail and proceed to analyze it.

Algorithm 1 Deadline TSP

Let $\gamma = 1.5$.

Input: metric $(\{r\} \cup V, c)$, deadlines $\{D_v\}_{v \in V}$, rewards $\{\pi_v\}_{v \in V}$; an α-approximation algorithm $\mathcal{A}$ for P2P orienteering.

Output: An r-rooted path P such that every $v \in P$ is visited by time D_v.

D1 Initialize $\mathcal{Q} \leftarrow \emptyset$. Let $u_0 := r$. For $k = 0, 1, \ldots, \frac{\log n\Delta}{\log \gamma}$, and every choice of nodes $w_0, u_1, w_1, \ldots, u_k, w_k$, perform the following steps.

- **D1.1** For $i = 0, \ldots, k$, and $v \in V'$, define $\mathsf{lb}_{i,v} := \sum_{j=0}^{i-1} \max\{c_{u_j w_j} + c_{w_j u_{j+1}}, c_{u_j u_{j+1}} + \gamma^j\} + c_{u_i v}$. Define $N^i := \{v \in V' : D_v \geq \mathsf{lb}_{i,v}\}$. If $u_i \notin N^i$ or $w_i \notin N^i$ for some $i \in \{0, \ldots, k\}$, then reject this guess – i.e., omit steps 12–14 – and move on to the next choice of u_i, w_i nodes.
- **D1.2** Use subroutine Greedy, or subroutine LP-Round below to obtain paths $Q^0, Q^1, \ldots, Q^k$, where each Q^i is a u_i-w_i path with $c^{\mathsf{reg}}_{u_i}(Q^i) \leq \lceil \gamma^i \rceil - 1$ visiting only nodes of N^i.
- **D1.3** For $\ell \in \{0, 1, 2\}$, let $Z^{(\ell)}$ be the path given by the node-sequence $r, \{Q^j\}_{0 \leq j \leq k : j = \ell \bmod 3}$.
- **D1.4** Add the path $Z^{\max} \in \{Z^{(0)}, Z^{(1)}, Z^{(2)}\}$ that gathers maximum reward to $\mathcal{Q}$.

D2 Return the best solution found in $\mathcal{Q}$, shortcutting the path to retain only the first occurrence of each node.

Subroutine Greedy.

G1 For $i = 0, \ldots, k$, use algorithm $\mathcal{A}$ to (approximately) solve the P2P-orienteering instance with start node u_i, end node w_i, node-set $N^i \setminus \left(\bigcup_{j=0}^{i-1} P^j\right)$, and length bound $c_{u_i w_i} + \lceil \gamma^i \rceil - 1$ (and node rewards $\{\pi_v\}_{v \in V'}$), to obtain a simple path P^i (so if $u_i = w_i$, then $P^i = \{u_i\}$ or $P^i = \emptyset$).
Return paths $P^0, \ldots, P^k$.

Subroutine LP-Round.

L1 Let $\mathcal{P}^i := \mathcal{P}^i(u_i, w_i, N^i, \lceil \gamma^i \rceil - 1)$ denote the collection of simple u_i-w_i paths with $c^{\mathsf{reg}}_{u_i}$-length at most $\lceil \gamma^i \rceil - 1$ (so the c-length is at most $c_{u_i w_i} + \lceil \gamma^i \rceil - 1$) and visiting only nodes from N^i. Note that if $u_i = w_i$ and $u_i \in N^i$, then $\mathcal{P}^i$ consists of only the trivial singleton path $\{u_i\}$. Consider the following LP.

$$\max \quad \sum_{i=0}^{k} \sum_{v \in V} \pi_v x^i_v \tag{Ap-P}$$

$$\text{s.t.} \quad \sum_{P \in \mathcal{P}^i} z^i_P \le 1 \qquad \forall i = 0, \dots, k \tag{1}$$

$$x^i_v \le \sum_{P \in \mathcal{P}^i : v \in P} z^i_P \qquad \forall v \in V,\ \forall i = 0, \dots, k \tag{2}$$

$$\sum_{i=0}^{k} x^i_v \le 1 \qquad \forall v \in V \tag{3}$$

$$x^i_{u_i} = x^i_{w_i} = 1 \qquad \forall i = 0, \dots, k \tag{4}$$

$$x, z \ge 0.$$

L2 Use Lemma 3.6 to obtain in polytime a solution $(\overline{x}, \overline{z})$ (with polynomial-size support) such that: (i) $\sum_{i=0}^{k} \sum_{v \in V} \pi_v \overline{x}^i_v \ge OPT_{\text{Ap-P}}$, (ii) $\sum_{P \in \mathcal{P}^i : v \in P} \overline{z}^i_P \ge \overline{x}^i_v / \alpha$ for all $v \in V$ and $i = 0, \dots, k$, and (iii) $(\overline{x}, \overline{z})$ satisfies the remaining constraints of (Ap-P).

L3 For each $i = 0, \dots, k$, sample a random path $P^i \in \mathcal{P}^i$ from the $\{\overline{z}^i_P\}_{P \in \mathcal{P}^i}$ distribution. Return paths $P^0, \dots, P^k$.

Analysis. The running time stated in Theorem 3.1 follows since $k = O(\log n\Delta)$, and for each k, we enumerate over all sequences of $2k$ nodes.

Recall that P^* is an optimal solution. Let $\mathsf{opt} = \pi(P^*)$ be the optimal value. We will assume in the sequel that we have the right choice of k and the u_i, w_i nodes. Recall that $u_0 = r$. That is, w_0 is the last node on P^* such that $c^{\mathsf{reg}}_{u_0}(P_{u_0 w_0}) = 0$. Given u_i, for $i \ge 0$, we have that u_{i+1} is the first node v on P^* after u_i such that $c^{\mathsf{reg}}_{u_i}(P_{u_i v}) \ge \gamma^i$, and w_i is the predecessor of u_{i+1}. If there is no such node u_{i+1}, then u_{i+1} is undefined and w_i is the the end-node of P^*. Also, k is the largest index such that u_k is well defined. Note that $k \ge 0$.

We begin by proving Lemma 3.2.

Proof of Lemma 3.2. For any $j \in \{0, \dots, i-1\}$, we have that $c(P^*_{u_j u_{j+1}}) \ge c(P^*_{u_j w_j}) + c_{w_j u_{j+1}} \ge c_{u_j w_j} + c_{w_j u_{j+1}}$ and also $c(P^*_{u_j u_{j+1}}) = c^{\mathsf{reg}}_{u_j}(P^*_{u_j u_{j+1}}) + c_{u_j u_{j+1}} \ge \gamma^j + c_{u_j u_{j+1}}$, where the inequality follows from the definition of u_{j+1}. The visiting time of $v \in P^*_{u_i w_i}$ is $c_{r u_0} + \sum_{j=0}^{i-1} c(P^*_{u_j u_{j+1}}) + c_{u_i v}$, which, using the above bounds, is at least $\mathsf{lb}_{i,v}$. ◀

Next, Lemma 3.3 shows that the paths $Z^{(\ell)}$ obtained in step D1.3 are feasible. Since these paths together cover $\bigcup_{i=0}^{k} Q^i$, the path $Z^{\max}$ returned in step D1.4 gathers reward at least $\pi\big(\bigcup_{i=0}^{k} Q^i\big)/3$.

▶ **Lemma 3.3.** *Consider $\ell \in \{0, 1, 2\}$, and the path $Z^{(\ell)}$ computed in step D1.3. For every Q^j that is part of $Z^{(\ell)}$, and every $v \in Q^j$, the visiting time of this occurrence of v in $Z^{(\ell)}$ is at most $\mathsf{lb}_{j,v} \le D_v$.*

Proof. Consider any Q^j that is part of $Z^{(\ell)}$. So $0 \le j \le k$ and $j = \ell \bmod 3$. We argue by induction on j that the visiting time (in $Z^{(\ell)}$) of every node $v \in Q^j$ is at most $\mathsf{lb}_{j,v}$.

For the base case, when $j = \ell$, the visiting time of any $v \in Q^\ell$ is at most $c_{u_0u_\ell} + c_{u_\ell v} + \lceil\gamma^\ell\rceil - 1$ since $c^{\mathsf{reg}}_{u_\ell}(Q^\ell_{u_\ell v}) \le c^{\mathsf{reg}}_{u_\ell}(Q^\ell) \le \lceil\gamma^\ell\rceil - 1$. We can check by inspection that this bound is always at most $\mathsf{lb}_{\ell,v}$:

- for $\ell = 0$, the bound is $c_{u_0v} = \mathsf{lb}_{0,v}$;
- for $\ell = 1$, the bound is $c_{u_0u_1} + c_{u_1v} + 1 \le \mathsf{lb}_{1,v}$;
- for $\ell = 2$, the bound is at most $c_{u_0u_2} + c_{u_2v} + \gamma^2 \le (c_{u_0u_1} + 1) + (c_{u_1u_2} + \gamma) + c_{u_2v} \le \mathsf{lb}_{2,v}$.

Now suppose $j > \ell$. The visiting time of any $v \in Q^j$ is at most (visiting time of w_{j-3}) + $c_{w_{j-3}u_j} + c_{u_jv} + \gamma^j$ since the $c^{\mathsf{reg}}_{u_j}$-length of $Q^j_{u_jv}$ is at most $\lceil\gamma^j\rceil - 1 \le \gamma^j$. By our induction hypothesis, the visiting time of w_{j-3} is at most $\mathsf{lb}_{j-3,w_{j-3}}$. Since $\gamma^j \le \gamma^{j-2} + \gamma^{j-1}$, we can upper bound the visiting time of v by

$$\big(\mathsf{lb}_{j-3,w_{j-3}} + c_{w_{j-3}u_{j-2}}\big) + \big(c_{u_{j-2}u_{j-1}} + \gamma^{j-2}\big) + \big(c_{u_{j-1}u_j} + \gamma^{j-1}\big) + c_{u_jv} \le \mathsf{lb}_{j,u_j} + c_{u_jv} \le \mathsf{lb}_{j,v}.$$

The first inequality is because for any index $i \ge 1$, we have $\mathsf{lb}_{i,u_i} \ge \mathsf{lb}_{i-1,w_{i-1}} + c_{w_{i-1}u_i}$ and $\mathsf{lb}_{i,u_i} \ge \mathsf{lb}_{i-1,u_{i-1}} + c_{u_{i-1}u_i} + \gamma^{i-1}$. This completes the induction step and proves the lemma. ◀

Finally, we prove guarantees for the paths returned by subroutine Greedy and subroutine LP-Round.

▶ **Lemma 3.4** (Follows from [7]). *The paths $Q^0, \ldots, Q^k$ returned if we use subroutine* Greedy *in step D1.2 satisfy $\pi\big(Q^0 \cup \ldots \cup Q^k\big) \ge \mathsf{opt}/(\alpha+1)$.*

Proof. Clearly, for each $i = 0, \ldots, k$, we have that $P^*_{u_iw_i} \setminus \big(\bigcup_{j=0}^{i-1} Q^j\big)$ is a feasible solution to the P2P-orienteering instance that is fed as input to algorithm $\mathcal{A}$ in iteration i. So for each $i = 0, \ldots, k$, we have

$$\pi\Big(Q^i \setminus \big(\bigcup_{j=0}^{i-1} Q^j\big)\Big) \ge \frac{1}{\alpha} \cdot \pi\Big(P^*_{u_iw_i} \setminus \big(\bigcup_{j=0}^{i-1} Q^j\big)\Big). \tag{5}$$

Adding the above for $i = 0, \ldots, k$ yields an inequality whose LHS is $\pi\big(Q^0 \cup \ldots \cup Q^k\big)$, and whose RHS is at least $\frac{1}{\alpha} \cdot \pi\big(P^* \setminus (Q^0 \cup \ldots \cup Q^k)\big) \ge \frac{1}{\alpha} \cdot \big[\pi(P^*) - \pi\big(Q^0 \cup \ldots \cup Q^k\big)\big]$. It follows that $\pi\big(Q^0 \cup \ldots \cup Q^k\big) \ge \pi(P^*)/(\alpha+1)$. ◀

Combining Lemma 3.4 with Lemma 3.3 leads to the proof of Theorem 3.1 (a).

Part (b) of Theorem 3.1

We now analyze the paths returned by subroutine LP-Round and prove Theorem 3.1 (b).

We first observe in Claim 3.5 that $OPT_{\text{Ap-P}} \ge \pi(P^*) = \mathsf{opt}$. Lemma 3.6 shows that using our P2P-orienteering approximation algorithm, we can obtain in polytime a solution $(\overline{x}, \overline{z})$ satisfying the properties stated in step L2. Given this, Lemma 3.7 proves that the random paths $Q^0, \ldots, Q^k$ returned by subroutine LP-Round satisfy $\mathrm{E}\big[\pi\big(\bigcup_{i=0}^k Q^i\big)\big] \ge \big(1 - e^{-1/\alpha}\big) OPT_{\text{Ap-P}}$. This yields the randomized guarantee stated in Theorem 3.1 (b). We then show how to derandomize the algorithm without affecting its guarantee or running time.

The following claim simply observes that, for each $i = 0, \ldots, k$, setting $z^i_{P^*_{u_iw_i}} = 1$ and x^i to be the indicator vector of the node-set of $P^*_{u_iw_i}$ yields a feasible solution to (Ap-P).

▷ Claim 3.5. We have $OPT_{\text{Ap-P}} \ge \pi(P^*)$.

Lemma 3.6 proves the main result regarding the polytime solvability of (Ap-P). Its proof is a bit technical, and is deferred to the end of this section.

▶ **Lemma 3.6.** *Using algorithm $\mathcal{A}$, we can obtain in polynomial time a solution $(\overline{x}, \overline{z})$ such that: (i) $\sum_{i=0}^{k} \sum_{v \in V} \pi_v \overline{x}_v^i \geq OPT_{Ap\text{-}P}$, (ii) $\sum_{P \in \mathcal{P}^i : v \in P} \overline{z}_P^i \geq \overline{x}_v^i / \alpha$ for all $v \in V$ and $i = 0, \ldots, k$, and (iii) $(\overline{x}, \overline{z})$ satisfies the remaining constraints of* (Ap-P).

▶ **Lemma 3.7.** *The paths $Q^0, \ldots, Q^k$ returned if we use subroutine* LP-Round *in step D1.2 satisfy $\mathrm{E}\big[\pi\big(\bigcup_{i=0}^{k} Q^i\big)\big] \geq \big(1 - e^{-1/\alpha}\big) OPT_{Ap\text{-}P}$.*

Proof. Defining $\rho_v := \Pr[v \in (Q^0 \cup \ldots \cup Q^k)]$ for $v \in V$, we have $\mathrm{E}\big[\pi\big(\bigcup_{i=0}^{k} Q^i\big)\big] = \sum_{v \in V} \pi_v \rho_v$. The paths $Q^0, \ldots, Q^k$ are chosen independently, and $\Pr[v \in Q^i] \geq \overline{x}_v^i / \alpha$ for each $v \in V$ (by Lemma 3.6 (ii)). So for every $v \in V$, we have

$$\rho_v \geq 1 - \prod_{i=0}^{k} \Big(1 - \tfrac{\overline{x}_v^i}{\alpha}\Big) \geq 1 - \Big(1 - \tfrac{\sum_{i=0}^{k} \overline{x}_v^i / \alpha}{k}\Big)^k \geq \Big[1 - \Big(1 - \tfrac{1}{\alpha k}\Big)^k\Big] \cdot \sum_{i=0}^{k} \overline{x}_v^i \geq \big(1 - e^{-1/\alpha}\big) \sum_{i=0}^{k} \overline{x}_v^i$$

which proves the lemma. The second inequality above follows from the AM-GM inequality, and the third uses the fact that the function $f(x) = 1 - \big(1 - \frac{x}{k}\big)^k$ is concave, and so $f(x) \geq x \cdot \frac{f(1/\alpha) - f(0)}{1/\alpha}$ for all $x \leq 1/\alpha$. ◀

Finishing up the proof of Theorem 3.1 (b). We first prove the randomized guarantee. The expected value of the solution returned is $\mathrm{E}\big[\max_{P \in \mathcal{Q}} \pi(P)\big]$, which is at least $\max_{P \in \mathcal{Q}} \mathrm{E}\big[\pi(P)\big]$. We lower bound the latter quantity by focusing on the path in $\mathcal{Q}$ returned for the right choice of k and the u_i-w_i nodes.

Fixing this choice, Claim 3.5 and Lemma 3.7 show that $\mathrm{E}\big[\pi\big(\bigcup_{i=0}^{k} Q^i\big)\big] \geq \big(1 - e^{-1/\alpha}\big)\pi(P^*)$. Since the $Z^{(\ell)}$ paths for $\ell = 0, 1, 2$ are feasible (Lemma 3.3) and together cover $\bigcup_{i=0}^{k} Q^i$, the path $Z^{\max}$ returned in step D1.4 satisfies $\mathrm{E}\big[\pi(Z^{\max})\big] \geq \frac{1 - e^{-1/\alpha}}{3} \cdot \pi(P^*)$.

Derandomization. The above guarantee can be easily derandomized. We use randomization only in sampling, for each $i = 0, \ldots, k$ independently, a random path Q^i from a polynomial-size distribution of u_i-w_i paths. Let $\mathcal{C}^i$ denote the support of this distribution, and for $P \in \mathcal{C}^i$, let λ_P^i denote the probability of choosing path P. The quantity of interest that determines the performance guarantee is $\Phi_0 = \Phi(\lambda^0, \ldots, \lambda^k) := \mathrm{E}\big[\pi\big(\bigcup_{i=0}^{k} Q^i\big)\big]$. For $P \in \mathcal{C}^i$, let $\mathbb{1}_P$ be the distribution that chooses P deterministically with probability 1. We show how to deterministically choose the Q^is so that $\Phi(\mathbb{1}_{Q^0}, \ldots, \mathbb{1}_{Q^k}) \geq \Phi_0$. We have $\Phi = \sum_{v \in V} \pi_v \rho_v$, where

$$\rho_v = \rho_{0,v} + (1 - \rho_{0,v})\rho_{1,v} + \ldots + (1 - \rho_{0,v})(1 - \rho_{1,v}) \cdots (1 - \rho_{k-1}, v)\rho_{k,v},$$
$$\text{and} \quad \rho_{i,v} = \rho_{i,v}(\lambda^0, \ldots, \lambda^k) = \sum_{P \in \mathcal{C}^i} \lambda_P^i \quad \forall i = 0, \ldots, k.$$

It is evident that each ρ_v is linear in λ^i, and therefore Φ is linear in λ^i. Therefore, to derandomize: (1) we choose $Q^0 \in \mathcal{C}^0$ so that $\Phi(\mathbb{1}_{Q^0}, \lambda^1, \ldots, \lambda^k) \geq \Phi_0$; (2) given that we have chosen $Q^0, \ldots, Q^{i-1}$, we choose $Q^i \in \mathcal{C}^i$ so that $\Phi(\mathbb{1}_{Q^0}, \ldots, \mathbb{1}_{Q^i}, \lambda^{i+1}, \ldots, \lambda^k) \geq \Phi(\mathbb{1}_{Q^0}, \ldots, \mathbb{1}_{Q^{i-1}}, \lambda^i, \ldots, \lambda^k)$. ◀

Proof of Lemma 3.6. Since (Ap-P) has an exponential number of variables, we consider the dual (Ap-D). The dual has polynomially many constraints corresponding to the polynomially many x_v^i primal variables, and an exponential number of constraints corresponding to the z_P^i primal variables. Using standard ideas (see, e.g., [13]), we show that we can use algorithm $\mathcal{A}$ to approximately separate over these exponentially many constraints, and hence leverage

the ellipsoid method to obtain the desired primal solution. Let $\theta^i \geq 0$ and $\mu_v^i \geq 0$ denote respectively the dual variables corresponding to constraints (1), (2). The dual constraints corresponding to the z_P^i variables are:

$$\sum_{v \in P} \mu_v^i \leq \theta^i \qquad \forall i = 0, \ldots, k, \ \forall P \in \mathcal{P}^i \tag{P2P}$$

For $b \geq 0$, we let (P2P$_b$) denote (P2P) with the RHS changed to θ^i/b. The effect of this on the primal is that it changes the RHS of (1) to b; let (Ap-P$_b$) denote (Ap-P) with this modified version of (1).

We focus on constraints (P2P) and do not explicitly write down the remaining (polynomially many) dual constraints (including nonnegativity constraints); we collectively denote these constraints by (Ap-D-*). Letting β denote the remaining dual variables, the objective function of (Ap-D) is of the form $\sum_{i=0}^k \theta^i + h^T\beta$, where h is a fixed vector.

Define $\mathcal{K}(\nu; b) := \left\{(\beta, \mu, \theta) : \mu, \theta \geq 0, \ \text{(Ap-D-*)}, \ \text{(P2P}_b\text{)}, \ \sum_{i=0}^k \theta^i + h^T\beta \leq \nu\right\}$. Note that the optimal value of the dual, and hence (Ap-P), is the smallest ν such that $\mathcal{K}(\nu; 1) \neq \emptyset$. Given ν, (β, μ, θ), we can use $\mathcal{A}$ to either show that $(\beta, \mu, \theta) \in \mathcal{K}(\nu; 1)$, or find a hyperplane separating (β, μ, θ) from $\mathcal{K}(\nu; \alpha)$. We first check if $\mu, \theta \geq 0$, (Ap-D-*) and $\sum_{i=0}^k \theta^i + h^T\beta \leq \nu$ hold, and if not use the violated inequality as the separating hyperplane. Next, for each $i = 0, \ldots, k$, we run $\mathcal{A}$ on the P2P-orienteering instance with start and end nodes u_i, w_i respectively, length bound $c_{u_i w_i} + \lceil \gamma^i \rceil - 1$, and node-rewards $\{\mu_v^i\}_{v \in V'}$. If for some i, we obtain a path $P \in \mathcal{P}^i$ with reward greater than θ^i/α, then we return $\sum_{v \in P} \mu_v^i \leq \theta^i/\alpha$ as the separating hyperplane. Otherwise, for all $i = 0, \ldots, k$ and all $P \in \mathcal{P}^i$, we know that $\sum_{v \in P} \mu_v^i \leq \theta^i$, and so $(\beta, \mu, \theta) \in \mathcal{K}(\nu; 1)$. Thus, for a fixed ν, in polynomial time, the ellipsoid method either certifies that $\mathcal{K}(\nu; \alpha) = \emptyset$, or returns a point $(\beta, \mu, \theta) \in \mathcal{K}(\nu; 1)$.

It is easy to find an upper bound UB such that $\mathcal{K}(\text{UB}; 1) \neq \emptyset$. For a given $\epsilon > 0$, we use binary search in the range $[0, \text{UB}]$ to find ν^* such that the ellipsoid method when run for ν^* (with the above separation oracle) returns $(\beta^*, \mu^*, \theta^*) \in \mathcal{K}(\nu^*; 1)$, and when run for $\nu^* - \epsilon$ certifies that $\mathcal{K}(\nu^* - \epsilon; \alpha) = \emptyset$. Since $\mathcal{K}(\nu^*; 1) \neq \emptyset$, we have that $OPT_{\text{Ap-P}} \leq \nu^*$, and $\mathcal{K}(\nu^* - \epsilon; \alpha) = \emptyset$ implies that the optimal value of (Ap-P$_\alpha$) is at least $\nu^* - \epsilon$. For $\nu^* - \epsilon$, the inequalities returned by the separation oracle during the execution of the ellipsoid method together with the inequality $\sum_{i=0}^k \theta^i + h^T\beta \leq \nu^* - \epsilon$ yield a polynomial-size certificate for the emptiness of $\mathcal{K}(\nu^* - \epsilon; \alpha)$. By duality (or Farkas' lemma), this implies that if we restrict (Ap-P$_\alpha$) to only use the (polynomially many) z_P^i variables corresponding to the violated inequalities of type (P2P$_\alpha$) returned during the execution of the ellipsoid method, we obtain a polynomial-size feasible solution $(\overline{x}, \hat{z})$ to (Ap-P$_\alpha$) of value at least $\nu^* - \epsilon$. If we take ϵ to be inverse exponential in the input size, this also implies $(\overline{x}, \hat{z})$ has value at least $\nu^* \geq OPT_{\text{Ap-P}}$. Finally, setting $\overline{z} = \hat{z}/\alpha$, we obtain that $(\overline{x}, \overline{z})$ has the desired properties. ◀

▶ Remark 3.8. It is worthwhile to note that *none* of our arguments above rely on the symmetry of the underlying metric, and so *the reduction in Theorem 3.1 also holds in asymmetric metrics*. Given an asymmetric metric $\{c_{u,v}\}_{u,v \in V \cup \{r\}}$, we define regret distances in the same way – $c_u^{\text{reg}}(v, w) := c_{u,v} + c_{v,w} - c_{u,w}$ – and they continue to form an asymmetric metric.

▶ Remark 3.9. In step D1, we only need to let k go up to $\log_\gamma D_{\max}$ (where $D_{\max} := \max_v D_v \leq n\Delta$), and so the running time is $O\left(n^{\log D_{\max}}\right)$. Also, for any integer $c \geq 1$, we can obtain an $O(c)$-approximation in $O\left(c \cdot n^{\frac{\log D_{\max}}{c}}\right)$ time, as follows. We divide the indices $0, 1, \ldots, k$ (where $k \leq \log_\gamma D_{\max}$) groups of (roughly) $\frac{k}{c}$ consecutive indices, essentially run our algorithm for each group *separately*, and return the best solution found. To elaborate, for a group $\{a, a+1, \ldots, b\}$, we guess the corresponding nodes $u_a, w_a, \ldots, u_b, w_b$, and obtain a

u_i-w_i path Q^i for each $i \in \{a, \ldots, b\}$ such that $c^{\mathsf{reg}}_{u_i}(Q^i) \leq \lceil \gamma^i \rceil - 1$, and $(Q^a \cup \ldots \cup Q^b)$ obtains reward $\Omega\big(\pi(P^*_{u_a w_b})\big)$. Since we are considering each group in isolation and do not know the u_j, w_j nodes for $j < a$, we need to define $\mathsf{lb}_{i,v}$ differently now (which then specifies the node-set N^i that Q^i is allowed to visit); we now define $\mathsf{lb}_{i,v} := \sum_{j=0}^{a-1} \gamma^j + \sum_{j=a}^{i-1} \max\{c_{u_j w_j} + c_{w_j u_{j+1}}, c_{u_j u_{j+1}} + \gamma^j\} + c_{u_i v}$. Similar to before, for $\ell \in \{0, 1, 2\}$, we define $Z^{(\ell)}$ as the path with node-sequence r, $\{Q^j\}_{a \leq j \leq b : j-a = \ell \bmod 3}$, and it is not hard to mimic the earlier arguments to infer that each $Z^{(\ell)}$ is a feasible deadline-TSP solution.

Note that taking $c = \log_\gamma D_{\max}$, this shows that the best of the r, Q^i-paths, for $i = 0, \ldots, \log_\gamma D_{\max}$ yields a (very simple) $O(\log D_{\max})$-approximation in polynomial time.

▶ Remark 3.10. There are a couple of ways of improving the efficiency of algorithm LP-Round, while incurring some associated loss of approximation factor. First, as noted earlier, Friggstad and Swamy [14] gave a polynomial-size LP-relaxation for P2P orienteering, and an LP-rounding 6-approximation algorithm for P2P orienteering. Their algorithm explicitly yields the distributional guarantee proved in Lemma 3.6, namely, it returns a (polynomial-support) distribution of P2P-orienteering solutions that visits each node with probability at least $x^i_v/6$, where x^i_v has the same meaning as above (see Section 5). One could replace the exponential-size LP (Ap-P) in algorithm LP-Round with their compact LP, and sample paths from the distribution output by their algorithm. This yields a much more efficient guarantee relative to an LP upper bound, but the approximation guarantee degrades to $\frac{3}{1-e^{-1/6}} \approx 19.542$. (While this is worse than the guarantee obtained using Greedy and the $(2+\epsilon)$-approximation for orienteering [6], the benefit is that this guarantee is with respect to an LP upper bound; also, the orienteering algorithm in [14] is simpler than the one in [6].)

Second, one can replace the use of the ellipsoid method to approximately solve (Ap-P) by the multiplicative-weights method, incurring a small loss in approximation.

4 Extensions

Our techniques can be applied to handle various extensions, including monotone-reward TSP, which we discuss here, and some other extensions that we discuss in Section 4.2.

4.1 Monotone-reward TSP

Recall that in monotone-reward TSP, each node v has a non-increasing reward function $\pi_v : \mathbb{R}_+ \mapsto \mathbb{R}_+$, where we set $\pi_r(x) = 0$ for all x for notational ease. We overload notation, and for a rooted path P, we now *define* $\pi(P) := \sum_{v \in P} \pi_v(c_P(v))$, and call this the reward of P. The goal is to find a simple rooted path that obtains maximum reward. Each $\pi_v(.)$ function is specified via a value oracle, and we treat each call to this oracle as an elementary operation. Recall that we assume that c_{uv} is an integer for all $u, v \in V \cup \{r\}$.

We show that monotone-reward TSP can be reduced to deadline TSP. This reduction incurs a slight loss and increases the size of the instance. We also show that the algorithms and analysis from Section 3 carry over easily to monotone-reward TSP.

▶ **Theorem 4.1.**

(1) *For any $\epsilon > 0$, given a monotone-reward TSP instance $\mathcal{I}$ with n clients, we can obtain in polytime a deadline-TSP instance $\mathcal{I}'$ with $O\big(\frac{n}{\epsilon} \cdot \log \frac{n}{\epsilon}\big)$ clients such that an α-approximation for $\mathcal{I}'$ yields an $\alpha/(1-2\epsilon)$-approximation to $\mathcal{I}$.*

(2) *The algorithms for deadline TSP described in Section 3 can be easily adapted to monotone-reward TSP and yield the same guarantees as those stated in Theorem 3.1.*

Proof of part (1) of Theorem 4.1. Recall that Δ is the diameter of the metric. In any simple rooted path, each client is visited by time $n \cdot \Delta$; so we define $\pi_v(t) = 0$ for all $t > n\Delta$ for notational ease. In the deadline-TSP instance, we use the same metric, but for each node $v \in V$ and every time $t \in \{0, 1, \ldots, n\Delta\}$, we create a co-located client (v, t) with deadline t and reward $\pi_v(t) - \pi_v(t+1)$ (which is nonnegative since $\pi_v(.)$ is non-increasing). Thus, if v is visited at time t in the original graph, then we collect reward $\pi_v(t)$ in total from its co-located clients $(v, t), (v, t+1), \ldots$. This is a lossless, but pseudo-polytime, reduction.

To make this efficient, we apply a geometric bucketing idea on the rewards. Consider a fixed $\epsilon > 0$. Observe that $\mathsf{LB} := \max_{v \in V} \pi_v(c_{rv})$ is the maximum reward that any solution can collect from node v. Also, we can obtain a path with reward at least LB by considering the path r, v for the node v that attains the maximum. We we have $\mathsf{LB} \leq \mathsf{opt}_{\mathcal{I}} \leq n \cdot \mathsf{LB}$, where $\mathsf{opt}_{\mathcal{I}}$ is the optimal value for the monotone-reward TSP instance $\mathcal{I}$. Now for each $v \in V$, instead of creating $n \cdot \Delta$ co-located clients, we consider each integer $i \geq 0$ such that $(1-\epsilon)^i \geq \frac{\epsilon}{n}$; note that there are at most $O\left(\frac{1}{\epsilon} \cdot \log \frac{n}{\epsilon}\right)$ such values. For each such i, we use binary search (using the value oracle) to find the largest integer $t_{v,i}$ such that $\pi_v(t_{v,i}) \geq (1-\epsilon)^i \cdot \mathsf{LB}$. In the deadline TSP instance $\mathcal{I}'$, we create a client located at v with deadline $t_{v,i}$ and reward $(1-\epsilon)^i \cdot \mathsf{LB} - (1-\epsilon)^{i+1} \cdot \mathsf{LB}$. So, if a path reaches location v by time $t_{v,i}$ it will collect reward at least $(1-\epsilon)^i \cdot \mathsf{LB} - \frac{\epsilon}{n} \cdot \mathsf{LB}$.

Let P^* be an optimum monotone-reward TSP solution. Consider the value of P^* as a solution to the new deadline-TSP instance. Consider each v on P^*, and say v was visited at time t along P^*. We know that $\pi_v(t) \leq \mathsf{LB}$. Let i be the smallest integer such that $(1-\epsilon)^i \cdot \mathsf{LB} \leq \pi_v(t)$; so we also have $(1-\epsilon)^i \cdot \mathsf{LB} \geq (1-\epsilon)\pi_v(t)$. By the construction of $\mathcal{I}'$, we know that $t_{v,i} \geq t$, and so we collect total reward at least $(1-\epsilon)^i \cdot \mathsf{LB} - \frac{\epsilon}{n} \cdot \mathsf{LB} \geq (1-\epsilon)\pi_v(t) - \frac{\epsilon}{n} \cdot \mathsf{LB}$ from the clients co-located at v. So the total reward of P^* when viewing it as a deadline-TSP solution is at least $(1-\epsilon) \cdot \mathsf{opt}_{\mathcal{I}} - \epsilon \cdot \mathsf{LB} \geq (1-2\epsilon) \cdot \mathsf{opt}_{\mathcal{I}}$.

Conversely, any deadline-TSP solution P' when viewed as a monotone-reward TSP solution produces at least as much reward, since for any time t and any client v, $\pi_v(t)$ is at least the total reward that will be collected in instance $\mathcal{I}'$ by visiting the clients located at v at time t. Therefore, a solution to instance $\mathcal{I}'$ of value at least $\mathsf{opt}_{\mathcal{I}'}/\alpha$ yields a monotone-reward TSP solution with value at least $\frac{1-2\epsilon}{\alpha} \cdot \mathsf{opt}_{\mathcal{I}}$. ◀

Proof of part (2) of Theorem 4.1

We briefly sketch the changes to the algorithms and analyses from Section 3. The only changes to Algorithm 1 involve changes to the constituent subroutines Greedy and LP-Round, and entail figuring out what *fixed* node rewards to use when we consider a P2P-orienteering instance in step G1, or in LP (Ap-P). (As before, P^* denotes an optimal solution, and the right choice of u_i, w_i nodes continues to be as defined in Section 3; consequently Lemma 3.2 continues to hold. Also, steps D1.4 and D2 hold as is, given our modified definition of the reward of a path.)

Two observations guide the choice of rewards we use in step G1 and in (Ap-P). First, we know that the visiting time of a node $v \in P^*_{u_i w_i}$ is at least $\mathsf{lb}_{i,v}$. Second, suppose for each i, we obtain a u_i-w_i path Q^i with $c^{\mathsf{reg}}_{u_i}(Q^i) \leq \lceil \gamma^i \rceil - 1$ and construct the paths $\{Z^{(0)}, Z^{(1)}, Z^{(2)}\}$ as in step D1.3; then, by Lemma 3.3, for every $Z^{(\ell)}$, every $Q^j \subseteq Z^{(\ell)}$, and every $v \in Q^j$, we know that the visiting time of this occurrence of v is at most $\mathsf{lb}_{i,v}$. Consequently, in subroutine Greedy and subroutine LP-Round, we use the node rewards $\{\pi_v(\mathsf{lb}_{i,v})\}_{v \in V'}$ for the i-th orienteering instance; also, the node-set N^i is now simply V. Let (MRAp-P) denote the resulting analogue of (Ap-P), where the coefficient multiplying x^i_v in the objective is now $\pi_v(\mathsf{lb}_{i,v})$.

Essentially, all the statements proved for deadline TSP hold here as well, with minor notational tweaks, and for mostly the same reasons. (Recall that for a rooted path P, we now define $\pi(P) := \sum_{v \in P} \pi_v(c_P(v))$, and call this the reward of P.)

Algorithm Greedy is unchanged, and the guarantee of Lemma 3.4 still holds, but it needs to be stated more precisely, and its proof needs to be tweaked. The modified statement is that we have $\sum_{i=0}^{k} \sum_{v \in Q^i \setminus (Q^0 \cup \ldots \cup Q^{i-1})} \pi_v(\mathsf{lb}_{i,v}) \geq \mathsf{opt}/(\alpha+1)$. In the proof, in place of (5), for all $i = 0, \ldots, k$, we now have

$$\sum_{v \in Q^i \setminus (\bigcup_{j=0}^{i-1} Q^j)} \pi_v(\mathsf{lb}_{i,v}) \geq \frac{1}{\alpha} \cdot \sum_{\substack{v \in P^*_{u_i w_i} \\ v \notin \bigcup_{j=0}^{i-1} Q^j}} \pi_v(\mathsf{lb}_{i,v}) \geq \frac{1}{\alpha} \cdot \left[\sum_{v \in P^*_{u_i w_i}} \pi_v\big(c_{P^*}(v)\big) - \sum_{\substack{v \in P^*_{u_i w_i} \\ v \in \bigcup_{j=0}^{i-1} Q^j}} \pi_v(\mathsf{lb}_{i,v}) \right] \quad (6)$$

Consider adding, for all $i = 0, \ldots, k$, the inequality in (6) involving the LHS of (6) and the final RHS of (6). The LHS of the resulting inequality is $\sum_{i=0}^{k} \sum_{v \in Q^i \setminus (Q^0 \cup \ldots \cup Q^{i-1})} \pi_v(\mathsf{lb}_{i,v})$. For a node v, consider its total contribution (across all i) to the negative terms on the final RHS of (6). Node v gets counted at most once among all these negative terms, and if i is the smallest index such that $v \in Q^i$, then the negative term where v is counted is due to some index $j > i$, and so is at least $-\pi_v(\mathsf{lb}_{i,v})$. Therefore, the total contribution of the negative terms on the RHS is at least $-\frac{1}{\alpha} \cdot \sum_{i=0}^{k} \sum_{v \in Q^i \setminus (Q^0 \cup \ldots \cup Q^{i-1})} \pi_v(\mathsf{lb}_{i,v})$. Thus, adding (6) for all $i = 0, \ldots, k$ and simplifying yields the inequality $\sum_{i=0}^{k} \sum_{v \in Q^i \setminus (Q^0 \cup \ldots \cup Q^{i-1})} \pi_v(\mathsf{lb}_{i,v}) \geq \mathsf{opt}/(\alpha+1)$.

Subroutine LP-Round is also unchanged, except for the change in (Ap-P) due to the above-mentioned node rewards and since $N^i = V$. The new LP (MRAp-P) can be approximately solved as in Lemma 3.6. Let $\overline{x}$ denote the values of the x_i^v variables in the solution so obtained. Then we have $\overline{z}$ such that the objective value of $(\overline{x}, \overline{z})$ is at least $OPT_{\text{(MRAp-P)}}$, $\sum_{P \in \mathcal{P}^i : v \in P} \overline{z}^i_P \geq \overline{x}^i_v/\alpha$ for all $v \in V$ and $i = 2, \ldots, k$, and $(\overline{x}, \overline{z})$ satisfies the remaining constraints of (MRAp-P).

Claim 3.5 gets replaced by

$$\sum_{i=0}^{k} \sum_{v \in V} \pi_v(\mathsf{lb}_{i,v}) \overline{x}^i_v \geq \sum_{i=0}^{k} \sum_{v \in P^*_{u_i w_i}} \pi_v(\mathsf{lb}_{i,v}) \geq \sum_{v \in P^*} \pi_v(c_{P^*}(v)) = \mathsf{opt} \quad (7)$$

The guarantee of Lemma 3.7 still holds, but it needs to be stated more precisely and requires a different proof, which we defer to the end of this section.

▶ **Lemma 4.2.** *For each $i = 0, \ldots, k$, let Q^i be a random path obtained such that $\Pr[v \in Q^i] \geq \overline{x}^i_v/\alpha$ for all $v \in V$, for some $\alpha \geq 1$. Then*

$$\mathrm{E}\left[\sum_{i=0}^{k} \sum_{v \in Q^i \setminus (Q^2 \cup \ldots \cup Q^{i-1})} \pi_v(\mathsf{lb}_{i,v})\right] \geq \big(1 - e^{-1/\alpha}\big) \cdot \left(\sum_{i=0}^{k} \sum_{v \in V} \pi_v(\mathsf{lb}_{i,v}) \overline{x}^i_v\right).$$

To finish up the analysis, as before we lower bound $\max_{\ell=0,1,2} \mathrm{E}\big[\pi(Z^{(\ell)})\big]$. Since Lemma 3.3 continues to hold, for every node v, we know that if i is the smallest index such that $v \in Q^i$, then v is visited by some path $Z^{(\ell)}$ by time $\mathsf{lb}_{i,v}$. Therefore, $\max_{\ell=0,1,2} \mathrm{E}\big[\pi(Z^{(\ell)})\big] \geq \frac{1}{3} \cdot \mathrm{E}\big[\sum_{i=0}^{k} \sum_{v \in Q^i \setminus (Q^2 \cup \ldots \cup Q^{i-1})} \pi_v(\mathsf{lb}_{i,v})\big]$. Combining this with Lemma 4.2 and (7), we obtain the same approximation guarantee as in part (b) of Theorem 3.1.

Finally, the derandomization also proceeds as before. Let $\mathcal{C}^i$ denote the polynomial-size support of the distribution from which the u_i-w_i path Q^i is sampled, and let λ^i_P denote the probability of choosing a path $P \in \mathcal{C}^i$. We are now interested in the quantity $\Phi_0 =$

$\Phi(\lambda^0,\ldots,\lambda^k) := \mathrm{E}\big[\sum_{i=0}^{k}\sum_{v\in Q^i\setminus(Q^2\cup\ldots\cup Q^{i-1})}\pi_v(\mathsf{lb}_{i,v})\big]$. We now have $\Phi = \sum_{v\in V}\mathsf{rewd}_v$, where

$$\mathsf{rewd}_v = \rho_{0,v}\pi_v(\mathsf{lb}_{0,v}) + (1-\rho_{0,v})\rho_{1,v}\pi_v(\mathsf{lb}_{1,v}) + \ldots + \Big(\prod_{i=0}^{k}(1-\rho_{i,v})\Big)\rho_{k,v}\pi_v(\mathsf{lb}_{k,v}), \text{ and}$$
$$\rho_{i,v} = \rho_{i,v}(\lambda^0,\ldots,\lambda^k) = \sum_{P\in\mathcal{C}^i}\lambda_P^i \quad \forall i = 0,\ldots,k.$$

Since each rewd_v is linear in λ^i for all i, as before, we can deterministically choose Q^is for all i so that $\Phi(\mathbb{1}_{Q^0},\ldots,\mathbb{1}_{Q^k}) \geq \Phi_0$.

Proof of Lemma 4.2. The following claim will be useful.

▷ Claim 4.3. Let $a_1 \geq a_2 \geq \ldots \geq a_q \geq a_{q+1} := 0$. Let $y \in [0,1]^q$, and $t \geq \sum_{i=1}^{q} y_i$. Define $F(y_1,\ldots,y_q) := y_1a_1 + (1-y_1)y_2a_2 + \ldots + (1-y_1)(1-y_2)\cdots(1-y_{q-1})y_qa_q$. We have: (i) $F(y) \geq (1-e^{-t})\cdot\frac{\sum_{i=1}^{q}a_iy_i}{t}$; and (ii) $F(z) \leq F(y)$ for any z such that $0 \leq z \leq y$.

Proof. The proof of part (i) follows from elementary arguments (see, e.g., [11]). For any $i = 1,\ldots,q$, we have

$$y_1 + (1-y_1)y_2 + \ldots + (1-y_1)(1-y_2)\cdots(1-y_{i-1})y_i = \Big(1-(1-y_1)(1-y_2)\cdots(1-y_i)\Big)$$
$$\geq 1 - \Big(1-\frac{\sum_{j=1}^{i}y_j}{i}\Big)^i \geq \Big[1-\Big(1-\tfrac{t}{i}\Big)^i\Big]\cdot\frac{\sum_{j=1}^{i}y_j}{t} \geq \big(1-e^{-t}\big)\cdot\frac{\sum_{j=1}^{i}y_j}{t}$$

The first inequality is the AM-GM inequality, the second uses the fact that the function $f(x) = 1-\big(1-\frac{x}{i}\big)^i$ is concave, and so $f(x) \geq x\cdot\frac{f(t)-f(0)}{t}$ for $x \leq t$. That is, we have

$$y_1 + (1-y_1)y_2 + \ldots + (1-y_1)(1-y_2)\cdots(1-y_{i-1})y_i \geq \big(1-e^{-t}\big)\cdot\frac{\sum_{j=1}^{i}y_j}{t} \tag{8}$$

Multiplying (8) by $a_i - a_{i+1}$ and adding the resulting inequalities for $i = 1,\ldots,q$ yields the stated bound.

Part (ii) follows from the fact that

$$\frac{\partial F}{\partial y_i} = (1-y_1)\cdots(1-y_{i-1})\Big[a_i - \Big(a_{i+1}y_{i+1} + (1-y_{i+1})y_{i+2}a_{i+2} + \ldots + (1-y_{i+1})\cdots(1-y_{q-1})y_qa_q\Big)\Big]$$

which is nonnegative since $a_i \geq a_{i+1},\ldots,a_q$. ◁

We now have everything to prove Lemma 4.2. Let $\rho_v^i := \Pr[v \in Q^i]$ for $v \in V$, and $i = 0,\ldots,k$. Let Φ denote the quantity on the LHS of the inequality in the lemma. Since $Q^0,\ldots,Q^k$ are chosen independently, we have $\Phi = \sum_{v\in V}\mathsf{rewd}_v$, where

$$\mathsf{rewd}_v = \rho_{0,v}\pi_v(\mathsf{lb}_{0,v}) + (1-\rho_{0,v})\rho_{1,v}\pi_v(\mathsf{lb}_{1,v}) + \ldots + \Big(\prod_{i=0}^{k}(1-\rho_{i,v})\Big)\rho_{k,v}\pi_v(\mathsf{lb}_{k,v})$$
$$\geq \tfrac{\overline{x}_v^0}{\alpha}\cdot\pi_v(\mathsf{lb}_{0,v}) + \Big(1-\tfrac{\overline{x}_v^0}{\alpha}\Big)\tfrac{\overline{x}_v^1}{\alpha}\cdot\pi_v(\mathsf{lb}_{1,v}) + \ldots + \Big(\prod_{i=0}^{k}\Big(1-\tfrac{\overline{x}_v^i}{\alpha}\Big)\Big)\tfrac{\overline{x}_v^k}{\alpha}\cdot\pi_v(\mathsf{lb}_{k,v}).$$

The inequality follows from part (ii) of Claim 4.3 since $\rho_v^i \geq \overline{x}_v^i/\alpha$ for all $i = 0,\ldots,k$. Applying Claim 4.3 with $a_i = \pi_v(\mathsf{lb}_{i,v})$ and $y_i = \frac{\overline{x}_v^i}{\alpha}$ for all $i = 0,\ldots,k$, and $t = \frac{1}{\alpha}$, we therefore obtain that $\mathsf{rewd}_v \geq \big(1-e^{-1/\alpha}\big)\sum_{i=0}^{k}\pi_v(\mathsf{lb}_{i,v})\overline{x}_v^i$. It follows that $\Phi \geq \big(1-e^{-1/\alpha}\big)\cdot\big(\sum_{i=0}^{k}\sum_{v\in V}\pi_v(\mathsf{lb}_{i,v})\overline{x}_v^i\big)$. ◀

4.2 Further extensions of Deadline TSP

Point-to-point {deadline, monotone-reward} TSP. In the point-to-point (P2P) version of {deadline, monotone-reward} TSP, in addition to the root node r, we are also given an end-node t and a length-bound B. The goal is to find an r-t path of length at most B that collects maximum reward.

The P2P-version easily reduces to the standard version of the problem. For deadline TSP, we can incorporate the above requirements by modifying the deadline of each node $v \in V'$ to $D_v^{\mathsf{new}} := \min\{D_v, B - c_{vt}\}$. If P is a rooted path ending at a node s such that all $v \in P$ are visited by time D_v^{new}, then $P' = P, st$ is an r-t path such that all $v \in P'$ are visited by time D_v, and $c(P') \leq D_s^{\mathsf{new}} + c_{st} \leq B$.

Similarly, for monotone-reward TSP, we modify the reward function of each $v \in V'$ to $\pi_v^{\mathsf{new}}(x) = \pi_v(x)$ if $x \leq B - c_{vt}$, and 0 if $x > B - c_{vt}$. As before if P is an r-s path earning a certain modified reward, then $P' = P, st$ is an r-t path of length at most B earning the same reward.

TSP with release dates, and TSP with increasing rewards. In TSP with release dates, each node v has a release date rel_v instead of a deadline, and we have a length bound B. A feasible solution is a rooted path P, and a traversal of P starting from the root node where we are allowed to also wait at nodes, so that the visiting time of each node $v \in P$ is at least rel_v and at most B; we seek a feasible solution that gathers maximum reward. Set $\mathsf{rel}_r = 0$ for notational convenience.

As noted in [2], this can be reduced to deadline TSP as follows. If P is a feasible solution ending at node t, then we must have $\mathsf{rel}_t \leq B$ and $c(P_{vt}) \leq B - \mathsf{rel}_v$ for all $v \in P$. Conversely any r-t path P with $c(P_{vt}) \leq B - \mathsf{rel}_v$ for all $v \in P$ yields a feasible solution, where we wait at r for $B - c(P)$ time and then traverse P without any subsequent waiting. We thus infer that we seek a feasible solution to P2P-deadline TSP (and hence deadline TSP) with start node t, end-node r, length bound B, and deadlines $\{B - \mathsf{rel}_v\}_{v \in V'}$. Trying all possible choices of t completes the reduction.

Analogous to how monotone-reward TSP generalizes deadline TSP, we can consider a generalization of TSP with release dates wherein each node v has a *non-decreasing* reward function $\pi_v : \mathbb{R}_+ \mapsto \mathbb{R}_+$, and we seek a rooted path P with $c(P) \leq B$, and a traversal of P that yields maximum reward. As above, if we know the end-node t of an optimal solution, then this reduces to solving an instance of P2P monotone-reward TSP, where we seek a path starting at t and ending at r path of length at most B, and the reward of node v is given by the *non-increasing* function $\pi'_v(x) = \pi_v(B - x)$.

Orienteering with time windows. Chekuri et al. [6] obtain (among other results) an $O\big(\max\{\log \mathsf{opt}, \log \frac{L_{\max}}{L_{\min}}\}\big)$-approximation for orienteering with time windows, where opt is the optimal value, and $L_{\max}$ and $L_{\min}$ are the lengths of the longest and shortest time windows with non-zero length. The $\log \mathsf{opt}$ term in their guarantee is because they use the logarithmic approximation of [2] for deadline TSP.[3] Replacing this algorithm with our deadline-TSP algorithm therefore yields an $O\big(\log \frac{L_{\max}}{L_{\min}}\big)$-approximation for orienteering with time windows in $O\big(n^{\log n \Delta}\big)$ time.

[3] Chekuri et al. [6] work with $\{0, 1\}$-rewards, and in this setting, the approximation factor obtained in [2] for deadline TSP can be seen to be $O(\log \mathsf{opt})$.

5 LP-rounding algorithm for P2P-orienteering in [14]

We briefly discuss the LP-rounding algorithm for P2P-orienteering by Friggstad and Swamy [14] that directly yields the distributional guarantee utilized in subroutine LP-Round. Recall that $(V' = \{r\} \cup V, c)$ is the underlying metric, and our P2P-orienteering instance $\mathcal{I}$ is specified by node-set $N \subseteq V'$, start and end-nodes $s, t \in N$ respectively, length-bound B, and node-rewards $\{\pi_v\}_{v \in N}$.

Let $D = (N, A)$ denote the complete (bidirected) graph on N, where the cost of an arc $(u, v) \in A$ is set to c_{uv}; thus c induces a metric on D. Let P^* be an optimal solution to $\mathcal{I}$. The idea underlying the LP relaxation is to "guess" the node $v \in P^*$ that maximizes $c_{sv} + c_{vt}$. The LP then searches for an $s \rightsquigarrow v$ path and a $v \rightsquigarrow t$ path – encoded by requiring an s-v flow and v-t flow of value 1 – that visit only nodes $u \in N$ such that $c_{su} + c_{ut} \leq c_{sv} + c_{vt}$, have total length at most B, and together earn the maximum reward. Also, we replace the "guessing" step by having indicator variables z^v_v to denote if v is the node on P^* with maximum $c_{sv} + c_{vt}$ value.

This leads to the following LP. For every $v \in N$, we have the following set of variables (in addition to z^v_v). We use $x_v \in [0, 1]$ to denote the extent to which v is visited. We let y^{sv} denote an s-v flow of value z^v_v, and y^{vt} denotes a v-t flow of value x^v_v. We impose that $y^{sv}\big(\delta^{\text{in}}(u)\big) = y^{vt}\big(\delta^{\text{in}}(u)\big) = 0$ whenever $c_{su} + c_{ut} > c_{sv} + c_{vt}$. We use z^{sv}_u and z^{vt}_u to denote respectively the $s \rightsquigarrow u$ connectivity under x^{sv} and the $v \rightsquigarrow u$ connectivity under x^{vt}. So in an integral solution, z^{sv}_u and z^{vt}_u indicate respectively if u lies on the s-v portion or the v-t portion of the optimum path. We connect the x and z variables by imposing that $x_u = \sum_{v \in N}(z^{sv}_u + z^{vt}_u)$ for every $u \in N$. For nodes $v, p, q \in N$, and $\kappa \geq 0$, define

$$\mathcal{F}_v(p, q, \kappa) := \Big\{ y \in \mathbb{R}^A_+ : \quad y\big(\delta^{\text{out}}(p)\big) = \kappa = y\big(\delta^{\text{in}}(q)\big), \qquad y\big(\delta^{\text{in}}(p)\big) = 0 = y\big(\delta^{\text{out}}(q)\big)$$
$$y\big(\delta^{\text{in}}(w)\big) - y\big(\delta^{\text{out}}(w)\big) = 0 \qquad \forall w \in V' \setminus \{p, q\}$$
$$y\big(\delta^{\text{in}}(w)\big) = 0 \qquad \forall w \in V' : c_{sw} + c_{wt} > c_{sv} + c_{vt} \Big\}$$

Note that if $\kappa > 0$, then $\mathcal{F}(u, u, \kappa) = \emptyset$ for every u.

$$\max \quad \sum_{u,v \in N} \pi_u x_u \tag{P2P-O}$$
$$\text{s.t.} \quad y^{sv} \in \mathcal{F}_v(s, v, z^v_v),\ y^{vt} \in \mathcal{F}_v(v, t, z^v_v) \qquad \forall v \in N$$
$$y^{sv}\big(\delta^{\text{in}}(S)\big) \geq z^{sv}_u \qquad \forall v \in N, S \subseteq N \setminus \{s\}, u \in S \tag{9}$$
$$y^{vt}\big(\delta^{\text{in}}(S)\big) \geq z^{vt}_u \qquad \forall v \in N, S \subseteq N \setminus \{v\}, u \in S \tag{10}$$
$$\sum_{a \in A} c_a(y^{sv}_a + y^{vt}_a) \leq B z^v_v \qquad \forall v \in N \tag{11}$$
$$\sum_{v \in N}(z^{sv}_u + z^{vt}_u) = x_u \qquad \forall u \in N \tag{12}$$
$$\sum_{v \in N} z^v_v = 1, \qquad y, z \geq 0, \quad x \in [0, 1]^N.$$

As noted in [14], we can rephrase the cut constraints (9), (10) using additional flow variables and constraints to obtain a polynomial-size formulation.

Friggstad and Swamy [14] devise the following algorithm for rounding an LP solution (x, y, z). They show that for each $v \in N$, we can utilize y^{sv} to obtain a polynomial collection of s-rooted paths, and weights for these paths such that:

(i) if a path P in the collection ends at a node u, then $P' = P, ut$ is an s-t path with $c(P') \leq B$;
(ii) the total weight of paths in the collection is at most $3z_v^v$; and
(iii) the total weight of paths containing a node u is at least z_u^{sv}, for all $u \in N$.

Similarly, we can utilize y^{vt} to obtain a suitable weighted collection of paths, each of which yields an s-t path of length at most B. Taking these collections for all $v \in N$, we obtain that the total weight of paths is at most 6, and for each $u \in N$, the total weight of paths containing u is at least $\sum_v (z_u^{sv} + z_u^{vt}) = x_u$. The distribution obtained by scaling the weights by 6 yields the desired distribution.

References

1 H. C. An, R. Kleinberg, and D. B. Shmoys. Improving Christofides' algorithm for the s-t path TSP. *Journal of the ACM*, 62(5):34, 2015.
2 N. Bansal, A. Blum, S. Chawla, and A. Meyerson. Approximation algorithms for deadline-TSP and vehicle routing with time windows. In *Proceedings of 36th STOC*, pages 166–174, 2004.
3 A. Blum, S. Chawla, D. R. Karger, T. Lane, and A. Meyerson. Approximation algorithms for orienteering and discount-reward TSP. *SIAM J. Comput.*, 37(2):653–670, 2007.
4 R. Carr and S. Vempala. Randomized metarounding. *Random Structures and Algorithms*, 20(3):343–352, 2002.
5 K. Chaudhuri, B. Godfrey, S. Rao, and K. Talwar. Paths, trees, and minimum latency tours. In *Proceedings of 44th FOCS*, pages 36–45, 2003.
6 C. Chekuri, N. Korula, and M. Pál. Improved algorithms for orienteering and related problems. *ACM Transactions on Algorithms*, 8(3), 2012.
7 C. Chekuri and A. Kumar. Maximum coverage problem with group budget constraints and applications. In *Proceedings of 7th APPROX*, pages 72–83, 2004.
8 C. Chekuri and M. Pál. A recursive greedy algorithm for walks in directed graphs. In *Proceedings of 46th FOCS*, pages 245–253, 2005.
9 N. Christofides. Worst-case analysis of a new heuristic for the traveling salesman problem. Technical Report 388, Graduate School of Industrial Administration, CMU, 1976.
10 B. Farbstein and A. Levin. Discount reward TSP. *Algorithmica*, 80(2):472–495, 2018.
11 L. Fleischer, M. Goemans, V. Mirrokni, and M. Sviridenko. Tight approximation algorithms for maximum general assignment problems. In *Proceedings of 17th SODA*, pages 611–620, 2006.
12 Z. Friggstad, M. R. Salavatipour, and Z. Svitkina. Asymmetric traveling salesman path and directed latency problems. *SIAM J. Comput*, 42(4):1596–1619, 2013.
13 Z. Friggstad and C. Swamy. Approximation algorithms for regret-bounded vehicle routing and applications to distance-constrained vehicle routing. In *Proceedings of 46th STOC*, pages 744–753, 2014. Detailed version posted on *CS arXiv*, Nov 2013.
14 Z. Friggstad and C. Swamy. Compact, provably-good LPs for orienteering and regret-bounded vehicle routing. In *Proceedings of 19th IPCO*, pages 199–211, 2017.
15 B. L. Golden, L. Levy, and R. Vohra. The orienteering problem. *Naval Research Logistics*, 34:307–318, 1987.
16 E. Halperin and R. Krauthgamer. Polylogarithmic inapproximability. In *Proceedings of 35th STOC*, pages 585–594, 2003.
17 A. Köhne, V. Traub, , and J. Vygen. The asymmetric traveling salesman path LP has constant integrality ratio. In *Proceedings of 20th IPCO*, pages 288–298, 2019.
18 R. Lavi and C. Swamy. Truthful and near-optimal mechanism design via linear programming. *Journal of the ACM*, 58(6), 2011.
19 C.-L. Li, D. Simchi-Levi, and M. Desrochers. On the distance constrained vehicle routing problem. *Operations research*, 40:790–799, 1992.

20 V. Nagarajan and R. Ravi. The directed minimum latency problem. In *Proceedings of 11th APPROX*, pages 193–206, 2008.

21 V. Nagarajan and R. Ravi. The directed orienteering problem. *Algorithmica*, 60(4):1017–1030, 2011.

22 V. Nagarajan and R. Ravi. Approximation algorithms for distance constrained vehicle routing problems. *Networks*, 59(2):209–214, 2012.

23 I. Post and C. Swamy. Linear-programming based techniques for multi-vehicle minimum latency problems. In *Proceedings of 26th SODA*, pages 512–531, 2015.

24 A. Sebo and A. van Zuylen. The salesman's improved paths: A $3/2 + 1/34$ approximation. In *Proceedings of 57th FOCS*, pages 118–127, 2016.

25 A. Sebo and J. Vygen. Shorter tours by nicer ears: 7/5-approximation for the graph-TSP, 3/2 for the path version, and 4/3 for two-edge-connected subgraphs. *Combinatorica*, 34(5):597–629, 2014.

26 O. Svensson, J. Tarnawski, and L. Vegh. A constant-factor approximation algorithm for the asymmetric traveling salesman problem. In *Proceedings of 50th STOC*, pages 204–213, 2018.

27 P. Toth and eds D. Vigo. *The Vehicle Routing Problem*. SIAM Monographs on Discrete Mathematics and Applications, Philadelphia, 2002.

28 V. Traub and J. Vygen. Approaching $\frac{3}{2}$ for the s-t path TSP. In *Proceedings of 29th SODA*, pages 1854–1864, 2018.

29 R. Zenklusen. A 1.5-approximation for path TSP. In *Proceedings of 30th SODA*, pages 1539–1549, 2019.

Random Order Vertex Arrival Contention Resolution Schemes for Matching, with Applications

Hu Fu
ITCS, Shanghai University of Finance and Economics, China

Zhihao Gavin Tang
ITCS, Shanghai University of Finance and Economics, China

Hongxun Wu
IIIS, Tsinghua University, Beijing, China

Jinzhao Wu
Peking University, Beijing, China

Qianfan Zhang
IIIS, Tsinghua University, Beijing, China

Abstract

With a wide range of applications, stochastic matching problems have been studied in different models, including prophet inequality, Query-Commit, and Price-of-Information. While there have been recent breakthroughs in all these settings for bipartite graphs, few non-trivial results are known for general graphs.

In this paper, we study the random order vertex arrival contention resolution scheme for matching in general graphs, which is inspired by the recent work of Ezra et al. (EC 2020). We design an $\frac{8}{15}$-selectable batched RCRS for matching and apply it to achieve $\frac{8}{15}$-competitive/approximate algorithms for all the three models. Our results are the first non-trivial results for random order prophet matching and Price-of-Information matching in general graphs. For the Query-Commit model, our result substantially improves upon the 0.501 approximation ratio by Tang et al. (STOC 2020). We also show that no batched RCRS for matching can be better than $\frac{1}{2} + \frac{1}{2e^2} \approx 0.567$-selectable.

2012 ACM Subject Classification Theory of computation → Online algorithms

Keywords and phrases Matching, Contention Resolution Scheme, Price of Information, Query-Commit

Digital Object Identifier 10.4230/LIPIcs.ICALP.2021.68

Category Track A: Algorithms, Complexity and Games

1 Introduction

Matching is a fundamental object of algorithmic studies, with wide applications. Its recent use in e-commerce often presents two general features: (i) one does not see the entire input graph in the beginning, but there is often prior stochastic information on each edge's value or presence, thanks to accumulated data from the past; depending on the application, the actual value/presence is revealed either according to some order or at the control of the matching algorithm; (ii) in either case, the matching algorithm must be online – that is, once an edge's information is revealed, the algorithm must make an irrevocable decision whether to include the edge in the matching. Examples of such applications include online advertisement [26, 10, 25], kidney exchange [9], and ride-sharing

platforms [4, 19, 20]. Mathematically, this type of scenarios has been modeled as prophet inequality (e.g. [24]), Query-Commit [9], and Price of Information [27] problems, according to the way the information is revealed.

The Online Contention Resolution Scheme (OCRS), first proposed by Feldman et al. [14], is a powerful framework that has found applications in various online decision problems such as prophet inequalities [14, 13], oblivious posted pricing [7], and stochastic probing [17]. Ezra et al. [13] recently proposed a generalization of OCRS, that with batched arrivals, and used it to give prophet inequalities for matching in bipartite graphs when vertices on both sides arrive online.

Our main results are two-fold. We follow Ezra et al. and define a batched OCRS for matching in general graphs with vertex arrival, and give the first non-trivial such OCRS for random arrival order. We then give an alternative perspective on Gamlath et al. [15]'s recent progress for the Query-Commit and Price of Information problems on bipartite matching; viewing their algorithms as reductions to batched OCRS with random batched arrivals, we immediately obtain similar reductions and consequently the first non-trivial results for the Query-Commit and Price of Information problems for matching in general graphs. Before elaborating on these results, we give more background on the connection between contention resolution schemes and online decision problems, and motivate the batched OCRSs we study.

Contention Resolution Schemes (CRSs) are first proposed as rounding algorithms in submodular maximization [8]. Such an algorithm treats a fractional solution $\boldsymbol{x}$ as a product distribution, samples elements according to it so that each element i is *active* independently with probability x_i, and selects a feasible subset of active elements, guaranteeing that each element, when active, is selected with probability at least $\alpha > 0$, which in turn guarantees that the selected subset retains performance comparable in expectation to (typically at least α fraction of) that of $\boldsymbol{x}$. Such a CRS is said to be α-selectable.

The Online Contention Resolution Schemes (OCRSs) were introduced by Feldman et al. [14]. Its difference from a CRS is that the algorithm does not see the set of active elements upfront, but observes each element's status (of being active or not) one by one online as each element arrives. Intuitively, in an online stochastic decision problem, the expected performance of the offline optimal can be calculated as a fractional solution: a realization of the stochastic procedure gives rise to an optimal (integral) solution, and taking expectation over the stochastic procedure is equivalent to taking a convex combination of these solutions, yielding a fractional solution. An online algorithm, to be competitive, emulates the performance of the fractional solution but only sees partial realizations of the stochastic procedure in an online fashion. For example, in the *prophet inequality* problem, one has prior knowledge in the form of a distribution over the elements' values, but the online algorithm only sees an element's value when it arrives and has to decide immediately whether to keep the element in the solution.

From this perspective, while Feldman et al. [14]'s setup of OCRS neatly fits the needs of many online decision problems, for other problems it is natural to go beyond and, as Ezra et al. [13] did, to consider settings in which (i) the elements arrive not necessarily one by one but in batches, and (ii) the elements' being active may not be independent events (which treats the fractional solution as a product distribution) but correlated ones. Ezra et al. [13] termed such schemes *batched OCRSs*. The concrete choices in the batched OCRS should adapt to the underlying online decision problem being solved.

For example, in online matching, it is often interesting to consider vertex arrival rather than edge arrival – with vertex arrival, the vertices arrive one by one, and at the arrival of each vertex, all the edges connecting it to the vertices that have arrived are revealed

in a batch (i.e., simultaneously), whereas with edge arrival, individual edges arrive one by one. Correlation in elements' status (of being active or not) arises naturally when such batch arrivals are allowed. For example, consider prophet inequalities in bipartite matching with vertex arrival: in a given graph $G = (U, V, E)$, each edge $e \in E$ has a value $v_e \geq 0$ drawn independently from a given distribution F_e, and the goal is to obtain a matching with maximum value. The offline (fractional) solution $\boldsymbol{x} \in [0,1]^E$ has x_e representing the probability that edge e is in the maximum value matching. In the standard reduction from edge arrival prophet inequality to OCRS, when an element e arrives and reveals its value v_e, one (re)samples the values of all the other edges according to their distributions, and labels e active if v_e is high enough to make e in the maximum value matching among the sampled edges. Now with vertex arrival, when vertex v arrives, the information of all edges connecting v to the vertices that arrived before v is revealed. Let B_v denote this set of edges. It is natural for the reduction to sample values for all the other edges, and labels at most one edge in B_v as active if that edge is in the maximum value matching among the (re)sampled edges. Here, the status of the edges in B_v in the (batched) OCRS are naturally correlated, even though their values in the original prophet inequality problem are independent.[1]

The batched OCRS we study in this work precisely results from this reduction, when the vertices arrive in an order that is uniformly at random. To be precise, we are given a vector $\boldsymbol{x} \in [0,1]^E$, such that for each vertex v, $\sum_{e \in \delta(v)} x_e \leq 1$, where $\delta(v)$ denotes the set of edges incident to v. Vertices arrive in an order uniformly at random; when vertex v arrives, the status of all edge in B_v are revealed: each edge $e \in B_v$ is active with probability x_e, but at most one of them is active. If an edge is active, we must decide immediately whether to select it into the solution. We must keep the set of selected edges a valid matching and, for as large a constant $\alpha > 0$ as possible, guarantee that any edge is selected with probability at least αx_e. We refer to such an algorithm an *α-selectable batched Random order Contention Resolution Scheme (RCRS)*.[2] The existence of a $\frac{1}{2}$-selectable batched RCRS is known, after all, Ezra et al. [13] constructed a simple $\frac{1}{2}$-selectable batched OCRS for arbitrary vertex arrival order. Their method is very similar to the $\frac{1}{2}$-selectable OCRS for rank 1 matroid due to Alaei [2]. On the contrary, we were unable to generalize existing RCRS (e.g., [23]) to a batched RCRS with a selectability better than $\frac{1}{2}$. Instead, we construct an $\frac{8}{15}$-selectable batched RCRS via a prune-greedy algorithm, which is our main technical result. As with other settings of RCRS, getting beyond the simplest algorithm is technically involved; in contrast with most other RCRS with non-trivial selectability guarantees [23, 1], our batched RCRS is non-adaptive, in the sense that the algorithm's decision to select an edge does not depend on the time when it arrives, besides checking the feasibility of selecting it.

We also show that such batched RCRSs find applications beyond prophet inequalities as in [13]. Recently there is a breakthrough by Gamlath et al. [15] on the Query-Commit and Price of Information problems, where they gave $(1 - \frac{1}{e})$-approximation algorithms for both problems by reducing them to the *prophet secretary* problem in [12, 11]. We notice that the prophet secretary algorithm can be easily replaced by the $(1 - \frac{1}{e})$-selectable RCRS for rank 1 matroid in [23] and still resulting a $(1 - \frac{1}{e})$-approximation algorithm. We find this perspective is not only conceptually simpler, but also powerful in the sense that if we replace RCRS for rank 1 matroid by our batched RCRS, their algorithms (with a slight modification) also apply for general graphs. As a result, we show that our batched RCRSs imply best-known results for Query-Commit and Price of Information for matching in general graphs.

[1] See [13] for details of this reduction.

[2] As explained above, batched OCRS as defined by Ezra et al. is more general. Our shorthand term should cause no confusion, as we consider only one OCRS setting in this work.

In the **Query-Commit** matching problem [9], we are given a graph $G = (V, E)$, a probability p_e for each edge e with which e is present, independently of the other edges, and a value v_e of e if e is present. At each step, we may choose to query an edge, but we must take it if it is present. At all times, we must keep the set of edges taken a valid matching, and our goal is to maximize the expected sum of values of the edges taken. Gamlath et al. [15] gave a clever algorithm that is $(1 - \frac{1}{e})$-competitive for bipartite graphs. Their algorithm solves an offline linear program relaxation and rounds the solution through a prophet secretary algorithm. We show that this rounding algorithm can be seen as a reduction to a batched RCRS with random one-sided vertex arrival. This perspective allows us to directly derive an $\frac{8}{15}$-competitive algorithm on general graphs, using our new batched RCRS.

In the same work, Gamlath et al. [15] gave a $(1 - \frac{1}{e})$-approximation algorithm for the closely related **Price of Information** (PoI) problem for matching in bipartite graphs. The PoI problems were introduced by Singla [27] as a generalization of the Pandora's Box problem from the consumer search literature [29, 22], which in turn is closely related to Bayesian bandits. In the PoI problem for matching [15], we are given a graph $G = (V, E)$, a search cost c_e and a value distribution F_e for each edge e. An invisible value v_e is drawn independently from F_e, and is revealed only when the algorithm chooses to *search* the edge e, at the cost of c_e. Our goal is to maximize, in expectation, the maximum value of matching among searched edges, minus the search costs paid along the way. From a new proof for the optimal algorithm for the Pandora's Box problem, given by Kleinberg et al. [22], surfaced a reduction from PoI with Bernoulli distributions to Query-Commit. This reduction was generalized by Singla [27]. Gamlath et al. [15] showed that their algorithm for Query-Commit admits some modifications that allow it to give a $(1 - \frac{1}{e})$-approximation for the general PoI problem for bipartite matching. Our perspective of their algorithm as a reduction to RCRS with random vertex arrival admits the same modifications, and therefore we immediately obtain an $\frac{8}{15}$-approximation for the PoI problem for matching in general graphs.

Other Related Works. Naturally, the most related work is the recent paper by Gamlath et al. [15]. They studied the query-commit matching problem and the price-of-information problem in *bipartite* graphs and achieved $(1 - \frac{1}{e})$ approximation ratios for both settings. Their paper has a comprehensive review of the related literature, and we only discuss briefly some most related works that are not covered there.

A closely related setting is the oblivious matching problem. It is also a query-commit model, while the input instance is adversarial rather than stochastic. In other words, there is no information about the existence probabilities of edges. Obviously, results in this harder model directly apply to our setting. The celebrated Ranking algorithm by Karp et al. [21] gives a $(1 - \frac{1}{e})$-approximation for the unweighted oblivious matching problem in bipartite graphs. Later, the Ranking algorithm is extended to general graphs and shown to achieve an approximation ratio of 0.526 [6]. Another well-studied algorithm in this literature is called modified randomized greedy. It was introduced by Aronson et al. [3] and shown to be $(0.5 + \Omega(1))$-approximate in general graphs. The approximation ratio was recently improved to 0.531 by Tang et al. [28]. For the edge-weighted case, Tang et al. [28] achieved the first non-trivial 0.501-approximation. As we remarked, this result applies to the Query-Commit setting studied in our current work, but our approximation ratio is significantly larger. Moreover, our techniques are entirely different from the previous papers and we believe they provide novel insights on the problem.

Offline contention resolution schemes for matchings also achieved some progress recently. Bruggmann and Zenklusen [5] gave an optimal monotone contention resolution scheme for bipartite matching with its application to submodular maximization, and Guruganesh and Lee [18] studied the connection between correlation gap and contention resolution scheme.

Gupta et al. [16] introduced a Markovian price of information model and use online contention resolution schemes to round the optimum of linear programming. However, as our RCRS is vertex-arrival model and batched, it can not be directly applied to their result.

2 Preliminaries

Price of Information. In a *Price of Information* problem, we are given a universe U and a feasibility system $\mathcal{F} \subseteq 2^U$. For each element $i \in U$, we are also given a *search cost* $c_i \in \mathbb{R}_+$ and a distribution F_i from which an invisible value $v_i \geq 0$ is independently drawn. In a valid algorithm for the problem, at each step we may choose to terminate the search or to inquire a new element i; when we choose to do the latter, we incur a cost c_i, and the value $v_i \sim F_i$ is revealed. At the termination of the algorithm, if S is the set of elements inquired by the algorithm, our utility is $\max_{A \subseteq S : A \in \mathcal{F}} \sum_{i \in A} v_i - \sum_{i \in S} c_i$. Our goal is to maximize the expected utility, where expectation is taken over both the realization of the values and the internal randomness of our algorithm. In this paper, we focus on the problem where the universe is the set of edges E in a given graph $G = (V, E)$, and $\mathcal{F}$ is the set of all matchings in E. An algorithm is said to be α-approximation if it achieves at least α fraction of the optimal utility. That is, we compare to the optimal algorithm.

The Query-Commit Problem for Matching. In a Query-Commit problem, we are given a universe U, a feasibility system $\mathcal{F} \subseteq 2^U$, and, for each element $i \in U$, a probability $p_i \in [0, 1]$ and a value $v_i \geq 0$. Each element $i \in U$ is *active* independently with probability p_i, but this status can be known only via a *query*. For the Query-Commit problem, at each step our algorithm can choose to query the status of an element; if the queried element is active, it must be accepted to the solution. At all time, we need to make sure that the set of accepted elements is feasible (i.e., in $\mathcal{F}$). Let S be the set of elements accepted by an algorithm by the time it terminates, the performance of the algorithm is $\sum_{i \in S} v_i$, the total value of the accepted elements. An algorithm is *α-competitive* if its performance is at least α fraction of the offline optimal, i.e., the expected performance of an algorithm that knows the set of active elements beforehand. In this work, we focus on the case where the universe U is the edge set E of a given graph $G = (V, E)$, and $\mathcal{F}$ is the set of all matchings in G.

Batched RCRS for Matching in General Graphs. We are given a graph $G = (V, E, \boldsymbol{x})$, where $\boldsymbol{x} \in [0, 1]^E$ satisfies $x_u \stackrel{\text{def}}{=} \sum_{e \in \delta(u)} x_e \leq 1$ for each $u \in V$. All vertices of G arrive online in a uniformly random order. Upon the arrival of a vertex v, the status of the edges connecting v and the vertices that have arrived are sampled and revealed to the algorithm. The sampling is such that, each edge e of these is *active* with probability x_e, and at most one of these edges can be active; that is, these edges' being active are mutually exclusive events. We refer to this sampling scheme as S. A batched random order contention resolution scheme for matching (henceforth batched RCRS) decides, upon the arrival of each vertex, irrevocably whether to select the active edge (if any exists). At any point in time, the selected edges must form a matching. A batched RCRS is *c-selectable* if $\Pr[e \text{ is selected} \mid e \text{ is active}] \geq c$ for every $e \in E$.

Throughout the paper, we use $\delta(u) \subseteq E$ to denote the set of edges incident to a vertex u.

3 Batched RCRS for Matching In General Graphs

In this section we give an $\frac{8}{15}$-selectable batched RCRS for matching in general graphs.

3.1 Batched RCRS and Its Proof Sketch

▶ **Theorem 1.** *There is a polynomial-time computable, $\frac{8}{15}$-selectable batched RCRS for matching in general graphs with random vertex arrival.*

Our proof consists of two steps. Given a graph $G = (V, E, \boldsymbol{x})$. Let S be the corresponding sampling scheme defined in Section 2. An instance is called 1-regular if $x_u = 1$ for all u. We refer to x_u as the degree of u. Our first lemma states that without loss of generality, we can focus on designing batched RCRS for 1-regular instances.

▶ **Lemma 2.** *If c-selectable batched RCRS exists for all 1-regular instances, then c-selectable batched RCRS exists for all instances.*

Proof. Assume c-selectable batched RCRS exists for all 1-regular instances. Consider an arbitrary instance $G = (V, E, \boldsymbol{x})$, we first convert it into a 1-regular instance G' by introducing dummy vertices and edges. Let S be the associated sampling scheme of the instance G. We correspondingly construct a sampling scheme S' on G' from S by simulating the arrival of dummy vertices. Finally, we apply the c-selectable batched RCRS in the converted 1-regular instance to achieve c-selectability in the original instance G.

For the first step, there can be multiple ways of converting a graph into a 1-regular graph. We give a concrete construction below and remark that the reduction does not rely on the details of the construction. Let $n = |V|$ be the number of vertices of G. We introduce n dummy vertices into the graph. For each vertex $u \in V$, we add n dummy edges between u and all dummy vertices, where each edge has active probability $x'_e = \frac{1-x_u}{n}$. Note that now every vertex in V has degree 1 and all dummy vertices have the same degree. Finally, we add a clique to all dummy vertices and set the weight/active probability of those edges appropriately so that all vertices have degree 1.

Recall G' is the converted instance and let $\boldsymbol{x}'$ be the corresponding vector of active probability. Next, we show how to construct the corresponding sampling scheme S' from S. There are $2n$ time slots in total for vertex arrivals. We select uniformly at random n of them to pair with n dummy vertices uniformly at random. Upon the arrival of a dummy vertex, we sample at most one of the edges connecting it to previously arrived vertices with probability consistent with $\boldsymbol{x}'$. Upon the arrival of an empty slot, we pair it with the next vertex arrival in the original instance and call S. If there exists an active edge realized by S, let it also be active in the new instance. Otherwise, we further sample at most one of the dummy edges connecting the current vertex to previously arrived dummy vertices with appropriate probability, so that the overall active probability is consistent with $\boldsymbol{x}'$. This is implementable since the degree of each vertex is exactly 1.

Now, we have defined a sampling scheme S'. Together with G' and $\boldsymbol{x}'$, it is a 1-regular instance and we can apply the c-selectable batched RCRS for 1-regular instances. Whenever an edge of G (i.e., edges between real vertices) is selected by the batched RCRS, we also select it in the real run of the instance. It is straightforward to verify that this is a c-selectable batched RCRS for the original instance. ◀

Next, we design an $\frac{8}{15}$-selectable batched RCRS for 1-regular instances.

▶ **Lemma 3.** *For every 1-regular instance, there exists an $\frac{8}{15}$-selectable batched RCRS.*

Before going into the details of our analysis, we present our algorithm and provide a proof sketch to highlight the essential ideas. The actual proof of Lemma 3 is deferred to Section 3.3.

Proof Sketch. Consider a 1-regular instance $G = (V, E, \boldsymbol{x})$. We apply the following prune-greedy algorithm: 1) Prune the active probability of each edge from x_e to $f(x_e) \stackrel{\text{def}}{=} \frac{3x_e}{3+2x_e}$; 2) Run greedy on the pruned instance.

The first step is a preprocessing step and we refer to it as the pruning step. Before the instance starts, we independently flip a coin for each edge e and decide whether to consider it later. In particular, an edge e survives with probability $\frac{f(x_e)}{x_e}$ and otherwise we would ignore it regardless being active or not in the future. The second step is just a greedy algorithm, that always select an active edge if adding it does not violate the matching constraint. This procedure is equivalent to assuming that each edge is active with probability $f(x_e)$ and we run greedy. For the ease of presentation, in the rest of the proof, we assume that each edge e is active with probability $f(x_e)$. Our specific choice of the function f is to benefit the analysis, and it be explained in the full proof.

For analysis purpose, we use the following interpretation of the random arrival order assumption. Let t_u be the arrival time of each vertex $u \in V$, that is drawn independently from $U[0, 1]$. All vertices arrive in the ascending order of their arrival times.

Fixing an arbitrary edge $(u, v) \in E$, we use $v \to u$ to denote the event that edge (u, v) is selected by our algorithm upon the arrival of vertex v. To prove the selectability, we need a lower bound on $\Pr[v \to u]$. Fixing the arrival time $t_v = t$, since we use a greedy algorithm, this event happens if 1) u arrives before t; 2) u remains unmatched at t; 3) (u, v) is active. Thus,

$$\begin{aligned}\Pr[v \to u \mid t_v = t] &= \Pr[t_u \le t, u \text{ unmatched } @t, (u,v) \text{ active} \mid t_v = t] \\ &= f(x_{uv}) \cdot \Pr[t_u \le t, u \text{ unmatched } @t \mid t_v = t] \\ &= f(x_{uv}) \cdot (t - \Pr[u \text{ matched } @t \mid t_v = t]). \qquad (*)\end{aligned}$$

Consider the following warm-up analysis,

$$\Pr[u \text{ matched } @t \mid t_v = t] = \sum_{z \neq u,v} \Pr[(u,z) \text{ matched before } t \mid t_v = t] \le \sum_{z \neq u,v} t^2 f(x_{uz}) \le t^2,$$

where the inequality follows since the probability that both u, z arrives before t is t^2 and the edge is matched only if it is active, that happens with probability $f(x_{uz})$. Applying this relaxation to equation $(*)$ and integrating over t gives a simple but weak bound of the selectablity of our algorithm, where the ratio is smaller than $\frac{1}{2}$. Observe that the warm-up analysis does not make use of the pruning step, as well as the 1-regularity of the instance.

Instead, we do recursive type of analysis for $\Pr[(u, z) \text{ matched before } t \mid t_v = t]$:

$$\begin{aligned}&\Pr[(u,z) \text{ matched before } t \mid t_v = t] \\ &\qquad = \int_0^t \Big(\Pr[u \to z \mid t_u = s, t_v = t] + \Pr[z \to u \mid t_z = s, t_v = t] \Big) \mathrm{d}s. \qquad (**)\end{aligned}$$

We can thus expand the term $\Pr[u \to z \mid t_v = t, t_u = s]$ as in $(*)$, and we would have a term $\Pr[z \text{ matched before } s \mid t_u = s, t_v = t]$ therein. However, since our final goal is to derive a lower bound of $(*)$, we need a lower bound of this term rather than an upper bound as we derived in the warm-up analysis. This is where we use the 1-regularity of the instance.

Intuitively, the larger the degree of a vertex, the larger the probability it is matched. It is not difficult to construct a counterexample showing that our prune-greedy algorithm is indeed worse than $\frac{1}{2}$-selectable for non 1-regular instances. For technical reasons, we also want to avoid edges with large active probabilities and our pruning step is designed to take care of such edges.

Roughly speaking, $(*)$ together with $(**)$ expands $\Pr[v \to u \mid t_v = t]$ to terms of form $\Pr[u \to z \mid t_u = s, t_v = t]$. The formal analysis recursively applies such expansion for two steps and uses the simple bound in the warm-up analysis in the last step to bound $\Pr[j \text{ matched } @r \mid t_v = t, t_u = s, t_i = r]$. We defer the proof to Section 3.3.

3.2 Hardness

We complement our positive result with a hardness result, showing that no algorithm can be better than $\frac{1}{2} + \frac{1}{2e^2} \approx 0.567$-selectable.

▶ **Theorem 4.** *No batched RCRS for matching is better than $(\frac{1}{2} + \frac{1}{2e^2})$-selectable.*

Proof. Consider a complete graph G with n vertices. Each edge e has $x_e = \frac{1}{n-1}$. Fix an algorithm and let $y_i \cdot n$ be the number of matched vertices after the arrival of the i-th vertex, where $y_i \in [0, 1]$ is increasing and $y_1 = 0$. Notice that all edges are symmetric and every vertex has degree 1, the selectability of the algorithm is thus upper bounded by $\mathbf{E}[y_n]$. Moreover, we have that

$$\mathbf{E}\left[y_{i+1} \mid y_i\right] \leq y_i + 2 \cdot \left(\frac{i}{n-1} - y_i\right),$$

since we can select an edge only if 1) there exists an active edge, which happens with probability $\frac{i}{n-1}$ and 2) the corresponding vertex is unmatched. Taking expectation over y_i, we have that

$$\mathbf{E}\left[y_{i+1}\right] - \mathbf{E}\left[y_i\right] \leq 2 \cdot \left(\frac{i}{n-1} - \mathbf{E}\left[y_i\right]\right).$$

When n goes to infinity, the above family of inequalities converges to $\frac{dz_t}{dt} \leq 2 \cdot (t - z_t)$, where z_t corresponds to $\mathbf{E}[y_{\lfloor t \cdot n \rfloor}]$. Solving the differential equation gives us

$$z_t \leq t - \frac{1}{2} + \frac{1}{2}e^{-2t}, \forall t \in [0, 1] \ .$$

Hence, the total portion of matched vertices is upper bounded by $z_1 \leq \frac{1}{2} + \frac{1}{2e^2}$ when n goes to infinity, that implies no RCRS can be better than $\left(\frac{1}{2} + \frac{1}{2e^2}\right) \approx 0.567$-selectable. ◀

3.3 Proof of Lemma 3

Proof of Lemma 3. Fix an edge (u, v), our goal is to lower bound the probability of

$$\Pr[v \to u] = \int_0^1 \Pr[v \to u \mid t_v = r_v] dr_v. \tag{1}$$

As we discussed in the proof sketch, the probability term on the right side can be expanded as following

$$
\begin{aligned}
&\Pr[v \to u \mid t_v = r_v] \\
&= f(x_{uv}) \cdot (\Pr[u \text{ arrives before } t_v] - \Pr[u \text{ is matched at } t_v]) \\
&= f(x_{uv}) \cdot \left(t_v - \sum_{\substack{(u,i)\in E \\ i \neq u,v}} \Pr[(u,i) \text{ is selected before } t_v \mid t_v = r_v] \right) \\
&= f(x_{uv}) \cdot \left(t_v - \sum_{\substack{(u,i)\in E \\ i \neq u,v}} (\Pr[u \to i \text{ and } t_u \le t_v \mid t_v = r_v] + \Pr[i \to u \text{ and } t_i \le t_v \mid t_v = r_v]) \right).
\end{aligned} \tag{2}
$$

Since both terms in the right side of (2) are symmetric, we will only show how to upper bound the first term $\Pr[u \to i \text{ and } t_u \le t_v \mid t_v = r_v]$. As explained in the sketch, we further recursively expand it in essentially the same way as $\Pr[v \to u \mid t_v = r_v]$.

$$
\begin{aligned}
&\Pr[u \to i \text{ and } t_u \le t_v \mid t_v = r_v] \\
&= \int_0^{t_v} \Pr[u \to i \mid t_v = r_v, t_u = r_u] dr_u \\
&= \int_0^{t_v} f(x_{ui})(t_u - \Pr[i \text{ is matched at } t_u \mid t_v = r_v, t_u = r_u]) dr_u \\
&= f(x_{ui}) \cdot \int_0^{t_v} \left(t_u - \sum_{\substack{(i,j)\in E \\ j \neq i,u,v}} \Pr[(i,j) \text{ is selected before } t_u \mid t_u = r_u, t_v = r_v] \right) dr_u \\
&= f(x_{ui}) \cdot \int_0^{t_v} \Bigg(t_u - \sum_{\substack{(i,j)\in E \\ j \neq i,u,v}} (\Pr[i \to j, t_i \le t_u \mid t_u = r_u, t_v = r_v] \\
&\qquad\qquad + \Pr[j \to i, t_j \le t_u \mid t_u = r_u, t_v = r_v]) \Bigg) dr_u.
\end{aligned} \tag{3}
$$

Now we further expand the last equation and apply similar argument as the warm up analysis in the proof sketch.

$$
\begin{aligned}
&\Pr[i \to j, t_i \le t_u \mid t_u = r_u, t_v = r_v] \\
&= \int_0^{t_u} \Pr[i \to j \mid t_i = r_i, t_u = r_u, t_v = r_v] dr_i \\
&= \int_0^{t_u} f(x_{ij}) \left(t_i - \Pr[j \text{ is matched at } t_i \mid t_i = r_i, t_u = r_u, t_v = r_v] \right) dr_i.
\end{aligned} \tag{4}
$$

As in the proof sketch, we enumerate the vertex k which j is matched to.

$$
\begin{aligned}
&\Pr[j \text{ is matched at } t_i \mid t_i = r_i, t_u = r_u, t_v = r_v] \\
&\qquad = \sum_{\substack{(j,k)\in E \\ k \neq i,j,u,v}} \Pr[(j,k) \text{ is selected before } t_i \mid t_i = r_i, t_u = r_u, t_v = r_v].
\end{aligned} \tag{5}
$$

Notice that if an edge (j,k) is selected by the algorithm R before time t_i, it has to satisfy at least three conditions:

1. j arrives before t_i;
2. k arrives before t_i;
3. the edge is sampled by the sampling scheme S and selected by the algorithm.

One can observe that these three events are independent. In order to bound (4), we can apply the trivial upper bound to (5)

$$\Pr[(j,k) \text{ is selected before } t_i \mid t_i = r_i, t_u = r_u, t_v = r_v] \leq t_i^2 f(x_{jk})$$

and we obtain

$$\begin{aligned}
&\Pr[i \to j, t_i \leq t_u \mid t_u = r_u, t_v = r_v] \\
&\geq f(x_{ij}) \cdot \int_0^{t_u} t_i \left(1 - t_i \sum_{\substack{(j,k)\in E \\ k \neq i,j,u,v}} f(x_{jk})\right) dt_i \\
&\geq f(x_{ij}) \cdot \int_0^{t_u} t_i \cdot (1 - t_i \cdot (1 - x_{ij}))\, dt_i = f(x_{ij}) \cdot \left(\frac{1}{2}t_u^2 - \frac{1}{3}t_u^3(1 - x_{ij})\right).
\end{aligned}$$

Similarly, it also holds that

$$\Pr[j \to i, t_j \leq t_u \mid t_u = r_u, t_v = r_v]) \geq f(x_{ij}) \cdot \left(\frac{1}{2}t_u^2 - \frac{1}{3}t_u^3(1 - x_{ij})\right).$$

Now we start to plug them back to the two-level recursive analysis. First, plugging these two inequalities into (3), we can bound the first term in the right side of (2).

$$\begin{aligned}
&\Pr[u \to i \text{ and } t_u \leq t_v \mid t_v = r_v] \\
&\leq f(x_{ui}) \cdot \int_0^{t_v} \left(t_u - \sum_{\substack{(i,j)\in E \\ j \neq i,u,v}} f(x_{ij}) \cdot \left(t_u^2 - \frac{2}{3}t_u^3(1 - x_{ij})\right)\right) dt_u \\
&= f(x_{ui}) \cdot \left(\frac{1}{2}t_v^2 - \sum_{\substack{(i,j)\in E \\ j \neq i,u,v}} f(x_{ij}) \cdot \left(\frac{1}{3}t_v^3 - \frac{1}{6}t_v^4(1 - x_{ij})\right)\right).
\end{aligned}$$

By symmetry,

$$\Pr[i \to u \text{ and } t_i \leq t_v \mid t_v = r_v] \leq f(x_{ui}) \cdot \left(\frac{1}{2}t_v^2 - \sum_{\substack{(u,j)\in E \\ j \neq i,u,v}} f(x_{uj}) \cdot \left(\frac{1}{3}t_v^3 - \frac{1}{6}t_v^4(1 - x_{uj})\right)\right).$$

Then, by plugging in both terms back to (2), we have

$$\begin{aligned}
(2) \geq f(x_{uv}) \cdot \Bigg(t_v &- \sum_{\substack{(u,i)\in E \\ i \neq u,v}} f(x_{ui}) \cdot \left(\frac{1}{2}t_v^2 - \sum_{\substack{(i,j)\in E \\ j \neq i,u,v}} f(x_{ij}) \cdot \left(\frac{1}{3}t_v^3 - \frac{1}{6}t_v^4(1 - x_{ij})\right)\right) \\
&- \sum_{\substack{(u,i)\in E \\ i \neq u,v}} f(x_{ui}) \cdot \left(\frac{1}{2}t_v^2 - \sum_{\substack{(u,j)\in E \\ j \neq i,u,v}} f(x_{uj}) \cdot \left(\frac{1}{3}t_v^3 - \frac{1}{6}t_v^4(1 - x_{uj})\right)\right)\Bigg).
\end{aligned}$$

Therefore, (1) can be further bounded as follows:

$$
\begin{aligned}
(1) &\geq f(x_{uv}) \cdot \left(\frac{1}{2} - \sum_{\substack{(u,i)\in E \\ i\neq u,v}} f(x_{ui}) \cdot \left(\frac{1}{6} - \sum_{\substack{(i,j)\in E \\ j\neq i,u,v}} f(x_{ij}) \cdot \left(\frac{1}{20} + \frac{1}{30} x_{ij} \right) \right) \right. \\
&\qquad \left. - \sum_{\substack{(u,i)\in E \\ i\neq u,v}} f(x_{ui}) \cdot \left(\frac{1}{6} - \sum_{\substack{(u,j)\in E \\ j\neq i,u,v}} f(x_{uj}) \cdot \left(\frac{1}{20} + \frac{1}{30} x_{uj} \right) \right) \right) \\
&= f(x_{uv}) \cdot \left(\frac{1}{2} - \sum_{\substack{(u,i)\in E \\ i\neq u,v}} f(x_{ui}) \cdot \left(\frac{1}{6} - \sum_{\substack{(i,j)\in E \\ j\neq i,u,v}} \frac{1}{20} x_{ij} \right) - \sum_{\substack{(u,i)\in E \\ i\neq u,v}} f(x_{ui}) \cdot \left(\frac{1}{6} - \sum_{\substack{(u,j)\in E \\ j\neq i,u,v}} \frac{1}{20} x_{uj} \right) \right) \\
&= f(x_{uv}) \cdot \left(\frac{1}{2} - \sum_{\substack{(u,i)\in E \\ i\neq u,v}} f(x_{ui}) \cdot \left(\frac{1}{3} - \frac{1}{20}(1 - x_{ui} - x_{vi}) - \frac{1}{20}(1 - x_{ui} - x_{uv}) \right) \right) \\
&= f(x_{uv}) \cdot \left(\frac{1}{2} - \sum_{\substack{(u,i)\in E \\ i\neq u,v}} f(x_{ui}) \cdot \left(\frac{7}{30} + \frac{1}{20}(2x_{ui} + x_{vi} + x_{uv}) \right) \right).
\end{aligned}
$$

▶ Remark. Note here for the first equality, we use the fact that $f(x)\left(\frac{1}{20} + \frac{1}{30}x\right) = cx$ for $c = \frac{1}{20}$. (Actually, we only need one side that it is larger than cx.) The choice of $f(x) = \frac{3x}{3+2x}$ is to maximize c while keeping $f(x) \leq x$ for all $x \in [0,1]$ so that it is a valid pruning. For the second equality (where again we only need one side), we use the fact that $\sum_{\substack{(i,j)\in E \\ j\neq i,u,v}} x_{i,j} \geq 1 - x_{u,i} - x_{v,i}$ which follows from the 1-regularity of our graph.

Finally we are ready to bound $\Pr[(u,v) \text{ is selected}]$. Without loss of generality, we can assume that $x_{uv} = 0$ for those $(u,v) \notin E$. Recall that $\Pr[(u,v) \text{ is selected}] = \Pr[u \to v] + \Pr[v \to u]$ by definition and we have

$$
\begin{aligned}
&\Pr[(u,v) \text{ is selected}] \\
&\geq f(x_{uv}) \cdot \left(1 - \sum_{i\neq u,v} \left(f(x_{ui}) \cdot \left(\frac{7}{30} + \frac{1}{20}(2x_{ui} + x_{vi} + x_{uv}) \right) + \right. \right. \\
&\qquad \left. \left. f(x_{vi}) \cdot \left(\frac{7}{30} + \frac{1}{20}(2x_{vi} + x_{ui} + x_{uv}) \right) \right) \right) \\
&\geq f(x_{uv}) \cdot \left(1 - \frac{1}{10} x_{uv} - \sum_{i\neq u,v} \left(f(x_{ui}) \cdot \left(\frac{7}{30} + \frac{1}{20}(2x_{ui} + x_{vi}) \right) \right. \right. \\
&\qquad \left. \left. + f(x_{vi}) \cdot \left(\frac{7}{30} + \frac{1}{20}(2x_{vi} + x_{ui}) \right) \right) \right) \\
&\geq f(x_{uv}) \cdot \left(1 - \frac{1}{10} x_{uv} - \sum_{i\neq u,v} \frac{7}{30} \cdot (x_{ui} + x_{vi}) \right) \\
&= f(x_{uv}) \cdot \left(1 - \frac{1}{10} x_{uv} - \frac{7}{30} \cdot (2 - 2x_{uv}) \right) \\
&= \frac{3x_{uv}}{3 + 2x_{uv}} \cdot \left(\frac{8}{15} + \frac{11}{30} x_{uv} \right) \geq \frac{8}{15} x_{uv}.
\end{aligned}
$$

Here the second inequality uses the fact that $\sum_{i\neq u,v} f(x_{ui}) \leq \sum_{i\neq u,v} x_{ui} \leq 1$ and $\sum_{i\neq u,v} f(x_{vi}) \leq \sum_{i\neq u,v} x_{vi} \leq 1$. We are left to prove the third inequality, which we state as the following lemma.

▶ **Lemma 5.** *The inequality*

$$f(x)\cdot\left(\frac{7}{30}+\frac{1}{20}(2x+y)\right)+f(y)\cdot\left(\frac{7}{30}+\frac{1}{20}(2y+x)\right)\leq\frac{7}{30}\cdot(x+y)$$

holds for every $x, y \geq 0$.

Proof. It is equivalent to prove that

$$\frac{1}{20}f(x)(2x+y)+\frac{1}{20}f(y)(2y+x)\leq\frac{7}{30}(x-f(x)+y-f(y)).$$

Expand $f(x)$ and we get

$$\frac{1}{10}\left(\frac{3x^2}{3+2x}+\frac{3y^2}{3+2y}\right)+\frac{1}{20}\left(\frac{3xy}{3+2x}+\frac{3xy}{3+2y}\right)\leq\frac{7}{15}\left(\frac{3x^2}{3+2x}+\frac{3y^2}{3+2y}\right).$$

Rewriting the formula, we find that it's equivalent to prove

$$30x^2+30y^2-54xy+2x^2y+2xy^2\geq 0.$$

Notice that $x, y \geq 0$, thus $2x^2y+2xy^2 \geq 0$. Also, $30x^2+30y^2-54xy \geq 30x^2+30y^2-60xy \geq 30(x-y)^2 \geq 0$, which completes our proof. ◀

This finishes the proof of Lemma 3. ◀

4 Applications

Our definition of the batched RCRS for matching with random vertex arrival is inspired by the standard reduction from prophet matching to online contention resolution schemes. Therefore, Theorem 1 immediately implies the first nontrivial result for prophet matching in general graphs with random vertex arrival.

▶ **Theorem 6.** *There is an* $\frac{8}{15}$*-competitive algorithm for prophet matching in general graphs with random vertex arrival.*

In this section, we mainly discuss the application of our batched RCRS in Query-Commit and Price-of-Information problems for matching. The algorithms below can be seen as reinterpretations of Gamlath et al. [15]'s algorithms for the problems on bipartite matching. For Query-Commit, their algorithm first solves a linear program relaxation of the corresponding problem, then, using characterization of polymatroids, interprets the fractional solution as follows: each vertex u on the left samples a permutation over the edges in δ_u, and *proposes* to query these edges in that order; literally following these proposals for every vertex obviously leads to collisions, and so vertices on the right need to resolve potential collisions by turning down some of the proposals. The algorithm lets the vertices on the left arrive in a uniformly random order to propose their permutations. Each vertex on the right then runs a *prophet secretary* algorithm, which guarantees that, in expectation, the proposed queries turned down by the nodes on the right do not carry too much value. We suggest that, one may bypass the more complex prophet secretary setup, by seeing the last step as an online contention resolution step with random vertex arrival: when vertex u arrives and proposes its permutation, if we let edge $e \in \delta_u$ be *active* if it is the first edge present when we query the edges following the proposed order, then a c-selectable batched RCRS with vertex arrival keeps each active edge with probability at least c, which leads to a c-approximation algorithm. The description at this level omits many details and twists, and the Price-of-Information algorithm has even more of those. Nonetheless, our perspective easily generalizes to the non-bipartite cases, and allows us to use the batched RCRS in Section 3 to these problems. We give the details below.

4.1 The (Weighted) Query-Commit Problem for Matching

We now present an $\frac{8}{15}$-competitive algorithm for the Query-Commit problem by a reduction to batched RCRS.

▶ **Theorem 7.** *There is a polynomial-time computable, $\frac{8}{15}$-competitive algorithm for the Query-Commit problem for weighted matching in graphs not necessarily bipartite.*

4.1.1 Bounding the optimal utility

First we construct a linear program, where for every edge $e \in E$, x_e represents the probability that e is present and included in the solution. We show that the value of the linear program is an upper bound on even the offline optimal. Therefore, if we implement an algorithm so that each edge e is selected with probability at least $\alpha \cdot x_e^*$, then the competitive ratio of the algorithm will be at least α.

Think of x_e as the probability with which edge e is present and included in the offline solution. For a subset of edges $F \subseteq E$, let $f(F)$ be the probability that at least one edge in F exists, i.e., $f(F) = 1 - \prod_{e \in F}(1 - p_e)$. For any $F \subseteq \delta_u$, since in a matching no more than one edge in F can be in the solution, $\sum_{e \in F} x_e$ is the probability any edge in F is in the offline solution, and this should be upper bounded by the probability any edge in F is present. The following linear program $\mathrm{LP}_{\mathrm{QC}}$ therefore upper bounds the offline optimal:

$$\begin{aligned} \text{max:} \quad & \sum_{e \in E} x_e \cdot v_e \\ \text{s.t.} \quad & \sum_{e \in F} x_e \le f(F), && \forall u \in V, F \subseteq \delta_u; \\ & x_e \ge 0, && \forall e \in E. \end{aligned}$$

▶ **Lemma 8** ([15]). *In the edge-weighted Query-Commit problem for matching, the offline optimal is upper bounded by the value of $\mathrm{LP}_{\mathrm{QC}}$. Furthermore, $\mathrm{LP}_{\mathrm{QC}}$ is polynomial-time solvable.*

Lemma 8 is a straightforward generalization of Lemma 2.1 and Lemma 2.2 in [15]. In the following, we need to assume $0 < p_e < 1$ for every $e \in E$ since some proofs requires strict monotonicity and strict submodularity of f. It turns out that we can ignore those edges with zero probabilities and scale down probabilities by $1 - \gamma$ for other edges due to the following lemma.

▶ **Lemma 9** (Lemma 2.3 of [15]). *For $0 < \gamma < 1$, let $\tilde{p}_e = (1 - \gamma)p_e$ for every $e \in E$. Define $\tilde{f}$ using $\tilde{p}$ instead of p. Similarly, define $\widetilde{\mathrm{LP}}_{\mathrm{QC}}$ use $\tilde{p}, \tilde{f}$ instead of of p, f. Then the value of $\widetilde{\mathrm{LP}}_{\mathrm{QC}}$ is at least $(1 - \gamma)$ times the value of $\mathrm{LP}_{\mathrm{QC}}$.*

For any solution $\boldsymbol{x}$ to $\mathrm{LP}_{\mathrm{QC}}$, the following lemma defines a "decomposition into permutations" over (subsets of) δ_u for each $u \in V$. It is essentially Lemma 2.6 in [15] restated on general graphs.

▶ **Lemma 10** ([15]). *Suppose $0 < p_e < 1$ for every $e \in E$. Let $\boldsymbol{x}^*$ be an optimal solution to $\mathrm{LP}_{\mathrm{QC}}$. For every $u \in V$, fix any subset $\delta'_u \subseteq \delta_u$. Then there exists a polynomial-time samplable distribution $\mathcal{D}_u^{\mathrm{QC}}$ over the permutations on subsets of δ'_u, so that, if one queries according to the permutation sampled from $\mathcal{D}_u^{\mathrm{QC}}$, each edge e is the first present one with probability x_e^*.*

The meaning that the permutations are over *subsets of* δ'_u is that some edges may not be present in the permutation (which indicates that they should not be queried).

4.1.2 Rounding with batched RCRS

The game plan becomes clearer with the decomposition from Lemma 10. Ideally, if one may naïvely follow the permutation sampled from $\mathcal{D}_u^{\mathrm{QC}}$ for each u, and take the first present edge, then one may recover the offline optimal. This is of course infeasible, because matching constraints may be violated by collisions caused by doing this for different vertices. Gamlath et al. [15] handled this via a clever application of prophet secretary algorithms. Instead, we replace this with a reduction to the batched RCRS we developed in Section 3.

We may view the naïve algorithm which always commits to the first present edge in $\sigma_u \sim \mathcal{D}_u^{\mathrm{QC}}$ as a sampling scheme S_{QC} that indicates the first present edge as *active*, and apply our batched RCRS. The matching constraint is then respected by the feasibility of RCRS solutions. However, there is one more subtlety: in the Query-Commit problem, we must commit to the edge we just queried, whereas a batched RCRS assumes that it can see an active edge and discards it. Therefore S_{QC} should not query the presence of an edge if the batched RCRS algorithm R would not accept it *even if it is present*. To handle this issue, we need to change the order of the events by modifying the sampling scheme S_{QC}: upon the arrival of each vertex u, S_{QC} first obtains an indicator vector $\boldsymbol{I}$ from the batched RCRS algorithm R, where $I_e = 1$ if and only if R is willing to accept edge e if S_{QC} indicates that e is active. If $I_e = 0$, instead of actual querying the presence of edge e, S_{QC} simply tosses a coin to simulate a query. We remark that Gamlath et al.'s algorithm has a similar argument. Following is the formal description for our reduction from a Query-Commit problem to batched RCRS.

Our algorithm. Our algorithm first solves $\mathrm{LP}_{\mathrm{QC}}$ to get an optimal solution $\boldsymbol{x}^*$. Let R be a batched RCRS instance corresponding to graph $\langle V, E, \boldsymbol{x}^* \rangle$. Let S_{QC} be the sampling scheme to be defined later. Additionally, for each vertex $u \in V$ we sample $t_u \sim U[0,1]$ as its arrival time. We define $\delta'_u = \{(u,w) \in \delta_u \mid t_w < t_u\}$ to be the edges batch that arrives with u.

Our algorithm iterates over all vertices $u \in V$ in the increasing order of t_u. For each vertex u, it first obtains the indicator vector $\boldsymbol{I}$ where I_e indicates whether R is willing to accept e if it is active. Passing $\boldsymbol{I}$ to the sampling scheme S_{QC}, it obtains the active edge $e \in \delta'_u$ for RCRS algorithm R. We commit to e if R accepts it.

The sampling scheme S_{QC}. Let u be the vertex just arrived in batched RCRS. First sample a permutation σ_u from $\mathcal{D}_u^{\mathrm{QC}}$ by Lemma 10, which is a permutation containing a subset of δ'_u. Consider each edge $e = (u,w)$ in the order of σ_u. There are two cases:

- If $I_e = 1$: query if e is active in the query-commit instance. If it is active, report e as the active edge to R and exit; otherwise continue to the next edge.
- If $I_e = 0$: with probability p_e, report e as the active edge to R and exit; otherwise continue to the next edge.

For the analysis, we should first verify that S_{QC} is a valid sampling scheme. First, it will sample at most one active edge for a vertex, and the sampling result is independent from the indicator vector $\boldsymbol{I}$ since when considering an edge e, in both cases it essentially do a coin flip which heads up with probability p_e and use the result to determine whether e is active. Having observed the independency, we can show that x_e^* is the probability of edge e being active in S_{QC} for any arrival times by using Lemma 10.

▶ **Lemma 11.** *Fix arrival times $\{t_u\}_{u \in V}$. For every vertex $u \in V$ and every edge $e \in \delta'_u$, the probability of e being active in S_{QC} upon the arrival of u equals x_e^*.*

Proof. Fixing a vertex $u \in V$, we have that the probability of every edge $e \in \delta'_u$ being the first active edge in the random permutation σ_u equals x^*_e, according to Lemma 10. The problem is that in the second case, the algorithm does not even check whether an edge is active in the query-commit instance.

Nevertheless, in both cases each edge e will be active independently with probability p_e, which is exactly the probability of e being active in the query-commit instance. Therefore, the random process for determining the active edge is identical to finding the first active element in σ_u. And we have the probability of e being active is exactly x^*_e. ◀

At last, $\sum_{e\in\delta_u} x^*_e \le 1$ hold for every $u \in V$ trivially by constraints in $\mathrm{LP}_{\mathrm{QC}}$. We conclude the validity of S_{QC} with Lemma 12.

▶ **Lemma 12.** *S_{QC} is a valid sampling scheme for batched RCRS. In particular, the following three conditions hold:*

1. *each time S_{QC} will sample at most one edge and the result is independent from $\boldsymbol{I}$;*
2. *each edge $e \in E$ will be active with probability x^*_e for any arrival times $\{t_u\}_{u\in V}$;*
3. *$\sum_{e\in\delta_u} x^*_e \le 1$ holds for every $u \in V$.*

We now are ready to prove the competitive ratio for the algorithm by the selectability of the batched RCRS.

Proof of Theorem 7. The validity of batched RCRS is already proven in Lemma 12. We start the rest of the proof by showing the correctness of the reduction. First, those committed edges must form a matching by the correctness of R. And once a query of e has succeed, R will always decide to take e because we only query e when the indicator vector $I_e = 1$. Finally, no edge will be committed without a query since every edge e that has not been queried will be active to R only when $I_e = 0$.

Then we can easily show the ratio of the reduction equals the selectability of the batched RCRS. By Lemma 11 and the fact that $\cup_{u\in V}\delta'_u = E$, the probability of every edge $e \in E$ being active in R is x^*_e. By Theorem 1, the expected utility for our algorithm is at least $\frac{8}{15}\cdot\sum_{e\in E} x^*_e \cdot v_e$. Further by Lemma 8, we conclude it is an $\frac{8}{15}$-competitive algorithm for the query-commit problem on general graphs. ◀

4.2 The Price of Information Problem for Matching

The Price of Information (PoI) problem has search costs but imposes no obligation on an algorithm to immediately accept a queried element regardless of what is revealed. Kleinberg et al. [22] upper bounded the optimal utility using a variant of query-commit, by giving a new proof for the optimal algorithm in the special case where $\mathcal{F}$ is the set of singleton sets (known as the Pandora's Box problem [29]); Singla [27] generalized the bound and proposed approximation algorithms using the bound. Gamlath et al. [15] studied the Price of Information problem for bipartite matching and made use of the upper bound.

Their method converts the original "price-of-information" world to "free-information" world by setting a threshold value τ_i for each element. In "free-information" world, there is no search cost c_i, but the algorithm gets a lower utility $\kappa_i = \min\{v_i, \tau_i\}$ (instead of v_i) for accepting an element i. Intuitively, the $(v_i - \tau_i)_+$ part[3] of the utility pays for the search cost. However, this is only the case if the algorithm accepts element i in the end. If the algorithm queried element i without accepting it, the utility of the algorithm in "free-information" world

[3] $(z)_+$ denotes $\max\{z, 0\}$

will only be an upper bound of that in the "price-of-information" world. In Section 4.2.1, we upper bound the optimal utility in "free-information" world. Therefore, to match this upper bound, the algorithm has to always accept element i whenever element i is queried and $v_i > \tau_i$.

The intuition above is summarized by the following lemma from [27].

▶ **Lemma 13** ([27]). *In any instance of Price of Information problem, for each element $i \in U$, let τ_i be the unique solution to the equation $\mathbf{E}_{v_i \sim F_i}[(v_i - \tau_i)_+] = c_i$, where $(z)_+$ denotes $\max\{z, 0\}$. Let κ_i be $\min\{v_i, \tau_i\}$. Then no algorithm's utility exceeds $\mathbf{E}[\max_{S \in \mathcal{F}} \sum_{i \in S} \kappa_i]$. In particular, in order for algorithm $\mathcal{A}$ to match this upper bound, $\mathcal{A}$ must accept each element i such that i is queried by $\mathcal{A}$ and $v_i > \tau_i$.*

Similar as query-commit setting, our algorithm is essentially the same as that of [15], but we provide the perspective that the use of prophet secretary in [15] can be replaced by batched RCRS and extend their results to general graphs.

▶ **Theorem 14.** *There is a polynomial time, $\frac{8}{15}$-approximate algorithm for the price of information problem for weighted matching in graphs not necessarily bipartite.*

4.2.1 Bounding the optimal utility

We start by presenting the LP which upper bounds the optimal utility in the "free-information" world. Therefore by Lemma 13, it is also an upper bound for the optimal utility in the "price-of-information" world. The LP and the proofs are essentially the same as that in [15] with the only difference that this is for general graph.

Recall in the PoI problem for matching, there is an undirected graph $G = (V, E)$. For each edge $e \in E$, its value is a random variable $v_e \sim F_e$. Then we set τ_e and κ_e as in Lemma 13.

Without loss of generality, we assume that the distributions of κ_e are discrete.[4] Let K_e be the set of possible values of κ_e. For all $u \in V$, let $E_u = \{(e, \kappa) : e \in \delta_u, \kappa \in K_e\}$ be the edge-value pairs incidents to u and $E_{\text{all}} = \cup_{u \in U} E_u$ be the set of all edge-value pairs in the graph. For each $F \subseteq E_u$, we define $f(F)$ be the probability that $\kappa_e = \kappa$ for at least one of the edge-value pair $(e, \kappa) \in F$. Namely $f(F) = \prod_{v \in V}(1 - \sum_{e:(e,\kappa) \in F} p_{e,\kappa})$ where $p_{e,\kappa} = \Pr[\kappa_e = \kappa]$.

For any algorithm $\mathcal{A}$ for the PoI problem for matching, let $x_{e,\kappa}$ be the probability that $\mathcal{A}$ accepts edge e and $\kappa_e = \kappa$. For any $F \subseteq E_u$, similar with Section 4.1, we know that $\sum_{(e,\kappa) \in F} x_{e,\kappa} \leq f(F)$. Therefore it is natural to consider the following LP, which is called LP_{PoI}.

$$\begin{aligned} \max: \quad & \sum_{(e,\kappa) \in E_{\text{all}}} x_{e,\kappa} \cdot \kappa \\ \text{s.t.} \quad & \sum_{(e,\kappa) \in F} x_{e,\kappa} \leq f(F), && \forall u \in V, F \subseteq E_u; \\ & x_e \geq 0, && \forall e \in E_{\text{all}}. \end{aligned}$$

We restate Lemma 3.2 and Lemma 3.3 of [15] below for general graphs.

[4] In the case where the distributions are continuous. We can discretize them by geometric grouping edge weights into classes.

▶ **Lemma 15** ([15]). *In the price of information problem for matching, the optimal expected utility* $\mathrm{OPT}_{\mathrm{PoI}}$ *is upper bounded by the value of* $\mathrm{LP}_{\mathrm{PoI}}$*. Furthermore,* $\mathrm{LP}_{\mathrm{PoI}}$ *is polynomial-time solvable.*

4.2.2 Rounding with batched RCRS

Let $\boldsymbol{x}^*$ be an optimal solution of $\mathrm{LP}_{\mathrm{PoI}}$. We proceed to "round" it to an algorithm for the PoI problem. Let $\langle V, E, \boldsymbol{x}_{\mathrm{PoI}}\rangle$ be the batched RCRS instance. Note the graph of the RCRS instance is the same as the original graph. Before the formal description of S_{PoI}, we first sketch it here:

From the LP solution $\boldsymbol{x}^*$, we first obtain a polynomial-time samplable distribution $\mathcal{D}_u^{\mathrm{PoI}}$ over permutations of edge-value pairs (e,κ). S_{PoI} first draw a permutation σ from $\mathcal{D}_u^{\mathrm{PoI}}$. An edge-value pair (e,κ) is said to be active if $\kappa_e = \kappa$. Roughly speaking, S_{PoI} returns the first active edge-value pair in σ as the active edge. However, as Lemma 15 suggests, our algorithm must accept edge e if e is queried and $v_e = \tau_e$. Therefore, there is an additional requirement that in each σ drawn from $\mathcal{D}_u^{\mathrm{PoI}}$, the edge-value pairs of the same edge should be in decreasing order of their values. Hence (e,τ_e) is always the first pair associated with edge e in σ.

The following lemma formally defines the distribution $\mathcal{D}_u^{\mathrm{PoI}}$. It is a restatement of Lemma 3.6 in [15].

▶ **Lemma 16** ([15]). *Suppose* $0 < p_{e,\kappa} < 1$ *for every* $(e,\kappa) \in E_{\mathrm{all}}$*. Let* $\boldsymbol{x}^*$ *be an optimal solution to* $\mathrm{LP}_{\mathrm{PoI}}$*. For every* $u \in V$*, fix any subset* $\delta'_u \subseteq \delta_u$*, and let* $E'_u = \{(e,\kappa) \in E_u, e \in \delta'_u\}$ *be the corresponding edge-value pairs.*

We call an edge-value pair (e,κ) *active if* $\kappa_e = \kappa$*. Let*

$$y_{e,\kappa} = \Pr_{\sigma \sim \mathcal{D}_u^{\mathrm{PoI}}, \{\kappa_e\}}[(e,\kappa) \text{ is the first active pair in } \sigma].$$

Then there exists a polynomial-time samplable distribution $\mathcal{D}_u^{\mathrm{PoI}}$ *over the permutations on (subsets of)* E'_u *such that the following holds:*

1. *For all permutation* σ *drawn from* $\mathcal{D}_u^{\mathrm{PoI}}$*, if edge-value pair* (e,κ) *appears in* σ*, then the edge-value pair* (e,w) *appears before* (e,κ) *in* σ *for all* $w \in K_e$ *such that* $w \geq \kappa$*.*
2. $\sum_{\kappa \in K_e} y_{e,\kappa} = \sum_{\kappa \in K_e} x^*_{e,\kappa}$ *for all* $e \in \delta'_u$*.*
3. $\sum_{\kappa \in K_e} y_{e,\kappa} \cdot \kappa \geq \sum_{\kappa \in K_e} x^*_{e,\kappa} \cdot \kappa$ *for all* $e \in \delta'_u$*.*

Now we formally define the sampling scheme S_{PoI}. Initially, each vertex $u \in V$ has its arrival time $t_u \sim U[0,1]$. We define $\delta'_u = \{(u,w) \in \delta_u \mid t_w < t_u\}$ to be the edges batch that arrives with u, and let $E'_u = \{(e,\kappa) \mid e \in \delta'_u, \kappa \in K_e\}$ be the corresponding edge-value pairs. When u arrives, using Lemma 16, S_{PoI} draw a permutation σ from $\mathcal{D}_u^{\mathrm{PoI}}$ over (subsets of) E'_u.

The subtlety in S_{QC} still exists in S_{PoI}, i.e., S_{PoI} should not query the value of an edge if batched RCRS algorithm R do not accept it. Therefore, S_{PoI} first obtains an indicator vector $\boldsymbol{I}$ from RCRS algorithm R. $I_e = 1$ if and only if R would accept edge e if S_{PoI} choose it to be active. S_{PoI} draws variables κ'_e from the same distribution of κ_e for all $e \in \delta'_u$. If $I_e = 0$, namely R will not take the edge, instead of query the true κ_e, S_{PoI} simply use κ'_e to replace it. In this way, the exact behavior of S_{PoI} depends on $\boldsymbol{I}$. Nevertheless, the returned edge of S_{PoI} is still independent of $\boldsymbol{I}$.

S_{PoI} handles each pair $\sigma_i = (e_i, \kappa_i)$ in order as follows:

- If $I_{e_i} = 1$: S_{PoI} queries the value of κ_{e_i} if it is not queried before. If $\kappa_{e_i} = \kappa_i$, S_{PoI} returns e_i as active edges. Otherwise, it continues to the next edge.
- If $I_{e_i} = 0$: S_{PoI} looks at κ'_{e_i} instead since R will not accept the edge anyway. If $\kappa'_{e_i} = \kappa_i$, S_{PoI} returns e_i as the active edges. Otherwise, it continues to the next edge.

The proof of validity for S_{PoI} is similar to the proof for S_{QC}.

▶ **Lemma 17.** *S_{PoI} is a valid sampling scheme for batched RCRS. In particular, the following three conditions hold:*

1. *each time S_{PoI} will sample at most one edge and the result is independent from $\boldsymbol{I}$.*
2. *Each edge $e \in E$ is active with probability $x_e = \sum_{\kappa \in K_e} x^*_{e,\kappa}$ for any arrival times $\{t_u\}_{u \in V}$. Specifically, the probability that S_{PoI} returns at edge-value pair (e, κ) is exactly $y_{e,\kappa}$ as defined in Lemma 16.*
3. *$\sum_{e \in \delta_u} x_e \le 1$ for all $u \in V$.*

Proof. Firstly, by the procedure of S_{PoI}, it returns only one edge. Since κ' and κ has the same distribution, the edge returned by S_{PoI} is independent of $\boldsymbol{I}$. Secondly, because of this independence, if we define $y_{e,\kappa}$ as in Lemma 16, the probability of S_{PoI} to return at edge-value pair (e, κ) is exactly $y_{e,\kappa}$. By the definition of x_e and part 2 of Lemma 16, we know that $x_e = \sum_{\kappa \in K_e} y_{e,\kappa} = \sum_{\kappa \in K_e} x^*_{e,\kappa}$. Thirdly, from the definition of LP_{PoI}, when $F = E_u$, we have $\sum_{(e,\kappa) \in E_u} x^*_{e,\kappa} \le 1$. Namely, $\sum_{e \in \delta_u} x_e \le 1$. ◀

In order to let the $(v_e - \tau_e)_+$ part of the utility pays for the search cost, the algorithm must accept an edge e if its v_e is larger than τ_e (which only happens when $\kappa_e = \tau_e$). The definition of S_{PoI} guarantees such property, and it will be useful in the proof of Theorem 14.

▶ **Lemma 18.** *Suppose an edge e is queried by the reduction and $\kappa_e = \tau_e$. Then the edge e is always accepted in the end.*

Proof. Since e is queried by the reduction, we know that $I_e = 1$, namely the RCRS algorithm R is willing to take this edge. By part 1 of Lemma 16, if (e, τ_e) is the first edge-value pair associated with e in σ. Therefore, as e is queried by S_{PoI}, it returns e as the active edge when handling (e, τ_e). Since $I_e = 1$, R accepts edge e as well. ◀

Now we are ready to prove Theorem 14, i.e. the approximation ratio of the algorithm.

Proof of Theorem 14. By Lemma 15, we know the optimum of LP_{PoI} is an upper bound of the optimal utility for the price of information problem for weighted matching. Namely, $\sum_{(e,\kappa) \in E_{\text{all}}} x^*_{e,\kappa} \cdot \kappa \ge \text{OPT}_{\text{PoI}}$. Recall K_e is the set of possible values of κ_e, and let V_e be the set of possible values of v_e.

On the other hand, by Lemma 18 we know when $v_e \ge \tau_e$, the edge is always accepted when queried. Together with the definition of τ_e, we know that the cost of query is

$$\begin{aligned}\sum_{e \in E} \Pr[e \text{ is queried}] \cdot c_e &= \sum_{e \in E} \Pr[e \text{ is queried}] \cdot \mathbf{E}_{v_i \sim F_i}\left[(v_e - \tau_e)_+\right] \\ &= \sum_{e \in E} \Pr[e \text{ is accepted}] \cdot \mathbf{E}_{v_i \sim F_i}\left[(v_e - \tau_e)_+\right].\end{aligned}$$

The utility of our algorithm is therefore

$$\begin{aligned}&\sum_{e \in E} \sum_{v \in V_e} \Pr[e \text{ is accepted and } v_e = v] \cdot v - \sum_{e \in E} \Pr[e \text{ is accepted}] \cdot \mathbf{E}_{v_i \sim F_i}\left[(v_e - \tau_e)_+\right] \\ =&\sum_{e \in E} \sum_{\kappa \in K_e} \Pr[e \text{ is accepted and } \kappa_e = \kappa] \cdot \kappa.\end{aligned}$$

The equality follows from $\kappa_e = v_e - (v_e - \tau_e)_+$. Let $y_{e,\kappa}$ be defined as Lemma 16. By Lemma 17, S_{PoI} is a valid sampling scheme. So we can apply the batched RCRS in Theorem 1. Since the only use of κ_e is to determine the active edge in S_{PoI}, conditioning on e is active, whether e is accepted is independent of κ_e.

Hence

$$\begin{aligned}
&\Pr[e \text{ is accepted and } \kappa_e = \kappa] \\
=\ &\Pr[e \text{ is accepted} \mid e \text{ is active and } \kappa_e = \kappa] \cdot \Pr[e \text{ is active and } \kappa_e = \kappa] \\
=\ &\Pr[e \text{ is accepted} \mid e \text{ is active}] \cdot \Pr[e \text{ is active and } \kappa_e = \kappa] \\
\geq\ &\frac{8}{15} y_{e,\kappa}.
\end{aligned}$$

Then by part 2 of Lemma 16, the utility of our algorithm is

$$\sum_{e \in E} \sum_{\kappa \in K_e} \frac{8}{15} y_{e,\kappa_e} \cdot \kappa \geq \frac{8}{15} \sum_{e \in E} \sum_{\kappa \in K_e} x^*_{e,\kappa} \cdot v = \frac{8}{15} \operatorname{OPT}_{\text{PoI}},$$

where the inequality follows from part 3 of Lemma 16. ◀

References

1. Marek Adamczyk and Michal Wlodarczyk. Random order contention resolution schemes. In Mikkel Thorup, editor, *59th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2018, Paris, France, October 7-9, 2018*, pages 790–801. IEEE Computer Society, 2018.
2. Saeed Alaei. Bayesian combinatorial auctions: Expanding single buyer mechanisms to many buyers. In *2011 IEEE 52nd Annual Symposium on Foundations of Computer Science*, pages 512–521, 2011.
3. Jonathan Aronson, Martin Dyer, Alan Frieze, and Stephen Suen. Randomized greedy matching. ii. *Random Structures & Algorithms*, 6(1):55–73, 1995.
4. Itai Ashlagi, Maximilien Burq, Chinmoy Dutta, Patrick Jaillet, Amin Saberi, and Chris Sholley. Edge weighted online windowed matching. In *EC*, pages 729–742. ACM, 2019.
5. Simon Bruggmann and Rico Zenklusen. An optimal monotone contention resolution scheme for bipartite matchings via a polyhedral viewpoint. *Mathematical Programming*, pages 1–51, 2020.
6. T.-H. Hubert Chan, Fei Chen, Xiaowei Wu, and Zhichao Zhao. Ranking on arbitrary graphs: Rematch via continuous linear programming. *SIAM J. Comput.*, 47(4):1529–1546, 2018.
7. Shuchi Chawla, Jason D. Hartline, David L. Malec, and Balasubramanian Sivan. Multi-parameter mechanism design and sequential posted pricing. In Leonard J. Schulman, editor, *Proceedings of the 42nd ACM Symposium on Theory of Computing, STOC 2010, Cambridge, Massachusetts, USA, 5-8 June 2010*, pages 311–320. ACM, 2010.
8. Chandra Chekuri, Jan Vondrák, and Rico Zenklusen. Submodular function maximization via the multilinear relaxation and contention resolution schemes. *SIAM J. Comput.*, 43(6):1831–1879, 2014.
9. Ning Chen, Nicole Immorlica, Anna R. Karlin, Mohammad Mahdian, and Atri Rudra. Approximating matches made in heaven. In *Automata, languages and programming. Part I*, volume 5555 of *Lecture Notes in Comput. Sci.*, pages 266–278. Springer, Berlin, 2009.
10. Nikhil R. Devanur and Kamal Jain. Online matching with concave returns [extended abstract]. In *STOC'12—Proceedings of the 2012 ACM Symposium on Theory of Computing*, pages 137–143. ACM, New York, 2012.
11. Soheil Ehsani, MohammadTaghi Hajiaghayi, Thomas Kesselheim, and Sahil Singla. Prophet secretary for combinatorial auctions and matroids. In *Proceedings of the twenty-ninth annual acm-siam symposium on discrete algorithms*, pages 700–714. SIAM, 2018.
12. Hossein Esfandiari, MohammadTaghi Hajiaghayi, Vahid Liaghat, and Morteza Monemizadeh. Prophet secretary. *SIAM Journal on Discrete Mathematics*, 31(3):1685–1701, 2017.
13. Tomer Ezra, Michal Feldman, Nick Gravin, and Zhihao Gavin Tang. Online stochastic max-weight matching: Prophet inequality for vertex and edge arrival models. In Péter Biró, Jason Hartline, Michael Ostrovsky, and Ariel D. Procaccia, editors, *EC '20: The 21st ACM Conference on Economics and Computation, Virtual Event, Hungary, July 13-17, 2020*, pages 769–787. ACM, 2020.

14 Moran Feldman, Ola Svensson, and Rico Zenklusen. Online contention resolution schemes. In Robert Krauthgamer, editor, *Proceedings of the Twenty-Seventh Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2016, Arlington, VA, USA, January 10-12, 2016*, pages 1014–1033. SIAM, 2016.

15 Buddhima Gamlath, Sagar Kale, and Ola Svensson. Beating greedy for stochastic bipartite matching. In Timothy M. Chan, editor, *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2019, San Diego, California, USA, January 6-9, 2019*, pages 2841–2854. SIAM, 2019.

16 Anupam Gupta, Haotian Jiang, Ziv Scully, and Sahil Singla. The markovian price of information. In *International Conference on Integer Programming and Combinatorial Optimization*, pages 233–246. Springer, 2019.

17 Anupam Gupta and Viswanath Nagarajan. A stochastic probing problem with applications. In Michel X. Goemans and José R. Correa, editors, *Integer Programming and Combinatorial Optimization - 16th International Conference, IPCO 2013, Valparaíso, Chile, March 18-20, 2013. Proceedings*, volume 7801 of *Lecture Notes in Computer Science*, pages 205–216. Springer, 2013.

18 Guru Guruganesh and Euiwoong Lee. Understanding the correlation gap for matchings. In *37th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science*, 2018.

19 Zhiyi Huang, Ning Kang, Zhihao Gavin Tang, Xiaowei Wu, Yuhao Zhang, and Xue Zhu. Fully online matching. *J. ACM*, 67(3):17:1–17:25, 2020.

20 Zhiyi Huang, Zhihao Gavin Tang, Xiaowei Wu, and Yuhao Zhang. Fully online matching II: beating ranking and water-filling. In *FOCS*, pages 1380–1391. IEEE, 2020.

21 Richard M. Karp, Umesh V. Vazirani, and Vijay V. Vazirani. An optimal algorithm for on-line bipartite matching. In *STOC*, pages 352–358. ACM, 1990.

22 Robert D. Kleinberg, Bo Waggoner, and E. Glen Weyl. Descending price optimally coordinates search. In Vincent Conitzer, Dirk Bergemann, and Yiling Chen, editors, *Proceedings of the 2016 ACM Conference on Economics and Computation, EC '16, Maastricht, The Netherlands, July 24-28, 2016*, pages 23–24. ACM, 2016.

23 Euiwoong Lee and Sahil Singla. Optimal online contention resolution schemes via ex-ante prophet inequalities. In Yossi Azar, Hannah Bast, and Grzegorz Herman, editors, *26th Annual European Symposium on Algorithms, ESA 2018, August 20-22, 2018, Helsinki, Finland*, volume 112 of *LIPIcs*, pages 57:1–57:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2018.

24 Brendan Lucier. An economic view of prophet inequalities. *SIGecom Exch.*, 16(1):24–47, 2017.

25 Aranyak Mehta. Online matching and ad allocation. *Found. Trends Theor. Comput. Sci.*, 8(4):265–368, 2012.

26 Aranyak Mehta, Amin Saberi, Umesh Vazirani, and Vijay Vazirani. AdWords and generalized online matching. *J. ACM*, 54(5):Art. 22, 19, 2007.

27 Sahil Singla. The price of information in combinatorial optimization. In Artur Czumaj, editor, *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2018, New Orleans, LA, USA, January 7-10, 2018*, pages 2523–2532. SIAM, 2018.

28 Zhihao Gavin Tang, Xiaowei Wu, and Yuhao Zhang. Towards a better understanding of randomized greedy matching. In *STOC*, pages 1097–1110. ACM, 2020.

29 Martin L Weitzman. Optimal search for the best alternative. *Econometrica: Journal of the Econometric Society*, pages 641–654, 1979.

A Subexponential Algorithm for ARRIVAL

Bernd Gärtner
Institute of Theoretical Computer Science,
Department of Computer Science, ETH Zürich, Switzerland

Sebastian Haslebacher
Department of Computer Science, ETH Zürich, Switzerland

Hung P. Hoang
Institute of Theoretical Computer Science,
Department of Computer Science, ETH Zürich, Switzerland

Abstract

The ARRIVAL problem is to decide the fate of a train moving along the edges of a directed graph, according to a simple (deterministic) pseudorandom walk. The problem is in NP ∩ coNP but not known to be in P. The currently best algorithms have runtime $2^{\Theta(n)}$ where n is the number of vertices. This is not much better than just performing the pseudorandom walk. We develop a subexponential algorithm with runtime $2^{O(\sqrt{n}\log n)}$. We also give a polynomial-time algorithm if the graph is almost acyclic. Both results are derived from a new general approach to solve ARRIVAL instances.

2012 ACM Subject Classification Theory of computation → Graph algorithms analysis

Keywords and phrases Pseudorandom walks, reachability, graph games, switching systems

Digital Object Identifier 10.4230/LIPIcs.ICALP.2021.69

Category Track A: Algorithms, Complexity and Games

Acknowledgements We thank Günter Rote for pointing out an error in an earlier version of the manuscript.

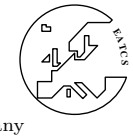
1 Introduction

Informally, the ARRIVAL problem is the following (we quote from Dohrau et al. [6]):

> Suppose that a train is running along a railway network, starting from a designated origin, with the goal of reaching a designated destination. The network, however, is of a special nature: every time the train traverses a switch, the switch will change its position immediately afterwards. Hence, the next time the train traverses the same switch, the other direction will be taken, so that directions alternate with each traversal of the switch.
>
> Given a network with origin and destination, what is the complexity of deciding whether the train, starting at the origin, will eventually reach the destination?

ARRIVAL is arguably the simplest problem in NP ∩ coNP that is not known to be in P. Due to its innocence and at the same time unresolved complexity status, ARRIVAL has attracted quite some attention recently. The train run can be interpreted as a deterministic simulation of a random walk that replaces random decisions at a switch by perfectly fair decisions. Such pseudorandom walks have been studied before under the names of *Eulerian walkers* [17], *rotor-router walks* [12], and *Propp machines* [4]. The reachability question as well as NP and coNP membership are due to Dohrau et al. [6].

Viewed somewhat differently, ARRIVAL is a *zero player game* (a process that runs without a controller); in contrast, three other well-known graph games in NP ∩ coNP that are not known to be in P are two-player (involving two controllers). These are *simple stochastic*



games, *mean-payoff games* and *parity games* [3, 21, 13]. Moreover, it is stated in (or easily seen from) these papers that the one-player variants (the strategy of one controller is fixed) have polynomial-time algorithms. In light of this, one might expect a zero-player game such as ARRIVAL to be *really* simple. But so far, no polynomial-time algorithm could be found.

On the positive side, the NP ∩ coNP complexity upper bound could be strengthened in various ways. ARRIVAL is in UP ∩ coUP, meaning that there are efficient verifiers that accept *unique* proofs [10]. A search version of ARRIVAL has been introduced by Karthik C. S. and shown to be in PLS [15], then in CLS [10], and finally in UniqueEOPL [10, 8]. The latter complexity class, established by Fearnley et al. [8], has an intriguing complete problem, but there is no evidence that ARRIVAL is complete for UniqueEOPL.

Concerning complexity lower bounds, there is one result: ARRIVAL is NL-hard [7]. This is not a very strong statement and means that every problem that can be solved by a nondeterministic log-space Turing machine reduces (in log-space) to ARRIVAL.

Much more interesting are the natural one- and two-player variants of ARRIVAL that have been introduced in the same paper by Fearnley et al. [7] and later expanded by Ani et al. [1]. These variants allow a better comparison with the previously mentioned graph games. It turns out that the one-player variants of ARRIVAL are NP-complete, and that the two-player variants are PSPACE-hard [7, 1]. This shows that the p-player variant of ARRIVAL is probably strictly harder than the p-player variants of the other graph games mentioned before, for $p = 1, 2$. This makes it a bit less surprising that ARRIVAL itself ($p = 0$) could so far not be shown to lie in P.

On the algorithmic side, the benchmark is the obvious algorithm for solving ARRIVAL on a graph with n vertices: simulate the train run. This is known to take at most $O(n2^n)$ steps (after this, we can conclude that the train runs forever) [6]. There is also an $\Omega(2^n)$ lower bound for the simulation [6]. The upper bound was improved to $O(p(n)2^{n/2})$ (in expectation) for some polynomial p, using a way to efficiently sample from the run [10]. The same bound was later achieved deterministically [11, 18], and the approach can be refined to yield a runtime of $O(p(n)2^{n/3})$, the currently best one for general ARRIVAL instances [18].

In this paper, we prove that ARRIVAL can be decided in subexponential time $2^{O(\sqrt{n}\log n)}$. While this is still far away from the desired polynomial-time algorithm, the new upper bound is making the first significant progress on the runtime. We also prove that polynomial runtime can be achieved if the graph is close to acyclic, meaning that it can be made acyclic by removing a constant number of vertices.

As the main technical tool from which we derive both results, we introduce a generalization of ARRIVAL. In this *multi-run* variant, there is a subset S of vertices where additional trains may start and also terminate. It turns out that if we start the right numbers of trains from the vertices in S, we also decide the original instance, so the problem is reduced to searching for these right numbers. We show that this search problem is well-behaved and can be solved by systematic guessing, where the number of guesses is exponential in $|S|$, not in n.

We are thus interested in cases where S is small but at the same time allows a sufficiently fast evaluation of a given guess. For the subexponential algorithm, we choose S as a set of size $O(\sqrt{n})$, with the property that a train can only take a subexponential number of steps until it terminates (in S or a destination). For almost acyclic graphs, we choose S as a minimum feedback vertex set, a set whose removal makes the graph acyclic. In this case, a train can visit any vertex only once before it terminates.

The multi-run variant itself is an interesting new approach to the ARRIVAL problem, and other applications of it might be found in the future.

2 ARRIVAL

The ARRIVAL problem was introduced by Dohrau et al. [6] as the problem of deciding whether the train arrives at a given destination or runs forever. Here, we work in a different but equivalent setting (implicitly established by Dohrau et al. already) in which the train always arrives at one of two destinations, and we have to decide at which one. The definitions and results from Dohrau et al. [6] easily adapt to our setting. We still provide independent proofs, derived from the more general setting that we introduce in Section 3.

Given a finite set of vertices V, an origin $o \in V$, two destinations $d, \overline{d} \notin V$ and two functions $s_{even}, s_{odd} : V \to V \cup \{d, \overline{d}\}$, the 6-tuple $A = (V, o, d, \overline{d}, s_{even}, s_{odd})$ is an *ARRIVAL instance*. The vertices $s_{even}(v)$ and $s_{odd}(v)$ are called the even and the odd successor of v.

An ARRIVAL instance A defines a directed graph, connecting each vertex $v \in V$ to its even and its odd successor. We call this the *switch graph* of A and denote it by $G(A)$. To avoid special treatment of the origin later, we introduce an artificial vertex $Y \notin V \cup \{d, \overline{d}\}$ (think of it as the "train yard") that only connects to the origin o. Formally, $G(A) = (V(A), E(A))$ where $V(A) = V \cup \{Y, d, \overline{d}\}$ and $E(A) = \{(Y, o)\} \cup \{(v, s_{even}(v)) : v \in V\} \cup \{(v, s_{odd}(v)) : v \in V\}$. We also refer to $E(A)$ simply as the edges of A. An edge $e \neq (Y, o)$ is called *proper*.

The *run procedure* is the following. For every vertex we maintain a current and a next successor, initially the even and the odd one. We put a token (usually referred to as the train) at o and move it along switch graph edges until it reaches either d or $\overline{d}$. Whenever the train is at a vertex v, we move it to v's current successor and then swap the current and the next successor; see Algorithm 1 for a formal description and Figure 1 for an example.

Algorithm 1 Run Procedure.

Input: ARRIVAL instance $A = (V, o, d, \overline{d}, s_{even}, s_{odd})$
Output: destination of the train: either d or $\overline{d}$
Let s_{curr} and s_{next} be arrays indexed by the vertices of V
for $v \in V$ **do**
  $s_{curr}[v] \leftarrow s_{even}(v)$
  $s_{next}[v] \leftarrow s_{odd}(v)$
$v \leftarrow o$ `/* traversal of edge (Y,o) */`
while $v \neq d$ *and* $v \neq \overline{d}$ **do**
  $w \leftarrow s_{curr}[v]$
  swap($s_{curr}[v]$, $s_{next}[v]$)
  $v \leftarrow w$ `/* traversal of edge (v,w) */`
return v

Algorithm 1 (Run procedure) may cycle, but we can avoid this by assuming that from every vertex $v \in V$, one of d and $\overline{d}$ is reachable along a directed path in $G(A)$. We call such an ARRIVAL instance *terminating*, since it guarantees that either d or $\overline{d}$ is eventually reached.

▶ **Lemma 1.** *Let $A = (V, o, d, \overline{d}, s_{even}, s_{odd})$ be a terminating ARRIVAL instance, $|V| = n$. Let $v \in V$ and suppose that the shortest path from v to a destination in $G(A)$ has length m. Then v is visited (the train is at v) at most 2^m times by Algorithm 1 (Run Procedure).*

Proof. Let $v = v_m, v_{m-1}, \ldots, v_0 \in \{d, \overline{d}\}$ be the sequence of vertices on a shortest path from v to $\{d, \overline{d}\}$. Consider the first 2^m visits to v (if there are less, we are done). Once every two consecutive visits, the train moves on to v_{m-1}, so we can consider the first 2^{m-1} visits to

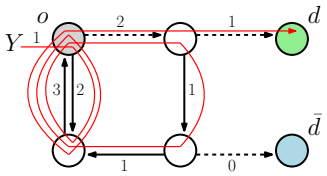


Figure 1 A terminating ARRIVAL instance and the train run. Bold edges go to the even successors, dashed edges to the odd successors. The two successors may coincide (lower left vertex). The numbers indicate how often each edge is traversed by the train.

v_{m-1} and repeat the argument from there to show that v_i is visited at least 2^i times for all i, before v exceeds 2^m visits. In particular, $v_0 \in \{d, \bar{d}\}$ is visited, so the run indeed terminates within at most 2^m visits to v. ◀

▶ **Lemma 2.** *Let $A = (V, o, d, \bar{d}, s_{even}, s_{odd})$ be a terminating ARRIVAL instance, $|V| = n$. Let ℓ be the maximum length of the shortest path from a vertex in V to a destination. Algorithm 1 (Run Procedure) traverses at most $(n - \ell + 2)2^\ell - 2$ proper edges.*

Proof. By Lemma 1, the total number of visits to vertices $v \in V$ is bounded by $\sum_{i=1}^{n} n_i 2^i$, where n_i is the number of vertices with a shortest path of length i to a destination. We have $n_i > 0$ if and only if $i \leq \ell$, and hence the sum is maximized if $n_i = 1$ for all $i < \ell$, and $n_\ell = n - \ell + 1$. In this case, the sum is $(n - \ell + 2)2^\ell - 2$. The number of proper edges being traversed (one after every visit of $v \in V$) is the same. ◀

Given a terminating instance, ARRIVAL is the problem of deciding whether Algorithm 1 (Run Procedure) returns d (YES instance) or $\bar{d}$ (NO instance). It is unknown whether ARRIVAL $\in$ P, but it is in NP $\cap$ coNP, due to the existence of *switching flows* that are certificates for the output of Algorithm 1 (Run Procedure).

2.1 Switching flows

For a vertex v and a set of edges E, we will denote the set of outgoing edges of v by $E^+(v)$. Analogously, we will denote the set of incoming edges of v by $E^-(v)$. Furthermore, for a function $x : E \to \mathbb{N}_0$, we will also use the notation x_e instead of $x(e)$ to denote the value of x at some edge $e \in E$. Lastly, given some vertex v, edges E and a function $x : E \to \mathbb{N}_0$, we will use $x^+(v) := \sum_{e \in E^+(v)} x_e$ to denote the *outflow* of x at v and $x^-(v) := \sum_{e \in E^-(v)} x_e$ to denote the *inflow* of x at v. For two functions $x, x' : E \to \mathbb{N}_0$, we write $x \leq x'$ if this holds componentwise, i.e. $x_e \leq x'_e$ for all $e \in E$.

▶ **Definition 3** (Switching Flow [6]). *Let $A = (V, o, d, \bar{d}, s_{even}, s_{odd})$ be a terminating ARRIVAL instance with edges E. A function $x : E \to \mathbb{N}_0$ is a* switching flow *for A if*

$$\begin{aligned} x^+(Y) &= 1, && \\ x^+(v) - x^-(v) &= 0, && v \in V \quad \textit{(flow conservation)} \\ x_{(v,s_{even}(v))} - x_{(v,s_{odd}(v))} &\in \{0,1\}, && v \in V \quad \textit{(switching behavior).} \end{aligned}$$

Moreover, x is called a switching flow to $t \in \{d, \bar{d}\}$ *if $x^-(t) = 1$.*

Note that due to flow conservation, a switching flow is a switching flow either to d or to $\bar{d}$: exactly one of the destinations must absorb the unit of flow emitted by Y. If we set x_e to the number of times the edge e is traversed in Algorithm 1 (Run Procedure), we obtain

a switching flow to the output; see Figure 1 for an example. Indeed, every time the train enters $v \in V$, it also leaves it; this yields flow conservation. The strict alternation between the successors (beginning with the even one) yields switching behavior.

Hence, the existence of a switching flow to the output is necessary for obtaining the output. Interestingly, it is also sufficient. For that, it remains to prove that we cannot have switching flows to both d and $\overline{d}$ for the same instance.

▶ **Theorem 4** (Switching flows are certificates [6]). *Let $A = (V, o, d, \overline{d}, s_{even}, s_{odd})$ be a terminating ARRIVAL instance, $t \in \{d, \overline{d}\}$. Algorithm 1 (Run Procedure) outputs t if and only if there exists a switching flow to t.*

The switching flow corresponding to the actual train run can be characterized as follows.

▶ **Theorem 5** (The run profile is the minimal switching flow [6]). *Let $A = (V, o, d, \overline{d}, s_{even}, s_{odd})$ be a terminating ARRIVAL instance with edges E. Let $\hat{x}$ be the* run profile *of A, meaning that $\hat{x}_e$ counts the number of times edge e is traversed during Algorithm 1 (Run Procedure). Then $\hat{x} \leq x$ for all switching flows x. In particular, $\hat{x}$ is the unique minimizer of the total flow $\sum_{e \in E} x_e$ over all switching flows.*

We note that this provides the missing direction of Theorem 4. Indeed, $\hat{x}$ is a switching flow and hence either has $\hat{x}^-(d) = 1$ or $\hat{x}^-(\overline{d}) = 1$. By $\hat{x} \leq x$, every switching flow x is to the same destination. In general, there can be switching flows $x \neq \hat{x}$ [6].

We will derive Theorem 5 as a special case of Theorem 8 in the next section.

3 A general framework

In order to solve the ARRIVAL problem, we can simulate Algorithm 1 (Run Procedure) which takes exponential time in the worst case [6]; alternatively, we can try to get hold of a switching flow; via Theorem 4, this also allows us to decide ARRIVAL.

According to Definition 3, a switching flow can be obtained by finding a feasible solution to an integer linear program (ILP); this is a hard task in general, and it is unknown whether switching flow ILPs can be solved more efficiently than general ILPs.

In this section, we develop a framework that allows us to reduce the problem to that of solving a number of more constrained ILPs. At the same time, we provide direct methods for solving them that do not rely on using general purpose ILP solvers.

3.1 The idea

Given a terminating ARRIVAL instance, we consider the switching flow conditions in Definition 3. Given an arbitrary fixed subset $S = \{v_1, \ldots, v_k\} \subseteq V$ of k vertices, we drop the flow conservation constraints at the vertices in S, but at the same time prescribe outflow values $x^+(v_1), \ldots, x^+(v_k)$ that we can think of as guesses for their values in a switching flow.

If we minimize the total flow subject to these guesses, we obtain a unique solution (Theorem 8 (i) below) and hence unique inflow values $x^-(v_1), \ldots, x^-(v_k)$ for the vertices in S. If we happen to stumble upon a fixed point of the mapping $x^+(v_1), \ldots, x^+(v_k) \to x^-(v_1), \ldots, x^-(v_k)$, we recover flow conservation also at S, which means that our guesses were correct and we have obtained a switching flow.

The crucial property is that the previously described mapping is *monotone* (Theorem 8 (ii) below), meaning that the theory of *Tarski fixed points* applies that guarantees the existence of a fixed point as well as efficient algorithms for finding it (Lemma 11 below).

Hence, we reduce the computation of a switching flow to a benign search problem (for a Tarski fixed point), where every search step requires us to solve a "guessing" ILP. We next present a "rail" way of solving the guessing ILP that turns out to be more efficient in the worst case (and also simpler) than general purpose ILP solvers. For suitable switch graphs and appropriate choices of the set S, it will be fast enough to yield the desired runtime results.

3.2 The Multi-Run Procedure

Given $S = \{v_1, \ldots, v_k\} \subseteq V$ and $w \in \mathbb{N}_0^k$ (guesses for the outflows from the vertices in S), we start one train from Y and w_i trains from v_i until they arrive back in S, or at a destination. In this way, we produce inflow values for the vertices in S.

By starting, we mean that we move each of the trains by one step: the one on Y moves to o, while $\lceil w_i/2 \rceil$ of the ones at v_i move to the even successor of v_i, and $\lfloor w_i/2 \rfloor$ to the odd successor. Trains that are now on vertices in $V \setminus S$ are called *waiting* (to move on).

For all $v \in V \setminus S$, we initialize current and next successors as before in Algorithm 1 (Run Procedure). Then we (nondeterministically) repeat the following until there are no more trains waiting.

We pick a vertex $v \in V \setminus S$ where some trains are waiting and call the number of waiting trains $t(v)$. We choose a number $\tau \in \{1, \ldots, t(v)\}$ of trains to move on; we move $\lceil \tau/2 \rceil$ of them to the current successor and $\lfloor \tau/2 \rfloor$ to the next successor. If τ is odd, we afterwards swap the current and the next successor at v.

Algorithm 2 (Multi-Run Procedure) provides the details. For $S = \emptyset$, the procedure becomes deterministic and is equivalent to Algorithm 1 (Run Procedure).

Algorithm 2 Multi-Run Procedure.

Input: Terminating ARRIVAL instance $A = (V, o, d, \overline{d}, s_{even}, s_{odd})$ with edges E; $S = \{v_1, v_2, \ldots, v_k\} \subseteq V$, $w = (w_1, w_2, \ldots, w_k) \in \mathbb{N}_0^k$ (one train starts from Y, and w_i trains start from v_i).
Output: number of trains arriving at $d, \overline{d}$, and in S, respectively
Let t be a zero-initialized array indexed by the vertices of $V \cup \{d, \overline{d}\}$
$t[o] \leftarrow 1$ `/* traversal of` (Y, o) `*/`
for $i = 1, 2, \ldots, k$ **do**
  $t[s_{even}(v_i)] \leftarrow t[s_{even}(v_i)] + \lceil w_i/2 \rceil$ `/*` $\lceil w_i/2 \rceil$ `traversals of` $(v_i, s_{even}(v_i))$ `*/`
  $t[s_{odd}(v_i)] \leftarrow t[s_{odd}(v_i)] + \lfloor w_i/2 \rfloor$ `/*` $\lfloor w_i/2 \rfloor$ `traversals of` $(v_i, s_{odd}(v_i))$ `*/`
Let s_{curr} and s_{next} be arrays indexed by the vertices of $V \setminus S$
for $v \in V \setminus S$ **do**
  $s_{curr}[v] \leftarrow s_{even}(v)$
  $s_{next}[v] \leftarrow s_{odd}(v)$
while $\exists v \in V \setminus S : t[v] > 0$ **do**
  pick $v \in V \setminus S$ such that $t[v] > 0$ and choose $\tau \in \{1, \ldots, t[v]\}$
  $t[v] \leftarrow t[v] - \tau$
  $t[s_{curr}(v)] \leftarrow t[s_{curr}(v)] + \lceil \tau/2 \rceil$ `/*` $\lceil \tau/2 \rceil$ `traversals of` $(v, s_{curr}(v))$ `*/`
  $t[s_{next}(v)] \leftarrow t[s_{next}(v)] + \lfloor \tau/2 \rfloor$ `/*` $\lfloor \tau/2 \rfloor$ `traversals of` $(v, s_{next}(v))$ `*/`
  if τ *is odd* **then**
    swap$(s_{curr}[v], s_{next}[v])$
return $(t[d], t[\overline{d}], t[v_1], t[v_2], \ldots, t[v_k])$

▶ **Lemma 6.** *Algorithm 2 (Multi-Run Procedure) terminates.*

Proof. This is a qualitative version of the argument in Lemma 1. Let $x : E \to \mathbb{N}_0$ record how many times each edge $e \in E$ has been traversed in total, at any given time of Algorithm 2 (Multi-Run Procedure). For $v \in V \setminus S$, we always have $x^+(v) = x^-(v) - t(v)$, where $t(v)$ is the number of trains currently waiting at v. Suppose for a contradiction that the Multi-Run procedure cycles. Then $x^-(v)$ is unbounded for at least one $v \in V \setminus S$, which means that $x^+(v)$ is also unbounded, since $t(v)$ is bounded. This in turn means that $x^-(s_{even}(v))$ and $x^-(s_{odd}(v))$ are unbounded as well, since we distribute $x^+(v)$ evenly between the two successors. Repeating this argument, we see that $x^-(w)$ is unbounded for all vertices w reachable from v. But as $x^-(d)$ and $x^-(\overline{d})$ are bounded (by the number of trains that we started), neither d nor $\overline{d}$ are reachable from v. This is a contradiction to A being terminating. ◀

3.3 Candidate switching flows

After Algorithm 2 (Multi-Run Procedure) has terminated, let $\hat{x}_e$ be the number of times the edge e was traversed. We then have flow conservation at $v \in V \setminus S$, switching behavior at $v \in V$ and outflow w_i from v_i. Indeed, every train that enters $v \in V \setminus S$ eventually also leaves it; moreover, the procedure is designed such that it simulates moving trains out of $v \in V$ individually, strictly alternating between successors. Finally, as we start w_i trains from $v_i \in S$ and stop all trains once they arrive in S, we also have outflow w_i from v_i.

We remark that we do not have any control over how many trains end up at d or $\overline{d}$. Also, $\hat{x}$ could in principle depend on the order in which we pick vertices, and on the chosen τ's. We will show in Theorem 8 below that it does not. So far, we have only argued that $\hat{x}$ is a *candidate switching flow* according to the following definition.

▶ **Definition 7** (Candidate Switching Flow). *Let $A = (V, o, d, \overline{d}, s_{even}, s_{odd})$ be a terminating ARRIVAL instance with edges E, $S = \{v_1, v_2, \ldots, v_k\} \subseteq V$, $w = (w_1, w_2, \ldots, w_k) \in \mathbb{N}_0^k$.*

A function $x : E \to \mathbb{N}_0$ is a candidate switching flow *for A (w.r.t. S and w) if*

$$\begin{array}{rcll} x^+(Y) & = & 1, & \\ x^+(v) - x^-(v) & = & 0, & v \in V \setminus S \quad \textit{(flow conservation at } V \setminus S\textit{)} \\ x^+(v_i) & = & w_i, & i = 1, 2, \ldots, k, \quad \textit{(outflow } w \textit{ at } S\textit{)} \\ x_{(v,s_{even}(v))} - x_{(v,s_{odd}(v))} & \in & \{0,1\}, & v \in V \quad \textit{(switching behavior).} \end{array} \tag{1}$$

▶ **Theorem 8** (Each Multi-Run profile is the minimal candidate switching flow). *Let A, E, S, w be as in Definition 7 and let $\hat{x}$ be a* Multi-Run profile *of A, meaning that $\hat{x}_e$ is the number of times edge $e \in E$ was traversed during some run of Algorithm 2 (Multi-Run Procedure). Then the following statements hold.*

- **(i)** *$\hat{x} \leq x$ for all candidate switching flows x (w.r.t. S and w). In particular, $\hat{x}$ is the unique minimizer of the total flow $\sum_{e \in E} x_e$ over all candidate switching flows.*
- **(ii)** *For fixed A, E, S, define $F(w) = (\hat{x}^-(v_1), \ldots, \hat{x}^-(v_k)) \in \mathbb{N}_0^k$. Then the function $F : \mathbb{N}_0^k \to \mathbb{N}_0^k$ is* monotone*, meaning that $w \leq w'$ implies that $F(w) \leq F(w')$.*

Proof. We prove part (i) by the *pebble argument* [6]: Let x be any candidate switching flow w.r.t. S and w. For every edge e, we initially put x_e pebbles on e, and whenever a train traverses e in Algorithm 2 (Multi-Run Procedure), we let it collect a pebble. If we can show that we never run out of pebbles, $\hat{x} \leq x$ follows. By "running out of pebbles", we concretely mean that we are for the first time trying to collect a pebble from an edge with no pebbles left.

Since x is a candidate switching flow, we cannot run out of pebbles while starting the trains. In fact, we exactly collect all the pebbles on the outgoing edges of $\{Y\} \cup S$. It remains to show that we cannot run out of pebbles while processing a picked vertex $v \in V \setminus S$. For this, we prove that we maintain the following additional invariants (which hold immediately after starting the trains). Let $p : E \to \mathbb{N}_0$ record for each edge e the remaining number of pebbles on e. Then for all $v \in V \setminus S$,

(a) $p^+(v) = p^-(v) + t(v)$, where $t(v)$ is the number of trains waiting at v;

(b) $p((v, s_{curr}(v))) - p((v, s_{next}(v))) \in \{0, 1\}$.

Suppose that these invariants hold when picking a vertex $v \in V \setminus S$. As we have not run out of pebbles before, $p^-(v) \geq 0$ and (a) guarantees that we have $q \geq t(v)$ pebbles on the outgoing edges; by (b), $\lceil q/2 \rceil$ of them are on $(v, s_{curr}(v))$ and $\lfloor q/2 \rfloor$ on $(v, s_{next}(v))$. From the former, we collect $\lceil \tau/2 \rceil$, and from the latter $\lfloor \tau/2 \rfloor$ where $\tau \leq t(v) \leq q$, so we do not run out of pebbles. We maintain (a) at v where both p^+ and t are reduced by τ. We also maintain (a) at the successors; there, the gain in t exactly compensates the loss in p^-. Finally, we maintain (b) at v: If τ is even, both $p((v, s_{curr}(v)))$ and $p((v, s_{next}(v)))$ shrink by $\tau/2$. If τ is odd, we have $p((v, s_{curr}(v))) - p((v, s_{next}(v))) \in \{-1, 0\}$ after collecting one more pebble from $(v, s_{curr}(v))$ than from $(v, s_{next}(v))$, but then we reverse the sign by swapping s_{curr} and s_{next}.

For $S = \emptyset$, this proves Theorem 5, and for general S, we have now proved (i). In particular, the order in which we move trains in Algorithm 2 (Multi-Run Procedure) does not matter.

The proof of (ii) is now an easy consequence; recall that the inflow $F(w)_i$ is the number of trains that arrive at v_i. If $w \leq w'$, we run Algorithm 2 (Multi Run Procedure) with input w' such that it first simulates a run with input w; for this, we keep the extra trains corresponding to $w' - w$ waiting where they are after the start, until all other trains have terminated. At this point, we have inflow $f \geq F(w)$ at S, where $f - F(w)$ corresponds to the extra trains that have already reached S right after the start. We finally run the extra trains that are still waiting, and as this can only further increase the inflows at S, we get $F(w') \geq f \geq F(w)$. ◀

We remark that since the total inflow to the vertices in $\{d, \overline{d}\} \cup S$ equals the total outflow from Y and vertices in S, Theorem 8 (i) implies that the inflows to the vertices in $\{d, \overline{d}\} \cup S$ are the same in every candidate switching flow (w.r.t. S and w). This means that in Theorem 8 (ii), the Multi-Run profile $\hat{x}$ can be replaced by an arbitrary candidate switching flow. This could in principle be easier to compute than the Multi-Run profile, but we currently do not know how.

3.4 Runtime

As we have proved in Theorem 8 (i), the Multi-Run procedure always generates the unique flow-minimal candidate switching flow. But the number of steps depends on the order in which vertices $v \in V \setminus S$ are picked, and on the chosen τ's. We start with an upper bound on the number of edge traversals that generalizes Lemma 2.

▶ **Lemma 9.** *Let $A = (V, o, d, \overline{d}, s_{even}, s_{odd})$ be a terminating ARRIVAL instance, $|V| = n$, $S = \{v_1, v_2, \ldots, v_k\} \subseteq V$, $w = (w_1, w_2, \ldots, w_k) \in \mathbb{N}_0^k$. Let ℓ be the maximum length of the shortest path from a vertex in $V \setminus S$ to a vertex in $\{d, \overline{d}\} \cup S$. Further suppose that at the beginning of some iteration in Algorithm 2 (Multi-Run Procedure), R trains are still waiting. Then all subsequent iterations traverse at most $R((n - \ell + 2)2^\ell - 2)$ edges in total.*

Proof. We continue to run each of the R waiting trains individually and proceed with the next one only when the previous one has terminated. In Algorithm 2 (Multi-Run Procedure), this corresponds to always using $\tau = 1$ and the next vertex v as the head of the previously traversed edge, for each of the R trains. So we effectively perform Algorithm 1 (Run Procedure) for R trains.

As each train terminates once it reaches a vertex in $S \cup \{d, \overline{d}\}$, Lemmata 1 and 2 are easily seen to hold also here, after redefining "destination" as any vertex in $S \cup \{d, \overline{d}\}$. As a consequence, each train traverses at most $(n - \ell + 2)2^\ell - 2$ edges until it reaches a vertex in $\{d, \overline{d}\} \cup S$. This leads to at most $R((n-\ell+2)2^\ell - 2)$ edge traversals overall. By Theorem 8 (i), this upper bound holds for all ways of continuing Algorithm 2 (Multi-Run Procedure). ◀

With $R = W := 1 + \sum_{i=1}^{k} w_i$, we obtain an upper bound for the total number of loop iterations since each iteration traverses at least one edge. But it turns out that we can be significantly faster (and polynomial in the encoding size of W) when we proceed in a greedy fashion, i.e. we always pick the next vertex as the one with the largest number of waiting trains, and move all these trains at once.

▶ **Lemma 10.** *Let A, n, S, w, ℓ as in Lemma 9, and suppose that in each iteration of Algorithm 2 (Multi-Run Procedure), we pick $v \in V \setminus S$ maximizing $t[v]$ and further choose $\tau = t[v]$. Then the number of iterations is at most $(\ln W + n)(n - k)((n - \ell + 2)2^\ell - 2)$, where $W = 1 + \sum_{i=1}^{k} w_i$.*

Proof. As in the proof of Theorem 8, we let each train collect a pebble as it traverses an edge, where we initially put $\hat{x}_e$ pebbles on edge e, with $\hat{x}$ being the unique Multi-Run profile. This means that we eventually collect all pebbles. Now consider an iteration and suppose that $R \leq W$ trains are still waiting. In the greedy algorithm, we move at least $R/(n - k)$ of them in this iteration and collect at least that many pebbles. On the other hand, with R trains still waiting, and with $T = (n - \ell + 2)2^\ell - 2$, there can be no more than RT pebbles left, as all of them will be collected in the remaining at most that many edge traversals, due to Lemma 9.

In summary, the number of pebbles is guaranteed to be reduced by a factor of

$$\left(1 - \frac{1}{(n-k)T}\right)$$

in each iteration, starting from at most WT pebbles before the first iteration. After $s = (\ln W + n)(n - k)T$ iterations, we therefore have at most

$$\left(1 - \frac{1}{(n-k)T}\right)^s WT \leq e^{-\ln W - n} WT < 1$$

pebbles left (using $T < e^n$). Hence, after at most s iterations, the greedy version of Algorithm 2 (Multi-Run Procedure) has indeed terminated. ◀

We remark that essentially the same runtime can be achieved by a round robin version that repeatedly cycles through $V \setminus S$ in some fixed order.

3.5 Tarski fixed points

Tarski fixed points arise in the study of order-preserving functions on complete lattices [19]. For our application, it suffices to consider finite sets of the form $L = \{0, 1, \ldots, N\}^k$ for some $N, k \in \mathbb{N}^+$. For such a set, Tarski's fixed point theorem [19] states that any monotone

function $D : L \to L$ has a fixed point, some $\hat{w} \in L$ such that $D(\hat{w}) = \hat{w}$. Moreover, the problem of finding such a fixed point has been studied: Dang, Qi and Ye [5] have shown that a fixed point can be found using $O(\log^k N)$ evaluations of D. Recently, Fearnley, Pálvölgyi and Savani [9] improved this to $O(\log^{2\lceil k/3 \rceil} N)$.

Via Theorem 8, we have reduced the problem of deciding a terminating ARRIVAL instance to the problem of finding a fixed point of a monotone function $F : \mathbb{N}_0^k \to \mathbb{N}_0^k$, assuming that we can efficiently evaluate F. Indeed, if we have such a fixed point, the corresponding (flow-minimal) candidate switching flow is an *actual* switching flow and hence decides the problem via Theorem 4.

The function F depends on a set $S \subseteq V$ of size k that we can choose freely (we will do so in the subsequent sections).

Here, we still need to argue that we can restrict F to a finite set $L = \{0, 1, \ldots, N\}^k$ so that the Tarski fixed point theorem applies. We already know that outflow (and hence inflow) values never exceed $N = 2^n$ in *some* switching flow, namely the run profile (Lemma 1), so we simply restrict F to this range and at the same time cap the function values accordingly.

▶ **Lemma 11.** *Let $A = (V, o, d, \overline{d}, s_{even}, s_{odd})$ be a terminating ARRIVAL instance, $S = \{v_1, \ldots, v_k\} \subseteq V$, $|V| = n$. Let F be the function defined in Theorem 8 (ii), let $N = 2^n$ and consider the function $D : \{0, 1, \ldots, N\}^k \to \{0, 1, \ldots, N\}^k$ defined by*

$$D(w) = \begin{pmatrix} \min(N, F(w)_1) \\ \min(N, F(w)_2) \\ \vdots \\ \min(N, F(w)_k) \end{pmatrix}, \quad w \in \{0, 1, \ldots, N\}^k.$$

Then D is monotone and has a fixed point $\hat{w}$ that can be found with $O(\log^{2\lceil k/3 \rceil} N)$ evaluations of D. Moreover, $\hat{w}$ is also a fixed point of F, and when we apply Theorem 8 (i) with $w = \hat{w}$, the flow-minimal candidate switching flow resulting from the multi-run procedure is a switching flow for A.

We remark that the switching flow obtained in this way is not necessarily flow-minimal, so we cannot argue that we obtain the run profile of A as defined in Theorem 5. The function D may have several fixed points, each of them leading to a different switching flow; to obtain the run profile, we would have to find a particular fixed point, the one that leads to the unique switching flow of smallest total flow. The known Tarski fixed point algorithms cannot do this, and we do not know of any efficient method for computing the run profile from a given switching flow.

Proof. Monotonicity is clear: if $w \leq w'$, then $F(w) \leq F(w')$ by monotonicity of F; see Theorem 8 (ii). But then also $D(w) \leq D(w')$ for the capped values. Hence, the Tarski fixed point theorem [19] yields a fixed point $\hat{w}$ of D, and the algorithm of Fearnley, Pálvölgyi and Savani [9] finds it using $O(\log^{2\lceil k/3 \rceil} N) = O(n^{2\lceil k/3 \rceil})$ evaluations.

It remains to prove that $\hat{w}$ is a fixed point of F. Suppose for a contradiction that it is not a fixed point. Then $F(\hat{w}) \neq D(\hat{w})$, i.e. some values were actually capped, and so $\hat{w}_j = D(\hat{w})_j = N < F(\hat{w})_j$ for at least one j. As we also have $\hat{w} = D(\hat{w}) \leq F(\hat{w})$, we get

$$\sum_{i=1}^{k} \hat{w}_i < \sum_{i=1}^{k} F(\hat{w})_i. \tag{2}$$

On the other hand, consider the candidate switching flow (1) with $w = \hat{w}$. At most the total flow emitted (at Y and the v_i's) is absorbed at S, so we have

$$\sum_{i=1}^{k} F(\hat{w})_i \leq 1 + \sum_{i=1}^{k} \hat{w}_i. \tag{3}$$

Putting this together with (2), we get an equality in (3). In particular, v_j is the only vertex whose inflow value was capped (by one), all emitted flow is absorbed at S, and no flow arrives at d or $\overline{d}$.

But this is a contradiction to $\hat{w}_j = N = 2^n$: By the same arguments as in the proof of Lemma 1, based on flow conservation (at all $v \neq v_j$) and switching behavior, one of these 2^n outflow units is guaranteed to arrive at $\{d, \overline{d}\}$. ◀

4 Subexponential algorithm for ARRIVAL

In this section, we present our main application of the general framework developed in the previous section.

Given a terminating ARRIVAL instance A with $|V| = n$, the plan is to construct a set $S \subseteq V$ of size $O(\sqrt{n})$ such that from any vertex, the length of the shortest path in $G(A)$ to a vertex in $S \cup \{d, \overline{d}\}$ is also bounded by roughly $O(\sqrt{n})$. Since S is that small, we can find a Tarski fixed point with a subexponential number of F-evaluations; and since shortest paths are that short, each F-evaluation can also be done in subexponential time using the Multi-Run procedure. An overall subexponential algorithm ensues.

▶ **Lemma 12.** *Let $A = (V, o, d, \overline{d}, s_{even}, s_{odd})$ be a terminating ARRIVAL instance with $|V| = n$. Let $\phi \in (0, 1)$ be a real number. In $O(n)$ time, we can construct a ϕ-*set *S, meaning a set $S \subseteq V$ such that*

- **(i)** *$|S| \leq \phi \cdot (n + 2)$;*
- **(ii)** *for all $v \in V$, the shortest path from v to $S \cup \{d, \overline{d}\}$ in $G(A)$ has length at most $\log_2(n + 2)/\phi$.*

Proof. We adapt the ball-growing technique of Leighton and Rao [16], as explained by Trevisan [20].

We first decompose the switch graph $G(A)$ into layers based on the distance of the vertices to a destination [11]. More formally, for $v \in V \cup \{d, \overline{d}\}$, we denote by $\mathrm{dist}(v)$ the length of the shortest path from v to $\{d, \overline{d}\}$ in $G(A)$. Then the layers are defined as $L_i := \{v \in V \cup \{d, \overline{d}\} : \mathrm{dist}(v) = i\}$ for $i \geq 0$. Define $\ell := \max\{\mathrm{dist}(v) : v \in V\}$. We can compute the layer decomposition $(L_0, \ldots, L_\ell)$ using breadth-first search in $O(n)$ time.

Consider Algorithm 3 which computes a ϕ-set as a union of layers. It is clear that the procedure is done in $O(n)$ time. To prove (i), we observe that whenever we add a layer L_i to S, we have $|L_i| < \phi|U|$; moreover, the U's considered in these inequalities are mutually disjoint subsets of $V \cup \{d, \overline{d}\}$. Hence, $|S| < \phi \cdot (n + 2)$.

For (ii), let $v \in V$. Then $v \in L_b$ for some $b \geq 1$. Let $0 \leq a \leq b$ be the largest index such that $L_a \subseteq S \cup \{d, \overline{d}\}$. Then the shortest path from v to a vertex in $S \cup \{d, \overline{d}\}$ has length at most $b - a$. It remains to bound $j := b - a$. The interesting case is $j > 0$.

Consider Algorithm 3. After the a-th iteration, we have $|U| = |L_a| \geq 1$. Moreover, $|L_i| \geq \phi|U|$ for $i = a + 1, \ldots, b$, meaning that for each iteration i in this range, the size of U has grown by a factor of at least $1 + \phi$. Hence, after the b-th iteration, $(1 + \phi)^j \leq |U| \leq n + 2$. This implies $j \leq \log_2(n + 2)/\log_2(1 + \phi) < \log_2(n + 2)/\phi$, where we use the inequality $\log_2(1 + \phi) > \phi$ for $\phi \in (0, 1)$. ◀

Algorithm 3 Procedure to compute a ϕ-set.

Input: ARRIVAL instance with layer decomposition $(L_0, \ldots, L_\ell)$, $\phi \in (0, 1)$
Output: a ϕ-set S
$S \leftarrow \emptyset$
$U \leftarrow L_0$
for $i = 1, \ldots, \ell$ **do**
  if $|L_i| < \phi|U|$ **then**
    $S \leftarrow S \cup L_i$
    $U \leftarrow \emptyset$
  $U \leftarrow U \cup L_i$
return S

▶ **Theorem 13.** *Let $A = (V, o, d, \overline{d}, s_{even}, s_{odd})$ be a terminating ARRIVAL instance with $|V| = n$. A can be decided in time $O(p(n)n^{1.633\sqrt{n}})$, for some polynomial p.*

Proof. By Lemma 12, we can find a ϕ-set S in $O(n)$ time, for any $\phi \in (0, 1)$. As $|S| \leq \phi \cdot (n+2)$, by Lemma 11, we can then decide A with $O(n^{2\lceil \phi \cdot (n+2)/3 \rceil})$ evaluations of the function D. Each evaluation in turn requires us to evaluate the function F in Theorem 8 (ii) for a given $w \in \{0, 1, \ldots, 2^n\}^{|S|}$. We can do this by applying Algorithm 2 (Multi-Run Procedure). By Lemma 10 and the definition of a ϕ-set in Lemma 12, running this algorithm in a greedy fashion requires at most $(\ln W + n)(n - |S|)((n - \ell + 2)2^\ell - 2)$ iterations, where $W = 1 + \sum_{i=1}^{|S|} w_i$ and $\ell = \log_2(n+2)/\phi$. Further, from the choice of w, we have $W \leq 2^n \phi(n+2) + 1$. Therefore, the number of iterations is $O(q(n)n^{1/\phi})$ for some polynomial q. At each iteration, we need to find the vertex with the highest number of waiting trains, as stated in Lemma 10, and move the trains from the chosen vertex. All these operations take polynomial time.

In total, the runtime of the whole process is $O(n^{2\lceil \phi \cdot (n+2)/3 \rceil} \cdot p(n)n^{1/\phi})$ for some polynomial p. Choosing $\phi = \sqrt{3}/\sqrt{2n}$, the runtime becomes $O(p(n)n^{1.633\sqrt{n}})$. ◀

5 Feedback vertex sets

In the previous section, we used our framework to obtain an improved algorithm for ARRIVAL in general. In this section, we will instantiate the framework differently to obtain a polynomial-time algorithm for a certain subclass of ARRIVAL.

A subset $S \subseteq V$ of vertices in a directed graph $G = (V, E)$ is called a *feedback vertex set* if and only if the subgraph induced by $V \setminus S$ is acyclic (i.e. it contains no directed cycle). Karp [14] showed that the problem of finding a smallest feedback vertex set is NP-hard. However, there exists a parameterized algorithm by Chen et al. [2] which can find a feedback vertex set of size k in time $O(n^4 4^k k^3 k!)$ in a directed graph on n vertices, or report that no such set exists.

If we apply Theorem 8 with a feedback vertex set S, it turns out that we can compute the Multi-Run profile in polynomial time, meaning that we get a polynomial-time algorithm for ARRIVAL if there is a feedback vertex set of constant size k.

▶ **Theorem 14.** *Let $A = (V, o, d, \overline{d}, s_{even}, s_{odd})$ be a terminating ARRIVAL instance with graph $G(A)$. If $G(A)$ has a feedback vertex set $S \subseteq V$ of size k (assumed to be fixed as $n = |V| \to \infty$), then A can be decided in time $O(n^{2(\lceil k/3 \rceil + 1)})$.*

Proof. Using the algorithm by Chen et al. [2], we can find a feedback vertex set S in $O(n^4)$ time if it exists. According to Lemma 11, we can then decide A with $O(n^{2\lceil k/3 \rceil})$ evaluations of the function D. Each evaluation in turn requires us to evaluate the function F in Theorem 8 (ii) for a given $w \in \{0, 1, \dots, 2^n\}^k$. To do this, we apply Algorithm 2 (Multi-Run Procedure) where we pick vertices $v \in V \setminus S$ in topological order and choose $\tau = t[v]$ always. As we never send any trains back to vertices that have previously been picked, we terminate within $n - k$ iterations, each of which can be performed in time $O(n)$ as it involves $O(n)$-bit numbers. Hence, $F(w)$ can be computed in $O(n^2)$ time.

Overall, this gives a runtime of $O(n^4 + n^{2(\lceil k/3 \rceil + 1)})$. For $k \geq 1$, the second term dominates the first one. For $k = 0$, we can check if a graph is acyclic, for example, via topological sorting in $O(n)$ time and do not need to employ the algorithm by Chen et al. [2]. The claimed runtime follows. ◀

We remark that even if k is not constant, we can still beat the subexponential algorithm in Section 4, as long as $k = O(n^\alpha)$ for some $\alpha < 1/2$.

References

1. Joshua Ani, Erik D. Demaine, Dylan H. Hendrickson, and Jayson Lynch. Trains, games, and complexity: 0/1/2-player motion planning through input/output gadgets, 2020. arXiv:2005.03192.
2. Jianer Chen, Yang Liu, Songjian Lu, Barry O'Sullivan, and Igor Razgon. A fixed-parameter algorithm for the directed feedback vertex set problem. *J. ACM*, 55(5):Art. 21, 19, 2008.
3. Anne Condon. The complexity of stochastic games. *Information and Computation*, 96(2):203–224, 1992.
4. Joshua Cooper, Benjamin Doerr, Joel Spencer, and Gábor Tardos. Deterministic random walks on the integers. *European Journal of Combinatorics*, 28(8):2072–2090, 2007. EuroComb '05 - Combinatorics, Graph Theory and Applications.
5. Chuangyin Dang, Qi Qi, and Yinyu Ye. Computations and complexities of Tarski's fixed points and supermodular games, 2020. arXiv:2005.09836.
6. Jérôme Dohrau, Bernd Gärtner, Manuel Kohler, Jiří Matoušek, and Emo Welzl. ARRIVAL: a zero-player graph game in $NP \cap coNP$. In *A journey through discrete mathematics*, pages 367–374. Springer, Cham, 2017.
7. John Fearnley, Martin Gairing, Matthias Mnich, and Rahul Savani. Reachability switching games. In *45th International Colloquium on Automata, Languages, and Programming*, volume 107 of *LIPIcs. Leibniz Int. Proc. Inform.*, pages Art. No. 124, 14. Schloss Dagstuhl. Leibniz-Zent. Inform., Wadern, 2018.
8. John Fearnley, Spencer Gordon, Ruta Mehta, and Rahul Savani. Unique end of potential line. *J. Comput. System Sci.*, 114:1–35, 2020.
9. John Fearnley, Dömötör Pálvölgyi, and Rahul Savani. A faster algorithm for finding Tarski fixed points, 2020. arXiv:2010.02618.
10. Bernd Gärtner, Thomas Dueholm Hansen, Pavel Hubáček, Karel Král, Hagar Mosaad, and Veronika Slívová. ARRIVAL: next stop in CLS. In *45th International Colloquium on Automata, Languages, and Programming*, volume 107 of *LIPIcs. Leibniz Int. Proc. Inform.*, pages Art. No. 60, 13. Schloss Dagstuhl. Leibniz-Zent. Inform., Wadern, 2018.
11. Bernd Gärtner and Hung P. Hoang. ARRIVAL with two vertices per layer. Manuscript in preparation, 2021.
12. Alexander E. Holroyd and James Propp. Rotor walks and Markov chains. In *Algorithmic probability and combinatorics*, volume 520 of *Contemp. Math.*, pages 105–126. Amer. Math. Soc., Providence, RI, 2010.
13. Marcin Jurdziński. Deciding the winner in parity games is in UP ∩ co-UP. *Information Processing Letters*, 68(3):119–124, 1998.

14 Richard M. Karp. Reducibility among combinatorial problems. In *Complexity of computer computations (Proc. Sympos., IBM Thomas J. Watson Res. Center, Yorktown Heights, N.Y., 1972)*, pages 85–103, 1972.

15 C. S. Karthik. Did the train reach its destination: the complexity of finding a witness. *Inform. Process. Lett.*, 121:17–21, 2017.

16 Tom Leighton and Satish Rao. Multicommodity max-flow min-cut theorems and their use in designing approximation algorithms. *J. ACM*, 46(6):787–832, 1999.

17 Viacheslav B. Priezzhev, Deepak Dhar, Abhishek Dhar, and Supriya Krishnamurthy. Eulerian walkers as a model of self-organized criticality. *Phys. Rev. Lett.*, 77:5079–5082, 1996.

18 Günter Rote. Personal communication, 2020.

19 Alfred Tarski. A lattice-theoretical fixpoint theorem and its applications. *Pacific J. Math.*, 5:285–309, 1955.

20 Luca Trevisan. Approximation algorithms for unique games. In *46th Annual IEEE Symposium on Foundations of Computer Science (FOCS'05)*, pages 197–205, 2005.

21 Uri Zwick and Mike Paterson. The complexity of mean payoff games on graphs. *Theoretical Computer Science*, 158:343–359, 1996.

Universal Algorithms for Clustering Problems

Arun Ganesh
Department of Computer Science, University of California at Berkeley, CA, USA

Bruce M. Maggs
Department of Computer Science, Duke University, Durham, NC, USA
Emerald Innovations, Cambridge, MA, USA

Debmalya Panigrahi
Department of Computer Science, Duke University, Durham, NC, USA

Abstract

This paper presents *universal* algorithms for clustering problems, including the widely studied k-median, k-means, and k-center objectives. The input is a metric space containing all *potential* client locations. The algorithm must select k cluster centers such that they are a good solution for *any* subset of clients that actually realize. Specifically, we aim for low *regret*, defined as the maximum over all subsets of the difference between the cost of the algorithm's solution and that of an optimal solution. A universal algorithm's solution SOL for a clustering problem is said to be an (α, β)-approximation if for all subsets of clients C', it satisfies $\text{SOL}(C') \leq \alpha \cdot \text{OPT}(C') + \beta \cdot \text{MR}$, where $\text{OPT}(C')$ is the cost of the optimal solution for clients C' and MR is the minimum regret achievable by any solution.

Our main results are universal algorithms for the standard clustering objectives of k-median, k-means, and k-center that achieve $(O(1), O(1))$-approximations. These results are obtained via a novel framework for universal algorithms using linear programming (LP) relaxations. These results generalize to other ℓ_p-objectives and the setting where some subset of the clients are *fixed*. We also give hardness results showing that (α, β)-approximation is NP-hard if α or β is at most a certain constant, even for the widely studied special case of Euclidean metric spaces. This shows that in some sense, $(O(1), O(1))$-approximation is the strongest type of guarantee obtainable for universal clustering.

2012 ACM Subject Classification Theory of computation → Facility location and clustering

Keywords and phrases universal algorithms, clustering, k-median, k-means, k-center

Digital Object Identifier 10.4230/LIPIcs.ICALP.2021.70

Category Track A: Algorithms, Complexity and Games

Related Version *Full Version*: https://arxiv.org/abs/2105.02363

Funding *Arun Ganesh*: Supported in part by NSF Award CCF-1535989.
Bruce M. Maggs: Supported in part by NSF grant CCF-1535972.
Debmalya Panigrahi: Supported in part by NSF grants CCF-1535972, CCF-1955703, an NSF Career Award CCF-1750140, and the Indo-US Virtual Networked Joint Center on Algorithms Under Uncertainty.

1 Introduction

In *universal*[1] approximation (e.g., [8, 9, 10, 16, 20, 22, 27, 39, 40]), the algorithm is presented with a set of *potential* input points and must produce a solution. After seeing the solution, an adversary selects some subset of the points as the actual *realization* of the input, and the

[1] In the context of clustering, universal facility location sometimes refers to facility location where facility costs scale with the number of clients assigned to them. This problem is unrelated to the notion of universal algorithms studied in this paper.

cost of the solution is based on this realization. The goal of a universal algorithm is to obtain a solution that is near-optimal for *every* possible input realization. For example, suppose that a network-based-service provider can afford to deploy servers at k locations around the world and hopes to minimize latency between clients and servers. The service provider does not know in advance which clients will request service, but knows where clients are located. A universal solution provides guarantees on the quality of the solution regardless of which clients ultimately request service. As another example, suppose that a program committee chair wishes to invite k people to serve on the committee. The chair knows the areas of expertise of each person who is qualified to serve. Based on past iterations of the conference, the chair also knows about many possible topics that might be addressed by submissions. The chair could use a universal algorithm to select a committee that will cover the topics well, regardless of the topics of the papers that are submitted. The situation also arises in targeting advertising campaigns to client demographics. Suppose a campaign can spend for k advertisements, each targeted to a specific client type. While the entire set of client types that are potentially interested in a new product is known, the exact subset of clients that will watch the ads, or eventually purchase the product, is unknown to the advertiser. How does the advertiser target her k advertisements to address the interests of any realized subset of clients?

Motivated by these sorts of applications, this paper presents the first universal algorithms for clustering problems, including the classic k-median, k-means, and k-center problems. The input to these algorithms is a metric space containing all locations of *clients* and *cluster centers*. The algorithm must select k cluster centers such that this is a good solution for *any* subset of clients that actually realize.

It is tempting to imagine that, in general, for some large enough value of α, one can find a solution SOL such that for all realizations (i.e., subsets of clients) C', $\text{SOL}(C') \leq \alpha \cdot \text{OPT}(C')$, where $\text{SOL}(C')$ denotes SOL's cost in realization C' and $\text{OPT}(C')$ denotes the optimal cost in realization C'. But this turns out to be impossible for many problems, including the clustering problems we study, and indeed this difficulty may have limited the study of universal algorithms. For example, suppose that the input for the k-median problem is a uniform metric on $k+1$ points, each with a cluster center and client. In this case, for any solution SOL with k cluster centers, there is some realization C' consisting of a single client that is not co-located with any of the k cluster centers in SOL. Then, $\text{SOL}(C') > 0$ but $\text{OPT}(C') = 0$. Since it is not possible to provide a strict approximation guarantee for every realization, we instead seek to minimize the *regret*, defined as the maximum difference between the cost of the algorithm's solution and the optimal cost across all realizations. The solution that minimizes regret is called the *minimum regret solution*, or MRS for short, and its regret is termed *minimum regret* or MR. More formally, $\text{MR} = \min_{\text{SOL}} \max_{C'} [\text{SOL}(C') - \text{OPT}(C')]$. We now seek a solution SOL that achieves, for all input realizations C', $\text{SOL}(C') - \text{OPT}(C') \leq \text{MR}$, i.e., $\text{SOL}(C') \leq \text{OPT}(C') + \text{MR}$. But, obtaining such a solution turns out to be **NP**-hard for many problems, and one has to settle for an approximation: $\text{SOL}(C') \leq \alpha \cdot \text{OPT}(C') + \beta \cdot \text{MR}$. The algorithm is then called an (α, β)-approximate universal algorithm for the problem. Note that in the aforementioned example with $k+1$ points, any solution must pay MR (the distance between any two points) in some realization where $\text{OPT}(C') = 0$ and only one client appears (in which case paying MR might sound avoidable or undesirable). This example demonstrates that stricter notions of regret and approximation than (α, β)-approximation are infeasible in general, suggesting that (α, β)-approximation is the least relaxed guarantee possible for universal clustering.

1.1 Problem Definitions and Results

We are now ready to formally define our problems and state our results. In all the clustering problems that we consider in this paper, the input is a metric space on all the potential client locations C and cluster centers F. The special case where $F = C$ has also been studied in the clustering literature, e.g., in [23, 14], although the more common setting, as in our work, is to not make this assumption. Of course, all results, including ours, without this assumption also apply to the special case. If $F = C$, the constants in our bounds improve, but the results are qualitatively the same. We note that some sources refer to the k-center problem when $F \neq C$ as the k-supplier problem instead, and use k-center to refer exclusively to the case where $F = C$.

Let c_{ij} denote the metric distance between points i and j. The solution produced by the algorithm comprises k cluster centers in F; let us denote this set by SOL. Now, suppose a subset of clients $C' \subseteq C$ realizes in the actual input. Then, the cost of each client $j \in C'$ is given as the distance from the client to its closest cluster center, i.e., $\text{COST}(j, \text{SOL}) = \min_{i \in \text{SOL}} c_{ij}$. The clustering problems differ in how these costs are combined into the overall minimization objective. The respective objectives are given below:

- ***k*-median** (e.g., [14, 25, 5, 34, 11]): $\text{SOL}(C') = \sum_{j \in C'} \text{COST}(j, \text{SOL})$.
- ***k*-center** (e.g., [23, 15, 24, 30, 37]): $\text{SOL}(C') = \max_{j \in C'} \text{COST}(j, \text{SOL})$.
- ***k*-means** (e.g., [35, 28, 33, 21, 1]): $\text{SOL}(C') = \sqrt{\sum_{j \in C'} \text{COST}(j, \text{SOL})^2}$.

We also consider **ℓ_p-clustering** (e.g., [21]) which generalizes all these individual clustering objectives. In ℓ_p-clustering, the objective is the ℓ_p-norm of the client costs for a given value $p \geq 1$, i.e., $\text{SOL}(C') = \left(\sum_{j \in C'} \text{COST}(j, \text{SOL})^p\right)^{1/p}$. Note that k-median and k-means are special cases of ℓ_p-clustering for $p = 1$ and $p = 2$ respectively. k-center can also be defined in the ℓ_p-clustering framework as the limit of the objective for $p \to \infty$; moreover, it is well-known that ℓ_p-norms only differ by constants for $p > \log n$, thereby allowing the k-center objective to be approximated within a constant by ℓ_p-clustering for $p = \log n$.

Our main result is to obtain $(O(1), O(1))$-approximate universal algorithms for k-median, k-center, and k-means. We also generalize these results to the ℓ_p-clustering problem.

▶ **Theorem 1.** *There are $(O(1), O(1))$-approximate universal algorithms for the k-median, k-means, and k-center problems. More generally, there are $(O(p), O(p^2))$-approximate universal algorithms for ℓ_p-clustering problems, for any $p \geq 1$.*

▶ Remark. The bound for k-means is by setting $p = 2$ in ℓ_p-clustering. For k-median and k-center, we use separate algorithms to obtain improved bounds than those provided by the ℓ_p-clustering result. This is particularly noteworthy for k-center where ℓ_p-clustering only gives poly-logarithmic approximation.

Universal Clustering with Fixed Clients. We also consider a more general setting where some of the clients are *fixed*, i.e., are there in any realization, but the remaining clients may or may not realize as in the previous case. (Of course, if no client is fixed, we get back the previous setting as a special case.) This more general model is inspired by settings where a set of clients is already present but the remaining clients are mere predictions. This surprisingly creates new technical challenges, that we overcome to get:

▶ **Theorem 2.** *There are $(O(1), O(1))$-approximate universal algorithms for the k-median, k-means, and k-center problems with fixed clients. More generally, there are $(O(p^2), O(p^2))$-approximate universal algorithms for ℓ_p-clustering problems, for any $p \geq 1$.*

Hardness Results. Next, we study the limits of approximation for universal clustering. In particular, we show that the universal clustering problems for all the objectives considered in this paper are **NP**-hard in a rather strong sense. Specifically, we show that both α and β are separately bounded away from 1, *irrespective of the value of the other parameter*, showing the necessity of both α and β in our approximation bounds. Similar lower bounds continue to hold for universal clustering in Euclidean metrics, even when PTASes are known in the offline (non-universal) setting [4, 31, 33, 37, 1].

▶ **Theorem 3.** *In universal ℓ_p-clustering for any $p \geq 1$, obtaining $\alpha < 3$ or $\beta < 2$ is* **NP***-hard. Even for Euclidean metrics, obtaining $\alpha < 1.8$ or $\beta \leq 1$ is* **NP***-hard. The lower bounds on α (resp., β) are independent of the value of β (resp., α).*

Interestingly, our lower bounds rely on realizations where sometimes as few as one client appears. This suggests that e.g. redefining regret to be some function of the number of clients that appear (rather than just their cost) cannot subvert these lower bounds.

1.2 Techniques

Before discussing our techniques, we discuss why standard approximations for clustering problems are insufficient. It is known that the *optimal* solution for the realization that includes all clients gives a $(1, 2)$-approximation for universal k-median (this is a corollary of a more general result in [29]; we do not know if their analysis can be extended to e.g. k-means), giving universal algorithms for "easy" cases of k-median such as tree metrics. But, the clustering problems we consider in this paper are **NP**-hard in general; so, the best we can hope for in polynomial time is to obtain optimal *fractional* solutions, or *approximate* integer solutions. Unfortunately, the proof of [29] does not generalize to *any* regret guarantee for the optimal *fractional* solution. Furthermore, for all problems considered in this paper, even $(1 + \epsilon)$-approximate (integer) solutions for the "all clients" instance are not guaranteed to be (α, β)-approximations for any finite α, β. These observations fundamentally distinguish universal approximations for **NP**-hard problems like the clustering problems in this paper from those in **P**, and require us to develop new techniques for universal approximations.

In this paper, we develop a general framework for universal approximation based on linear programming (LP) relaxations that forms the basis of our results on k-median, k-means, and k-center (Theorem 1) as well as the extension to universal clustering with fixed clients (Theorem 2).

The first step in our framework is to write an LP relaxation of the regret minimization problem. In this formulation, we introduce a new regret variable that we seek to minimize and is constrained to be at least the difference between the (fractional) solution obtained by the LP and the optimal integer solution *for every realizable instance*. Abstractly, if the LP relaxation of the optimization problem is given by $\min\{\mathbf{c} \cdot \mathbf{x} : \mathbf{x} \in P\}$, then the new *regret minimization* LP is given by

$$\min\{\mathbf{r} : \mathbf{x} \in P; \mathbf{c}(I) \cdot \mathbf{x} \leq \text{OPT}(I) + \mathbf{r}, \ \forall I\}.$$

(For problems like k-means with non-linear objectives, the constraint $\mathbf{c}(I) \cdot \mathbf{x} \leq \text{OPT}(I) + \mathbf{r}$ cannot be replaced with a constraint that is simultaneously linear in $\mathbf{x}, \mathbf{r}$. However, for a fixed value of $\mathbf{r}$, the corresponding non-linear constraints still give a convex feasible region, and so the techniques we discuss in this section can still be used.)

Here, I ranges over all realizable instances of the problem. Hence, the LP is exponential in size, and we need to invoke the ellipsoid method via a separation oracle to obtain an optimal fractional solution. It suffices to design a separation oracle for the new set of constraints

$\mathbf{c}(I)\cdot\mathbf{x} \leq \text{OPT}(I) + \mathbf{r}$, $\forall I$. This amounts to determining the regret of a fixed solution given by $\mathbf{x}$, which unfortunately, is **NP**-hard for our clustering problems. So, we settle for designing an approximate separation oracle, i.e., approximating the regret of a given solution. For k-median, we reduce this to a submodular maximization problem subject to a cardinality constraint, which can then be (approximately) solved via standard greedy algorithms. For k-means, and more generally ℓ_p-clustering, as well as the setting with fixed clients, the situation is more complex, but can still be reduced to submodular maximization.

The next step in our framework is to round these fractional solutions to integer solutions for the regret minimization LP. Typically, in clustering problems such as k-median, LP rounding algorithms give *average* guarantees, i.e., although the overall objective in the integer solution is bounded against that of the fractional solution, individual connection costs of clients are not (deterministically) preserved in the rounding. But, average guarantees are too weak for our purpose: in a realized instance, an adversary may only select the clients whose connection costs increase by a large factor in the rounding thereby causing a large regret. Ideally, we would like to ensure that the connection cost of *every* individual client is preserved up to a constant in the rounding. However, this may be impossible in general. Consider a uniform metric over $k+1$ points. One fractional solution is to make $\frac{k}{k+1}$ fraction of each point a cluster center. In any integer solution, since there are only k cluster centers but $k+1$ points overall, there is one client that has connection cost of 1, which is $k+1$ times its fractional connection cost.

To overcome this difficulty, we allow for a uniform *additive* increase in the connection cost of every client. We show that such a rounding also preserves the regret guarantee of our fractional solution within constant factors. The clustering problem we now solve has a modified objective: for every client, the distance to the closest cluster center is now discounted by the additive allowance, with the caveat that the connection cost is 0 if this difference is negative. This variant is a generalization of a problem appearing in [19], and we call it clustering *with discounts* (e.g., for k-median, we call this problem *k-median with discounts*.) Our main tool in the rounding then becomes an approximation algorithm for ℓ_p^p-clustering with discounts. For k-median, we use a Lagrangian relaxation of this problem to the classic facility location problem to design such an approximation. For k-means and ℓ_p-clustering, extra work is needed to relate the ℓ_p and ℓ_p^p objectives. For k-center, we give a purely combinatorial (greedy) algorithm.

1.3 Related Work

For all previous universal algorithms, the approximation factor corresponds to our parameter α, i.e., these algorithms are $(\alpha, 0)$-approximate. The notion of regret was not considered. As we have explained, however, it is not possible to obtain such results for universal clustering. Furthermore, it may be possible to trade-off some of the large values of α in these results, e.g., $\Omega(\sqrt{n})$ for set cover, by allowing $\beta > 0$.

Universal algorithms have been of large interest in part because of their applications as online algorithms where all the computation is performed ahead of time. Much of the work on universal algorithms has focused on TSP, starting with the seminal work of Jia et al. [26] (later improved by [20]), with following work giving better approximations for Euclidean metrics [39], minor-free metrics [22], and tree metrics [40]. The universal metric Steiner tree problem was also considered by Jia et al. [26], with nearly matching lower bounds [2, 26, 9]. The problem has also been considered for general graphs and minor-free graphs [10]. Finally, for universal (weighted) set cover, Jia et al. [26] (see also [17]) provide an algorithm and an almost matching lower bound.

The problem of minimizing regret has been studied in the context of robust optimization, with a focus on tree metrics. The robust 1-median problem was introduced for tree metrics by Kouvelis and Yu in [32] and several faster algorithms and for general metrics were developed in the following years (e.g. see [7]). For robust k-center, Averbakh and Berman[7] gave a reduction to ordinary k-center problems, which are tractable on tree metrics.

Roadmap. We present the constant approximation algorithms (Theorem 1) for universal k-median, a sketch for k-means, and k-center in Sections 2, 4, and 5 respectively. The k-means result is given in full detail as a more general ℓ_p-clustering result in the full paper. In describing these algorithms, we defer the clustering with discounts algorithms used in the rounding to the appendix. We also give the extension to universal clustering with fixed clients for k-median in Section 3, with the extensions for k-means and k-center in the full paper. Finally, the hardness results (Theorem 3) appear in Section 6.

2 Universal k-Median

In this section, we prove the following theorem:

▶ **Theorem 4.** *There exists a $(27, 49)$-approximate universal algorithm for the k-median problem.*

The algorithm has two components. The first component is a separation oracle for the regret minimization LP based on submodular maximization, which we define below.

Submodular Maximization with Cardinality Constraints. A (non-negative) function $f : 2^E \to \mathbb{R}_0^+$ is said to be *submodular* if for all $S \subseteq T \subseteq E$ and $x \in E$, we have $f(T \cup \{x\}) - f(T) \leq f(S \cup \{x\}) - f(S)$. It is said to be *monotone* if for all $S \subseteq T \subseteq E$, we have $f(T) \geq f(S)$. The following theorem for maximizing monotone submodular functions subject to a cardinality constraint is well-known.

▶ **Theorem 5** (Fisher et al. [38])**.** *For the problem of finding $S \subseteq E$ that maximizes a monotone submodular function $f : 2^E \to \mathbb{R}_0^+$, the natural greedy algorithm that starts with $S = \emptyset$ and repeatedly adds $x \in E$ that maximizes $f(S \cup \{x\})$ until $|S| = k$, is a $\frac{e}{e-1} \approx 1.58$-approximation.*

We give the reduction from the separation oracle to submodular maximization in Section 2.1, and then employ the above theorem.

***k*-median with Discounts.** The second component of our framework is a rounding algorithm that employs the k-median with discounts problem, which we define next. In the k-median with discounts problem, we are given a k-median instance, but where each client j has an additional (non-negative) parameter r_j called its *discount*. Just as in the k-median problem, our goal is to place k cluster centers that minimize the total connection costs of all clients. But, the connection cost for client j can now be discounted by up to r_j, i.e., client j with connection cost c_j contributes $(c_j - r_j)^+ := \max\{0, c_j - r_j\}$ to the objective of the solution.

Let OPT be the cost of an optimal solution to the k-median with discounts problem. We say an algorithm ALG that outputs a solution with connection cost c_j for client j is a (γ, σ)-approximation if:

$$\sum_{j \in C} (c_j - \gamma \cdot r_j)^+ \leq \sigma \cdot \text{OPT}.$$

That is, a (γ, σ)-approximate algorithm outputs a solution whose objective function when computed using discounts $\gamma \cdot r_j$ for all j is at most σ times the optimal objective using discounts r_j. In the case where all r_j are equal, [19] gave a $(9, 6)$-approximation algorithm for this problem based on the classic primal-dual algorithm for k-median. The following lemma generalizes their result to the setting where the r_j may differ:

▶ **Lemma 6.** *There exists a (deterministic) polynomial-time* $(9, 6)$*-approximation algorithm for the* k*-median with discounts problem.*

We give details of the algorithm and the proof of this lemma in the full paper. We note that when all r_j are equal, the constants in [19] can be improved (see e.g. [13]); we do not know of any similar improvement when the r_j may differ. In Section 2.2, we give the reduction from rounding the fractional solution for universal k-median to the k-median with discounts problem, and then employ the above lemma.

2.1 Universal k-median: Fractional Algorithm

The standard k-median polytope (see e.g., [25]) is given by:

$$P = \{(x, y) : \sum_i x_i \leq k; \forall i, j : y_{ij} \leq x_i; \forall j : \sum_i y_{ij} \geq 1; \forall i, j : x_i, y_{ij} \in [0, 1]\}.$$

Here, x_i represents whether point i is chosen as a cluster center, and y_{ij} represents whether client j connects to i as its cluster center. Now, consider the following LP formulation for minimizing regret r:

$$\min\{r : (x, y) \in P; \forall C' \subseteq C : \sum_{j \in C'} \sum_i c_{ij} y_{ij} - \text{OPT}(C') \leq r\}, \tag{1}$$

where $\text{OPT}(C')$ is the cost of the (integral) optimal solution in realization C'. Note that the new constraints: $\forall C' \subseteq C : \sum_{j \in C'} \sum_i c_{ij} y_{ij} - \text{OPT}(C') \leq r$ (we call it the regret constraint set) require that the regret is at most r in all realizations.

In order to solve LP (1), we need a separation oracle for the regret constraint set. Note that there are exponentially many constraints corresponding to realizations C'; moreover, even for a single realization C', computing $\text{OPT}(C')$ is **NP**-hard. So, we resort to designing an *approximate* separation oracle. Fix some fractional solution (x, y, r). Overloading notation, let $S(C')$ denote the cost of the solution with cluster centers S in realization C'. By definition, $\text{OPT}(C') = \min_{S \subseteq F, |S| = k} S(C')$. Then designing a separation oracle for the regret constraint set is equivalent to determining if the following inequality holds:

$$\max_{C' \subseteq C} \quad \max_{S \subseteq F, |S| = k} \left[\sum_{j \in C'} \sum_i c_{ij} y_{ij} - S(C') \right] \leq r.$$

We flip the order of the two maximizations, and define $f_y(S)$ as follows:

$$f_y(S) = \max_{C' \subseteq C} \left[\sum_{j \in C'} \sum_i c_{ij} y_{ij} - S(C') \right].$$

Then designing a separation oracle is equivalent to maximizing $f_y(S)$ for $S \subseteq F$ subject to $|S| = k$. The rest of the proof consists of showing that this function is monotone and submodular, and efficiently computable.

▶ **Lemma 7.** *Fix y. Then, $f_y(S)$ is a monotone submodular function in S. Moreover, $f_y(S)$ is efficiently computable for a fixed S.*

Proof. Let $d(j,S) := \min_{i' \in S} c_{i'j}$ denote the distance from client j to the nearest cluster center in S. If $S = \emptyset$, we say $d(j,S) := \infty$. The value of C' that defines $f_y(S)$ is the set of all clients closer to S than to the fractional solution y, i.e., $\sum_i c_{ij} y_{ij} > \min_{i' \in S} c_{i'j}$. This immediately establishes efficient computability of $f_y(S)$. Moreover, we can equivalently write $f_y(S)$ as follows:

$$f_y(S) = \sum_{j \in C} (\sum_i c_{ij} y_{ij} - d(j,S))^+.$$

A sum of monotone submodular functions is a monotone submodular function, so it suffices to show that for all clients j, the new function $g_{y,j}(S) := (\sum_i c_{ij} y_{ij} - d(j,S))^+$ is monotone submodular.

- $g_{y,j}$ is *monotone*: for $S \subseteq T$, $d(j,T) \leq d(j,S)$, and thus $(\sum_i c_{ij} y_{ij} - d(j,S))^+ \leq (\sum_i c_{ij} y_{ij} - d(j,T))^+$.
- $g_{y,j}$ is submodular if:

  $$\forall S \subseteq T \subseteq F, \forall x \in F : g_{y,j}(S \cup \{x\}) - g_{y,j}(S) \geq g_{y,j}(T \cup \{x\}) - g_{y,j}(T)$$

  Fix S, T, and x. Assume $g_{y,j}(T \cup \{x\}) - g_{y,j}(T)$ is positive (if it is zero, by monotonicity the above inequality trivially holds). This implies that x is closer to client j than any cluster center in T (and hence S too), i.e., $d(j,x) \leq d(j,T) \leq d(j,S)$. Thus, $d(j,x) = d(j, S \cup \{x\}) = d(j, T \cup \{x\})$ which implies that $g_{y,j}(S \cup \{x\}) = g_{y,j}(T \cup \{x\})$. Then we just need to show that $g_{y,j}(S) \leq g_{y,j}(T)$, but this holds by monotonicity. ◀

By standard results (see e.g., GLS [18]), we get an (α, β)-approximate fractional solution for universal k-median via the ellipsoid method if we have an approximate separation oracle for LP (1) that given a fractional solution (x,y,r) does either of the following:

- Declares (x,y,r) feasible, in which case (x,y) has cost at most $\alpha \cdot \text{OPT}(\mathbf{I}) + \beta \cdot r$ in all realizations, or
- Outputs an inequality violated by (x,y,r) in LP (1).

The approximate separation oracle does the following for the regret constraint set (all other constraints can be checked exactly): Given a solution (x,y,r), find an $\frac{e-1}{e}$-approximate maximizer S of f_y via Lemma 7 and Theorem 5. Let C' be the set of clients closer to S than the fractional solution y (i.e., the realization that maximizes $f_y(S)$). If $f_y(S) > r$, the separation oracle returns the violated inequality $\sum_{j \in C'} \sum_i c_{ij} y_{ij} - S(C') \leq r$; else, it declares the solution feasible. Whenever the actual regret of (x,y) is at least $\frac{e}{e-1} \cdot r$, this oracle will find S such that $f_y(S) > r$ and output a violated inequality. Hence, we get the following lemma:

▶ **Lemma 8.** *There exists a deterministic algorithm that in polynomial time computes a fractional $\frac{e}{e-1} \approx 1.58$-approximate solution for* LP (1) *representing the universal k-median problem.*

2.2 Universal k-Median: Rounding Algorithm

Let FRAC denote the $\frac{e}{e-1}$-approximate fractional solution to the universal k-median problem provided by Lemma 8. We will use the following property of k-median, shown by Archer et al. [3].

▶ **Lemma 9** ([3]). *The integrality gap of the natural* LP *relaxation of the k-median problem is at most* 3.

Lemmas 8 and 9 imply that that for any set of clients C',

$$\frac{1}{3} \cdot \text{OPT}(C') \leq \text{FRAC}(C') \leq \text{OPT}(C') + \frac{e}{e-1} \cdot \text{MR}. \tag{2}$$

Our overall goal is to obtain a solution SOL that minimizes $\max_{C' \subseteq C} [\text{SOL}(C') - \text{OPT}(C')]$. But, instead of optimizing over the exponentially many different $\text{OPT}(C')$ solutions, we use the surrogate $3 \cdot \text{FRAC}(C')$ which has the advantage of being defined by a fixed solution FRAC, but still approximates $\text{OPT}(C')$ by Eq. 2. This suggests minimizing the following objective instead: $\max_{C'} [\text{SOL}(C') - 3 \cdot \text{FRAC}(C')]$. Minimizing this objective is equivalent to the k-median with discounts problem, where the discount for client j is $3f_j$. This allows us to invoke Lemma 6 for the k-median with discounts problem.

Thus, our overall algorithm is as follows. First, use Lemma 8 to find a fractional solution $\text{FRAC} = (x, y, r)$. Let $f_j := \sum_i c_{ij} y_{ij}$ be the connection cost of client j in FRAC. Then, construct a k-median with discounts instance where client j has discount $3f_j$, and use Lemma 6 on this instance to obtain the final solution to the universal k-median problem. Theorem 4 follows using the above lemmas; we defer the proof to the full paper.

3 Universal k-Median with Fixed Clients

In this section, we extend the techniques from Section 2 to prove the following theorem:

▶ **Theorem 10.** *If there exists a deterministic polynomial time γ-approximation algorithm for the k-median problem, then for every $\epsilon > 0$ there exists a $(54\gamma + \epsilon, 60)$-approximate universal algorithm for the universal k-median problem with fixed clients.*

By using the derandomized version of the $(2.732 + \epsilon)$-approximation algorithm of Li and Svensson [34] for the k-median problem, and appropriate choice of both ϵ parameters, we obtain the following corollary from Theorem 10.

▶ **Corollary 11.** *For every $\epsilon > 0$, there exists a $(148 + \epsilon, 60)$-approximate universal algorithm for the k-median problem with fixed clients.*

Our high level strategy follows similarly to the previous section. In Section 3.2, we show how to find a good fractional solution by approximately solving a linear program. In Section 3.3, we describe how to round the fractional solution in a manner that preserves its regret guarantee within constant factors. Similar techniques in conjunction with the techniques in Sections 4 and 5 are used for the universal k-means and k-center problems with fixed clients; due to space constraints, we only focus on universal k-median with fixed clients here.

3.1 Preliminaries

In addition to the preliminaries of Section 2, we will use the following tools:

Submodular Maximization over Independence Systems. An *independence system* comprises a ground set E and a set of subsets (called *independent sets*) $\mathcal{I} \subseteq 2^E$ with the property that if $A \subseteq B$ and $B \in \mathcal{I}$ then $A \in \mathcal{I}$ (the *subset closed* property). An independent set S in $\mathcal{I}$ is *maximal* if there does not exist $S' \supset S$ such that $S' \in \mathcal{I}$. Note that one can define an

independence system by specifying the set of maximal independent sets $\mathcal{I}'$ only, since the subset closed property implies $\mathcal{I}$ is simply all subsets of sets in $\mathcal{I}'$. An independence system is a 1-*independence system* (or 1-*system* in short) if all maximal independent sets are of the same size. The following result on maximizing submodular functions over 1-independence systems follows from a more general result given implicitly in [38] and more formally in [12].

▶ **Theorem 12.** *There exists a polynomial time algorithm that given a* 1*-independence system* $(E, \mathcal{I})$ *and a non-negative monotone submodular function* $f : 2^E \to \mathbb{R}^+$ *defined over it, finds a* $\frac{1}{2}$*-maximizer of* f*, i.e. finds* $S' \in \mathcal{I}$ *such that* $f(S') \geq \frac{1}{2} \max_{S \in \mathcal{I}} f(S)$*.*

The algorithm in the above theorem is the natural greedy algorithm, which starts with $S' = \emptyset$ and repeatedly adds to S' the element u that maximizes $f(S' \cup \{u\})$ while maintaining that $S' \cup \{u\}$ is in $\mathcal{I}$, until no such addition is possible.

Incremental ℓ_p-Clustering. We will also use the *incremental ℓ_p-clustering* problem which is defined as follows: Given an ℓ_p-clustering instance and a subset of the cluster centers S (the "existing" cluster centers), find the minimum cost solution to the ℓ_p-clustering instance with the additional constraint that the solution must contain all cluster centers in S. When $S = \emptyset$, this is just the standard ℓ_p-clustering problem, and this problem is equivalent to the standard ℓ_p-clustering problem by the following lemma:

▶ **Lemma 13.** *If there exists a γ-approximation algorithm for the ℓ_p-clustering problem, there exists a γ-approximation for the incremental ℓ_p-clustering problem.*

The lemma follows by an approximation-preserving reduction between the two problems, which simply adds many clients to the locations of cluster centers in S, forcing any low-cost solution to place cluster centers at these locations even in the standard ℓ_p-clustering problem.

3.2 Obtaining a Fractional Solution for Universal k-Median with Fixed Clients

Let $C_f \subseteq C$ denote the set of fixed clients and for any realization of clients C' satisfying $C_f \subseteq C' \subseteq C$, let $\textsc{opt}(C')$ denote the cost of the optimal solution for C'. The same LP we used for universal k-median applies here, except we remove constraints on regret corresponding to realizations $C' \not\subseteq C_f$. Recall that to design an approximate separation oracle, it suffices to find a realization approximately maximizing the regret of the fractional solution.

Let $S(C')$ denote the cost of the solution $S \subseteq F$ in realization C' (that is, $S(C') = \sum_{j \in C'} \min_{i \in S} c_{ij}$). Since $\textsc{opt}(C') = \min_{S : S \subseteq F, |S| = k} S(C')$, exactly deciding the feasibility of the constraints on regret in the LP is equivalent to deciding if the following holds:

$$\forall S : S \subseteq F, |S| = k : \quad \max_{C' : C_f \subseteq C' \subseteq C} \left[\sum_{j \in C'} \sum_{i \in F} c_{ij} y_{ij} - S(C') \right] \leq r. \tag{3}$$

By splitting the terms $\sum_{j \in C'} \sum_{i \in F} c_{ij} y_{ij}$ and $S(C')$ into terms for C_f and $C' \setminus C_f$, we can rewrite Eq. (3) as follows:

$$\forall S \subseteq F, |S| = k : \max_{C^* \subseteq C \setminus C_f} \left[\sum_{j \in C^*} \sum_{i \in F} c_{ij} y_{ij} - S(C^*) \right] \leq S(C_f) - \sum_{j \in C_f} \sum_{i \in F} c_{ij} y_{ij} + r$$

For fractional solution y, let

$$f_y(S) = \max_{C^* : C^* \subseteq C \setminus C_f} \left[\sum_{j \in C^*} \sum_{i \in F} c_{ij} y_{ij} - S(C^*) \right]. \tag{4}$$

Note that we can compute $f_y(S)$ for any S easily since the maximizing value of C^* is the set of clients j for which S has connection cost less than $\sum_{i \in F} c_{ij} y_{ij}$. We already know $f_y(S)$ is submodular. But, the term $S(C_f)$ is not fixed with respect to S, so maximizing $f_y(S)$ does not suffice for separating the LP. To overcome this difficulty, for every possible cost M on the fixed clients, we replace $S(C_f)$ with M and only maximize over solutions S for which $S(C_f) \leq M$ (for convenience, we will call any solution S for which $S(C_f) \leq M$ an *M-cheap* solution):

$$\forall M \in \left\{ 0, 1, \ldots, |C_f| \max_{i,j} c_{ij} \right\} : \max_{S : S \subseteq F, |S| = k, S(C_f) \leq M} f_y(S) \leq M - \sum_{j \in C_f} \sum_{i \in F} c_{ij} y_{ij} + r. \tag{5}$$

Note that this set of inequalities is equivalent to Eq. (3), but it has the advantage that the left-hand side is approximately maximizable and the right-hand side is fixed. Hence, these inequalities can be approximately separated. However, there are exponentially many inequalities; so, for any fixed $\epsilon > 0$, letting $Z_\epsilon := \left\{ 0, 1, 1+\epsilon, \ldots, (1+\epsilon)^{\lceil \log_{1+\epsilon}(|C_f| \max_{i,j} c_{ij}) \rceil + 1} \right\}$ we relax to the following polynomially large set of inequalities:

$$\forall M \in Z_\epsilon : \max_{S : S \subseteq F, |S| = k, S(C_f) \leq M} f_y(S) \leq M - \sum_{j \in C_f} \sum_{i \in F} c_{ij} y_{ij} + r. \tag{6}$$

Separating inequality Eq. (6) for a fixed M corresponds to submodular maximization of $f_y(S)$, but now subject to the constraints $|S| = k$ and $S(C_f) \leq M$ as opposed to just $|S| = k$. Let $\mathcal{S}_M$ be the set of all $S \subseteq F$ such that $|S| = k$ and $S(C_f) \leq M$. Since $f_y(S)$ is monotone, maximizing $f_y(S)$ over $\mathcal{S}_M$ is equivalent to maximizing $f_y(S)$ over the independence system $(F, \mathcal{I}_M)$ with maximal independent sets $\mathcal{S}_M$.

Then all that is needed to approximately separate Eq. (6) corresponding to a fixed M is an oracle for deciding membership in $(F, \mathcal{I}_M)$. Recall that $S \subseteq F$ is in $(F, \mathcal{I}_M)$ if there exists a set $S' \supseteq S$ such that $|S'| = k$ and $S'(C_f) \leq M$. But, even deciding membership of the empty set in $(F, \mathcal{I}_M)$ requires one to solve a k-median instance on the fixed clients, which is in general NP-hard. More generally, we are required to solve an instance of the incremental k-median problem (see Section 3.1) with existing cluster centers in S.

While exactly solving incremental k-median is NP-hard, we have a constant approximation algorithm for it (call it A), by Lemma 13. So, we could define a new system $(F, \mathcal{I}'_M)$ that contains a set $S \subseteq F$ if the output of A for the incremental k-median instance with existing cluster centers S has cost at most M. But, due to the unpredictable behavior of A, $(F, \mathcal{I}'_M)$ may no longer be a 1-system, or even an independence system. To restore the subset closed property, the membership oracle needs to ensure that: (a) if a subset $S' \subseteq S$ is determined to not be in $(F, \mathcal{I}'_M)$, then S is not either, and (b) if a superset $S' \supseteq S$ is determined to be in $(F, \mathcal{I}'_M)$, then so is S.

We now describe the modified greedy maximization algorithm GreedyMax that we use to try to separate one of the inequalities in Eq. (6), which uses a built-in membership oracle that ensures the above properties hold. Pseudocode is given in the full paper, and we informally describe it here. GreedyMax initializes $S_0 = \emptyset$, $F_0 = F$, and starts with a M-cheap k-median solution T_0 (generated by running a γ-approximation on the k-median instance involving only fixed clients C_f). In iteration l, GreedyMax starts with a partial

solution S_{l-1} with $l-1$ cluster centers, and it is considering adding cluster centers in F_{l-1} to S_{l-1}. For each cluster center i in F_{l-1}, GreedyMax generates some k-median solution $T_{l,i}$ containing $S_{l-1} \cup \{i\}$ to determine if $S_{l-1} \cup \{i\}$ is in the independence system. If a previously generated solution, T_0 or $T_{l',i'}$ for any l', i', contains $S_{l-1} \cup \{i\}$ and is M-cheap, then $T_{l,i}$ is set to this solution. Otherwise, GreedyMax runs the incremental k-median approximation algorithm on the instance with existing cluster centers in $S_{l-1} \cup \{i\}$, the only cluster centers in the instance are F_{l-1}, and the client set is C_f. It sets $T_{l,i}$ to the solution generated by the approximation algorithm.

After generating the set of solutions $\{T_{l,i}\}_{i \in F_{l-1}}$, if one of these solutions contains $S_{l-1} \cup \{i\}$ and is M-cheap, then GreedyMax concludes that $S_{l-1} \cup \{i\}$ is in the independence system. This, combined with the fact that these solutions may be copied from previous iterations ensures property (b) holds (as the M-cheap solutions generated by GreedyMax are implicitly considered to be in the independence system). Otherwise, since GreedyMax was unable to find an M-cheap superset of $S_{l-1} \cup \{i\}$, it considers $S_{l-1} \cup \{i\}$ to not be in the independence system. In accordance with these beliefs, GreedyMax initializes F_l as a copy of F_{l-1}, and then removes any i such that it did not find an M-cheap superset of $S_{l-1} \cup \{i\}$ from F_l and thus from future consideration, ensuring property (a) holds. It then greedily adds to S_{l-1} the i in F_l that maximizes $f_y(S_{l-1} \cup \{i\})$ as defined before to create a new partial solution S_l. After the kth iteration, GreedyMax outputs the solution S_k.

Our approximate separation oracle, SepOracle, can then use GreedyMax as a subroutine. Pseudocode is given in the full paper, and we give an informal description of the algorithm here. SepOracle checks all constraints not involving the regret, and then outputs any violated constraints it finds. If none are found, it then runs a k-median approximation algorithm on the instance containing only the fixed clients to generate a solution T_0. For each M in Z_ϵ, if T_0 is M-cheap, it then invokes GreedyMax for this value of M (otherwise, GreedyMax will consider the corresponding independence system to be empty, so there is no point in running it), passing T_0 to GreedyMax. If then checks the inequality $\sum_{j \in C'} \sum_i c_{ij} y_{ij} - S(C') \leq M - \sum_{j \in C_f} \sum_i c_{ij} y_{ij} + r$ for the solution S outputted by GreedyMax, and outputs this inequality if it is violated.

Using the ellipsoid method where SepOracle is used as the separation oracle now gives the following lemma. The proof is deferred to the full paper.

▶ **Lemma 14.** *If there exists a deterministic polynomial-time γ-approximation algorithm for the k-median problem, then for every $\epsilon > 0$ there exists a deterministic algorithm that outputs a $(2\gamma(1+\epsilon), 2)$-approximate fractional solution to the universal k-median problem in polynomial time.*

3.3 Rounding the Fractional Solution for Universal k-Median with Fixed Clients

The rounding algorithm for universal k-median with fixed clients is almost identical to the rounding algorithm for universal k-median without fixed clients. The only difference is that in constructing a k-median with discounts problem, we give the fixed clients a discount of 0 rather than a discount of $3f_j$, as these clients will always appear and thus we want their connection cost to always factor into the cost of the k-median with discounts instance. The cost of a solution ALG to the k-median with discounts instance and the regret of ALG against an adversary with costs $3f_j$ now differs by $\sum_{j \in C_f} 3f_j$ (before, they were equal). However, as before $\sum_{j \in C_f} 3f_j$ is at most some constant times $\text{OPT}(C_f) + \text{MR}$, which lower bounds $\text{OPT}(C') + \text{MR}$ for all realizations $C' \supseteq C_f$. So an analysis of the rounding similar to that in Section 2 still allows us to prove Theorem 10, as the the offset $\sum_{j \in C_f} 3f_j$ (and multiples of it appearing in the analysis) can be absorbed into the (α, β)-approximation guarantee.

4 Universal k-means

In this section, we sketch our universal algorithm for k-means with the following guarantee:

▶ **Corollary 15.** *There exists a* $(108, 412)$*-approximate universal algorithm for the* k*-means problem.*

This follows as a special case of a more general ℓ_p-clustering result, given in the full paper; due to space constraints, we focus on k-means here.

Before describing further details of the universal k-means algorithm, we note a rather unusual feature of the universal clustering framework. Typically algorithms effectively optimize the ℓ_2^2 objective (i.e., sum of squared distances) instead of the k-means objective because these are equivalent in the following sense: an α-approximation for the k-means objective is equivalent to an α^2-approximation for the ℓ_2^2 objective. But, this equivalence fails in the setting of universal algorithms for reasons that we discuss below. Indeed, we first give a universal ℓ_p^p-clustering algorithm, which is a simple extension of the k-median algorithm, and then outline our ℓ_p-clustering algorithm in the setting $p = 2$, which turns out to be much more challenging.

Similar to k-median, we use the primitive of an algorithm for the ℓ_p^p-clustering with discounts problem: In this problem, are given a ℓ_p^p-clustering instance, but where each client j has an additional (non-negative) parameter r_j called its *discount*. Our goal is to place k cluster centers that minimize the total connection costs of all clients. But, the connection cost for client j can now be discounted by up to r_j^p, i.e., client j with connection cost c_j contributes $(c_j^p - r_j^p)^+ := \max\{0, c_j^p - r_j^p\}$ to the objective of the solution. (Note that the k-median with discounts problem that we described in the previous section is a special case of this problem for $p = 1$.)

Let OPT be the cost of an optimal solution to the ℓ_p^p-clustering with discounts problem. We say an algorithm ALG that outputs a solution with connection cost c_j for client j is a (γ^p, σ)-approximation[2] if $\sum_{j \in C}(c_j^p - \gamma^p \cdot r_j^p)^+ \leq \sigma \cdot \text{OPT}$. That is, a (γ^p, σ)-approximate algorithm outputs a solution whose objective function computed using discounts $\gamma \cdot r_j$ for all j is at most σ times the optimal objective using discounts r_j. We give the following result about the ℓ_p^p-clustering with discounts problem (see full paper for details):

▶ **Lemma 16.** *There exists a (deterministic) polynomial-time* $(9^p, \frac{2}{3} \cdot 9^p)$*-approximation algorithm for the* ℓ_p^p*-clustering with discounts problem.*

The rest of this section is dedicated to sketching our algorithm for the universal k-means problem. As for k-median, we have two stages, the fractional algorithm and the rounding algorithm, that we sketch in the next two subsections.

[2] We refer to this as a (γ^p, σ)-approximation instead of a (γ, σ)-approximation to emphasize the difference between the scaling factor for discounts γ and the loss in approximation factor γ^p.

4.1 Universal k-means: Fractional Algorithm

Let us start by describing the fractional relaxation of the universal k-means problem[3] (again, P is the k-median polytope defined as in Section 2.1):

$$\min\{r : (x,y) \in P; \forall C' \subseteq C : \left(\sum_{j \in C'} \sum_i c_{ij}^2 y_{ij}\right)^{1/2} - \textsc{opt}(C') \leq r\}, \tag{7}$$

As described earlier, when minimizing regret, the k-means and ℓ_2^2 objectives are no longer equivalent. For instance, recall that one of the key steps in Lemma 8 was to establish the submodularity of the function $f_y(S)$ denoting the maximum regret caused by any realization when comparing two given solutions: a fractional solution y and an integer solution S. Indeed, the worst case realization had a simple structure: choose all clients that have a smaller connection cost for S than for y. This observation continues to hold for the ℓ_2^2 objective because of the linearity of $f_y(S)$ as a function of the realized clients once y and S are fixed. But, the k-means objective is not linear even after fixing the solutions, and as a consequence, we lose both the simple structure of the maximizing realization as well as the submodularity of the overall function $f_y(S)$. For instance, consider two clients: one at distances 1 and 0, and another at distances $1+\epsilon$ and 1, from y and S respectively. Using the ℓ_p objective, the regret with both clients is $(2+\epsilon)^{1/2} - 1 < 1$, whereas with just the first client the regret is 1.

The above observation results in two related difficulties: first, that $f_y(S)$ is not submodular and hence standard submodular maximization techniques do not apply, but also that given y and S, we cannot even compute the function $f_y(S)$ efficiently. To overcome this difficulty, we further refine the function $f_y(S)$ to a collection of functions $f_{y,Y}(S)$ by also fixing the cost of the fractional solution y to at most a given value Y. Let $\textsc{frac}_2, \textsc{frac}_2^2$ denote the k-means and ℓ_2^2-objectives of a given fractional solution, and S_2, S_2^2 the same for the solution using the set of cluster centers S. We can show that:

$$\max_{C' \subseteq C} \left[\textsc{frac}_2(C') - S_2(C')\right] \simeq_2 \max_Y \max_{C' \subseteq C : \textsc{frac}_2^2(C') \leq Y} \left[\frac{\textsc{frac}_2^2(C') - S_2^2(C')}{Y^{1/2}}\right],$$

where $\simeq_2$ denotes equality to within a factor of 2. In turn, by guessing the maximizing value of Y we can (approximately) reduce maximizing the difference in k-means objectives to maximizing the difference in ℓ_2^2 objectives, subject to the constraint $\textsc{frac}_2^2(C') \leq Y$.

A separation oracle then just needs to (approximately) compute $\max\{\textsc{frac}_2^2(C') - S_2^2(C') : C' \subseteq C, \textsc{frac}_2^2(C') \leq Y\}$ for each fixed (discretized) value of Y. To do so, we show that allowing an adversary to choose *fractional* realizations of clients does not give them an advantage.

▶ **Lemma 17.** *For any two solutions y, S, there exists a global maximum of $\textsc{frac}_2(\mathbf{I}) - S_2(\mathbf{I})$ over fractional realizations $\mathbf{I} \in [0,1]^C$ where all the clients are integral, i.e., $\mathbf{I} \in \{0,1\}^C$. Therefore,*

$$\max_{\mathbf{I} \in [0,1]^C} \left[\textsc{frac}_2(\mathbf{I}) - S_2(\mathbf{I})\right] = \max_{C' \subseteq C} \left[\textsc{frac}_2(C') - S_2(C')\right].$$

[3] The constraints are not simultaneously linear in y and r, although fixing r, we can write these constraints as $\sum_{j \in C'} \sum_i c_{ij}^p y_{ij} \leq (\textsc{opt}(C') + r)^p$, which is linear in y. In turn, to solve this program we bisection search over r, using the ellipsoid method to determine if there is a feasible point for each fixed r.

We then show that $f_{y,Y}(S) := \max\{\text{FRAC}_2^2(\mathbf{I}) - S_2^2(\mathbf{I}) : \mathbf{I} \in [0,1]^C, \text{FRAC}_2^2(\mathbf{I}) \leq Y\}$ is a submodular function. Since we are allowed to use fractional clients, computing $f_{y,Y}(S)$ for a given S is a fractional knapsack problem which can be solved in polynomial time (whereas computing $\max\{\text{FRAC}_2^2(C') - S_2^2(C') : C' \subseteq C, \text{FRAC}_2^2(C') \leq Y\}$ requires solving an integer knapsack problem), giving an efficient separation oracle using the greedy algorithm for submodular maximization.

4.2 Universal k-Means: Rounding Algorithm

At a high level, we use the same strategy for rounding the fractional k-means solution as we did with k-median. Namely, we use Lemma 16 to solve a discounted version of the problem where the discount for each client is equal to the (scaled) cost of the client in the fractional solution. However, if we apply this directly to the k-means objective, we run into several problems. In particular, the linear discounts are incompatible with the non-linear objective defined over the clients. A more promising idea is to use these discounts on the ℓ_2^2 objective, which in fact is defined as a linear combination over the individual client's objectives. But, for this to work, we will first need to relate the regret bound in the ℓ_2^2 objective to that in the k-means objective. We show that, roughly speaking, the realization that maximizes the regret of an algorithm ALG against a fixed solution SOL in both objectives is the same under a "farness" condition:

▶ **Lemma 18.** *Suppose* ALG *and* SOL *are two solutions to a k-means instance, such that there is a subset of clients C^* with the following property: for every client in C^*, the connection cost in* ALG *is greater than* 2 *times the connection cost in* SOL*, while for every client not in C^*, the connection cost in* SOL *is at least the connection cost in* ALG*. Then, C^* is a $1/2$-maximizer of $\text{ALG}_2(C') - \text{SOL}_2(C')$.*

Given any solution SOL, it is easy to define a *virtual* solution $\widetilde{\text{SOL}}$ whose individual connection costs are bounded by 2 times that in SOL, and $\widetilde{\text{SOL}}$ satisfies the farness condition. This allows us to relate the regret of ALG against $\widetilde{\text{SOL}}$ (and thus against 2 times SOL) in the ℓ_2^2 objective to its regret in the k-means objective.

5 Universal k-Center

In this section, we prove the following guarantee for universal k-center:

▶ **Theorem 19.** *There exists a $(3,3)$-approximate algorithm for the universal k-center problem.*

First, note that for every client j, its distance to the closest cluster center in the minimum regret solution MRS is at most $\text{MR}_j := \min_{i \in F} c_{ij} + \text{MR}$; otherwise, in the realization with only client j, MRS would have regret $> \text{MR}$. We first design an algorithm ALG that 3-approximates these distances MR_j, i.e., for every client j, its distance to the closest cluster center in ALG is at most 3MR_j. Since $\min_{i \in F} c_{ij}$ lower bounds $\text{OPT}(C')$ for any C' containing j, this gives a $(3,3)$-approximation. This algorithm actually satisfies a more general property: given *any* value r, it produces a set of cluster centers such that every client j is at a distance $\leq 3r_j$ from its closest cluster center, where $r_j := \min_{i \in F} c_{ij} + r$. Moreover, if $r \geq \text{MR}$, then the number of cluster centers selected by ALG is at most k (for smaller values of r, ALG might select more than k cluster centers).

Our algorithm ALG is a natural greedy algorithm. We order clients j in increasing order of r_j, and if a client j does not have a cluster center within distance $3r_j$ in the current solution, then we add its closest cluster center in F to the solution.

▶ **Lemma 20.** *Given a value r, the greedy algorithm* ALG *selects cluster centers that satisfy the following properties:*

- *every client j is within a distance of $3r_j = 3(\min_{i \in F} c_{ij} + r)$ from their closest cluster center.*
- *If $r \geq$ MR, then* ALG *does not select more than k cluster centers, i.e., the solution produced by* ALG *is feasible for the k-center problem.*

Proof. The first property follows from the definition of ALG. To show that ALG does not pick more than k cluster centers, we map the cluster center i added in ALG by some client j to its closest cluster center i' in MRS. Now, we claim that no two cluster centers i_1, i_2 in ALG can be mapped to the same cluster center i' in MRS. Clearly, this proves the lemma since MRS has only k cluster centers.

Suppose i_1, i_2 are two cluster centers in ALG mapped to the same cluster center i' in MRS. Assume without loss of generality that i_1 was added to ALG before i_2. Let j_1, j_2 be the clients that caused i_1, i_2 to be added; since i_2 was added later, we have $r_{j_1} \leq r_{j_2}$. The distance from j_2 to i_1 is at most the length of the path (j_2, i', j_1, i_1) , which is at most $2r_{j_2} + r_{j_1} \leq 3r_{j_2}$. But, in this case j_2 would not have added a new cluster center i_2, thus arriving at a contradiction. ◀

Theorem 19 follows since there are only polynomially many possibilities for the k-center objective across all realizations (namely, the set of all cluster center to client distances) and thus polynomially many possible values for MR (the set of all differences between all possible solution costs). So we can simply run the algorithm of Lemma 20 with r equal to each of these values, and then choose the solution corresponding to the smallest r that results in the algorithm using at most k cluster centers, which will be at most MR by Lemma 20.

We note that the greedy algorithm described above can be viewed as an extension of the k-center algorithm in [24] to a $(3, 3)$-approximation for the "k-center with discounts" problem, where the discounts are the minimum distances $\min_{i \in F} c_{ij}$.

6 Hardness of Universal Clustering for General Metrics

In this section we give some hardness results to help contextualize the algorithmic results. Much like the hardness results for k-median, all our reductions are based on the NP-hardness of approximating set cover (or equivalently, dominating set) due to the natural relation between the two types of problems. We state our hardness results in terms of ℓ_p-clustering. Setting $p = 1$ gives hardness results for k-median, and setting $p = \infty$ (and using the convention $1/\infty = 0$ in the proofs as needed) gives hardness results for k-center.

The results in this section can be extended to Euclidean metrics by building off the reductions in [36], albeit with worse constants. Due to space constraints, we defer our results for Euclidean metrics to the full paper.

6.1 Hardness of Approximating α

▶ **Theorem 21.** *For all $p \geq 1$, finding an (α, β)-approximate solution for universal ℓ_p-clustering where $\alpha < 3$ is NP-hard.*

Proof. To prove the theorem, given an instance of set cover, we construct the following instance of universal ℓ_p-clustering:

- For each element, there is a corresponding client in the universal ℓ_p-clustering instance.
- For each set S, there is a cluster center which is distance 1 from the clients corresponding to elements in S and 3 from other all clients.

If there is a set cover of size k, the corresponding cluster centers are an optimal solution for any realization of clients, i.e. $\text{MR} = 0$. Furthermore, in any single-client realization, $\text{OPT} = 1$. So an (α, β)-approximate universal ℓ_p-clustering algorithm must find a solution within distance of α of every client to satisfy the regret guarantee in single-client realizations. If $\alpha < 3$, this implies it is distance 1 from every client, in which case its cluster centers also correspond to a set cover. So this algorithm can be used to find set covers of size at most k if they exist, which is an NP-hard task. ◀

Note that for e.g. k-median, we can classically get an approximation ratio of less than 3. So this theorem shows that the universal version of the problem is harder, even if we are willing to use arbitrary large β.

6.2 Hardness of Approximating β

We give the following result on the hardness of universal ℓ_p-clustering.

▶ **Theorem 22.** *For all $p \geq 1$, finding an (α, β)-approximate solution for universal ℓ_p-clustering where $\beta < 2$ is NP-hard.*

Proof. Given an instance of dominating set $G = (V, E)$, we construct the following instance of universal ℓ_p-clustering:

- For each vertex $v \in V$, replace it with a k-clique.
- For each $(u, v) \in E$, add an edge from every vertex in u's clique to every vertex in v's clique.
- To turn this modified graph into a clustering instance, place a client and cluster center at each vertex, and impose the shortest path metric on the clients, where all edges are length 1.

Suppose a dominating set of size k exists in the modified graph (and thus in the original graph). Then the corresponding cluster centers are distance at most 1 from every client. Since any solution is distance 0 from at most k clients and distance at least 1 from all other clients, these cluster centers have regret at most $k^{1/p}$. It now suffices to prove the claim that any k cluster centers that aren't a dominating set have regret at least $2k^{1/p}$ in a realization where OPT has cost 0. The theorem follows since an (α, β)-approximate algorithm would produce a solution with cost at most $\beta k^{1/p}$ in any such realization, i.e. can be used to find dominating sets of size at most k if they exist when $\beta < 2$. The claim follows since if the cluster centers aren't a dominating set, there is a k-clique they are distance 2 or more from. The realization containing exactly the clients in this k-clique satisfies the desired properties. ◀

7 Future Directions

In this paper, we gave the first universal algorithms for clustering problems. While we achieve constant approximation guarantees for these problems, the actual constants are orders of magnitude larger than the best (non-universal) approximations known for these problems. In part to ensure clarity of presentation, we did not attempt to optimize these constants. But it is unlikely that our techniques will lead to *small* constants for the k-median and k-means problems. On the other hand, we show that in general it is **NP**-hard to find an (α, β)-approximation algorithm for a universal clustering problem where α matches the approximation factor for the standard clustering problem. Therefore, it is not entirely clear what one should expect: *are there universal algorithms for clustering with approximation factors of the same order as the classical (non-universal) bounds?*

Another open research direction pertains to Euclidean clustering. Here, we showed that in $\mathbb{R}^d$ for $d \geq 2$, α needs to be bounded away from 1, which is in stark contrast to non-universal clustering problems that admit PTASes in constant-dimension Euclidean space. But for universal clustering on a line, the picture is not as clear. On a line, the lower bounds on α are no longer valid, which brings forth the possibility of (non-bicriteria) approximations of regret. Indeed, there is 2-approximation for universal k-median on a line [29], and even better, an *optimal* algorithm for universal k-center on a line [6]. This raises the natural question: *can we design a PTAS for the universal k-median problem on a line?*

References

1 Sara Ahmadian, Ashkan Norouzi-Fard, Ola Svensson, and Justin Ward. Better guarantees for k-means and Euclidean k-median by primal-dual algorithms. In *Proceedings of the 58th Annual IEEE Symposium on Foundations of Computing*, pages 61–72, October 2017. doi:10.1109/FOCS.2017.15.

2 N. Alon and Y. Azar. On-line steiner trees in the Euclidean plane. In *Proceedings of the 8th Annual Symposium on Computational Geometry*, pages 337–343, 1992.

3 Aaron Archer, Ranjithkumar Rajagopalan, and David B. Shmoys. Lagrangian relaxation for the k-median problem: New insights and continuity properties. In Giuseppe Di Battista and Uri Zwick, editors, *Algorithms - ESA 2003: 11th Annual European Symposium, Budapest, Hungary, September 16-19, 2003. Proceedings*, pages 31–42. Springer Berlin Heidelberg, Berlin, Heidelberg, 2003. doi:10.1007/978-3-540-39658-1_6.

4 Sanjeev Arora, Prabhakar Raghavan, and Satish Rao. Approximation schemes for Euclidean k-medians and related problems. In *Proceedings of the Thirtieth Annual ACM Symposium on Theory of Computing*, STOC '98, pages 106–113, New York, NY, USA, 1998. ACM. doi:10.1145/276698.276718.

5 Vijay Arya, Naveen Garg, Rohit Khandekar, Adam Meyerson, Kamesh Munagala, and Vinayaka Pandit. Local search heuristic for k-median and facility location problems. In *Proceedings of the Thirty-third Annual ACM Symposium on Theory of Computing*, STOC '01, pages 21–29, New York, NY, USA, 2001. ACM. doi:10.1145/380752.380755.

6 I. Averbakh and Oded Berman. Minimax regret p-center location on a network with demand uncertainty. *Location Science*, 5(4):247–254, 1997. doi:10.1016/S0966-8349(98)00033-3.

7 Igor Averbakh and Oded Berman. Minmax regret median location on a network under uncertainty. *INFORMS Journal on Computing*, 12(2):104–110, 2000. doi:10.1287/ijoc.12.2.104.11897.

8 D. Bertsimas and M. Grigni. Worst-case examples for the spacefilling curve heuristic for the Euclidean traveling salesman problem. *Operations Research Letter*, 8(5):241–244, October 1989.

9 Anand Bhalgat, Deeparnab Chakrabarty, and Sanjeev Khanna. Optimal lower bounds for universal and differentially private Steiner trees and TSPs. In Leslie Ann Goldberg, Klaus Jansen, R. Ravi, and José D. P. Rolim, editors, *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques*, pages 75–86, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg.

10 Costas Busch, Chinmoy Dutta, Jaikumar Radhakrishnan, Rajmohan Rajaraman, and Srinivasagopalan Srivathsan. Split and join: Strong partitions and universal Steiner trees for graphs. In *53rd Annual IEEE Symposium on Foundations of Computer Science, FOCS 2012, New Brunswick, NJ, USA, October 20-23, 2012*, pages 81–90, 2012.

11 Jaroslaw Byrka, Fabrizio Grandoni, Thomas Rothvoß, and Laura Sanità. Steiner tree approximation via iterative randomized rounding. *J. ACM*, 60(1):6:1–6:33, 2013.

12 Gruia Calinescu, Chandra Chekuri, Martin Pál, and Jan Vondrák. Maximizing a monotone submodular function subject to a matroid constraint. *SIAM J. Comput.*, 40(6):1740–1766, 2011. doi:10.1137/080733991.

13 Deeparnab Chakrabarty and Chaitanya Swamy. Approximation algorithms for minimum norm and ordered optimization problems. In *Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing*, STOC 2019, page 126–137, New York, NY, USA, 2019. Association for Computing Machinery. doi:10.1145/3313276.3316322.

14 Moses Charikar, Sudipto Guha, Éva Tardos, and David B. Shmoys. A constant-factor approximation algorithm for the k-median problem (extended abstract). In *Proceedings of the Thirty-first Annual ACM Symposium on Theory of Computing*, STOC '99, pages 1–10, New York, NY, USA, 1999. ACM. doi:10.1145/301250.301257.

15 Teofilo F. Gonzalez. Clustering to minimize the maximum intercluster distance. *Theor. Comput. Sci.*, 38:293–306, 1985.

16 Igor Gorodezky, Robert D. Kleinberg, David B. Shmoys, and Gwen Spencer. Improved lower bounds for the universal and a priori TSP. In Maria Serna, Ronen Shaltiel, Klaus Jansen, and José Rolim, editors, *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques*, pages 178–191, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg.

17 F. Grandoni, A. Gupta, S. Leonardi, P. Miettinen, P. Sankowski, and M. Singh. Set covering with our eyes closed. In *Proceedings of the 49th Annual IEEE Symposium on Foundations of Computer Science*, October 2008.

18 Martin Grötschel, László Lovász, and Alexander Schrijver. The ellipsoid method and its consequences in combinatorial optimization. *Combinatorica*, 1(2):169–197, 1981.

19 Sudipto Guha and Kamesh Munagala. Exceeding expectations and clustering uncertain data. In *Proceedings of the Twenty-Eighth ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, PODS '09, page 269–278, New York, NY, USA, 2009. Association for Computing Machinery. doi:10.1145/1559795.1559836.

20 Anupam Gupta, Mohammad T. Hajiaghayi, and Harald Räcke. Oblivious network design. In *Proceedings of the Seventeenth Annual ACM-SIAM Symposium on Discrete Algorithm*, SODA '06, pages 970–979, Philadelphia, PA, USA, 2006. Society for Industrial and Applied Mathematics. URL: http://dl.acm.org/citation.cfm?id=1109557.1109665.

21 Anupam Gupta and Kanat Tangwongsan. Simpler analyses of local search algorithms for facility location. *ArXiv*, abs/0809.2554, 2008. arXiv:0809.2554.

22 Mohammad T. Hajiaghayi, Robert Kleinberg, and Tom Leighton. Improved lower and upper bounds for universal TSP in planar metrics. In *Proceedings of the Seventeenth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 649–658, 2006.

23 Dorit S. Hochbaum and David B. Shmoys. A best possible heuristic for the k-center problem. *Math. Oper. Res.*, 10(2):180–184, May 1985. doi:10.1287/moor.10.2.180.

24 Dorit S. Hochbaum and David B. Shmoys. A unified approach to approximation algorithms for bottleneck problems. *J. ACM*, 33(3):533–550, May 1986. doi:10.1145/5925.5933.

25 Kamal Jain and Vijay V. Vazirani. Approximation algorithms for metric facility location and k-median problems using the primal-dual schema and lagrangian relaxation. *J. ACM*, 48(2):274–296, 2001. doi:10.1145/375827.375845.

26 L. Jia, G. Lin, G. Noubir, R. Rajaraman, and R. Sundaram. Universal algorithms for TSP, Steiner tree, and set cover. In *Proceedings of the 36th Annual ACM Symposium on Theory of Computing*, 2005.

27 Lujun Jia, Guevara Noubir, Rajmohan Rajaraman, and Ravi Sundaram. GIST: Group-independent spanning tree for data aggregation in dense sensor networks. In Phillip B. Gibbons, Tarek Abdelzaher, James Aspnes, and Ramesh Rao, editors, *Distributed Computing in Sensor Systems*, pages 282–304, Berlin, Heidelberg, 2006. Springer Berlin Heidelberg.

28 Tapas Kanungo, David M. Mount, Nathan S. Netanyahu, Christine D. Piatko, Ruth Silverman, and Angela Y. Wu. A local search approximation algorithm for k-means clustering. In *Proceedings of the Eighteenth Annual Symposium on Computational Geometry*, SCG '02, pages 10–18, New York, NY, USA, 2002. ACM. doi:10.1145/513400.513402.

29 Adam Kasperski and Pawel Zielinski. On the existence of an FPTAS for minmax regret combinatorial optimization problems with interval data. *Oper. Res. Lett.*, 35:525–532, 2007.

30 Samir. Khuller and Yoram J. Sussmann. The capacitated k-center problem. *SIAM Journal on Discrete Mathematics*, 13(3):403–418, 2000. doi:10.1137/S0895480197329776.

31 Stavros G. Kolliopoulos and Satish Rao. A nearly linear-time approximation scheme for the Euclidean k-median problem. In Jaroslav Nešetřil, editor, *Algorithms - ESA' 99*, pages 378–389, Berlin, Heidelberg, 1999. Springer Berlin Heidelberg.

32 Panos Kouvelis and Gang Yu. *Robust 1-Median Location Problems: Dynamic Aspects and Uncertainty*, pages 193–240. Springer US, Boston, MA, 1997. doi:10.1007/978-1-4757-2620-6_6.

33 Amit Kumar, Yogish Sabharwal, and Sandeep Sen. A simple linear time $(1+\epsilon)$-approximation algorithm for k-means clustering in any dimensions. In *Proceedings of the 45th IEEE Symposium on Foundations of Computer Science*, pages 454–462, 2004.

34 Shi Li and Ola Svensson. Approximating k-median via pseudo-approximation. In *Proceedings of the Forty-fifth Annual ACM Symposium on Theory of Computing*, pages 901–910, 2013.

35 Stuart P. Lloyd. Least squares quantization in pcm. *IEEE Trans. Information Theory*, 28:129–136, 1982.

36 Stuart G. Mentzer. Approximability of metric clustering problems. Unpublished manuscript, March 2016.

37 Viswanath Nagarajan, Baruch Schieber, and Hadas Shachnai. The Euclidean k-supplier problem. In Michel Goemans and José Correa, editors, *Integer Programming and Combinatorial Optimization*, pages 290–301, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg.

38 G. L. Nemhauser, L. A. Wolsey, and M. L. Fisher. An analysis of approximations for maximizing submodular set functions—i. *Mathematical Programming*, 14(1):265–294, 1978. doi:10.1007/BF01588971.

39 Loren K. Platzman and John J. Bartholdi, III. Spacefilling curves and the planar travelling salesman problem. *J. ACM*, 36(4):719–737, 1989. doi:10.1145/76359.76361.

40 Frans Schalekamp and David B. Shmoys. Algorithms for the universal and a priori TSP. *Operations Research Letters*, 36(1):1–3, 2008. doi:10.1016/j.orl.2007.04.009.

LF Successor: Compact Space Indexing for Order-Isomorphic Pattern Matching

Arnab Ganguly
Department of Computer Science, University of Wisconsin – Whitewater, WI, USA

Dhrumil Patel
School of EECS, Louisiana State University, Baton Rouge, LA, USA

Rahul Shah
School of EECS, Louisiana State University, Baton Rouge, LA, USA

Sharma V. Thankachan
Department of Computer Science, University of Central Florida, Orlando, FL, USA

Abstract

Two strings are order isomorphic iff the relative ordering of their characters is the same at all positions. For a given text $T[1, n]$ over an ordered alphabet of size σ, we can maintain an order-isomorphic suffix tree/array in $O(n \log n)$ bits and support (order-isomorphic) pattern/substring matching queries efficiently. It is interesting to know if we can encode these structures in space close to the text's size of $n \log \sigma$ bits. We answer this question positively by presenting an $O(n \log \sigma)$-bit index that allows access to any entry in order-isomorphic suffix array (and its inverse array) in $t_{\mathsf{SA}} = O(\log^2 n / \log \sigma)$ time. For any pattern P given as a query, this index can count the number of substrings of T that are order-isomorphic to P (denoted by occ) in $O((|P| \log \sigma + t_{\mathsf{SA}}) \log n)$ time using standard techniques. Also, it can report the locations of those substrings in additional $O(occ \cdot t_{\mathsf{SA}})$ time.

2012 ACM Subject Classification Theory of computation → Pattern matching

Keywords and phrases Succinct data structures, Pattern Matching

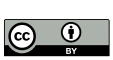
Digital Object Identifier 10.4230/LIPIcs.ICALP.2021.71

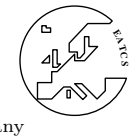
Category Track A: Algorithms, Complexity and Games

1 Introduction

An index of a text $T[1, n]$ is a data structure that is capable of counting/reporting all those *substrings* of T that "*match*" (as per the problem specific definition of match) with any pattern P given as a query. We use Σ to denote the alphabet set (of size σ) from which the characters in T are drawn from. WLOG, we assume $T[n] = \$$, a special character that does not appear anywhere else in T. Two fundamental indexes for exact pattern matching are the suffix tree (ST) [23] and the suffix array (SA) [17]. Both takes $\Theta(n \log n)$ bits of space, which could be much larger than the $n\lceil \log \sigma \rceil$ bits needed to store T optimally. The first succinct indexes that use close to $n \log \sigma$ bits are the Compressed Suffix Array (CSA) [13] and the FM-index [7]. The crucial component of FM Index is Burrows-Wheeler Transform (BWT) [2] and its associated operation called the *Last-to-Front* (LF) *mapping*. The subsequent work lead to fully functional suffix trees in succinct space [22]. See [20] for further reading.

The parameterized ST [1, 18] and the order-isomorphic ST [5] are two popular ST variants under the class known as *suffix trees with missing suffix links* [4]. As they do not hold some critical structural properties of the original ST, their compression is challenging. Recently, Ganguly *et al.* showed that it is indeed possible to compress the parameterized suffix arrays. They implemented LF mapping using a BWT-like transformation called the parameterized BWT [10]. However, such a transformation is hard to define for order-isomorphic ST because





LF mapping could lead to multiple changes in the (encoding of) associated suffixes. To that end, we present a novel technique for implementing the LF mapping (named LF *Successor*), leading to the first compact space index for order-isomorphic pattern matching.

1.1 Generalizing the Philosophy of BWT and LF Mapping

We present an overview of our approach using three text indexing problems for (i) traditional/exact matching, (ii) parameterized matching, and (iii) order-isomorphic matching, in that order, to show gradation and successive generalization of the LF mapping approach.

Indexing for Traditional Matching. The classic solution is the suffix tree (ST), which is lexicographic arrangement of all strings in $\mathcal{S} = \{T[t, n] \mid 1 \leq t \leq n\}$ as a compacted trie. We use $\mathsf{path}(u)$ to denote the concatenation of edge labels on the path from the root to node u. Let ℓ_i denotes the ith leftmost leaf. Then, $\mathsf{path}(\ell_i)$ is exactly the ith smallest string in S in the lexicographic order. The suffix array $\mathsf{SA}[1, n]$ is such that $\mathsf{SA}[i] = n + 1 - |\mathsf{path}(\ell_i)|$. Also, its inverse array $\mathsf{SA}^{-1}[1, n]$ is such that $\mathsf{SA}^{-1}[\mathsf{SA}[i]] = i$. For convenience, we use the term "suffix i" for $T[\mathsf{SA}[i], n]$. For all i, where $\mathsf{SA}[i] \neq 1$, we define the Last-to-Front (LF) mapping as $\mathsf{LF}(i) = \mathsf{SA}^{-1}[\mathsf{SA}[i] - 1]$. Therefore, *suffix* $\mathsf{LF}(i)$ is obtained by prepending to *suffix* i the character in T which occurs just before the starting location of suffix i. The Burrows-Wheeler Transform is an array $\mathsf{BWT}[1, n]$, such that $\mathsf{BWT}[i] = T[\mathsf{SA}[i] - 1]$ (define $T[0] = \$$). Computing LF mapping is central to BWT based pattern matching, and in some sense, the BWT enables efficient computation of LF mapping in succinct space. Therefore, once we store the BWT and its associated counting structures, we can replace the costly (space-wise) suffix array with a (cheaper) sampled suffix array [7].

Indexing for Parameterized Matching. Here, P matches with T at position i iff there is one-to-one correspondence between the characters of P and $T[i, i + |P| - 1]$. For example, $xwyx$ can match with $abca$ as x can be mapped to a, b to w, and c to y. However, $abca$ does not match with $xyxw$ because both a and c cannot be mapped to x. Baker [1] presented an encoding called $\mathsf{prev}(S)$ which encodes every character in the string by replacing it by its distance to the previous occurrence of the same character and using 0 if the character has not occurred before. For example, $\mathsf{prev}(xwxyywx) = 0020144$. It is not hard to see that two strings X and Y are a parameterized match iff $\mathsf{prev}(X) = \mathsf{prev}(Y)$. The parameterized ST is a lexicographic arrangement of all strings in $\mathcal{S}' = \{\mathsf{prev}(T[i, n - 1]) \circ \$ \mid 1 \leq i < n\} \cup \{\$\}$ as a compacted trie, where $\circ$ denotes concatenation. The matching of P in T can be performed via exact matching of $\mathsf{prev}(P)$ in this suffix tree. The same notion of LF-mapping can be defined and implemented in succinct space using a BWT-like transform [10].

Indexing for Order-isomorphic Matching. This problem has received significant attention [3, 5, 14, 16, 19] due to its simple formulation and the ability to model complex matching problems in other domains where the relative ordering of characters has to be matched rather than the string itself. Here, there is a total ordering between the symbols in Σ. The pattern P matches with text $T[1, n]$ at position i if for any j, k in $[1, |P|]$, $P[j] < P[k]$ iff $T[i + j - 1] < T[i + k - 1]$. Similar constraints apply for $P[j] > P[k]$ and $P[j] = P[k]$. For example, 1423 can match with 2957 but not with 2657 because $6 < 7$ and $4 > 3$. A new encoding "pred" works in this case. This is a slight modification of the scheme in [5].

▶ **Definition 1** (pred encoding). *Given a character $S[i]$ in string S, its predecessor is a character q which occurs in $S[1, i - 1]$ such that $q \leq S[i]$ and there is no other character r in $S[1, i - 1]$ such that $q < r \leq S[i]$. Given a string S, $\mathsf{pred}(S)[i]$ is defined as follows: let*

alphabet symbol q be the predecessor of $S[i]$ in $S[1, i-1]$ and let position j be the rightmost occurrence of q in $S[1, i-1]$. Then, $\mathsf{pred}(S)[i] = (i-j)$ *if* $q \neq S[i]$, $(i-j)'$ *if* $q = S[i]$, *and* 0 *if q does not exist. Thus* $\mathsf{pred}(S)$ *is a string over the alphabet* $\{0, 1, 1', 2, 2', \ldots, |S|-1, (|S|-1)'\}$.

Thus, in pred encoding, every position (character) in T points to its closest predecessor on the left. For e.g., $\mathsf{pred}(0869514371) = 0\ 1\ 2\ 2\ 4\ 5\ 1\ 2\ 6\ 4'$. We refer to *primed* characters as an *equality* version of their non-primed counterparts. For example, $2'$ is equality variant of 2. It is easy to see that two strings X and Y are order-isomorphic iff $\mathsf{pred}(X) = \mathsf{pred}(Y)$.

The *order-isomorphic* ST [5] of T is a lexicographic arrangement of all strings in $\mathcal{S}'' = \{\mathsf{pred}(T[i, n-1]) \circ \$ \mid 1 \leq i < n\} \cup \{\$\}$ as a compacted trie. We order the encoded characters as: $0 < 1 < 1' < 2 < 2' < \cdots < n-1 < (n-1)' < \$$. Then, the order-isomorphic matching of P in T can be performed via exact matching of $\mathsf{pred}(P)$ in this suffix tree. As earlier, we can define the (order-isomorphic) suffix array SA, its inverse array and the LF mapping.

1.2 Challenges in Implementing (Generalised) LF Mapping Compactly

The challenge here is in deciding what needs to be precomputed and stored, so that $\mathsf{LF}(i)$ for any i can be computed efficiently. At its root, we need to solve the following: given two leaves ℓ_i and ℓ_j with $i < j$, how quickly can we decide whether $\mathsf{LF}(i) < \mathsf{LF}(j)$ or $\mathsf{LF}(i) > \mathsf{LF}(j)$.

In the case of **traditional matching**, the order between $\mathsf{LF}(i)$ and $\mathsf{LF}(j)$ will stay the same if the corresponding suffixes have the same *previous character* (which are $\mathsf{BWT}[i]$ and $\mathsf{BWT}[j]$). It will flip iff the previous character of the suffix corresponding to j is smaller than that of i in the lexicographic order. Therefore pair-wise comparison between such i and j can be computed in "bulk" for i against all j's, enabling "quick" computation of $\mathsf{LF}(i)$ [7].

In the case of **parameterized matching**, this order determination is more sophisticated [10]. Here, it becomes essential to see how prepending the previous character changes the canonical encoding of a suffix and how can this information be stored compactly. For example, consider $T[1, n] = abcabbadcb$ and the suffix $T[4, n] = abbadcb$. Its previous character $T[3]$ is c. When we prepend this character, the suffix (in traditional ST) becomes $cabbadcb$. The string corresponding to $T[4, n]$ in the parameterized suffix tree is $\mathsf{prev}(T[4, n]) = 0013004$. When $T[4, n]$ is prepended with c and prev is applied, apart from the insertion (of 0) at the beginning, there is one change within prev of $T[4, n]$, which is at the first occurrence of c in $T[4, n]$. Thus, the second last character in the encoding switches from 0 to 6, i.e., $\mathsf{prev}(T[3, n]) = 00013064$. Ganguly *et al.* [10] show how to record this change-location for each suffix succinctly using the paramaterized-BWT, which supports LF mapping. Again, as in the case of traditional pattern matching, we can compare two suffixes in terms of their LF mapping by comparing which suffix changes first – in case at least one of them changes before their longest common prefix (LCP). See [11, 15, 9] for some related results.

We now illustrate **order-isomorphic matching** using an example text $T[1, n] = 20869514371\$$. Then, $T[2, n] = 0869514371\$$ and $\mathsf{pred}(T[2, n-1]) \circ \$ = 0\ 1\ 2\ 2\ 4\ 5\ 1\ 2\ 6\ 4'\ \$$. However, pred after prepending $T[1]$, i.e., $\mathsf{pred}(T[1, n-1]) \circ \$$ is $0\ 0\ 2\ 3\ 2\ 5\ 5\ \ 7\ 8\ \ 6\ 4'\ \$$. Observe how the encoding changes when we go from $T[2, n]$ to $T[1, n]$. Apart from the obvious 0 in front, there are "five" other entries whose predecessor changed due to the newly inserted 2. Both earlier problems, traditional and parameterized, incurred only a constant (1 or 2) number of changes per suffix, and hence it was possible to record this information compactly. However, the number of changes here can be as large as σ, which makes it challenging and the existing techniques do not seem adequate.

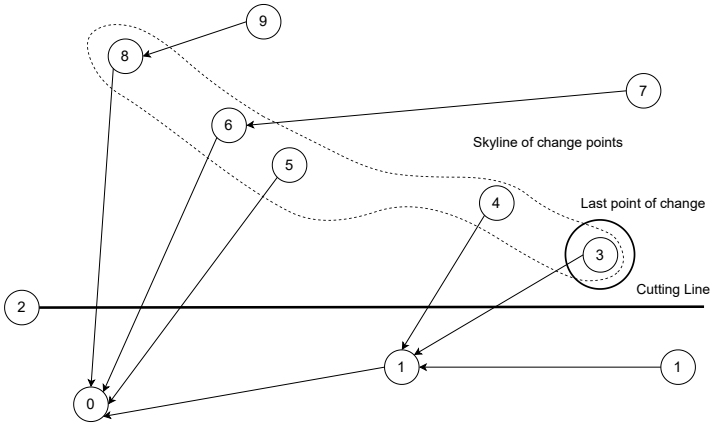


Figure 1 Geometric interpretation of the change in pred encoding of 0869514371 when prepended with 2. The cutting line corresponds to the prepended character.

Our approach. Even though many positions change, and they cannot be explicitly stored, the structural properties of this problem show that the *last point of change* (the rightmost value which changes) during LF is what matters. In the example above, the rightmost character which changes its encoding is 3 and its encoding changes from 2 to 8. The good part is that once we know this, we can deterministically pinpoint which other previous (to the left) locations changed their encoding (we present this formally as Lemma 5). Thus, we can register/store one particular value and all previous changes can be captured based on that. Yet this only gives us existential dependency and not an algorithmic tool.

1.3 Our Contribution

The existing results on compressed text indexing for order-isomorphic pattern matching are partial and conditional. For example, the $O(n \log\log n)$-bit by Gagie et al. [8] can answer only counting queries, that too for short patterns of size $O(\log^{O(1)} n)$. Another result by Decaroli et al. [6] is based on heuristics. We show:

▶ **Theorem 2.** *Let $T[1,n]$ be a given text over an ordered alphabet Σ of size σ. We can encode its order-isomorphic suffix array $\mathsf{SA}[1,n]$ in $O(n \log \sigma)$ bits and answer both $\mathsf{SA}[\cdot]$ and $\mathsf{SA}^{-1}[\cdot]$ queries in $t_{\mathsf{SA}} = O(\log^2 n / \log \sigma)$ time.*

Using the standard binary search algorithm, we can easily answer counting queries in time $O((|P| \log \sigma + t_{\mathsf{SA}}) \log n)$ and then reporting in time t_{SA} per match. At the heart of proving Theorem 2 lies a novel way of implementing LF mapping. We call this as LF Successor. It goes one step beyond the current approach of simulating *Suffix Array using LF mapping*.

2 Structural Properties of the Order Isomorphic Suffixes

In this section we introduce two key lemmas explaining the structural properties of the pred encoding. In other words, we see where the changes occur when a new character is prepended to the suffix. Firstly, we formally define a *change point* as follows,

▶ **Definition 3** (Change Point). *Given a string $T[r,z]$ along with its* pred *encoding* $\mathsf{pred}(T[r,z])$, *point $i \in [r,z]$ is a change point if* $\mathsf{pred}(T[r-1,z])[i-r+2] \neq \mathsf{pred}(T[r,z])[i-r+1]$.

In other words, when a character is prepended to $T[r, z]$ (making it $T[r-1, z]$) the encoding of the character $T[i]$ changes. Here point i means position in the text.

▶ **Definition 4** (Skyline). *A point j in text substring $T[r, z]$ covers a point i iff $j < i$ and $T[j] \leq T[i]$. γ-skyline of $T[r, z]$ is set of all points $i \in [r, z]$ such that $T[i] \geq \gamma$ and i is not covered by any point $j \in [r, i-1]$ such that $T[j] \geq \gamma$. When $\gamma = T[r-1]$, we simply refer to this as skyline of $T[r, z]$. Given a point $d \in T[r, z]$, the skyline induced by d is same as $T[d]$-skyline of $T[r, z]$ (i.e., the one obtained by setting $\gamma = T[d]$).*

Lemma 5 proves that all the change points of $T[r, z]$ are exactly the ones that are on the *skyline* (See Figure 1 for geometric interpretation). Secondly, as mentioned earlier, although there are many change points in the order isomorphic setting, given the rightmost or last change point we can uniquely determine all the previous change points. More formally, it can be stated as follows.

▶ **Lemma 5** (Skyline Lemma). *Given a text substring $T[r, z]$ and it rightmost change point d of the substring, all the change points in $T[r, z]$ can be determined based on d. These are precisely the points in $T[d]$-skyline of $T[r, z]$.*

Proof. Firstly, let's consider any change point $i \in T[r, z]$. Since its pred-encoding changes due to prepending of $T[r-1]$ the new predecessor of point i in $T[r-1, z]$ must be $r-1$ (i.e., $\mathsf{pred}(i) = i - r + 1$). This means $T[i] \geq T[r-1]$. Also if point i was covered by point j such that $j < i$ and $T[j] \geq T[r-1]$, then predecessor of i in $T[r-1, z]$ would still be j.

For the other way around, consider a point i on the skyline of $T[r, z]$. The predecessor of i in $T[r, z]$ cannot be a point j such that $T[j] \geq T[r-1]$ (by definition of cover), Thus, when $T[r-1]$ is prepended, it will become the new predecessor of i. Hence, i is a change point.

Now, if d is the rightmost change point of T then no character value in $T[r-1, z]$ is in between $T[r-1]$ and $T[z]$. That is, there is no $i \in [r, d-1]$ such that $T[r-1] \leq T[i] \leq T[d]$. Thus, this indeed is the same as $T[d]$-skyline. Also, since there are no change points after d they will not be on the skyline. ◀

Next, given two suffixes and their last common change points, all their previous change points will be the same. We state this as a lemma below. Here we define $\mathsf{rank}(x, T[r, z])$ as the number of values in $T[r, z]$ that are less than or equal to x. It follows from the definition of rank that if there are two order isomophic substrings $T[r, r+l-1]$ and $T[s, s+l-1]$, then for any point $1 \leq d \leq l$, $\mathsf{rank}(T[r+d-1], T[r, r+l-1] = \mathsf{rank}(T[s+d-1], T[s, s+l-1]$.

▶ **Lemma 6** (Last Common Point of Change (LCPC) Lemma). *Given two text substrings $T[r, r+l-1]$ and $T[s, s+l-1]$ such that $\mathsf{pred}(T[r, r+l-1]) = \mathsf{pred}(T[s, s+l-1])$, let d be the greatest value such that $r+d-1$ and $s+d-1$ are the change points in $T[r, r+l-1]$ and $T[s, s+l-1]$ respectively. Thus, the dth point is the last common change point of substrings $T[r, r+l-1]$ and $T[s, s+l-1]$. Then for every $e \in [1, d-1]$, $r+e-1$ is a change point in $T[r, r+l-1]$ if and only if $s+e-1$ is a change point in $T[s, s+l-1]$.*

Proof. Firstly, w.l.o.g, let $\mathsf{rank}(T[r-1], T[r, r+l-1]) < \mathsf{rank}(T[s-1], T[s, s+l-1])$. Now, there is no point p such that $1 < p < d$ and $\mathsf{rank}(T[r-1], T[r, r+l-1]) < \mathsf{rank}(T[r+p-1], T[r, r+l-1]) < \mathsf{rank}(T[s-1], T[s, s+l-1])$. This is because if there was such a point p, then d cannot be a change point of $T[r, r+l-1]$, because d will be covered by point p. Secondly, if $e \in [1, d-1]$ is a change point of $T[r, r+l-1]$ and suppose q was the predecessor of e before prepending of the new point $T[r-1]$, then $\mathsf{rank}(T[r+q+1], T[r, r+l-1]) < \mathsf{rank}(T[r-1], T[r, r+l-1]) < \mathsf{rank}(T[r+e+1], T[r, r+l-1])$. Therefore, we can say that

$\mathsf{rank}(T[r+q+1], T[r, r+l-1]) < \mathsf{rank}(T[r-1], T[r, r+l-1]) < \mathsf{rank}(T[s-1], T[s, s+l-1]) < \mathsf{rank}(T[r+e+1], T[r, r+l-1])$. Here if we just consider the ranking orders of $T[s, s+l-1]$, then $\mathsf{rank}(T[s+q+1], T[s, s+l-1]) < \mathsf{rank}(T[s-1], T[s, s+l-1]) < \mathsf{rank}(T[s+e+1], T[s, s+l-1])$ because $\mathsf{pred}(T[r, r+l-1]) = \mathsf{pred}(T[s, s+l-1])$. This implies that $T[s-1]$ is the new predecessor of $T[s+e+1]$, which means e is also a change point of $T[s, s+l-1]$.

The encoding of characters which are not change points will stay the same in $\mathsf{pred}(T[r-1, r+d-1])$ and $\mathsf{pred}(T[s-1, s+d-1])$. On the characters which are change points, their $\mathsf{pred}(\cdot)$ values point to $T[r-1]$ (resp. $T[s-1]$). Since pred encodes distance to the predecessor character, these pred values will be the same for corresponding change points in $T[r-1, r+d-1]$ and $T[s-1, s+d-1]$. Thus, $\mathsf{pred}(\cdot)$ encoding for both agree up to the first $d+1$ characters. ◀

As an example of LCPC, consider two substrings $T[r-1, r+6] = \texttt{dkgcihfe}$ and $T[s-1, s+6] = \texttt{ckfaihdb}$. Now, $T[r, r+6]$ and $T[s, s+6]$ are both order-isomorphic with prev encoding of 0002334 (here we follow the usual ordering of English characters). Considering, the previous values $T[r-1]$ and $T[s-1]$, the change points for $T[r, r+6]$ are $\texttt{k}, \texttt{g}, \texttt{f}, \texttt{e}$ at locations r plus $\{0, 1, 5, 6\}$ within string $T[r, r+6]$ and the change points for $T[s, s+6]$ are $\texttt{k}, \texttt{f}, \texttt{d}$ at locations s plus $\{0, 1, 5\}$ within string $[s, s+6]$. Thus, last common point of change (LCPC) is at location 6 i.e., $T[r+5]$ and $T[s+5]$. Note that $T[r+6]$ is also a change point however it is not a common change point. Also, note that since all the earlier change points in both the strings are at same locations $\{1, 2\}$. All the common change points at locations $\{0, 1, 5\}$ are indeed skylines of $T[r, r+5]$ as well as that of $T[s, s+5]$. Alternatively, these skylines are indeed $T[r+5]$-skyline of $T[r, r+6]$ and $T[s+5]$-skyline of $T[s, s+6]$.

3 LF Successor and Order-Isomorphic Text Indexing

Recall our encoding scheme pred (Definition 1) and the lexicographic order of encoded symbols: $0 < 1 < 1' < 2 < 2' < \cdots < n-1 < (n-1)' < \$$. We will now introduce a few more terminologies related to the order-isomorphic suffix tree (ST). We shall refer to any character on any substring representing an edge label as a "point" in ST. An edge is labeled by a substring represented by that edge in ST. For any point c in ST, let $\mathsf{path}(c)$ denote the concatenation of labels from the root until c. We shall denote $\mathsf{char}(c)$ as an (pred encoded) character represented by point c. We will also refer to nodes in ST as points. In this case, the node will be represented by the character just above it (i.e., the last character of the label of its parent edge). For any point c, $\mathsf{depth}(c)$ is length of $\mathsf{path}(c)$ and $\alpha\mathsf{Depth}(c)$ = number of distinct symbols in $T[r, r+\mathsf{depth}(c)-1]$, where $T[r, n]$ is any suffix passing through c. Note that this $\alpha\mathsf{Depth}$ indeed refers back to the original text instead of encoded text (in terms of encoded text this would be the number of non-primed characters). We call this the alphabet depth of point c. We shall generalize this notion as alphabet length for any string S as $\alpha(S)$ = number of unique alphabet symbols in S. For any two suffixes i and j (i.e., suffixes corresponding to leaves ℓ_i and ℓ_j), let point $v = \mathsf{lca}(i, j)$ be the lowest common ancestor (LCA) of ℓ_i and ℓ_j. Then, the length of their longest common prefix, denoted by $\mathsf{LCP}(i, j)$ is $\mathsf{depth}(v)$. Also $\alpha\mathsf{LCP}(i, j) = \alpha\mathsf{Depth}(v)$.

The locus of a pattern P is the highest node u such that $\mathsf{pred}(P)$ is a prefix of $\mathsf{path}(u)$. Every leaf ℓ_i in the sub-tree of u corresponds to an occurrence of P at a position in T given by $\mathsf{SA}[i]$. Let $[sp, ep]$ be the suffix range of P, where ℓ_{sp} (resp. ℓ_{ep}) is the leftmost (resp. rightmost) suffix in the subtree of u. We note that in order to support pattern matching, we need to **(a)** compute the suffix range $[sp, ep]$ of P and **(b)** decode suffix array values $\mathsf{SA}[i]$, $i \in [sp, ep]$. Using a standard binary search on the suffix array along with the text, we can

find the suffix range. Storing $\mathsf{SA}[i]$ for every leaf ℓ_i is too costly as it will take $\Theta(n \log n)$ bits. The goal is to encode suffix array values in compact space so that they can be decoded efficiently. We show how to achieve this using a *sampled suffix array* and *LF mapping*.

Recall that LF mapping is defined as: $j = \mathsf{LF}(i)$ iff $\mathsf{SA}[j] = \mathsf{SA}[i] - 1$. We explicitly store $\mathsf{SA}[\cdot]$ values belonging to the set $\{1, 1+\Delta, 1+2\Delta, \ldots, n\}$, where Δ is a tunable parameter to be set later. For any suffix i, where $\mathsf{SA}[i]$ has not been stored, we repeatedly apply LF mapping operation (starting from i) until we reach j such that $\mathsf{SA}[j]$ has been sampled. Then, $\mathsf{SA}[i] = \mathsf{SA}[j] + k$, where k is the number of LF operations applied; note that $k \leq \Delta$. Thus, we have reduced the problem to that of computing $\mathsf{LF}(\cdot)$. For SA^{-1} queries, we store $\mathsf{SA}^{-1}[i]$ if i equals n or if i is a multiple of Δ. To compute $\mathsf{SA}^{-1}[j]$, we first find the smallest number $j' \geq j$, such that j' is a multiple of Δ (or $j' = n$). Compute $j'' = \mathsf{SA}^{-1}[j']$ from sampled-SA^{-1} in $O(1)$ time. Let $k = j' - j$. Starting from j'' carry out k successive LF operations and report the final index as $\mathsf{SA}^{-1}[j]$.

To compute LF, we introduce *LF successor*, defined as:

i' is called the LF-successor of i iff $\mathsf{LF}(i') = \mathsf{LF}(i) + 1$

We denote it as $i' = \mathsf{LFS}(i)$. Throughout this paper, we use i' to denote $\mathsf{LFS}(i)$ for any suffix i. Thus, the leaves ℓ_i and $\ell_{i'}$ are mapped by using LF operation to leaves ℓ_j and ℓ_{j+1} respectively. To compute LF mapping, we again use a sampling technique. Specifically, we explicitly store $\mathsf{LF}(\cdot)$ values in the set $\{1, 1+\Delta, 1+2\Delta, \ldots, n\}$, thereby reducing the problem of computing LF mapping to that of computing at most Δ number of LF successors. In Section 4, we show how to compute LF successor in time $t_{\mathsf{LFS}} = O(\log \sigma)$ by using an $O(n \log \sigma)$-bit index. Therefore, LF can be computed in time $t_{\mathsf{LF}} = \Delta \cdot t_{\mathsf{LFS}}$ and $t_{\mathsf{SA}} = O(\Delta \cdot t_{\mathsf{LF}})$. Theorem 2 follows immediately by fixing $\Delta = \log_\sigma n$.

4 Computing LF Successor in Time $O(\log \sigma)$ Using Compact Space

In this section, we shall describe what additional information should be augmented to each leaf of the suffix tree, so that given ith leaf ℓ_i, we can quickly identify which leaf is its LF successor $\mathsf{LFS}(i)$. We shall first describe the data structure and then the query algorithm for computing $\mathsf{LFS}(i)$. We saw earlier that we will be writing SA values and LF values only for n/Δ positions. Thus, this takes $O(n \log \sigma)$-bit space by choosing $\Delta = \log_\sigma n$. What remains to be seen is how to compute LF successor for a given suffix associated with the leaf ℓ_i. If we explicitly write it at all the leaves, it will take $\Theta(\log n)$ bits per leaf. Since there is no sampling here, this will lead to $\Theta(n \log n)$ bits which will defeat our purpose. Our approach is to store only $O(\log \sigma)$ bits per leaf and yet be able to compute the LF successor quickly.

4.1 Four Cases for Suffix and its LF Successor

For the discourse in this section, we use the following terminology. Let the starting position in the text for suffix denoted by leaf ℓ_i be r (i.e, $r = \mathsf{SA}[i]$), and that of $\ell_{i'}$ be r', where $i' = \mathsf{LFS}(i)$. Let $d = |\mathsf{LCP}(i, i')|$, the length of the longest common prefix of $\mathsf{pred}(T[r, n])$ and $\mathsf{pred}(T[r', n])$. Thus, $T[r, r+d-1]$ and $T[r', r'+d-1]$ are order isomorphic. Inevitably, we will also focus on suffixes $\mathsf{LF}(i)$ and $\mathsf{LF}(i')$ which are encodings of text suffixes $T[r-1, n]$ and $T[r'-1, n]$ respectively.

Now, we distinguish two cases with respect to leaf ℓ_i (and its LF successor $\ell_{i'}$) – case (1) if $T[r-1, r+d-1]$ is not order isomorphic with $T[r'-1, r'+d-1]$, and case (2) $T[r-1, r+d-1]$ is order isomorphic with $T[r'-1, r'+d-1]$ i.e., prepending of character $T[r-1]$ (resp., $T[r'-1]$) to the left still maintains order-isomorphism until the LCP i.e., $\mathsf{pred}(T[r-1, r+d-1]) = \mathsf{pred}(T[r'-1, r'+d-1])$.

First, we shall talk about case (1). Let us consider all the change points of $T[r, r+d-1]$ and $T[r', r'+d-1]$. Let e be their last common change point. If $T[r'-1] \neq T[r'+e-1]$ then we call it case (1a) - the *breakaway case*. Else, we call it case (1b) - the *equality case*. In case (1a), let g be the first change point after e for $T[r', r'+d-1]$. To proceed to case (2), we now define LF-image, which generalizes the concept of Wiener links.

▶ **Definition 7** (LF-image). *Let c be any point in the suffix tree and point p above c be such that for at least one of the suffixes $T[r, n]$ passing through c, p is the last change point before c. The LF-image of c with respect to a change point p, denoted by $\mathsf{LF}(c, p, EQBT)$ is a point representing the position of* (pred *encoding of*) $T[r-1, r+\mathsf{depth}(c)-1]$. *EQBT is called the equality bit and is set to* 1 *if p is an equality change point and* 0 *otherwise.*

For any such suffix i passing through c with change point p being the last one above c, $\mathsf{LF}(i)$ passes through $\mathsf{LF}(c, p, \text{EQBT})$. So if $q = \mathsf{LF}(c, p, \text{EQBT})$, $\mathsf{path}(q) = \mathsf{pred}(T[r-1, r+\mathsf{depth}(c)-1])$. Note that the same point c can have multiple LF-images based on which change point above c is taken as the last one and also if that is equality change point or not.

If leaf ℓ_i falls under case 2, we shall again break this case into cases (2a) and (2b). In case (2a) we consider $i < i'$ (we call this *ordered case*) and in case (2b) we consider $i' < i$ (we call this *inverting case*). We say that a suffix l *inverts* over suffix k iff $l < k$ and $\mathsf{LF}(l) > \mathsf{LF}(k)$.

▶ **Lemma 8.** *If suffixes i and $i' = \mathsf{LFS}(i)$ fall in case 2 then they have the same change points (and also the same type of change points - equality or not) until $\mathsf{lca}(i, i')$. Then i cannot have a change point immediately after $\mathsf{lca}(i, i')$. Moreover, if they fall in case (2b) then i' must have a change point immediately after $\mathsf{lca}(i, i')$.*

Proof. Let the point $c = \mathsf{lca}(i, i')$. For case (2) we know that $T[r-1, r+d-1]$ is order isomorphic with $T[r'-1, r'+d-1]$ i.e. $\mathsf{pred}(T[r-1, r+d-1]) = \mathsf{pred}(T[r'-1, r'+d-1])$. This means that i and i' have all the same change points until c.

Now let p be the last common change point of i and i' i.e. $p = \mathsf{LCPC}(i)$. Here, $\mathsf{LCPC}(i)$ denotes the last common point of change of i and its LFS i'. Additionally, suppose $b = \mathsf{LF}(c, p, \text{EQBT})$. So this means that $\mathsf{LF}(i)$ and $\mathsf{LF}(i')$ will pass through b. As per the definition of LF successor we know that, $\mathsf{LF}(i) < \mathsf{LF}(i')$. More specifically, $\mathsf{LF}(i') = \mathsf{LF}(i) + 1$.

Firstly, lets say that i has a change point right after c. It is easy to observe from the structural properties of order-isomorphic suffixes that both i and i' cannot change immediately after c. Hence, if we see all the branches under b, then $\mathsf{LF}(i)$ will fall under the rightmost branch (or just previous branch depending on whether that change point is of equality type or not). This leads to $\mathsf{LF}(i') < \mathsf{LF}(i)$ which is not possible as per the definition of LF successor. Thus, i cannot have a change point immediately after c.

Now, if we take the case (2b), then i' inverts over i because $\mathsf{LF}(i')$ must be greater than $\mathsf{LF}(i)$. For this to happen i' must have a change point immediately after c. ◀

The proof of the lemma above also leads us to the following fact.

▶ **Fact 1.** *Let c be a point immediately above a node v. Let $b = \mathsf{LF}(c, p, EQBT)$, where p is the last common change point (of type equality or non-equality) on $\mathsf{path}(c)$ for two case (2b) suffixes i and $i' = \mathsf{LFS}(i)$ passing through c. Then, i' has a change point immediately after v. Moreover, there cannot be another pair of case (2b) suffixes j and $j' = \mathsf{LFS}(j)$, which have the same last common point of change p, and j' changes immediately after v.*

Proof. If any two of the suffixes i' and j', where $i' = \mathsf{LFS}(i)$ and $j' = \mathsf{LFS}(j)$, passing through v have a change point right after the node v and their last common change point is p, then under the point $b = \mathsf{LF}(c, p, \text{EQBT})$ only one of their LF values (either $\mathsf{LF}(i')$ or $\mathsf{LF}(j')$) can be next to their respective $\mathsf{LF}(i)$ or $\mathsf{LF}(j)$. That implies only one of either $\mathsf{LF}(i') = \mathsf{LF}(i) + 1$ or $\mathsf{LF}(j') = \mathsf{LF}(j) + 1$ can be true. This is a contradiction, implying the fact is true. ◀

4.2 Storing Augmenting Information for each Leaf

We shall describe this section in terms of augmenting information stored with each leaf. However, one can easily see them as arrays that run parallel to the suffix array. We shall show that each of these augmenting fields in all the cases can be stored in $O(\log \sigma)$ bits. For each leaf ℓ_i, we can write in 2 bits which of the above 4 cases it belongs to. We denote this by $\mathsf{CASE}[i]$. We also store the same value with i' and in this case we shall call it $\overline{\mathsf{CASE}}[i']$.

If ℓ_i belongs to case (1b), then we intend to store e which we will denote as $\mathsf{LCPC}[i] = e$. Recall that e is defined as the rightmost (maximum value) common change point for $T[r, r+d-1]$ and $T[r', r'+d-1]$, and LCPC stands for *last common point of change*. Thus, LCPC is an array whose ith entry corresponds to leaf ℓ_i. However, storing the value e directly will require $\log n$ bits. Therefore, instead of e, we store number of distinct alphabet symbols in $T[r, r+e-1]$) (i.e., $\alpha(T[r, r+e-1])$). We will call this value $\alpha\mathsf{LCPC}[i]$. It is worth noting that since change points only occur at new (first occurrence) alphabets in the string, e can be uniquely decoded from $\alpha\mathsf{LCPC}$. We also store a complementary array of $\alpha\mathsf{LCPC}$ denoted as $\overline{\alpha\mathsf{LCPC}}$ such that $\overline{\alpha\mathsf{LCPC}}[i'] = \alpha\mathsf{LCPC}[i]$. Thus, this value is not only stored with leaf i but also replicated in leaf $i' = \mathsf{LFS}(i)$ - albeit under a differently named field.

Recall that for case (1a), g is the first change point after e for $T[r', r'+d-1]$. For the case (1a), we store g which we call the first point of break $\mathsf{FPB}[i]$. Again, we will not store the value g directly but an encoding $\alpha(T[r, r+g-1])$ which takes $\log \sigma$ bits. We will call this value $\alpha\mathsf{FPB}[i]$. Similarly, we store this value with i' as $\overline{\alpha\mathsf{FPB}}[i'] = \alpha\mathsf{FPB}[i]$.

For the case (2a), we maintain $\alpha\mathsf{LCPC}$ and $\overline{\alpha\mathsf{LCPC}}$ as in case (1b). We also maintain an extra-bit EQBT indicating which type of change point LCPC is - whether equality change point (indicated by EQBT = 1) or not. Similarly, we also store $\overline{\mathrm{EQBT}}$. We also store $\alpha(T[r, r+d-1])$ that is the number of distinct alphabet symbol occurring until $\mathsf{LCP}(i, i')$. We shall call it $\alpha\mathsf{LCP}[i]$. Again, we store the same value at leaf $\ell_{i'}$ so that $\overline{\alpha\mathsf{LCP}}[i'] = \alpha\mathsf{LCP}[i]$. Additionally to this, we store $\mathsf{FPC}[i]$ (read as *first point of change post LCA*) which in this case will be defined as the first change point of $T[r, n]$ after $T[r+d-1]$. Note that this point of change cannot be right after LCA at $T[r+d]$ because otherwise i will invert over i' (this would then be case (2b) Lemma 8) during LF mapping operation and $\mathsf{LF}(i)$ will be greater than $\mathsf{LF}(i')$. Once again we define $\overline{\mathsf{FPC}}[i'] = \mathsf{FPC}[i]$ and define $\alpha\mathsf{FPC}[i]$ and $\overline{\alpha\mathsf{FPC}}[i']$ in similar vein. In summary, we maintain $\alpha\mathsf{LCPC}, \mathrm{EQBT}, \alpha\mathsf{LCP}$ and $\alpha\mathsf{FPC}$ for each such leaf which falls in case (2a). We also store these values at their corresponding LF successors. One point to note here is that $\mathsf{FPC}, \mathsf{LCPC}, \mathsf{FPB}$ are all uniquely decodable from $\alpha\mathsf{FPC}, \alpha\mathsf{LCPC}, \alpha\mathsf{FPB}$ since they necessarily fall on the new alphabet which is yet unseen in the suffix. However, the same is not true of $\alpha\mathsf{LCP}$.

As an example, let us look at $T[r-1, n] = \texttt{caghhfbab}$... and $T[r'-1, n] = \texttt{cagjjebae}$.... Then, $\mathsf{pred}(T[r, n]) = \texttt{0111'456'2'}$... and $\mathsf{pred}(T[r', n]) = \texttt{0111'456'3'}$.... Their LCPC is at depth 5 which is encoded as 4 in the encodings of both the suffixes. Their $\alpha\mathsf{LCPC} = 4$, since there are 4 distinct alphabets in both the strings until that point (4 non-prime characters in their pred encoding). Length of their $\mathsf{LCP} = 7$, however the character $\texttt{a}$ which occurs their as encoded character $\texttt{6'}$ is not a new character. Hence, $\alpha\mathsf{LCP} = 5$ which points to character $\texttt{b}$ in both the original strings. If we try to decode $\alpha\mathsf{LCP}$, it will lead us to position 6 rather than 7. Finally, after the LF mapping, the encoded strings are $\texttt{00211'556'2'}$ and $\texttt{00211'556'3'}$.

For case (2b), our solution is more intricate so we only give a brief overview and defer details to Case (2b) section of the proof of correctness. In this case, i' inverts over i. Thus, i' has a change point right after the $\mathsf{lca}(i, i')$ at $T[r'+d]$. Just storing additional augmenting values to the leaves of the suffix tree is not sufficient. Like before, we shall store $\alpha\mathsf{LCPC}$ and $\alpha\mathsf{LCP}$ values. But we shall construct additional data structures called mini-trees and search for i' in an appropriate mini-tree identified by $\alpha\mathsf{LCPC}$ and $\alpha\mathsf{LCP}$ values of i. We will denote this mini-tree as $\tau_{\alpha\mathsf{LCPC}[i], \alpha\mathsf{LCP}[i]}$.

4.3 Query Algorithm

Now, we outline the pseudo-code for our query algorithm.

Computing $\mathsf{LFS}(i)$

- If ℓ_i falls in case (1a),
  $\ell_{i'}$ is the unique leaf under u s.t. $\overline{\mathsf{CASE}}[i'] = \mathsf{CASE}[i]$ and $\overline{\alpha\mathsf{FPB}}[i'] = \alpha\mathsf{FPB}[i]$, where u is the highest ancestor of ℓ_i with $\alpha\mathsf{Depth}(u) \geq \alpha\mathsf{FPB}[i]$
- ElseIf ℓ_i falls in case (1b)
  $\ell_{i'}$ is unique leaf under u s.t. $\overline{\mathsf{CASE}}[i'] = \mathsf{CASE}[i]$ and $\overline{\alpha\mathsf{LCPC}}[i'] = \alpha\mathsf{LCPC}[i]$, where u is the highest ancestor of ℓ_i with $\alpha\mathsf{Depth}(u) \geq \alpha\mathsf{LCPC}[i]$
- ElseIf ℓ_i falls in case (2a)
  Let c = point above $\mathsf{FPC}[i]$ on suffix $T[r, n]$ in the suffix tree. Then $\ell_{i'}$ is leftmost leaf after ℓ_i in the (subtree of $\alpha\mathsf{LCP}[i]$) $\setminus$ (subtree of c) s.t. $\overline{\mathsf{CASE}}[i'] = \mathsf{CASE}[i]$, $\alpha\mathsf{LCP}[i] = \overline{\alpha\mathsf{LCP}}[i']$, $\alpha\mathsf{LCPC}[i] = \overline{\alpha\mathsf{LCPC}}[i']$ and $\mathrm{EQBT}[i] = \overline{\mathrm{EQBT}}[i']$
- Else
  $i' = \mathsf{findSucc}\big(i, \alpha\mathsf{LCPC}[i], \alpha\mathsf{LCP}[i]\big)$, which is to be defined later

Note that all the arrays mentioned above can be represented in $O(n \log \sigma)$ bits, and the implementation uses standard succinct-data-structure techniques (see Section 4.5); the difficulty lies in proving the correctness of the algorithm, which is our focus next.

4.4 Proofs of Correctness

We shall show correctness of each case. In each case, we need to ensure that we would not end up with a wrong answer. This could happen if there is another pair j, j' such that $j' = \mathsf{LFS}(j)$ and this pair shares the same characteristics with the pair i, i'. In this case, pair j, j' may interfere in the search for i' leading to false answer j'.

4.4.1 Case (1a)

Let c be the first point (the character within an edge of ST) on $\mathsf{path}(\ell_i)$ such that $T[r, r + \mathsf{depth}(c) - 1]$ has exactly $\alpha\mathsf{FPB}[i]$ distinct characters. Thus, this is the first (encoded) character where $\mathsf{pred}(T[r-1, n])$ and $\mathsf{pred}(T[r'-1, n])$ differ; in other words, $\mathsf{path}(\ell_{\mathsf{LF}(i)})$ and $\mathsf{path}(\ell_{\mathsf{LF}(i')})$ bifurcate at the position given by $\mathsf{depth}(c) + 1$. Let $\hat{c}$ be the point in ST such that $\mathsf{path}(\hat{c}) = \mathsf{pred}(T[r-1, r + \mathsf{depth}(c) - 1])$ and $\hat{c'}$ be such that $\mathsf{path}(\hat{c'}) = \mathsf{pred}(T[r'-1, r' + \mathsf{depth}(c) - 1])$. These points are on sibling edges going down from the same node. Let v be the node just above $\hat{c}$ and $\hat{c'}$. For example, consider $T[r-1, n] = \texttt{jeabdh}...$ and $T[r'-1, n] = \texttt{gfabdh}....$ Then, $\mathsf{path}(c) = \mathsf{pred}(\texttt{eabdh}) = \mathsf{pred}(\texttt{fabdh}) = \texttt{00114}$. This makes $\mathsf{path}(\hat{c}) = \mathsf{pred}(\texttt{jeabdh}) = \texttt{000114}$. However, $\mathsf{path}(\hat{c'}) = \mathsf{pred}(\texttt{gfabdh}) = \texttt{000115}$. Note that $\texttt{5}$ is the highest encoded character (with an exception of $\texttt{5}'$) which branches out of the node v.

▶ **Lemma 9.** *There is only one pair of leaves i, i' in the subtree of c, such that $\alpha\mathsf{FPB}[i] = \overline{\alpha\mathsf{FPB}}[i'] = \alpha(T[r, r + \mathsf{depth}(c) - 1])$.*

Proof. Consider LF mapping of i and i'. $\mathsf{path}(\ell_{\mathsf{LF}(i)})$ and $\mathsf{path}(\ell_{\mathsf{LF}(i')})$ first bifurcate at points $\hat{c}$ and $\hat{c'}$ respectively. Since $i' = \mathsf{LFS}(i)$, $\mathsf{char}(\hat{c}) < \mathsf{char}(\hat{c'})$. Moreover, $\mathsf{char}(\hat{c'})$ is precisely $\mathsf{depth}(c)$ or its equality version i.e. $(\mathsf{depth}(c))'$. This is the highest (encoded) character, and thus the branch with $\hat{c'}$ will be one of the two rightmost branches among branches (depending on whether the change point c for suffix i' was based on "equality" or not). However, the point $\hat{c}$ will certainly be before the two rightmost branches at v. If there was any other pair j and j' of case (1a) under the subtree of c such that $j' = \mathsf{LFS}(j)$ and $\mathsf{FPB}(j) = \mathsf{FPB}(i)$, then both $\mathsf{LF}(i')$ and $\mathsf{LF}(j')$ will fall under the subtree of $\hat{c'}$ because as per the LCPC lemma all the change points of i' and j' are the same until c (including c). On the contrary, $\mathsf{LF}(i)$ and $\mathsf{LF}(j)$ cannot fall under this subtree as they are under the subtree of $\hat{c}$. Thus, depending on whether $\mathsf{LF}(i') < \mathsf{LF}(j')$ or not, only one pair out of $(\mathsf{LF}(i), \mathsf{LF}(i'))$ or $(\mathsf{LF}(j), \mathsf{LF}(j'))$ can be adjacent. Since, i' is indeed the LF successor of i, such a pair j, j' cannot exist. ◀

4.4.2 Case (1b)

Let c be the first point in ST on $\mathsf{path}(\ell_i)$ such that $T[r, r + \mathsf{depth}(c) - 1]$ has $\alpha\mathsf{LCPC}[i]$ distinct characters. In this case, c is a change point for both i and i'. For i', it is the equality change point while for i it is not (i.e., $T[r' - 1] = T[r' + \mathsf{depth}(c) - 1]$ and $T[r - 1] \neq T[r + \mathsf{depth}(c) - 1]$). Let point $\hat{c}$ correspond to $\mathsf{path}(T[r-1, r+\mathsf{depth}(c)-1])$ and $\hat{c'}$ correspond to $\mathsf{path}(T[r'-1, r'+\mathsf{depth}(c)-1])$. Let v be the node right above $\hat{c}$ (and also $\hat{c'}$) which can be identified by $\mathsf{path}(v) = T[r-1, r+\mathsf{depth}(c)-2]$. In this case, $\hat{c'}$ will fall in the rightmost branch at node v and $\hat{c}$ will fall in the branch previous to that. The character at point $\hat{c'}$ is precisely the equality (prime) version of the character at $\hat{c}$. For example, consider $T[r-1, n] = \mathtt{geabdh}$... and $T[r' - 1, n] = \mathtt{hfabdh}$.... Then, $\mathsf{path}(c) = \mathsf{pred}(\mathtt{eabdh}) = \mathsf{pred}(\mathtt{fabdh}) = \mathtt{00114}$. This makes $\mathsf{path}(\hat{c}) = \mathsf{pred}(\mathtt{geabdh}) = \mathtt{000115}$. However, $\mathsf{path}(\hat{c'}) = \mathsf{pred}(\mathtt{hfabdh}) = \mathtt{000115'}$. Here $\mathtt{5'}$ is the highest encoded character. Again, as in the case (1a), if there were any other pair j, j' falling in case (1b) under subtree of c such that $\mathsf{LCPC}(j) = \mathsf{LCPC}(i)$, then $\mathsf{LF}(j')$ will also fall in the rightmost branch at v while $\mathsf{LF}(j)$ will fall in the previous one. Again, by applying simple interval logic as in case (1a), we can show that only one of the pairs can satisfy the LF-successor definition.

4.4.3 Case (2a)

In this case, post $\mathsf{lca}(i, i')$, branch with ℓ_i is to the left of the branch with $\ell_{i'}$. Let c be the point just above $\mathsf{FPC}[i]$. Let ℓ_k be the rightmost leaf in the subtree of c. Note that since $\mathsf{FPC}[i]$ is not immediately after the $\mathsf{lca}(i, i')$, the subtree of c does not include i'. Therefore, the order between i and i' will not be inverted after taking LF mapping. Let f be the first point in ST on suffix $T[r, n]$ such that $\alpha(\mathsf{path}(f)) = \alpha\mathsf{LCP}[i]$. The actual $\mathsf{LCP}[i]$ will be somewhere in the subtree of f because $\mathsf{LCP}[i]$ is not uniquely decodable from $\alpha\mathsf{LCP}[i]$. Here $\mathsf{LCP}[i]$ denotes the $\mathsf{lcp}(i, i')$. Let j, j' be another pair in the subtree of f such that $j' = \mathsf{LFS}(j)$ and $\alpha\mathsf{LCP}[j] = \alpha\mathsf{LCP}[i]$ and $\mathsf{LCPC}[j] = \mathsf{LCPC}[i]$. All four leaves $\mathsf{LF}(i), \mathsf{LF}(i'), \mathsf{LF}(j), \mathsf{LF}(j')$ will be in the subtree of $\hat{f}$ which is the LF-image $\mathsf{LF}(f, \mathsf{LCPC}[i], \mathrm{EQBT})$. In other words, $\hat{f}$ is the locus of $\mathsf{pred}(T[r - 1, r + \mathsf{depth}(f) - 1])$ in ST.

▶ **Lemma 10.** *There does not exist a pair (j, j') such that $j' = \mathsf{LFS}(j)$, $\alpha\mathsf{LCP}[j] = \alpha\mathsf{LCP}[i]$, $\alpha\mathsf{LCPC}[j] = \alpha\mathsf{LCPC}[i]$ and j' lies in between k and i'.*

Proof. Consider any other pair j, j' in the subtree of f and with the same $\alpha\mathsf{LCPC}$, EQBT and $\alpha\mathsf{LCP}$ values such that $k < j' < i'$. We will show by contradiction that such a j' cannot exist. Firstly, since $i < k < j'$ and ℓ_k being the rightmost leaf in the subtree of c,

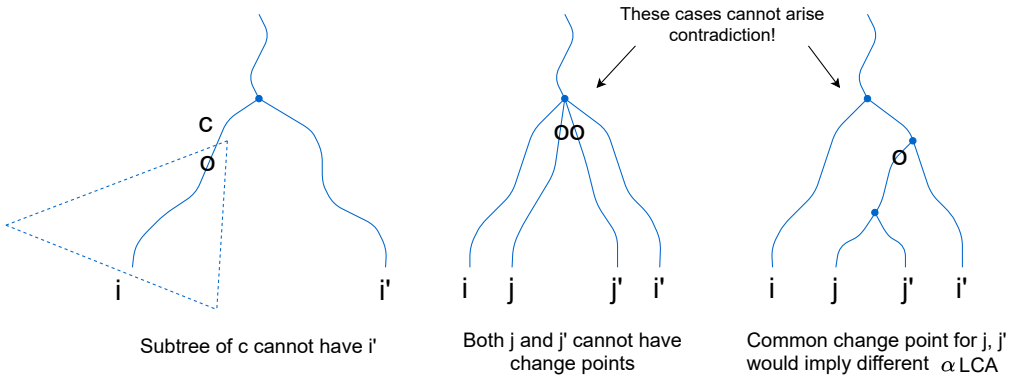


Figure 2 Illustration of case (2a). Small black circles are change points.

i cannot invert over j' after taking LF mapping. This is because c is the point just above $\mathsf{FPC}[i]$. Hence $\mathsf{LF}(i) < \mathsf{LF}(j')$. Also, since $\mathsf{LF}(i') = \mathsf{LF}(i) + 1$, $\mathsf{LF}(j')$ must be greater than $\mathsf{LF}(i')$. Secondly, the pair j, j' falls under case (2a) where $j < j'$ and $\mathsf{LF}(j) < \mathsf{LF}(j')$. Thus, $\mathsf{LF}(i') \leq \mathsf{LF}(j) < \mathsf{LF}(j')$ which means both j and j' invert over i' after LF operation.

Next, $j < j' < i'$ means $\mathsf{lca}(j, i')$ is equal to or above $\mathsf{lca}(j', i')$. Since j and j' invert over i', it must be at $\mathsf{lca}(j, i')$ and $\mathsf{lca}(j', i')$ respectively. If $\mathsf{lca}(j, i')$ is above $\mathsf{lca}(j', i')$, then j inverts above j' and it implies $\mathsf{LF}(j) > \mathsf{LF}(j')$ which is a contradiction. Now if $\mathsf{lca}(j, i') = \mathsf{lca}(j', i')$, then there are two cases. The first case is where j and j' invert from a common branch connecting path of i'. Here, j and j' will have a common change point at this branch which is post $\mathsf{lca}(j', i')$. It implies that there is another common change point for j, j' which leads to $\mathsf{LCPC}[j] > \mathsf{LCPC}[i]$ (a contradiction). In the second case, j and j' branch out at $\mathsf{lca}(j', i')$ but fall in different branches. However, according to Lemma 8, only one of j or j' can have a change point right after the $\mathsf{lca}(i, i')$. Hence, this case also leads to contradiction. Thus, j' does not lie in between k and i'. See Figures 2 and 3 for illustration. ◀

4.4.4 Case (2b)

For the case (2b), we know that suffix i' comes before suffix i in the suffix tree, i.e. $i' < i$. Additionally, for the case (2b), i' has a change point right after the node representing the $\mathsf{lca}(i, i')$. Moreover, under $\mathsf{lca}(i, i')$ the branch containing the suffix i' will be the only one that will have a change point tied with the same LCPC (see Fact 1). Since $i' = \mathsf{LFS}(i)$, after the LF mapping i' will invert over i making $\mathsf{LF}(i') = \mathsf{LF}(i) + 1$. See Figure 3.

As mentioned in Section 4.2, for the case (2b) we store $\alpha\mathsf{LCPC}[i]$ and $\alpha\mathsf{LCP}[i]$ values for each leaf ℓ_i as augmenting information. Additionally, we store their complements $\overline{\alpha\mathsf{LCPC}}[i']$ and $\overline{\alpha\mathsf{LCP}}[i']$ for each leaf $\ell_{i'}$. Now we consider an additional data structure called mini-trees that will help us in finding i' given i. Specifically, a particular mini-tree $\tau_{a,b}$ has set of all leaves ℓ_i and their corresponding LF successors $\ell_{i'}$ from the suffix tree that has $\alpha\mathsf{LCPC}[i] = \overline{\alpha\mathsf{LCPC}}[i'] = a$ and $\alpha\mathsf{LCP}[i] = \overline{\alpha\mathsf{LCP}}[i'] = b$. A particular leaf ℓ_i will not be in any mini-tree if that leaf does not fall under the case (2b). Thus, a leaf can be present in a mini-tree if it falls under case (2b) or it is an LF-successor of some other leaf which falls under the case (2b). Therefore, each leaf in the suffix tree will be in at most two mini-trees and some mini-trees are possibly empty. In other words, a mini-tree is a compacted subtrie of the suffix tree containing only those leaves selected for that mini-tree. Hence, overall size of all the mini-trees combined is $O(n)$.

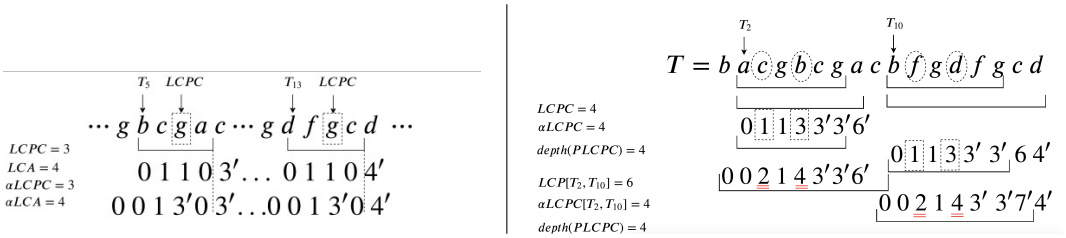


Figure 3 Illustration of case (2a) (left) and case (2b) (right). Red underline shows the character encoding that changes after taking LF.

To draw a correspondence between the leaves of the suffix tree and the leaves of the mini-trees, we use a bit-vector $B[1, n]$, where $B[i] = 1$ iff leaf i falls in case (2b) or leaf i is an LF-successor of the leaf which falls in case (2b). In other words, $B[i] = 1$ if a leaf from the suffix tree is present in at least one of the mini-trees, and $B[i] = 0$ otherwise. Next, we create two character vectors C and $\overline{C}$ as follows. If $B[i] = 0$, then $C[i] = \overline{C}[i] = 0$. Otherwise,

1. $C[i]$ stores an encoding of the pair $\alpha\mathsf{LCPC}[i], \alpha\mathsf{LCP}[i]$ as a combined character from an alphabet of size σ^2; essentially $C[i] = (\sigma - 1) \cdot \alpha\mathsf{LCPC}[i] + \alpha\mathsf{LCP}[i]$
2. $\overline{C}[i] = -C[i]$ if $\alpha\mathsf{LCPC}[i] = \overline{\alpha\mathsf{LCPC}}[i]$ and $\alpha\mathsf{LCP}[i] = \overline{\alpha\mathsf{LCP}}[i]$, and $\overline{C}[i] = (\sigma - 1) \cdot \overline{\alpha\mathsf{LCPC}}[i] + \overline{\alpha\mathsf{LCP}}[i]$ otherwise.

Now given a particular leaf ℓ_i in the suffix tree, for finding the corresponding leaf in the mini-tree, we first check if $B[i] = 1$. Since $a = \alpha\mathsf{LCPC}[i]$ and $b = \alpha\mathsf{LCP}[i]$, we can quickly identify the mini-tree $\tau_{a,b}$ it belongs to as augmenting information $\alpha\mathsf{LCPC}[i]$ and $\alpha\mathsf{LCP}[i]$ is stored for the leaf ℓ_i. To find out which leaf in $\tau_{a,b}$ corresponds to ℓ_i, all we have to do is figure out the number of leaves $j \leq i$ that satisfy $a = \alpha\mathsf{LCPC}[j] = a$ and $b = \alpha\mathsf{LCP}[j]$ or $\overline{\alpha\mathsf{LCPC}}[j] = a$ and $\overline{\alpha\mathsf{LCP}}[j] = b$; this is the same as the number of entries $j \leq i$ in the character vectors C such that $C[j] = C[i]$ plus the number of entries $k \leq i$ in the character vectors $\overline{C}$ such that $\overline{C}[k] = C[i]$. This is because the mini-tree is just a compacted subtrie of the original suffix tree consisting of only those leaves present in a particular mini-tree. To map a leaf from the mini-tree back to the leaf of the original suffix tree, we need to store a character vector for each mini-tree over the leaves of the mini-tree. Let $C_{a,b}$ be the character vector for the mini-tree $\tau_{a,b}$. This character array indicates whether the leaf has $a = \alpha\mathsf{LCPC}[i]$ and $b = \alpha\mathsf{LCP}[i]$ or $a = \overline{\alpha\mathsf{LCPC}}[i]$ and $b = \overline{\alpha\mathsf{LCP}}[i]$ or both. In other words, it simply specifies how the leaf was selected for that mini-tree using techniques similar to that described above. It is to be noted that all character vectors combined need $O(n \log \sigma)$ bits.

4.4.4.1 Identifying i'

We know that $\alpha\mathsf{LCPC}[i] = a$ and $\alpha\mathsf{LCP}[i] = b$. Let p_a be the first point in suffix tree where $\alpha(T[r + \mathsf{depth}(p_a) - 1])) = a$ and p_b be the first point such that $\alpha(T[r + \mathsf{depth}(p_b) - 1]) = b$. Thus, p_a and p_b are the points in suffix tree where $\alpha\mathsf{LCPC}[i]$ and $\alpha\mathsf{LCP}[i]$ are located. Note that p_a is above or the same as p_b. Now consider the mini-tree $\tau_{a,b}$. Let another pair j, j' where $j' = \mathsf{LFS}(j)$ fall under the same mini-tree (i.e., ℓ_j and ℓ'_j are also descendants of p_b and $\alpha\mathsf{LCPC}[j] = \alpha\mathsf{LCPC}[i]$ and $\alpha\mathsf{LCP}[j] = \alpha\mathsf{LCP}[i]$). Here j' will be on the left of j because they fall under the case (2b). We will focus here on searching i' as the first qualifying leaf to the left of i. Another pair j, j' could interfere with our process of searching if j' falls between i' and i. Formally, we say

▶ **Definition 11.** *A pair* j, j' interferes *with* i, i' *if* $i' < j' < i$ *and* $\alpha\mathsf{LCPC}[j] = \alpha\mathsf{LCPC}[i]$ *and* $\alpha\mathsf{LCP}[j] = \alpha\mathsf{LCP}[i]$. *Here,* $i' = \mathsf{LFS}(i)$ *and* $j' = \mathsf{LFS}(j)$

There are two cases of "interference" that can occur with respect to these two pairs – case (2b′) is where both j' and j are in between i' and i i.e. $i' < j' < j < i$ and case (2b*) where j is on the right of i i.e. $i' < j' < i < j$. As we know that $\alpha\mathsf{LCPC}[i] = \alpha\mathsf{LCPC}[j] = a$ and p_a is the first point in the suffix tree where $\alpha(T[r + \mathsf{depth}(p_a) - 1])) = a$. Suppose $x = \mathsf{LF}(\mathsf{lca}(i, i'), p_a, \text{EQBT})$ and $y = \mathsf{LF}(\mathsf{lca}(j, j'), p_a, \text{EQBT})$. Here EQBT is set to 1 if i' has an equality change point and 0 otherwise. Now in the case (2b′), after taking LF-mapping, j' inverts over j under y and i' inverts over all three of j, j', i under x – we call this the *nested case*. In case (2b*), j' and i both together (maintaining same order) invert over j under y and then i' inverts over all of them under x – we call this the *bulk-invert case*. Additionally, we will need to augment this mini-tree further so that we can distinguish the pair i, i' from the pair j, j'.

▶ **Lemma 12.** *If a pair* j, j' *interferes with* i, i'*, then* $\mathsf{lca}(i', i)$ *occurs above* $\mathsf{lca}(j', j)$ *in the suffix tree. Additionally, if* $i < j$*, then* $\mathsf{lca}(j', i)$ *is below* $\mathsf{lca}(j, j')$.

Proof. Note that in bulk invert case since j' and i both invert together over j, $\mathsf{lca}(j', i)$ must be below $\mathsf{lca}(j, j')$. Even though $\alpha\mathsf{LCP}[i] = \alpha\mathsf{LCP}[j]$, it cannot happen that LCAs of both the pairs are on the same node in the suffix tree (i.e. $\mathsf{lca}(i', i) = \mathsf{lca}(j', j)$). This is because from any node only one branch can have a change point at the next character below the node (see Fact 1). But we know that i' has a change point just below the node representing $\mathsf{lca}(i, i')$. Therefore, the branch containing j' cannot have a change point just below that node. This implies $j' \neq \mathsf{LFS}(j)$ since j falls under the case (2b). This holds a contradiction.

Therefore, for the case (2b′), it must be the case that $\mathsf{lca}(j', j)$ is below $\mathsf{lca}(i', i)$, implying that suffixes j' and j belong to the subtree at $\mathsf{lca}(i', i)$. In case (2b*), it cannot happen that $\mathsf{lca}(i', i)$ is below $\mathsf{lca}(j', j)$ because that would mean j' has a change point right below $\mathsf{lca}(j', j)$ which falls above $\mathsf{lca}(i', i)$. This would make $\alpha\mathsf{LCPC}[i]$ different than $\alpha\mathsf{LCPC}[j]$ because the suffixes i and i' will have an extra change point above $\mathsf{lca}(i, i')$ and below the $\mathsf{lca}(j, j')$. Hence, for the case (2b*) this leads to a contradiction and $\mathsf{lca}(j, j')$ cannot be above the $\mathsf{lca}(i, i')$. ◀

If $\mathsf{lca}(i', i)$ and $\mathsf{lca}(j', j)$ are not on the same root-to-leaf path (neither above nor below nor same as each other), then pairs i, i' and j, j' are non-interfering. So we need not consider that case as in some sense for i, our algorithm looks at the closest suffix to the left of i that has the same $\alpha\mathsf{LCP}$ and $\alpha\mathsf{LCPC}$ as the qualifying suffix for $\mathsf{LFS}(i)$.

Finally, from Fact 1 we can say that there exists a unique suffix i' marked with case (2b) under the point at $1 + \mathsf{depth}(\mathsf{lca}(i', i))$ depth such that $\alpha\mathsf{LCP}[i] = \overline{\alpha\mathsf{LCP}}[i']$ and $\alpha\mathsf{LCPC}[i] = \overline{\alpha\mathsf{LCPC}}[i']$, with the constraint that i' has a change point at $1 + \mathsf{depth}(\mathsf{lca}(i', i))$ depth.

4.4.4.2 Searching in Minitree

For any i, if we can identify $\mathsf{lca}(i', i)$ precisely, then i' is the leaf which has the same $\overline{\alpha\mathsf{LCPC}}$ and $\overline{\alpha\mathsf{LCP}}$ values (as that of i) and i' is in the subtree of a branch of $\mathsf{lca}(i', i)$ whose leading character in that branch is a change point. For this, we mark some nodes in the tree. More precisely, for each mini-tree, we mark a node v if a point at $(\mathsf{depth}(\mathsf{parent}(v)) + 1)$ depth is a change point for a suffix i' (in case (2b)) in the subtree of v. Note that only one child of a node can get marked (refer to Fact 1). Also note that there is only one marked node in a path from the root to a leaf because if there were another marked node w for a suffix j', then $\alpha\mathsf{LCPC}[i'] \neq \alpha\mathsf{LCPC}[j']$. But we know that all the leaves in a mini-trie have the same $\alpha\mathsf{LCPC}[i], \alpha\mathsf{LCP}$ (or their complement) values.

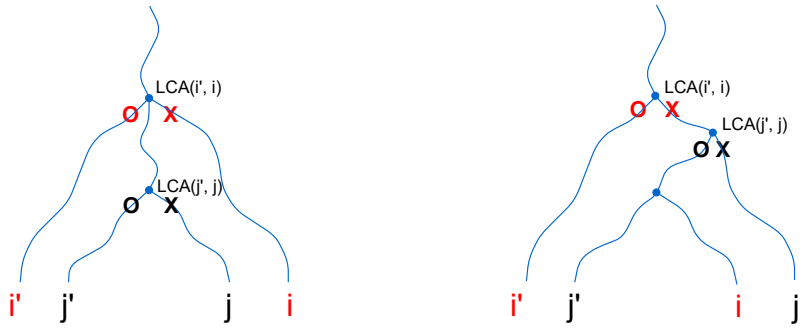


Figure 4 Mini-trees for case (2b). Red (resp. black) circle is the marked node after the change point of i' (resp. j') immediately after $\mathsf{lca}(i', i)$ (resp. $\mathsf{lca}(j', j)$). Red (resp. black) cross is the sibling of the marked node lying on the path from the LCA to the leaf ℓ_i (resp. ℓ_j).

Now lets say that a node x in the mini tree $\tau_{\alpha\mathsf{LCPC}[i],\alpha\mathsf{LCP}[i]}$ is the node corresponding to $\mathsf{lca}(i', i)$ in the suffix tree. Therefore, given i, our task simply becomes locating the leaf ℓ in the mini-tree that corresponds to i. Then, find the lowest ancestor of ℓ that has a marked child before ℓ in pre-order; observe that this lowest ancestor is precisely the node x corresponding to $\mathsf{lca}(i', i)$. Let y be the marked child of x. Within the subtree of y, we can find the unique leaf ℓ' corresponding to i', which can be mapped back to the original suffix tree. To find this unique leaf, we store a unary encoding at the marked node indicating which leaf we looking for; more precisely, if the desired leaf is the z^{th} leftmost leaf under the marked node, then store z in unary at the marked node. Since there is only one marked node from a leaf to root path in a mini-tree, the total length of all such unary encodings combined is bounded by the size of the mini-tree. The mapping to and from the suffix tree to a mini-tree can be carried out using the bit-vector and the character vectors defined earlier.

For the sake of completion, we summarize the discussion in the following findSucc method, which was used by pseudo-code in Section 4.3. See Figure 4 for an illustration.

findSucc(i, a, b)

- Use the bit-vector B and the character vectors C and $\overline{C}$ to identify the leaf ℓ in $\tau_{a,b}$ that corresponds to ℓ_i
- Find the lowest ancestor x of ℓ that has a marked child y before x in pre-order
- Use the unary encoding stored at y to locate the leaf ℓ' in $\tau_{a,b}$ corresponding to $\ell_{i'}$
- Finally, use the character vector $C_{a,b}$ to map ℓ' back to i'

4.5 Implementation and Complexity Analysis

We will rely on the following well-known data structures of Fact 2 and Fact 3.

▶ **Fact 2** (Wavelet Tree [12]). *Given an array $A[1, t]$ over Σ, by using a $t \log |\Sigma| + o(t \log |\Sigma|)$-bit structure, we can compute the following in $O(\log |\Sigma|)$ time:*

- $A[i]$
- $\mathsf{rank}_A(i, x)$ = *number of occurrences of x in $A[1, i]$*
- $\mathsf{select}_A(i, x)$ = *i-th occurrence of x in A*
- $\mathsf{prevValue}_A(i, y)$ = *rightmost position $j < i$ such that $A[j] \leq y$*

We drop the subscript A when the context is clear.

▶ **Fact 3** (Fully-Functional Succinct Tree [21]). *The topology of order-isomorphic suffix tree can be encoded in $O(n)$ bits to support the following operations in $O(1)$ time.*

- $\mathsf{pre\text{-}order}(u)/\mathsf{post\text{-}order}(u)$*: pre-order/post-order rank of node* u
- $\mathsf{parent}(u)$*: parent of node* u
- $\mathsf{nodeDepth}(u)$*: number of edges on the path from the root to* u
- $\mathsf{child}(u, q)$*: qth leftmost child of node* u
- $\mathsf{sibRank}(u)$*: number of children of* $\mathsf{parent}(u)$ *to the left of* u
- $\mathsf{lca}(u, v)$*: lowest common ancestor (LCA) of two nodes* u *and* v
- $\mathsf{sp}(u)/\mathsf{ep}(u)$*: leftmost/rightmost leaf in the subtree of* u
- $\mathsf{levelAncestor}(u, d)$*: ancestor of* u *such that* $\mathsf{nodeDepth}(u) = d$

Moving forward, we assume that any array has been pre-processed using Fact 2. We maintain the topology of the order-isomorphic suffix tree and the mini-trees (Case 2b) using Fact 3. Finally, we explicitly store $\alpha\mathsf{Depth}(u)$ for every node u in the order-isomorphic suffix tree. For the purpose of locating the node immediately below FPB or LCPC, we will rely on the following lemma.

▶ **Lemma 13.** *By maintaining an $O(n \log \sigma)$ bit data structure, given a leaf ℓ_i and an integer W, we can find the highest ancestor w of ℓ_i satisfying $\alpha\mathsf{Depth}(w) \geq W$ in $O(\log \sigma)$ time.*

Proof. Create an array A such that $A[k] = \alpha\mathsf{Depth}(w)$, where w is the node with pre-order rank k. Maintain A as a wavelet tree. Given ℓ_i, find the rightmost entry $r < \mathsf{pre\text{-}order}(\ell_i)$ in A such that $A[r] < W$ using $\mathsf{prevValue}_A(\mathsf{pre\text{-}order}(\ell_i), W-1)$. Let $v' = \mathsf{lca}(\ell_i, v)$, where v is the node with pre-order rank r. Then, $w = \mathsf{levelAncestor}(\ell_i, \mathsf{nodeDepth}(v')+1)$. To see why this is correct, observe that $\alpha\mathsf{Depth}(v') \leq \alpha\mathsf{Depth}(v) < W$. If $\alpha\mathsf{Depth}(w) < W$, the $\mathsf{prevValue}$-query should have returned w instead of v (since $\mathsf{pre\text{-}order}(v) < \mathsf{pre\text{-}order}(w) \leq \mathsf{pre\text{-}order}(\ell_i)$). ◀

4.5.1 Case (1a) and Case (1b)

In case (1a), i' is the only leaf marked with case (1a) in the sub-tree of $\mathsf{FPB}(i)$ that satisfies $\overline{\alpha\mathsf{FPB}}[i'] = \alpha\mathsf{FPB}[i]$. The first task is to find the subtree of $\mathsf{FPB}(i)$, i.e., the node just below $\mathsf{FPB}(i)$. This node, say v, can be found in $O(\log \sigma)$ time using Lemma 13 and by using $\alpha\mathsf{FPB}[i]$. Within the subtree of v, we simply find the only leaf i' marked with 1a such that $\overline{\mathsf{FPB}}[i'] = \mathsf{FPB}[i]$ using Fact 2. Since $\alpha\mathsf{FPB}$ and $\overline{\alpha\mathsf{FPB}}$ entries for case (1a) suffixes are at least one, in order to identify a valid case (1a) suffix, we simply set the $\alpha\mathsf{FPB}$ and $\overline{\alpha\mathsf{FPB}}$ entries for non case (1a) suffixes to zero.

In case (1b), the idea is the same, with the difference that we use $\alpha\mathsf{LCPC}$ and $\overline{\alpha\mathsf{LCPC}}$ arrays (instead of FPB and $\alpha\mathsf{FPB}$ arrays) for finding the node v and then i'. As in the previous case, we set the $\alpha\mathsf{LCPC}$ and $\overline{\alpha\mathsf{LCPC}}$ entries for non case (1b) suffixes to zero.

Note that the wavelet trees for the four arrays need $O(n \log \sigma)$ bits, and a wavelet tree query needs $O(\log \sigma)$ time.

4.5.2 Case (2a)

Let c be the point just above $\mathsf{FPC}[i]$. Let ℓ_k be the rightmost leaf in the subtree of c. By Lemma 10, it is evident that $i' > k$, $\overline{\alpha\mathsf{LCP}}[i'] = \alpha\mathsf{LCP}[i]$, $\overline{\alpha\mathsf{LCPC}}[i'] = \alpha\mathsf{LCPC}[i]$, and $\overline{\mathrm{EQBT}}[i'] = \mathrm{EQBT}[i]$. To properly identify a case (2a) suffix, we maintain a summary vector X defined as follows. For any suffix i lying in case (2a), $X[i] = (\sigma-1) \cdot \alpha\mathsf{LCP}[i] + \alpha\mathsf{LCPC}[i]$ if $\mathrm{EQBT}[i] = 1$, and $X[i] = -(\sigma-1) \cdot \alpha\mathsf{LCP}[i] - \alpha\mathsf{LCPC}[i]$ if $\mathrm{EQBT}[i] = 0$. For any suffix j not in case (2a), we let $X[i] = 0$. Likewise, we define $\overline{X}$ based on $\overline{\alpha\mathsf{LCP}}$, $\overline{\alpha\mathsf{LCPC}}$, and $\overline{\mathrm{EQBT}}$.

Note that any entry in X and $\overline{X}$ is from the set $[0, 2\sigma^2]$; hence, a wavelet over them needs $O(n \log \sigma)$ bits and supports queries in $O(\log \sigma)$ time. Thus, if we can find out the leaf ℓ_k, we can locate i' by using the wavelet-tree over the two summary vectors X and $\overline{X}$ in additional $O(\log \sigma)$ time.

To find ℓ_k, we use Lemma 13 and $\alpha\mathsf{FPB}$ to first find the highest node v such that $\alpha\mathsf{Depth}(v) \geq \alpha\mathsf{FPB}[i]$. Note that ℓ_k is the rightmost leaf in the subtree of $\mathsf{parent}(v)$ if $\mathsf{FPB}[i]$ is the first character of the edge on which it lies, and is the rightmost leaf in the subtree of v otherwise. We explicitly store a bit-vector to distinguish between the cases. Using these, ℓ_k is located in $O(\log \sigma)$ time.

4.5.3 Case (2b)

In our previous discussion, we have already addressed how to map a case (2b) leaf i in the suffix tree to its corresponding leaf in the mini-tree (refer to Section 4.4.4). We have also addressed that given the desired marked node (corresponding to i) in the mini-tree, how we can find the leaf in the mini-tree corresponding to the LF-successor i'. Finally, we also know how to map-back to i' from the mini-tree. Note that all of these can be achieved by storing the character vectors and the bit vector as a wavelet tree, and by using a succinct encoding of the mini trees. What is left to discuss is how to find the marked node. To this end, we present Lemma 14. Using this we can find the desired marked node in $O(1)$ time given the leaf corresponding to i in the mini-tree.

▶ **Lemma 14.** *Consider a tree having t nodes, where each non-leaf node has at least two children. Also, each node is marked or unmarked. By using an $O(t)$-bit data structure, given a leaf x, in $O(1)$ time, we can find the rightmost leaf $y < x$ such that the child of $\mathsf{lca}(y, x)$ on the path to y is marked.*

Proof. Let u be a node. We *associate* 1 with u iff $\mathsf{parent}(u)$ has a child v before u in pre-order, where v is marked. Pre-process the tree with Lemmas 15 and 16.

Given the query x, use Lemma 15 to locate the lowest ancestor u of x associated with a 1. We find the marked sibling v of u to its left using Lemma 16. The time needed is $O(1)$. ◀

▶ **Lemma 15.** *Consider a tree having t nodes, where each non-leaf node has at least two children. Also, each node is associated with a 0 or 1. By using an $O(t)$-bit data structure, in $O(1)$ time, we can find the lowest ancestor of a leaf that is associated with a 1.*

Proof. Starting from the leftmost leaf, every $g = c\lceil \log t \rceil$ leaves form a group, where c is a constant to be decided later. (The last group may have fewer than g leaves.) Mark the lca of the first and last leaf of each group. At each marked node, write the node-depth of its lowest ancestor which is associated with a 1. The space needed is $O(\frac{t}{g} \log t) = O(t)$ bits. Let τ_u be the subtree rooted at a marked node u. Since each node in τ_u is associated with a 0 or 1, the number of possible trees is at most 2^g (because τ_u has fewer than g non-leaf nodes). We store a pointer from u to τ_u. The total space needed for storing all pointers is $O(\frac{t}{g} \log 2^g) = O(t)$ bits. For each possible τ_u, store the following satellite data in an additional array. Consider the kth leftmost leaf ℓ_k in τ_u. Let v be the lowest node on the path from u to ℓ_k associated with a 1. If v exists, store the node-depth of v relative to u, else store -1. The space needed for each τ_u is $O(g \log g) = O(g \log \log t)$ bits. Therefore, the total space for all such trees is $O(2^g g \log \log t)$. By choosing $c = 1/2$, this space is bounded by $o(t)$ bits. Thus, the total space is bounded by $O(t)$ bits.

Given a query leaf ℓ_k, we first locate the lowest marked node $u^* = \mathsf{lca}(1 + g\lfloor k/g \rfloor, \max\{t, g(1 + \lfloor k/g \rfloor)\})$ of ℓ_k. Let d^* be the depth stored at u^*. Let $k' = k - g\lfloor k/g \rfloor$. Check the k'th entry of the satellite array of u^*, and let it be d. If $d = -1$, then assign $D = d^*$, else assign $D = \mathsf{nodeDepth}(u^*) + d$. The lowest ancestor of ℓ_k associated with a 1 is given by $\mathsf{levelAncestor}(\ell_k, D)$. ◀

▶ **Lemma 16.** *Consider a tree of t nodes, where some nodes are marked. By using an $O(t)$-bit data structure, in $O(1)$ time, given a node v, we can find a node u (if any) such that u is the rightmost marked child of $\mathsf{parent}(v)$ and $\mathsf{pre\text{-}order}(u) < \mathsf{pre\text{-}order}(v)$.*

Proof. For each node w, we store a bit-vector $B_w[t_w]$, where t_w is the number of children of w. Assign $B_w[i] = 1$ iff the i^{th} leftmost child of w, given by $\mathsf{child}(w, i)$, is marked. The total space needed is $O(t)$ bits. Given the query node v, we go to the bit vector $B_{v'}$, where $v' = \mathsf{parent}(v)$. Let $r = \mathsf{rank}_{B_{v'}}(\mathsf{sibRank}(v), 1)$. If $r = 0$, then u does not exist; otherwise, $u = \mathsf{child}(v', \mathsf{select}_{B_{v'}}(r, 1))$. ◀

References

1 Brenda S. Baker. A theory of parameterized pattern matching: algorithms and applications. In *Proceedings of the Twenty-Fifth Annual ACM Symposium on Theory of Computing, May 16-18, 1993, San Diego, CA, USA*, pages 71–80, 1993. doi:10.1145/167088.167115.

2 M. Burrows and D. J. Wheeler. A block-sorting lossless data compression algorithm, 1994.

3 Domenico Cantone, Simone Faro, and M. Oguzhan Külekci. The order-preserving pattern matching problem in practice. *Discret. Appl. Math.*, 274:11–25, 2020. doi:10.1016/j.dam.2018.10.023.

4 Richard Cole and Ramesh Hariharan. Faster suffix tree construction with missing suffix links. *SIAM J. Comput.*, 33(1):26–42, 2003. *An extended abstract appeared in STOC 2000.* doi:10.1137/S0097539701424465.

5 Maxime Crochemore, Costas S. Iliopoulos, Tomasz Kociumaka, Marcin Kubica, Alessio Langiu, Solon P. Pissis, Jakub Radoszewski, Wojciech Rytter, and Tomasz Walen. Order-preserving indexing. *Theor. Comput. Sci.*, 638:122–135, 2016. doi:10.1016/j.tcs.2015.06.050.

6 Gianni Decaroli, Travis Gagie, and Giovanni Manzini. A compact index for order-preserving pattern matching. In *2017 Data Compression Conference, DCC 2017, Snowbird, UT, USA, April 4-7, 2017*, pages 72–81, 2017. doi:10.1109/DCC.2017.35.

7 Paolo Ferragina and Giovanni Manzini. Indexing compressed text. *J. ACM*, 52(4):552–581, 2005. *An extended abstract appeared in FOCS 2000 under the title "Opportunistic Data Structures with Applications".* doi:10.1145/1082036.1082039.

8 Travis Gagie, Giovanni Manzini, and Rossano Venturini. An encoding for order-preserving matching. In *25th Annual European Symposium on Algorithms, ESA 2017, September 4-6, 2017, Vienna, Austria*, pages 38:1–38:15, 2017. doi:10.4230/LIPIcs.ESA.2017.38.

9 Arnab Ganguly, Wing-Kai Hon, Kunihiko Sadakane, Rahul Shah, Sharma V. Thankachan, and Yilin Yang. A framework for designing space-efficient dictionaries for parameterized and order-preserving matching. *Theor. Comput. Sci.*, 854:52–62, 2021. doi:10.1016/j.tcs.2020.11.036.

10 Arnab Ganguly, Rahul Shah, and Sharma V Thankachan. pBWT: Achieving succinct data structures for parameterized pattern matching and related problems. In *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 397–407. Society for Industrial and Applied Mathematics, 2017.

11 Arnab Ganguly, Rahul Shah, and Sharma V. Thankachan. Structural pattern matching - succinctly. In Yoshio Okamoto and Takeshi Tokuyama, editors, *28th International Symposium on Algorithms and Computation, ISAAC 2017, December 9-12, 2017, Phuket, Thailand*, volume 92 of *LIPIcs*, pages 35:1–35:13. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2017. doi:10.4230/LIPIcs.ISAAC.2017.35.

12 Roberto Grossi, Ankur Gupta, and Jeffrey Scott Vitter. High-order entropy-compressed text indexes. In *Proceedings of the Fourteenth Annual ACM-SIAM Symposium on Discrete Algorithms, January 12-14, 2003, Baltimore, Maryland, USA.*, pages 841–850, 2003.

13 Roberto Grossi and Jeffrey Scott Vitter. Compressed suffix arrays and suffix trees with applications to text indexing and string matching. *SIAM J. Comput.*, 35(2):378–407, 2005. *An extended abstract appeared in STOC 2000.* doi:10.1137/S0097539702402354.

14 Jinil Kim, Peter Eades, Rudolf Fleischer, Seok-Hee Hong, Costas S. Iliopoulos, Kunsoo Park, Simon J. Puglisi, and Takeshi Tokuyama. Order-preserving matching. *Theor. Comput. Sci.*, 525:68–79, 2014. doi:10.1016/j.tcs.2013.10.006.

15 Sung-Hwan Kim and Hwan-Gue Cho. Simpler fm-index for parameterized string matching. *Inf. Process. Lett.*, 165:106026, 2021. doi:10.1016/j.ipl.2020.106026.

16 Marcin Kubica, Tomasz Kulczyński, Jakub Radoszewski, Wojciech Rytter, and Tomasz Waleń. A linear time algorithm for consecutive permutation pattern matching. *Information Processing Letters*, 113(12):430–433, 2013.

17 Udi Manber and Eugene W. Myers. Suffix arrays: A new method for on-line string searches. *SIAM J. Comput.*, 22(5):935–948, 1993. doi:10.1137/0222058.

18 Juan Mendivelso, Sharma V. Thankachan, and Yoan J. Pinzón. A brief history of parameterized matching problems. *Discret. Appl. Math.*, 274:103–115, 2020. doi:10.1016/j.dam.2018.07.017.

19 Temma Nakamura, Shunsuke Inenaga, Hideo Bannai, and Masayuki Takeda. Order preserving pattern matching on trees and dags. In Gabriele Fici, Marinella Sciortino, and Rossano Venturini, editors, *String Processing and Information Retrieval - 24th International Symposium, SPIRE 2017, Palermo, Italy, September 26-29, 2017, Proceedings*, volume 10508 of *Lecture Notes in Computer Science*, pages 271–277. Springer, 2017. doi:10.1007/978-3-319-67428-5_23.

20 Gonzalo Navarro. *Compact data structures: A practical approach.* Cambridge University Press, 2016.

21 Gonzalo Navarro and Kunihiko Sadakane. Fully functional static and dynamic succinct trees. *ACM Trans. Algorithms*, 10(3):16:1–16:39, 2014. *An extended abstract appeared in SODA 2010.* doi:10.1145/2601073.

22 Kunihiko Sadakane. Compressed suffix trees with full functionality. *Theory Comput. Syst.*, 41(4):589–607, 2007. doi:10.1007/s00224-006-1198-x.

23 Peter Weiner. Linear pattern matching algorithms. In *14th Annual Symposium on Switching and Automata Theory, Iowa City, Iowa, USA, October 15-17, 1973*, pages 1–11, 1973. doi:10.1109/SWAT.1973.13.

Crossing-Optimal Extension of Simple Drawings

Robert Ganian
Algorithms and Complexity Group, TU Wien, Austria

Thekla Hamm
Algorithms and Complexity Group, TU Wien, Austria

Fabian Klute
Deptartment of Information and Computing Sciences, Utrecht University, The Netherlands

Irene Parada
TU Eindhoven, The Netherlands

Birgit Vogtenhuber
Graz University of Technology, Austria

Abstract

In extension problems of partial graph drawings one is given an incomplete drawing of an input graph G and is asked to complete the drawing while maintaining certain properties. A prominent area where such problems arise is that of crossing minimization. For plane drawings and various relaxations of these, there is a number of tractability as well as lower-bound results exploring the computational complexity of crossing-sensitive drawing extension problems. In contrast, comparatively few results are known on extension problems for the fundamental and broad class of simple drawings, that is, drawings in which each pair of edges intersects in at most one point. In fact, the extension problem of simple drawings has only recently been shown to be NP-hard even for inserting a single edge.

In this paper we present tractability results for the crossing-sensitive extension problem of simple drawings. In particular, we show that the problem of inserting edges into a simple drawing is fixed-parameter tractable when parameterized by the number of edges to insert and an upper bound on newly created crossings. Using the same proof techniques, we are also able to answer several closely related variants of this problem, among others the extension problem for k-plane drawings. Moreover, using a different approach, we provide a single-exponential fixed-parameter algorithm for the case in which we are only trying to insert a single edge into the drawing.

2012 ACM Subject Classification Theory of computation → Parameterized complexity and exact algorithms; Theory of computation → Computational geometry

Keywords and phrases Simple drawings, Extension problems, Crossing minimization, FPT-algorithms

Digital Object Identifier 10.4230/LIPIcs.ICALP.2021.72

Category Track A: Algorithms, Complexity and Games

Related Version *Full Version*: `https://arxiv.org/abs/2012.07457` [24]

Funding *Robert Ganian*: Supported by the Austrian Science Fund (FWF) via projects Y1329 (*Parameterized Analysis in Artificial Intelligence*) and P31336 (*New Frontiers for Parameterized Complexity*).
Thekla Hamm: Supported by the Austrian Science Fund (FWF) via projects Y1329 (*Parameterized Analysis in Artificial Intelligence*), P31336 (*New Frontiers for Parameterized Complexity*) and W1255-N23.
Fabian Klute: Supported by the Netherlands Organisation for Scientific Research (NWO) under project no. 612.001.651.
Birgit Vogtenhuber: Partially supported by Austrian Science Fund (FWF) within the collaborative DACH project *Arrangements and Drawings* as FWF project I 3340-N35.

Acknowledgements This work was started during the Austrian Computational Geometry Reunion Meeting in Strobl (Austria), August 10 to 14, 2020. We thank all the participants for the nice working atmosphere as well as fruitful discussions on this as well as other topics. The authors would also like to thank Eduard Eiben for his insightful comments.

1 Introduction

The study of the crossing number of graphs, that is, the minimum number of edge crossings necessary to draw a given graph, is a major research direction in the field of computational geometry [10, 33, 36]. More recently, there have been a number of works focusing on minimizing or restricting edge crossings when the task is not to draw a graph from scratch, but rather to extend a partial drawing that is provided on the input. Prominently, Chimani et al. [14] showed that extending a plane drawing with a star in a way that minimizes the number of crossings of the resulting drawing is polynomial-time tractable. Later, Angelini et al. [1] obtained a polynomial-time algorithm for extending plane drawings so that the crossing number remains 0 (i.e., the resulting drawing is plane).

While the two results mentioned above give rise to polynomial-time variants of crossing-minimization extension problems, a number of important cases are known to be NP-hard; a prototypical example is the Rigid Multiple Edge Insertion (RMEI) problem, which asks for a crossing-minimal insertion of k edges into a plane drawing of an n-vertex graph [15, 37]. To deal with this, in recent years the focus has broadened to also consider a weaker notion of tractability, namely, *fixed-parameter tractability* (FPT) [17, 19]. Chimani and Hliněný [15] have shown that RMEI is FPT, i.e., there is an algorithm which solves that problem in time $f(k) \cdot n^{\mathcal{O}(1)}$. Other works have considered various relaxations of crossing minimization; for instance, recently Eiben et al. [20] established the fixed-parameter tractability of extending drawings by k edges in a way which does not minimize the total number of crossings, but rather bounds the number of crossings per edge to at most 1.

For many problems in the intersection of crossing minimization and graph extension, an important goal is that the desired extension should maintain certain properties of the given partial representation. In the problems studied in [1] and [20], the input is a plane or 1-plane[1] drawing, respectively, and the desired extension must maintain the property of being (1-)plane. There have been a plethora of results exploring such extension problems, especially on plane drawings, for a range of other, often more restrictive properties [3, 9, 12, 13, 31, 32, 34].

Beyond planarity, the perhaps most prominent class of drawings with respect to crossing minimization are *simple drawings* (also called *good drawings* [8, 21], *simple topological graphs* [29], or simply *drawings* [26]). A drawing is simple if every pair of edges intersects in at most one point that is either a common endpoint or a proper crossing. Simplicity is an extremely natural restriction that is taken as a basic assumption in a range of settings, e.g., [2, 5, 11, 30], and that constitutes a necessary requirement for crossing-minimal drawings [36].

Contribution. In this work we study the extension problem for simple drawings in the context of crossing minimization. In other words, our aim is to extend a given simple drawing with k new edges while maintaining simplicity and restricting newly created crossings. Naturally, the most obvious way of restricting such crossings is by bounding their number, leading us to our first problem of interest:[2]

[1] A drawing of a graph is ℓ-plane if every edge is involved in at most ℓ crossings.

[2] *Decision* versions of problems are provided purely for complexity-theoretic reasons; every algorithm provided in this article is constructive and can also output a solution as a witness.

Simple Crossing-Minimal Edge Insertion (SCEI)	
Input:	A graph $G = (V, E)$ along with a connected simple drawing $\mathcal{G}$, an integer ℓ, and a set F of k edges of the complement of G.
Question:	Can $\mathcal{G}$ be extended to a simple drawing $\mathcal{G}'$ of the graph $G' = (V, E \cup F)$ such that the number of crossings in $\mathcal{G}'$ that involve an edge of F is at most ℓ?

Note that we require the initial drawing $\mathcal{G}$ to be connected. While this is a natural assumption that is well-justified in many situations, it would certainly also make sense to consider the more general setting in which this is not the case. A short discussion of how the connectivity of $\mathcal{G}$ is used in our proof is provided in Section 4.

SCEI was recently shown to be NP-complete already when $|F| = 1$ and $\ell \geq |E|$ (meaning that the aim is merely to obtain a simple drawing) [7]. On the other hand, dropping the simplicity requirement of the resulting drawing, the problem reduces to RMEI which is FPT.

The main contribution of this article is an FPT algorithm for SCEI parameterized by $k + \ell$. The result is obtained via a combination of techniques recently introduced in [20] and completely new machinery. A high-level overview of challenges posed by the problem and our strategies for overcoming them is provided in the next part of this introduction. Before that, let us mention other natural crossing-sensitive restrictions of simple drawing extension.

Instead of restricting the *total number* of newly created crossings, one may aim to extend $\mathcal{G}$ in a way which bounds the number of crossings involving each of the newly added edges – akin to the restrictions imposed by ℓ-planarity. We call this problem Simple Locally Crossing-Minimal Edge Insertion (SLCEI), where the role of ℓ is that it bounds the maximum number of crossings involving any one particular edge of F. Alternatively, one may simply require that *every* edge in the resulting drawing is involved in at most ℓ crossings, i.e., that the whole $\mathcal{G}'$ is ℓ-plane. This results in the Simple ℓ-Plane Edge Insertion (Sℓ-PEI) problem. Both of these problems are known to be NP-hard when either $\ell = 1$ or $k = 1$, meaning that we can drop neither of our parameters if we wish to achieve tractability.

One key strength of the framework we develop for solving SCEI is its universality. Notably, we obtain the fixed-parameter tractability of SLCEI as an immediate corollary of the proof of our main theorem, while the fixed-parameter tractability of Sℓ-PEI follows by a minor adjustment of the final part of our proof. Moreover, it is trivial to use the framework to solve the considered problems when one drops the requirement that the final drawing is simple – allowing us to, e.g., generalize the previously established fixed-parameter tractability of 1-Planar Edge Insertion [20] to ℓ-Planar Edge Insertion (ℓ-PEI).

Finally, we note that a core ingredient in our approach is the use of Courcelle's theorem [16], and hence the algorithms underlying our tractability results will have an impractical dependency on k. However, for the special case of $|F| = 1$ (i.e., when inserting a single edge), we use so-called *representative sets* to provide a single-exponential fixed-parameter algorithm which is tight under the Exponential Time Hypothesis [27].

Proof Overview. On a high level, our approach follows the general strategy co-developed by a subset of the authors in [20] for solving the problem of inserting k edges into a drawing while maintaining 1-planarity. This general strategy can be summarized as follows:

1. We preprocess G and the planarization of $\mathcal{G}$ to remove parts of $\mathcal{G}$ which are too far away to interact with our solution. This drawing is then translated into a graph representation of bounded *treewidth* [35].
2. We identify a combinatorial characterization that captures how the solution curves will be embedded into $\mathcal{G}$. Crucially, the characterization has size bounded by our parameters.
3. We perform brute-force branching over all characterizations to pre-determine the behavior of a solution in $\mathcal{G}$, and for each such characterization we employ *Courcelle's theorem* [16] to determine whether there exists a solution with such a characterization.

The specific implementation of this strategy differs substantially from the previous work [20] – for instance, the combinatorial characterization of solutions in Step 2 and the use of Courcelle's theorem in Step 3 are both different. But the by far greatest challenge in implementing this strategy occurs in Step 1. Notably, removing the parts of $\mathcal{G}$ required to obtain a bounded-treewidth graph representation creates *holes* in the drawing, and these could disconnect edges intersecting these holes. The graph representation can then lose track of "which edge parts belong to each other", which means we can no longer use it to determine whether the extended drawing is simple. We remark that specifically for Sℓ-PEI and ℓ-PEI, it would be possible to directly adapt Step 1 to ensure that no edge is disconnected in this manner, thus circumventing this difficulty. To handle this problem, we employ an in-depth geometric analysis combined with a careful use of the sunflower lemma and subroutines which invoke Courcelle's theorem to construct a representation which (a) still has bounded treewidth, and (b) contains partial information about which edge parts belong to the same edge in $\mathcal{G}$. A detailed overview of how this is achieved is presented at the beginning of Section 3.

Related Work. There have been two distinct lines of work that recently considered simple drawings in the context of drawing extension problems. The first studied a closely related notion of *saturated* simple drawings [25, 28], while the second studied the computational complexity of the extension problem for simple drawings [6, 7].

Statements where proofs or more details are provided in the full version are marked with ($\star$).

2 Preliminaries

We use standard terminology for undirected and simple graphs [18]. The *length* of a walk or a path is the number of edges it visits. For $r \in \mathbb{N}$, we write $[r]$ as shorthand for the set $\{1, \ldots, r\}$.

A *simple drawing* of a graph G is a drawing $\mathcal{G}$ of G in the plane such that every pair of edges shares at most one point that is either a unique crossing point or a common endpoint. In particular, no tangencies between edges are allowed, edges must not contain any vertices in their relative interior, and no three edges intersect in the same point. Given a simple drawing $\mathcal{G}$ of a graph G and a set of edges F of the complement of G we say that the edges in F can be *inserted* into $\mathcal{G}$ if there exists a simple drawing $\mathcal{G}^+$ of $G^+ = (V(G), E(G) \cup F)$ that contains $\mathcal{G}$ as a subdrawing. The *planarization* of a simple drawing $\mathcal{G}$ of G is the plane graph $\mathcal{G}^\times$ obtained from $\mathcal{G}$ by subdividing the edges of G at the crossing points of $\mathcal{G}$. We call each part of the subdivision of $e \in E(G)$ in $\mathcal{G}^\times$ an *edge segment* (of e). Furthermore, we consider the faces of $\mathcal{G}^\times$ as the *cells* of $\mathcal{G}$ and call $\mathcal{G}$ *connected* if $\mathcal{G}^\times$ is a connected graph.

Sunflower Lemma. One tool we use to obtain our results is the classical sunflower lemma of Erdős and Rado. A *sunflower* in a set family $\mathcal{F}$ is a subset $\mathcal{F}' \subseteq \mathcal{F}$ such that all pairs of elements in $\mathcal{F}'$ have the same intersection.

▶ **Lemma 1** ([22, 23]). *Let $\mathcal{F}$ be a family of subsets of a universe U, each of cardinality at most b, and let $a \in \mathbb{N}$. If $|\mathcal{F}| \geq b!(a-1)^b$, then $\mathcal{F}$ contains a sunflower $\mathcal{F}'$ of cardinality at least a. Moreover, $\mathcal{F}'$ can be computed in time polynomial in $|\mathcal{F}|$.*

Parameterized Complexity. In parameterized complexity [17, 19, 23], the complexity of a problem is studied not only with respect to the input size, but also with respect to some problem parameter(s). The core idea behind parameterized complexity is that the

combinatorial explosion resulting from the NP-hardness of a problem can sometimes be confined to certain structural parameters that are small in practical settings. We now proceed to the formal definitions.

A *parameterized problem* Q is a subset of $\Omega^* \times \mathbb{N}$, where Ω is a fixed alphabet. Each instance of Q is a pair (I, κ), where $\kappa \in \mathbb{N}$ is called the *parameter*. A parameterized problem Q is *fixed-parameter tractable* (FPT) [23, 19, 17], if there is an algorithm, called an FPT-*algorithm*, that decides whether an input (I, κ) is a member of Q in time $f(\kappa) \cdot |I|^{\mathcal{O}(1)}$, where f is a computable function and $|I|$ is the input instance size. The class FPT denotes the class of all fixed-parameter tractable parameterized problems. A parameterized problem Q is FPT-*reducible* to a parameterized problem Q' if there is an algorithm, called an FPT-*reduction*, that transforms each instance (I, κ) of Q into an instance (I', κ') of Q' in time $f(\kappa) \cdot |I|^{\mathcal{O}(1)}$, such that $\kappa' \leq g(\kappa)$ and $(I, \kappa) \in Q$ if and only if $(I', \kappa') \in Q'$, where f and g are computable functions.

Monadic Second Order Logic. We consider *Monadic Second Order* (MSO) logic on (edge-)labeled directed graphs in terms of their incidence structure, whose universe contains vertices and edges; the incidence between vertices and edges is represented by a binary relation. We assume an infinite supply of *individual variables* $x, x_1, x_2, \ldots$ and of *set variables* $X, X_1, X_2, \ldots$. The *atomic formulas* are Vx ("x is a vertex"), Ey ("y is an edge"), Ixy ("vertex x is incident with edge y"), $x = y$ (equality), $P_a x$ ("vertex or edge x has label a"), and Xx ("vertex or edge x is an element of set X"). *MSO formulas* are built up from atomic formulas using the usual Boolean connectives $(\neg, \wedge, \vee, \rightarrow, \leftrightarrow)$, quantification over individual variables ($\forall x$, $\exists x$), and quantification over set variables ($\forall X$, $\exists X$).

Free and bound variables of a formula are defined in the usual way. To indicate that the set of free individual variables of formula Φ is $\{x_1, \ldots, x_\ell\}$ and the set of free set variables of formula Φ is $\{X_1, \ldots, X_q\}$ we write $\Phi(x_1, \ldots, x_\ell, X_1, \ldots, X_q)$. If G is a graph, $v_1, \ldots, v_\ell \in V(G) \cup E(G)$ and $S_1, \ldots, S_q \subseteq V(G) \cup E(G)$ we write $G \models \Phi(v_1, \ldots, v_\ell, S_1, \ldots, S_q)$ to denote that Φ holds in G if the variables x_i are interpreted by the vertices or edges v_i, for $i \in [\ell]$, and the variables X_i are interpreted by the sets S_i, for $i \in [q]$.

The following result (the well-known Courcelle's theorem [16]) shows that if G has bounded treewidth [35] then we can find an assignment φ to the set of free variables $\mathcal{F}$ with $G \models \Phi(\varphi(\mathcal{F}))$ (if one exists) in linear time.

▶ **Theorem 2** (Courcelle's theorem [4, 16]). *Let $\Phi(x_1, \ldots, x_\ell, X_1, \ldots, X_q)$ be a fixed MSO formula with free individual variables $x_1, \ldots, x_\ell$ and free set variables $X_1, \ldots, X_\ell$, and let w a constant. Then there is a linear-time algorithm that, given a labeled directed graph G of treewidth at most w, either outputs $v_1, \ldots, v_\ell \in V(G) \cup E(G)$ and $S_1, \ldots, S_q \subseteq V(G) \cup E(G)$ such that $G \models \Phi(v_1, \ldots, v_\ell, S_1, \ldots, S_q)$ or correctly identifies that no such vertices $v_1, \ldots, v_\ell$ and sets $S_1, \ldots, S_q$ exist.*

We remark that since an understanding of the definition of *treewidth* is not required for our presentation, we merely refer to the literature for a discussion of the notion [17, 19, 35]. We denote the treewidth of a graph G as $\mathrm{tw}(G)$.

Problem Definition and Terminology. We formulate the following generalization of SLCEI in which we allow the numbers of crossings allowed for each newly added edge to differ. Note that this formulation also fixes a parameterization.

Simple Crossing-Restricted Edge Insertion (SCREI) Parameter: $k + \max_{i\in[k]} \ell_i$	
Input:	A graph $G = (V, E)$ along with a connected simple drawing $\mathcal{G}$, a set $F = \{e_1, \dots, e_k\}$ of k edges of the complement of G, and $\ell_1, \dots, \ell_k \in \mathbb{N}$.
Question:	Can $\mathcal{G}$ be extended to a simple drawing $\mathcal{G}'$ of the graph $G' = (V, E \cup F)$ such that the drawing of each edge $e_i \in F$ has at most ℓ_i crossings in $\mathcal{G}'$?

For an instance of SCREI we refer to elements in F as *added edges* and denote the endpoints of e_i as s_i and t_i (where $s_1, t_1 \dots, s_k, t_k$ are not necessarily distinct). For brevity we denote $\ell = \max_{i\in[k]} \ell_i$. Although SCREI is stated as a decision problem, we will want to speak about hypothetical *solutions* of SCREI, which will naturally correspond to the drawings of added edges in $\mathcal{G}'$ (if one exists) as the rest of $\mathcal{G}'$ is predetermined by $\mathcal{G}$. This means that a solution is a set of drawings of added edges in $\mathcal{G}'$ where $\mathcal{G}'$ witnesses the fact that the given instance is a `yes`-instance. If no such $\mathcal{G}'$ exists, then we say that the SCREI-instance *has no solution*.

The reason we focus our presentation on SCREI is that the fixed-parameter tractability of SCREI immediately implies the fixed-parameter tractability of both SLCEI parameterized by the number of added edges and crossings per added edge, and SCEI parameterized by the number of added edges and crossings of all added edges. The former is just a subcase of SCREI. The latter admits a straightforward FPT-reduction to SCREI by branching over the number ℓ_i of crossings each edge $e_i \in F$ is at most involved in. Hence obtaining a fixed-parameter algorithm for SCREI provides a unified reason for the fixed-parameter tractability of both SCEI and SLCEI. Furthermore, we will later show that the result for SCREI can be straightforwardly adapted to solve the other problems mentioned in the introduction.

3 Stitches

Let $\big(G, \mathcal{G}, F, (\ell_i)_{i\in[|F|]}\big)$ be an instance of SCREI. Recalling the Proof Overview provided in Section 1, we want to identify parts of $\mathcal{G}$ that may be considered "unimportant" because they can never be intersected by the drawing of any of the edges $s_it_i \in F$ with at most ℓ_i crossings. Formally, consider the dual G^* of the planarization $\mathcal{G}^\times$ of $\mathcal{G}$, and for each vertex $v \in V(G)$ let $U_v \subseteq G^*$ be the set of vertices that correspond to cells $\mathfrak{c}$ of $\mathcal{G}$ such that v lies on the boundary of $\mathfrak{c}$. We say a cell $\mathfrak{c}$ of $\mathcal{G}$ is *s_it_i-far* if it corresponds to a vertex $v_\mathfrak{c} \in V(G^*)$ at distance more than ℓ_i from U_{s_i} or U_{t_i}, and $\mathfrak{c}$ is *far* if it is s_it_i-far for all $i \in [k]$. Observe that in any solution of SCREI for $\big(G, \mathcal{G}, F, (\ell_i)_{i\in[|F|]}\big)$ no drawing of an edge in F can intersect far cells of $\mathcal{G}$. We refer to maximal unions of far cells in $\mathcal{G}$ which form subsets of $\mathbb{R}^2$ whose interior is connected as *holes*. The interiors of holes are a natural choice for information that is not immediately relevant for the insertion of drawings for F, in the sense that no intersections with these drawings can occur in far cells. However, as mentioned in the Proof Overview, omitting the interior of holes destroys the information about which parts of edges belong to the same edge whenever an edge is disconnected by the removal of a hole.

To transfer this information between different parts of one edge – parts which could be crossed by a hypothetical solution but which are disconnected by holes – we introduce *stitches* into the respective holes. More formally, for a hole H in $\mathcal{G}$ we call an edge e in $E(G)$ *H-torn* if e is split into at least two curves by the removal of the interior of H from $\mathcal{G}$. We call maximal subcurves of an H-torn edge after removing H *(edge) parts* of e and refer to the endpoints of these subcurves as *endpoints* of the corresponding edge part. Stitches will correspond to paths between the endpoints of edge parts of H-torn edges. To construct these paths we introduce so-called *threads* which are edges that we insert into a hole H to connect parts of H-torn edges and derive the stitches from them by considering their planarization.

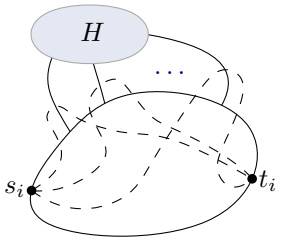


Figure 1 Assuming $\ell_i = 3$, then the potential drawings of s_it_i, depicted as dashed curves, can cross an arbitrary number of H-torn edges.

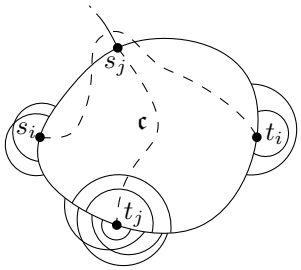


Figure 2 Assuming $\ell_i = \ell_j = 3$, then the drawing of s_jt_j has to go through $\mathfrak{c}$ and the drawing of s_it_i has to revisit cell $\mathfrak{c}$.

To ensure that the obtained combinatorialization of $\mathcal{G}$ has bounded treewidth, the main goal of this section will be to bound the number of stitches for each edge $s_it_i \in F$ and hole H by some function of $k + \ell_i$. We do this by considering which and how many edge parts of H-torn edges any simple s_it_i-curve in a hypothetical solution can cross. Here we face an apparent difficulty: it *is* possible that there is an unbounded number of edge parts which are crossed by drawings of an added edge s_it_i in hypothetical solutions and each edge part belongs to a different H-torn edge (see Figure 1). However, such situations can be safely avoided by restricting our attention to "reasonable" solutions, as we will see in Subsection 3.1. In particular, to specify "reasonable" solutions, we turn our attention to the behavior of drawings of added edges in a hypothetical solution when they *revisit* a cell of $\mathcal{G}$. Then, bounding the number of stitches we introduce for an added edge s_it_i and hole H is equivalent to showing that we can identify all but a bounded number of edges in $E(G)$ which are H-torn and cannot be crossed by a drawing of s_it_i in a "reasonable" hypothetical solution. This is what we focus on in Subsection 3.2.

After adding stitches, we are finally able to define an appropriate combinatorialization of $\mathcal{G}$ in Section 4 which we can use for the final application of Courcelle's theorem in Section 5.

3.1 Detours and Reasonable Solutions

Fix an added edge s_it_i, a hole H, and a cell $\mathfrak{c}$ of the original drawing of G. Note that a drawing of s_it_i in a hypothetical solution might revisit the cell $\mathfrak{c}$ to avoid crossing the drawing of a different added edge s_jt_j. Figure 2 exemplifies such a situation. Understanding how and why a solution might need to revisit a cell is a major component in establishing an upper bound on the number of stitches per hole. In fact, as we will see in this section, avoiding such crossings is the only reason why a cell might have to be revisited.

Let γ_i be a drawing of s_it_i in a hypothetical solution which revisits $\mathfrak{c}$. A $\mathfrak{c}$-*detour* (of γ_i) is a maximal subcurve of γ_i whose interior is disjoint from $\text{int}(\mathfrak{c})$ and has neither s_i nor t_i as an endpoint. Note that a $\mathfrak{c}$-detour might also consist of a singular point. This case occurs when γ_i crosses an edge segment on the boundary of $\mathfrak{c}$ which does not lie on the boundary of another cell. See Figure 3 for an illustration.

▶ **Definition 3.** *Let δ be a $\mathfrak{c}$-detour, and let the embedding $\mathcal{E}$ consist only of δ and the restriction of $\mathcal{G}$ to the boundary of $\mathfrak{c}$. Then δ partitions the boundary of $\mathfrak{c}$ into two connected parts: the part incident to the unbounded (i.e. outer) cell in $\mathcal{E}$, and the* δ-avoided part *which is not incident to the outer cell in $\mathcal{E}$.*

Additionally, we call the subset of $\mathbb{R}^2$ which is enclosed by δ and the δ-avoided part of the boundary of $\mathfrak{c}$ together with the δ-avoided part of the boundary of $\mathfrak{c}$ itself the δ-avoided region.

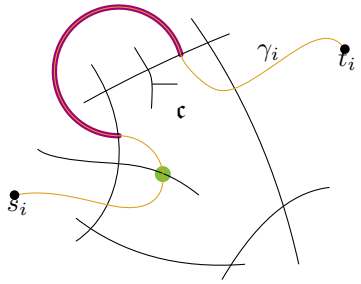


Figure 3 Drawing γ_i of s_it_i in a hypothetical solution with two $\mathfrak{c}$-detours: one is a curve (highlighted in purple) and the other is a point (highlighted in green).

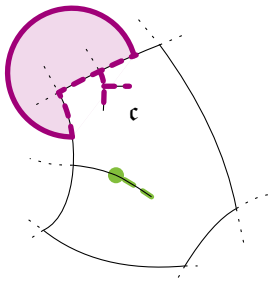


Figure 4 For the single point detour (green), the avoided part of the boundary of $\mathfrak{c}$ and the plane coincide and are dashed green. For the curve detour (purple), the avoided part of the boundary of $\mathfrak{c}$ is dashed purple and the avoided region is shaded purple.

See Figure 4 for an illustration. A $\mathfrak{c}$-detour δ is unremovable *if there exists an added edge s_jt_j with $j \neq i$ such that exactly one of s_j and t_j lies in the δ-avoided region of $\mathcal{G}$. In that case we say that the endpoint (s_j or t_j) in the δ-avoided region is* avoided *by δ, or that δ is* around *the endpoint. We call a $\mathfrak{c}$-detour* removable *if it is not unremovable.*

▶ **Lemma 4 (⋆).** *If there is a solution, then there exists a solution in which no drawing of any added edge contains a removable $\mathfrak{c}'$-detour for any cell $\mathfrak{c}'$ of $\mathcal{G}$.*

Lemma 4 allows us to restrict our attention to solutions which do not contain any removable detours (these are the solutions we intuitively referred to as "reasonable").

3.2 Defining and Finding Stitches

Let $s_it_i \in F$ and H be a hole. Our goal is to compute a, by our parameters, bounded number of edge parts in $E(G)$ which could be crossed by a drawing of s_it_i in some reasonable hypothetical solution. As we obviously do not know any hypothetical solution we cannot compute this set directly. Consequently, we identify and compute a slightly larger set: the set of all edge parts that can be crossed by some so-called *solution curve* for s_i and t_i that is superficially like an s_it_i-curve in a "reasonable" hypothetical solution (but which might induce double-crossings).

▶ **Definition 5.** *A* solution curve *for s_it_i is a simple curve γ_i that (i) starts in s_i and ends in t_i; (ii) produces at most ℓ_i crossings with $\mathcal{G}$; and (iii) whenever γ_i intersects a cell $\mathfrak{c}'$ in more than one maximal connected subcurve there is an added edge s_jt_j with $j \neq i$ such that exactly one of s_j and t_j lies in the ζ-avoided part of $\mathcal{G}$, where ζ is a maximal connected subcurve of γ_i outside of $\mathfrak{c}'$ between two intersections of γ_i with $\mathfrak{c}'$. A part of an H-torn edge $e \in E(G)$ is* crossable *for s_it_i if it is crossed by a solution curve for s_it_i.*

▶ **Lemma 6 (⋆).** *For every hole H and every added edge $s_it_i \in F$ there are less than $\ell_i(2\ell_i+1)! \cdot \left(4k(\ell_i+2)(\ell_i+1)^{\ell_i+1}\right)^{2\ell_i+1}$ parts of H-torn edges that are crossable for s_it_i.*

Proof Sketch. We show that there is a set K of less than $(2\ell_i+1)!\left(4k(\ell_i+2)(\ell_i+1)^{\ell_i+1}\right)^{2\ell_i+1}$ solution curves for s_it_i such that each crossable edge part for s_it_i is crossed by at least one of the curves in K. Then the claim follows as each solution curve crosses at most ℓ_i edges.

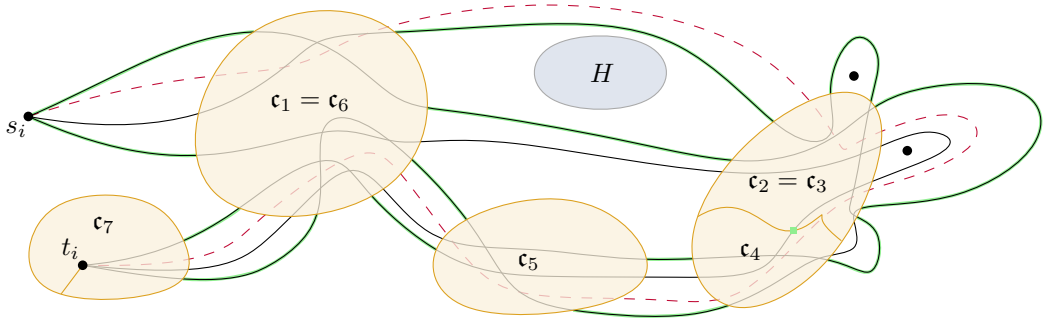


Figure 5 The cells $\mathfrak{c}_1, \ldots, \mathfrak{c}_7$ are in the core of the sunflower. The red dashed s_it_i curve cannot be part of the minimal set of curves K. The extremal subcurves are highlighted in green.

Assume for contradiction that the minimum set K that witnesses crossability of parts of H-torn crossable edges for s_it_i consists of at least $(2\ell_i+1)!\left(4k(\ell_i+2)(\ell_i+1)^{\ell_i+1}\right)^{2\ell_i+1}$ solution curves for s_it_i. Consider the restricted drawing $\mathcal{G}_H$ which is given by $\mathcal{G}$ restricted to the boundary of H, all H-torn edges in $E(G)$, as well as s_i and t_i.

We associate each s_it_i curve in K with the set of cells of $\mathcal{G}_H$ which it intersects and the set of edge segments in $\mathcal{G}_H^\times$ which it crosses. In this way, each curve in K is associated to a set of size at most $2\ell_i+1$. By the minimality of K, no two curves in K are associated to the same set of cells and edge segments. Using the sunflower lemma [22, 23] for the set system given by the sets of cells and edge segments associated to the s_it_i curves in K we obtain a set of at least $4k(\ell_i+2)(\ell_i+1)^{\ell_i+1}$ solution curves $K^\sunflower \subseteq K$ which all intersect pairwise different cells of $\mathcal{G}_H$ and edge segments of $\mathcal{G}_H^\times$, apart from the cells and edge segments in the core of a sunflower, which they all intersect. Moreover, as curves in K intersect at most ℓ_i edges we find at most ℓ_i+1 cells in the core.

By the pigeonhole principle there is a set of at least $4k(\ell_i+2)$ curves in $K^\sunflower$ which all intersect the cells in the core of the sunflower in the same order (taking into account repetitions of cells). Let $K^\sunflower_\sigma \subseteq K^\sunflower$ be such a set of curves and let $\sigma = \mathfrak{c}_1, \ldots, \mathfrak{c}_l$ with $l \leq \ell_i+1$ be the order in which these curves traverse the cells in the core of the sunflower.

As each $\mathfrak{c}_j$ with $j \in [l]$ is a cell in a restriction of $\mathcal{G}$ containing all H-torn edges, no part of an H-torn edge intersects the interior of $\mathfrak{c}_j$. In particular parts of H-torn edges are not crossed by any curve in $K^\sunflower_\sigma$ within $\mathrm{int}(\mathfrak{c}_\mathrm{j})$.

When considering subcurves of curves *between* each $\mathfrak{c}_j$ and $\mathfrak{c}_{j+1}$, we can find at most $4(k-1)$ "extremal" such subcurves which separate all other subcurves from H together with $\mathfrak{c}_j$ and $\mathfrak{c}_{j+1}$. These extremal subcurves together cross any crossable edge part of an H-torn edge intersected by any other considered subcurve. See Figure 5 for an illustration.

In this way we find at least one curve after the removal of which from K the same crossable edge parts of H-torn edges are intersected, contradicting our minimality assumption. ◀

While the fact that the number of crossable edge parts we want to introduce stitches for is bounded by a function in our parameters is reassuring, we need to be able to actually introduce these stitches before being able to give our final MSO encoding of hypothetical solutions. For this we invoke Courcelle's theorem in Lemma 7 independently of its final application. This then allows us to insert the corresponding stitches.

▶ **Lemma 7 (⋆).** *There is a fixed-parameter algorithm parameterized by $k+\ell$ which identifies, for an added edge s_it_i and a hole H, all parts of H-torn edges which are crossable for s_it_i.*

▶ **Definition 8.** *For a hole H in $\mathcal{G}$ and an added edge $s_it_i \in F$, a* thread *is a pair of two endpoints of two distinct edge parts of the same H-torn edge in $e \in E(G)$ satisfying the following properties: (i) both edge parts are crossable for s_it_i, (ii) there is no other crossable edge part between these edge parts along a traversal of e, and (iii) there is no other endpoint of one of the two edge parts along a traversal of e. We denote the set of all threads for H and s_it_i as T_{H,s_it_i}, and define the set of all threads for H as $T_H = \bigcup_{i\in[k]} T_{H,s_it_i}$.*

An embedding of T_H is a set of curves, contained completely in H, which connect each pair of two endpoints of edge parts in T_H.

▶ **Lemma 9 (⋆).** *There is a fixed-parameter algorithm parameterized by $k + \ell$ that computes, for a hole H in $\mathcal{G}$, a simple embedding of T_H.*

For the simple embedding of T_H into H computed in Lemma 9, define the set of *stitches* S_H of H as the planarization of the threads in this embedding.

4 The Patchwork Graph

After identifying a bounded number of stitches in each hole, we are finally able to define the *patchwork graph* and prove desirable properties which we will use in our final application of Courcelle's theorem. An illustration of the patchwork graph is provided in Figure 6. The following definition also doubles as a description of how to construct the patchwork graph from a given drawing. We remark that, unlike $\mathcal{G}$, the patchwork graph might be disconnected.

▶ **Definition 10.** *The* patchwork graph *P and its embedding $\mathcal{P}$ are given by the labeled graph derived from $\mathcal{G}$ in the following steps:*

1. *Planarize $\mathcal{G}$ and label the vertices which are newly introduced by this as* crossing vertices*. Label vertices which correspond to vertices of G as* real vertices*. Additionally label each s_i and t_i with label $i \in [k]$.*
2. *Subdivide each edge e in the planarization $\mathcal{G}^\times$ of $\mathcal{G}$ by k vertices*[3] *$v_1^e, \ldots, v_k^e$ which are labeled as* segment vertices *– each segment vertex of e will represent a possible crossing point of the drawing of one of the k edges in F and e.*
3. *Inside each face f of $\mathcal{G}^\times$, introduce a new vertex v_f and label it as* cell vertex*.*
4. *Inside each face f of $\mathcal{G}^\times$, trace the boundary of f creating a curve at ε-distance and create a vertex labeled as* shadow vertex *on this curve every time an endpoint of an edge in F or a segment vertex is encountered. Insert two edges for each shadow vertex; one connecting the shadow vertex to the corresponding endpoint of an edge in F or segment vertex; and one connecting the shadow vertex to v_f. Note that multiple shadow vertices can be introduced for the same vertex in G (e.g. the orange vertex in Figure 6). Shadow vertices allow to distinguish different ways, more formally positions in the rotation around an endpoint, of accessing that endpoint via the inserted drawing of an edge in F (see Figure 7); this is where the connectivity of $\mathcal{G}$ is used. In this way each shadow vertex of an endpoint corresponds to an* access direction*.*
5. *Delete every vertex that is in the interior of a hole H.*
6. *For each hole H insert all stitches S_H for H into the interior of H and label the inserted vertices as* crossing vertices*.*[4]

[3] If $k = 1$ we subdivide by 2 vertices for reasons that will become clear when we introduce *tracking labels*.
[4] This means they receive the same label as vertices introduced by planarizing $\mathcal{G}$.

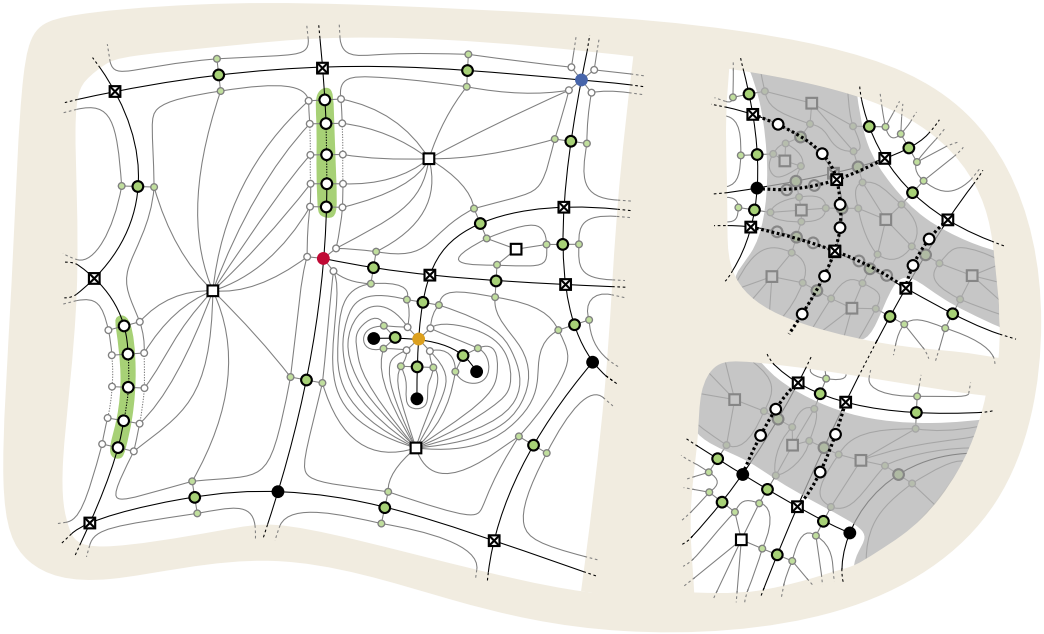


Figure 6 Illustration of a patchwork graph P. The remainder of P is hinted in beige. Black disks are original vertices. Colored disks are endpoints of edges in F. Crossing vertices are crosses. Green and white disks represent the edge segment/shadow vertices. Cell vertices are white squares. Holes are shaded in gray and stitches drawn with thick, dashed curves.

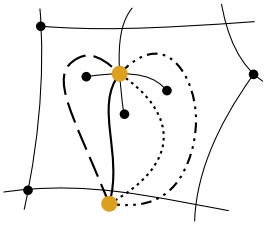


Figure 7 Illustration for different access directions. Each hypothetical drawing (indicated as thick dashes, normal, dotted, and dash-dotted lines) of the added edge between the orange vertices crosses the same edge segment of $\mathcal{G}^\times$ but separates the black vertices differently. In connected initial drawings, ways of separating vertices of the same cell by the drawing of an added edge are completely determined by potential crossing points of that drawing and its positions in the rotations around each of its endpoints. This is not the case for disconnected initial drawings.

7. *For technical reasons which will become apparent later (when we introduce* tracking labels*), we replace each edge in S_H by a path consisting of two vertices and three edges and label the inserted vertices as* segment vertices.[5]

We introduce additional crossability labels for segment vertices in the following way. For every segment vertex v corresponding to an edge segment σ of edge $e \in E(G)$, we label v as *crossable* for some edge $s_it_i \in F$ if one of the following two conditions holds:

- e is not H-torn for any hole H, or
- for *each* hole H in $\mathcal{G}$ for which e is H-torn, σ lies on a part (when considering parts arising from the removal of the interior of H) of e that is crossable for s_it_i.

[5] This means they receive the same label as vertices introduced by subdividing edge segments of the planarization of $\mathcal{G}$.

▶ **Lemma 11 (⋆).** *If there exists a solution for the considered SCREI instance, then there is a solution such that all segment vertices which correspond to edge segments of an edge that is crossed by the drawing of $s_it_i \in F$ in the solution are labeled as crossable for s_it_i.*

Note that Lemmas 7 and 9 and Definition 10 allow us to compute the patchwork graph in FPT time. Two important properties of the patchwork graph are encapsuled in Lemmas 12 and 13. The proof of Lemma 12 relies on obtaining a bound on the diameter of each connected component of the patchwork graph – a task which is intuitively clear, but requires to overcome technical challenges due to the addition of stitches. Lemma 13 later allows us to encode whether two edge segments in P belong to the same edge in $\mathcal{G}$ via an MSO formulation.

▶ **Lemma 12 (⋆).** *The patchwork graph P has treewidth bounded by $3(2 + 4(k-1))(4\ell + 8(kf(k,\ell)-1))$, where $f(k,\ell)$ is the bound on the number of crossable edge parts for a single added edge and hole obtained in Lemma 6.*

▶ **Lemma 13 (⋆).** *Segment vertices which correspond to edge segments of the same edge in $e \in E(G)$ and are labeled as crossable for s_it_i are connected via paths in P consisting only of segment and crossing vertices which correspond to segments and crossings of e and segments and crossings for threads that connect parts of e.*

Ideally, we would like Lemma 13 to lead to an MSO subformula that can check whether two segment vertices in P belong to the same edge – an important component of our algorithm for SCREI. The lemma provides us with a characterization that seems suitable for this task since it is easy to define a path in MSO, but there is an issue if we use P as it is currently defined: a crossing vertex is adjacent to 4 segment vertices, and P (viewed as a graph without an embedding) does not specify which of these segment vertices belong to the same edge. We resolve this by introducing *tracking labels*: for each crossing vertex v in P created by a crossing between edges e and e' in $\mathcal{G}$, we assign the label 1 to the two unique neighbors of v corresponding to e and the label 2 to the remaining two neighbors of v.

▶ **Corollary 14.** *Segment vertices which correspond to edge segments of the same edge in $e \in E(G)$ and are labeled as crossable for s_it_i are connected via paths in P consisting only of segment and crossing vertices with the following property: the two neighbors of each crossing vertex on the path are segment vertices with the same tracking label.*

5 Using the Patchwork Graph

Now that we have constructed the patchwork graph P and established that it has the properties we need, we can proceed to the final stage of our proof. Here, our aim will be to identify a combinatorial characterization which projects the behavior of a solution from $\mathcal{G}$ to P, establish a procedure that allows us to identify (and construct) solutions based on a characterization in P, and finally show how to find such characterizations. To streamline our presentation, at this stage we perform a brute-force branching procedure which will determine, for each $s_it_i \in F$, the number ℓ'_i of crossings between the curve connecting s_i to t_i and edges of $\mathcal{G}$ in the sought-after solution.

Consider a hypothetical solution S, and let f be a curve in S connecting vertex a to b. The *trace* r_f of f is a walk in P starting at a such that:

1. From a, r_f proceeds to the shadow vertex that corresponds to the access direction through which f connects to a, and then to the cell vertex of the first cell $\mathfrak{c}_1$ in $\mathcal{G}$ intersecting f.
2. For each intersection along f with an edge segment q between cells $\mathfrak{c}_i$ and $\mathfrak{c}_{i+1}$, r_f proceeds to the shadow vertex of a segment vertex v in $\mathfrak{c}_i$ on q, then to v, then to its shadow vertex in $\mathfrak{c}_{i+1}$, and then to the cell vertex of $\mathfrak{c}_{i+1}$, where v has the property that the number of

segment vertices of q on either side of v is at least as large as the number of drawings of added edges in F which intersect q on the respective side of its intersection with f. Such a segment vertex v exists, since there are $k = |F|$ segment vertices on q.

3. Finally, r_f continues to the shadow vertex that *corresponds* to the direction through which f enters b, and finally ends in b.

Observe that r_f visits precisely $4\ell'_i + 5$ vertices. Moreover, for two curves f, f' in S, their traces $r_f, r_{f'}$ may only intersect in cell vertices, the real vertices that form the endpoints of the curves, and the associated shadow vertices.

Now, let the *solution trace* (r_S,η_S) of S be a pair where $r_S = \{r_f | f \in S\}$ and η_S describes cyclic orders which will intuitively capture how edges cross into and out of each cell vertex in the solution. Let $R_S = \{v \mid \exists f \in S : v \in r_f\}$ be the set of all vertices occurring in the traces of S. η_S then is a mapping from each cell vertex $c \in R_S$ to a cyclic order $\prec_c$ over the shadow vertices in R_S that are incident to c. Specifically, $\prec_c$ is defined as the cyclic order given by the cycle on the neighborhood of c in P restricted to R_S.

Solution traces describe the way in which a solution can be related to a set of walks and cyclic orders in P. Of course we can abstract away from the explicit reference to a solution and define the more general notion of *preimages* whose combinatorial structure is the same as that of a solution trace but which does not arise and in particular does not even need to correspond to a solution. (Preimages and solution traces relate in a similar way as solution curves and solutions in Section 3.2.)

Formally, a *preimage* (α', β') is a tuple with the following properties. α' is a set of k walks in H which are labeled $\alpha'_1, \ldots, \alpha'_k$, where each α'_i has length $4(\ell_i + 1)$ and visits vertices with the same orders of labels as traces. Similarly, β' is a mapping from each cell vertex c visited by the walks in α' to the cyclic order over its neighbors that occur in α', along the cycle on $N_P(c)$ in P.

Obviously every solution trace is a preimage. Conversely, one can derive a drawing of all edges of F into $\mathcal{G}$ from a preimage (α', β') by the *assembly procedure* **A** introduced below. For each $\alpha'_i \in \alpha'$, **A** will draw a curve u_i that starts and ends at the two vertices labeled i (i.e., the endpoints of $s_i t_i \in F$) as described in the following steps.

1. u_i exits its starting vertex via the access direction given by the first shadow vertex in α'_i.
2. For each cell vertex c such that (e_1, v_1, c, v_2, e_2) forms a subsequence of visited vertices in α'_i, expand u_i by drawing a curve ι in c connecting the edge segment (or the real vertex) e_1 to the edge segment (or the real vertex) e_2 in the following way.
   - Consider an arbitrary other curve drawn in c by **A** up to now, say ζ, that was obtained from some subsequence $(e_1^\zeta, v_1^\zeta, c, v_2^\zeta, e_2^\zeta)$. ι will intersect ζ if and only if the shadow vertices of ι interleave with the shadow vertices of ζ in $\beta(c)$ (i.e., for instance, if $v_1 \prec_c v_1^\zeta \prec_c v_2 \prec_c v_2^\zeta \prec_c v_1$).
   - Such a drawing can be achieved by, e.g., having the curve ι follow the inside boundary of c in a clockwise manner while avoiding all curves it is not supposed to cross (as these will be either completely enveloped by or completely enveloping ι).
   - We remark that v_1 and v_2 may either be shadows of segment vertices or the actual endpoints s_i or t_i.
3. u_i ends by entering the final real vertex in α'_i from the direction specified by the last shadow vertex in α'_i.

The intuition here is that **A** interprets a preimage of a template trace as a specification of precisely which parts of $\mathcal{G}$ should be crossed by the drawings of each added edge (this information is provided in α'), while controlling when and how individual curves in the newly constructed solutions should cross each other (this information is provided in β'). Note that the output of **A** for an arbitrary preimage will in general not be a solution for our edge insertion problem, but – crucially – one can check whether it is in polynomial time.

Observe that, although preimages imply curves in $\mathcal{G}$ for all added edges in F, and we can check for each of them if they are a solution, we cannot iterate over them in FPT time as the number of preimages in P is generally not FPT. We will however be able to distill the structure of preimages, independently of their exact specification in P. For this we define *template traces*. A template trace is a tuple $\tau = (T, \alpha, \beta)$ where:

- T is a graph whose vertices are equipped with a labeling that matches the vertex-labeling used in P (i.e., some may be labeled as segment vertices, some as cell vertices, etc., and in addition some of them may be labeled as the endpoints of added edges in F);
- $\alpha = \{\alpha_1, \ldots, \alpha_k\}$ is a set of walks in T, where each walk α_i has length $4(\ell'_i + 1)$ and the types of vertices visited by α_i match the types of vertices visited by a trace (i.e., α_i starts with a real vertex labeled i, then proceeds with a shadow vertex, a cell vertex, followed by a sequence of ℓ'_i-many subsequences of shadow-, segment-, shadow-, cell vertices, and ends with a shadow vertex followed by a different real vertex labeled i); and
- β is a mapping from each cell vertex in T to a cyclic order over its adjacent shadow vertices.
- For simplicity, we require that each vertex and edge in T occurs in at least one walk in α.

▶ **Proposition 15 (⋆).** *There are at most $(k\ell)^{\mathcal{O}(k\ell)}$ distinct template traces. Moreover, the set of all template traces can be enumerated in time $(k\ell)^{\mathcal{O}(k\ell)}$.*

We say that a template trace (T, α, β) *matches* a preimage (α', β') if there is a label-preserving bijective mapping γ (called the *preimaging*) from the vertices on walks in α' to $V(T)$ such that (1) for each $\alpha'_i \in \alpha'$, $\gamma(\alpha'_i) = \alpha_i$ and (2) $\gamma(\beta')$ maps each c to $\beta(\gamma(c))$. For a template trace τ that matches a preimage (α', β'), we say that (α', β') is a *preimage* of τ. Intuitively, a preimage of a template trace is its firmly embedded counterpart in P. As every solution trace is a preimage, these definitions carry over to solution traces.

The following lemma shows that a template trace τ matching the solution trace of a hypothetical solution contains a sufficient amount of information to *almost* reconstruct a solution using $\mathbf{A}$ on a preimage of τ.

▶ **Lemma 16 (⋆).** *Let S be a solution which matches a template trace $\tau = (T, \alpha, \beta)$, and let (α', β') be a preimage of τ. Let S' be the output of $\mathbf{A}$ applied to (α', β'). Then S' is either a solution, or there exists an edge e of G that intersects some curve in S' more than once.*

Next, we show that the problem of finding a preimage of a template trace (or determining that there is none) can be encoded in Monadic Second Order (MSO) logic. Which is the last ingredient needed to prove our main result.

▶ **Lemma 17 (⋆).** *Let $\tau = (T, \alpha, \beta)$ be a template trace. There exists an MSO formula $\phi_\tau(V(T))$ of size independent of G and $\mathcal{G}$ which is satisfiable in P if and only if there exists a preimage for τ in P. Moreover, if the formula is true, then the interpretation of $V(T)$ defines a* preimaging *between a preimage of τ and τ.*

▶ **Theorem 18 (⋆).** *SCREI is fixed-parameter tractable.*

Theorem 18 implies the fixed-parameter tractability of SCEI and SLCEI parameterized by $k + \ell$. Moreover, the approach can also be used to obtain fixed-parameter tractability of the other problems defined in the introduction, with only minor adaptations required.

▶ **Theorem 19 (⋆).** *Sℓ-PEI, ℓ-PEI and Locally Crossing-Minimal Edge Insertion are fixed-parameter tractable when parameterized by $\ell + k$.*

6 Inserting a Single Edge

In this section we present a single-exponential fixed-parameter algorithm for SCEI parameterized by ℓ in the case where $|F| = 1$; we hereinafter denote this problem SC1EI. We remark that the parameter dependency of this algorithm is tight under the Exponential Time Hypothesis [27], since Arroyo et al. [7] gave a reduction from 3-SAT to the simple drawing extension problem with one extra edge, and the number of edges in the obtained graphs is linear in the size of the 3-SAT instance. We note that in the same work [7], the authors also presented a single-exponential parameterized algorithm for SCEI when $|F| = 1$, however the parameter used there is the total number of crossings in the original drawing.

As a first step we transform SC1EI to the problem of finding a colorful st-path (i.e., a path where no color is repeated) of length at most κ in a vertex-colored graph with coloring χ obtained from $\mathcal{G}^\times$. Using so-called *representative sets*, see e.g. [17, Chapter 12], we can show how to efficiently find a colorful path. Theorem 22 is then an immediate consequence.

▶ **Proposition 20 (⋆).** *There is a linear-time reduction that converts an instance $(G, \mathcal{G}, \{st\}, \ell)$ of SC1EI to an equivalent instance $(G^*, \chi, s, t, 2\ell + 3)$ of* Colorful Short Path.

▶ **Theorem 21 (⋆).** *Colorful Short Path can be solved in time $\mathcal{O}(2^{\mathcal{O}(\kappa)} \cdot |E(G)| \log |V(G)|)$.*

▶ **Theorem 22.** *SC1EI can be solved in time $\mathcal{O}(2^{\mathcal{O}(\ell)} \cdot |\mathcal{G}| \log |E(G)|)$.*

7 Conclusion

In this paper we established the fixed-parameter tractability of inserting a given set of edges into a given drawing while maintaining simplicity and adhering to various restrictions on the number of crossings in the solution. While the presented results make the reasonable assumption that the initial drawing is connected, the problem is of course also interesting in the general case. We believe that our framework and methodology can also be used to handle the extension problem for disconnected drawings, albeit only after overcoming a few additional technical challenges; moreover, the algorithm presented in Section 6 does not require connectivity at all. Other than connectivity, the most glaring question left open concerns the complexity of SCEI parameterized by ℓ alone. Recall that, in contrast to this open question, SLCEI is known to be NP-hard already for $\ell = 1$. Last but not least, while here we focused on the edge insertion problem, it would also be interesting to extend the scope to also allow for the addition of vertices into the drawing.

References

1 Patrizio Angelini, Giuseppe Di Battista, Fabrizio Frati, Vít Jelínek, Jan Kratochvíl, Maurizio Patrignani, and Ignaz Rutter. Testing planarity of partially embedded graphs. *ACM Transactions on Algorithms*, 11(4):32:1–32:42, 2015. doi:10.1145/2629341.

2 Patrizio Angelini, Michael A. Bekos, Franz J. Brandenburg, Giordano Da Lozzo, Giuseppe Di Battista, Walter Didimo, Michael Hoffmann, Giuseppe Liotta, Fabrizio Montecchiani, Ignaz Rutter, and Csaba D. Tóth. Simple k-planar graphs are simple $(k+1)$-quasiplanar. *Journal of Combinatorial Theory, Series B*, 142:1–35, 2020. doi:10.1016/j.jctb.2019.08.006.

3 Patrizio Angelini, Ignaz Rutter, and Sandhya T. P. Extending Partial Orthogonal Drawings. In *Proceedings of the 28th International Symposium on Graph Drawing and Network Visualization (GD'20)*, volume 12590 of *LNCS*, pages 265–278. Springer, 2020. doi:10.1007/978-3-030-68766-3_21.

4 Stefan Arnborg, Jens Lagergren, and Detlef Seese. Easy problems for tree-decomposable graphs. *Journal of Algorithms*, 12(2):308–340, 1991. doi:10.1016/0196-6774(91)90006-K.

5 Alan Arroyo, Julien Bensmail, and R. Bruce Richter. Extending drawings of graphs to arrangements of pseudolines. In *Proceedings of the 36th International Symposium on Computational Geometry (SoCG'20)*, volume 164 of *LIPIcs*, pages 9:1–9:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPIcs.SoCG.2020.9.

6 Alan Arroyo, Martin Derka, and Irene Parada. Extending simple drawings. In *Proceedings of the 27th International Symposium on Graph Drawing and Network Visualization (GD'19)*, volume 11904 of *LNCS*, pages 230–243. Springer, 2019. doi:10.1007/978-3-030-35802-0_18.

7 Alan Arroyo, Fabian Klute, Irene Parada, Raimund Seidel, Birgit Vogtenhuber, and Tilo Wiedera. Inserting one edge into a simple drawing is hard. In *Proceedings of the 46th International Workshop on Graph-Theoretic Concepts in Computer Science (WG'20)*, volume 12301 of *LNCS*, pages 325–338. Springer, 2020. doi:10.1007/978-3-030-60440-0_26.

8 Alan Arroyo, Dan McQuillan, R. Bruce Richter, and Gelasio Salazar. Levi's lemma, pseudolinear drawings of K_n, and empty triangles. *Journal of Graph Theory*, 87(4):443–459, 2018. doi:10.1002/jgt.22167.

9 Guido Brückner and Ignaz Rutter. Partial and constrained level planarity. In *Proceedings of the 28th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA'17)*, pages 2000–2011. SIAM, 2017. doi:10.1137/1.9781611974782.130.

10 Christoph Buchheim, Markus Chimani, Carsten Gutwenger, Michael Jünger, and Petra Mutzel. Crossings and planarization. In Roberto Tamassia, editor, *Handbook on Graph Drawing and Visualization*, pages 43–85. Chapman and Hall/CRC, 2013.

11 Jean Cardinal and Stefan Felsner. Topological drawings of complete bipartite graphs. *Journal of Computational Geometry*, 9(1):213–246, 2018. doi:10.20382/jocg.v9i1a7.

12 Erin W. Chambers, David Eppstein, Michael T. Goodrich, and Maarten Löffler. Drawing graphs in the plane with a prescribed outer face and polynomial area. *Journal of Graph Algorithms and Applications*, 16(2):243–259, 2012. doi:10.7155/jgaa.00257.

13 Timothy M. Chan, Fabrizio Frati, Carsten Gutwenger, Anna Lubiw, Petra Mutzel, and Marcus Schaefer. Drawing partially embedded and simultaneously planar graphs. *Journal of Graph Algorithms and Applications*, 19(2):681–706, 2015. doi:10.7155/jgaa.00375.

14 Markus Chimani, Carsten Gutwenger, Petra Mutzel, and Christian Wolf. Inserting a vertex into a planar graph. In *Proceedings of the 20th Annual ACM-SIAM Symposium on Discrete Algorithms, (SODA'09)*, pages 375–383. SIAM, 2009.

15 Markus Chimani and Petr Hlinený. Inserting multiple edges into a planar graph. In *Proceedings of the 32nd International Symposium on Computational Geometry (SoCG'16)*, volume 51 of *LIPIcs*, pages 30:1–30:15. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2016. doi:10.4230/LIPIcs.SoCG.2016.30.

16 Bruno Courcelle. The monadic second-order logic of graphs I: recognizable sets of finite graphs. *Information and Computation*, 85(1):12–75, 1990. doi:10.1016/0890-5401(90)90043-H.

17 Marek Cygan, Fedor V. Fomin, Lukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michal Pilipczuk, and Saket Saurabh. *Parameterized Algorithms*. Springer, 2015. doi:10.1007/978-3-319-21275-3.

18 Reinhard Diestel. *Graph Theory, 5th Edition*, volume 173 of *Graduate texts in mathematics*. Springer, 2012.

19 Rodney G. Downey and Michael R. Fellows. *Fundamentals of Parameterized Complexity*. Texts in Computer Science. Springer, 2013. doi:10.1145/2744447.2744454.

20 Eduard Eiben, Robert Ganian, Thekla Hamm, Fabian Klute, and Martin Nöllenburg. Extending partial 1-planar drawings. In *Proceedings of the 47th International Colloquium on Automata, Languages, and Programming (ICALP'20)*, volume 168 of *LIPIcs*, pages 43:1–43:19. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPIcs.ICALP.2020.43.

21 Paul Erdős and Richard K. Guy. Crossing number problems. *The American Mathematical Monthly*, 80(1):52–58, 1973. doi:10.1080/00029890.1973.11993230.

22 Paul Erdős and Richard Rado. Intersection theorems for systems of sets. *Journal of the London Mathematical Society*, 1(1):85–90, 1960.

23 Jörg Flum and Martin Grohe. *Parameterized Complexity Theory*, volume XIV of *Texts in Theoretical Computer Science. An EATCS Series*. Springer, Berlin, 2006. doi:10.1007/3-540-29953-X.

24 Robert Ganian, Thekla Hamm, Fabian Klute, Irene Parada, and Birgit Vogtenhuber. Crossing-optimal extension of simple drawings. *CoRR*, abs/2012.07457, 2020. arXiv:2012.07457.

25 Péter Hajnal, Alexander Igamberdiev, Günter Rote, and André Schulz. Saturated simple and 2-simple topological graphs with few edges. *Journal of Graph Algorithms and Applications*, 22(1):117–138, 2018. doi:10.7155/jgaa.00460.

26 Heiko Harborth. Empty triangles in drawings of the complete graph. *Discrete Mathematics*, 191(1-3):109–111, 1998. doi:10.1016/S0012-365X(98)00098-3.

27 Russell Impagliazzo, Ramamohan Paturi, and Francis Zane. Which problems have strongly exponential complexity? *Journal of Computer and System Sciences*, 63(4):512–530, 2001. doi:10.1006/jcss.2001.1774.

28 Jan Kynčl, János Pach, Radoš Radoičić, and Géza Tóth. Saturated simple and k-simple topological graphs. *Computational Geometry: Theory and Application*, 48(4):295–310, 2015. doi:10.1016/j.comgeo.2014.10.008.

29 Jan Kynčl. Enumeration of simple complete topological graphs. *European Journal of Combinatorics*, 30(7):1676–1685, 2009. doi:10.1016/j.ejc.2009.03.005.

30 Jan Kynčl. Simple realizability of complete abstract topological graphs simplified. *Discrete and Computational Geometry*, 64(1):1–27, 2020. doi:10.1007/s00454-020-00204-0.

31 Giordano Da Lozzo, Giuseppe Di Battista, and Fabrizio Frati. Extending upward planar graph drawings. *Computational Geometry: Theory and Applications*, 91:101668, 2020. doi:10.1016/j.comgeo.2020.101668.

32 Tamara Mchedlidze, Martin Nöllenburg, and Ignaz Rutter. Extending convex partial drawings of graphs. *Algorithmica*, 76(1):47–67, 2016. doi:10.1007/s00453-015-0018-6.

33 János Pach. Geometric graph theory. In Csaba D. Tóth, Joseph O'Rourke, and Jacob E. Goodman, editors, *Handbook of Discrete and Computational Geometry, Third Edition*, pages 257–279. CRC press, 2017. doi:10.1201/9781315119601.

34 Maurizio Patrignani. On extending a partial straight-line drawing. *International Journal of Foundations of Computer Science*, 17(5):1061–1070, 2006. doi:10.1142/S0129054106004261.

35 Neil Robertson and Paul D. Seymour. Graph minors. III. Planar tree-width. *Journal of Combinatorial Theory, Series B*, 36(1):49–64, 1984. doi:10.1016/0095-8956(84)90013-3.

36 Marcus Schaefer. *Crossing numbers of graphs*. CRC Press, 2018. doi:10.1201/9781315152394.

37 Thomas Ziegler. *Crossing minimization in automatic graph drawing*. PhD thesis, Saarland University, Saarbrücken, Germany, 2001.

Quantum Logspace Algorithm for Powering Matrices with Bounded Norm

Uma Girish
Department of Computer Science, Princeton University, NJ, USA

Ran Raz
Department of Computer Science, Princeton University, NJ, USA

Wei Zhan
Department of Computer Science, Princeton University, NJ, USA

Abstract

We give a quantum logspace algorithm for powering contraction matrices, that is, matrices with spectral norm at most 1. The algorithm gets as an input an arbitrary $n \times n$ contraction matrix A, and a parameter $T \leq \text{poly}(n)$ and outputs the entries of A^T, up to (arbitrary) polynomially small additive error. The algorithm applies only unitary operators, without intermediate measurements. We show various implications and applications of this result:

First, we use this algorithm to show that the class of quantum logspace algorithms with only quantum memory and with intermediate measurements is equivalent to the class of quantum logspace algorithms with only quantum memory without intermediate measurements. This shows that the deferred-measurement principle, a fundamental principle of quantum computing, applies also for quantum logspace algorithms (without classical memory). More generally, we give a quantum algorithm with space $O(S + \log T)$ that takes as an input the description of a quantum algorithm with quantum space S and time T, with intermediate measurements (without classical memory), and simulates it unitarily with polynomially small error, without intermediate measurements.

Since unitary transformations are reversible (while measurements are irreversible) an interesting aspect of this result is that it shows that any quantum logspace algorithm (without classical memory) can be simulated by a reversible quantum logspace algorithm. This proves a quantum analogue of the result of Lange, McKenzie and Tapp that deterministic logspace is equal to reversible logspace [15].

Finally, we use our results to show non-trivial classical simulations of quantum logspace learning algorithms.

2012 ACM Subject Classification Theory of computation → Quantum complexity theory

Keywords and phrases BQL, Matrix Powering, Quantum Circuit, Reversible Computation

Digital Object Identifier 10.4230/LIPIcs.ICALP.2021.73

Category Track A: Algorithms, Complexity and Games

Related Version *Full Version*: `https://eccc.weizmann.ac.il/report/2020/087/`

Funding Research supported by the Simons Collaboration on Algorithms and Geometry, by a Simons Investigator Award and by the National Science Foundation grant No. CCF-1714779.

Acknowledgements We would like to thank Dieter van Melkebeek and Subhayan Roy Moulik for very helpful suggestions and comments on a previous version of this work. We also thank the anonymous reviewers for their thorough feedback.

1 Introduction

Quantum computers hold great promise, but in the near future their memory is likely to be limited to a small number of qubits. This motivates the study of quantum complexity classes with bounded space. The most important of these classes is the class of problems solvable in quantum logarithmic space and polynomial time, first studied by Watrous [28].

In the literature, there are several variants of this class. One variant, BQL, is the class of problems solvable in quantum logarithmic space and polynomial time when intermediate measurements are allowed. Another variant, $\mathsf{BQ_UL}$, is the class of problems solvable in quantum logarithmic space and polynomial time when only unitary operators are allowed and intermediate measurements are not allowed. We note that in most previous works, the class BQL allows a quantum algorithm to use both quantum and classical memory (see for example [17, 27, 6]).

Our first main result, Theorem 18, gives a quantum logspace algorithm for powering matrices, a fundamental problem in computational complexity, which is not known to be in classical (deterministic or probabilistic) logspace. Our algorithm uses only unitary operators, without intermediate measurements, and hence it places the problem of powering matrices in the class $\mathsf{BQ_UL}$.

The algorithm gets as an input an arbitrary $n \times n$ matrix A, a parameter $T \leq \mathrm{poly}(n)$ and two indices $i, j \in \{1, \ldots, n\}$ and outputs the entry $(A^T)_{i,j}$, up to an additive error of $\frac{\|A\|^T}{\mathrm{poly}(n)} + \frac{1}{\mathrm{poly}(n)}$, where $\|A\|$ is the spectral norm of the matrix A. In particular, if A is a contraction matrix, that is, a matrix with spectral norm at most 1, the additive error is just $\frac{1}{\mathrm{poly}(n)}$.

We note that by an easy reduction, our algorithm can also solve another fundamental problem in computational complexity, the problem of iterative matrix multiplication. In this problem, the input is T matrices $A_1, \ldots, A_T$ of size $n \times n$ each, and the algorithm outputs the entries of the product $A_1 \cdot \ldots \cdot A_T$.

Besides giving a quantum logspace algorithm for a basic computational problem, our results shed light on several fundamental issues regarding bounded-space quantum computations, and have additional applications.

$\mathsf{BQ_QL}$ is Equal to $\mathsf{BQ_UL}$

We consider the class of quantum logspace algorithms with only quantum memory and with intermediate measurements and refer to it by $\mathsf{BQ_QL}$. We use our algorithm for powering matrices to show that the two classes $\mathsf{BQ_QL}$ and $\mathsf{BQ_UL}$ are exactly equal (Theorem 12). Moreover, the way that this equality is proved is by a *simulation*. Our second main result, Theorem 16, proves that there is a quantum logspace algorithm without intermediate measurements, that is, a $\mathsf{BQ_UL}$ algorithm, that gets the description of a quantum logspace algorithm with intermediate measurements, without classical memory, that is, a $\mathsf{BQ_QL}$ algorithm, and simulates it with polynomially small error. Theorem 16 is even more general and shows how to simulate a quantum logspace algorithm with *unital* channels that are given as an input, while even the very restricted special case of simulating an arbitrary unitary operator within $\mathsf{BQ_UL}$ seems to us interesting.

The Deferred-Measurement Principle

The deferred measurement principle is a fundamental result in quantum computing which states that delaying measurements until the end of a computation doesn't affect the output. In order for the principle to hold, the qubits that were supposed to be measured cannot further participate in the computation from that point on. However, a $\mathsf{BQ_QL}$ algorithm can only store a logarithmic number of qubits, while the number of intermediate measurements is potentially polynomial, and hence excluding the qubits that are supposed to be measured from further participating in the computation is infeasible.

Nevertheless, Theorem 12 and Theorem 16 imply that intermediate measurements are not necessary even when the space used by the quantum algorithm is logarithmic, but the way to eliminate the intermediate measurements is not as straightforward.

Reversible Computation

Landauer introduced the concept of time-reversible computation and argued that any irreversible operation must be accompanied by entropy increase [14] (see also [2]). An interesting aspect of Theorem 12 and Theorem 16 is that they show that any quantum logspace algorithm (without classical memory) can be implemented using only time-reversible operations (except for the final measurement that gives the final output). This is a quantum analogue of the result of Lange, McKenzie and Tapp that deterministic logspace algorithms can be implemented using only time-reversible operations [15].

Classical Simulations of Quantum Learning with Bounded Memory

A line of recent works studied the power of (classical) algorithms for online learning, under memory constraints, where a bounded-space learner tries to learn a concept class from a stream of samples. These works showed that for a large class of online learning problems, any classical learning algorithm requires either super-linear memory size or a super-polynomial number of samples (see for example [24, 26, 22, 13, 21, 18, 1, 8] and the references therein).

Here, we study the relative power of quantum and classical algorithms for online learning, under memory constraints. More concretely, we study the task of distinguishing between two families of distributions over the possible samples. Corollary 23 proves that any quantum algorithm with time T and space S for distinguishing between arbitrary two families of distributions, can be simulated classically in time $\mathrm{poly}(2^{S^2+\log^2 T})$ and space $O(S^2+\log^2 T)$. Moreover, Theorem 24 proves that if one family is a singleton, that is, the task is to distinguish between one distribution over the samples and a family of different distributions, then any quantum learning algorithm with time T and space S can be simulated classically in time $\mathrm{poly}(2^S \cdot T)$ and space $O(S+\log T)$.

Thus, an intriguing open problem is whether any quantum algorithm with time T and space S for distinguishing between two arbitrary families of distributions, can be simulated classically in time $\mathrm{poly}(2^S \cdot T)$ and space $O(S+\log T)$. Theorem 22 proves that this holds if and only if $\mathsf{promiseBQ_UL} = \mathsf{promiseBPL}$.

1.1 Techniques

We start by proving a lemma that shows how to implement an arbitrary contraction matrix A as a subsystem of a unitary quantum circuit (Lemma 6). Since A is not necessarily unitary, rather than implementing A, the lemma implements the unitary matrix

$$U_H = \begin{pmatrix} H & \sqrt{\mathbf{I}_{2m} - H^2} \\ \sqrt{\mathbf{I}_{2m} - H^2} & -H \end{pmatrix}$$

where H is the Hermitian contraction $\begin{pmatrix} & A \\ A^\dagger & \end{pmatrix}$. That is, the lemma shows how to apply the transformation U_H on a unit vector (quantum state) that is also given as an input. The unitary matrix U_H is called a block-encoding of A in some literature [4, 10], which admits various constructions (see [9] for an exhibition). In particular, our construction in Lemma 6 is in unitary quantum logspace.

The proof of Lemma 6 is inspired by, and uses techniques from, Ta-Shma's algorithm that inverts well-conditioned matrices in quantum logspace [27], whose general framework traces back to [12]. In particular, as in [27], the proof goes according to the following lines: Given a Hermitian matrix H,

- First apply the phase estimation over the unitary e^{iH} so that it maps $|u_\lambda\rangle$ to $|u_\lambda\rangle|\lambda\rangle$, where u_λ is an eigenvector of H with eigenvalue λ.
- For each eigenvector apply the unitary transformation $|\lambda\rangle \to \lambda|0\rangle|\lambda\rangle + \sqrt{1-\lambda^2}|1\rangle|\lambda\rangle$ according to the eigenvalue λ. This is where contraction matrices come into play, as the eigenvalues of H are required to be in $[-1, 1]$.
- Uncompute the eigenvalues by reversing the phase estimation over e^{iH}.

As a special case of Lemma 6, when we take the contraction A to be unitary, we get a space-efficient unitary implementation of any unitary matrix (Corollary 7).

We get our algorithms for powering contraction matrices (Theorem 10 and Corollary 15) by iteratively applying the unitary matrix U_H of Lemma 6. However, since Lemma 6 implements the matrix U_H, rather than A, we need to "throw away" the unwanted dimensions introduced by U_H, by permuting them into additional dimensions.

We get our algorithm for powering arbitrary matrices (Theorem 18), by a reduction to the algorithm for powering contraction matrices, by dividing the matrix by its norm. However, the known algorithm for computing the spectral norm of a matrix, by Ta-Shma [27], only works for contraction matrices. To bypass this, we apply Ta-Shma's algorithm on the matrix A divided by its Frobenius norm (which is always larger than the spectral norm).

Finally, we get our algorithms for simulating quantum logspace algorithms with intermediate measurements, or even *unital* channels that are given as an input (Lemma 11 and Theorem 16), by reducing any unital quantum algorithm to the contraction powering problem in the m^2-dimensional space of the $m \times m$ entries of the density matrix, where $m = 2^S$ and S is the space used by the algorithm. Note that this step already doubles the space used. At the end of this step, we only get polynomially small success probability, but that success probability can be amplified to a constant using a Grover-type technique inspired by [6], resulting in Lemma 11 that simulates the computation with constant error. The error is further reduced to be polynomially small in Theorem 16. Interestingly, to reduce the error and prove Theorem 16, we use Theorem 12, so, in a way, the results are used to improve themselves.

1.2 Related Work

Independently of our work, Fefferman and Remscrim have proven closely related results to ours [7]. They took a different route from ours by proving L-reductions between several well-conditioned versions of matrix problems which turned out to be $\mathsf{BQ_UL}$-complete. In particular, they obtained a stronger version of our Theorem 12, showing that $\mathsf{BQL} = \mathsf{BQ_UL}$.

2 Preliminaries

For an integer n, let $[n] = \{0, 1, \ldots, n-1\}$. Let $\mathbb{C}$ denote the set of complex numbers, and $\mathbb{C}^{m\times n}$ denote the set of m by n complex matrices. For a matrix $A \in \mathbb{C}^{m\times n}$, let $\mathrm{vec}(A)$ be the vectorization of A, which is a vector of dimension mn formed by stacking the columns of A on top of each other, that is

$$\mathrm{vec}(A)_{i+jm} = A_{i,j}, \quad \forall i \in [m], j \in [n].$$

Let $\mathcal{U}_m$ be the set of m by m unitary matrices, and $\mathcal{D}_m$ be the set of m by m density matrices, i.e. positive semidefinite Hermitians of trace 1. The m by m identity matrix is denoted by $\mathbf{I}_m$. Let $\|A\|$ denote the spectral norm of a complex matrix A, and $\|A\|_{\mathrm{F}}$ denote the Frobenius norm.

We use ε to denote small real numbers, and $|\epsilon\rangle$ to denote vectors with small norms. When we talk about errors, approximations and ε-closeness of matrices, they are measured in spectral norms.

As we work mostly with complex numbers, we often need corresponding concentration bounds. The following is a direct corollary of the Chernoff-Hoeffding inequality:

▶ **Lemma 1.** *Let X be a random complex number with $|X| \leq 1$, and $X_1, \ldots, X_n$ are n independent copies of X. Then*

$$\Pr\left[\left|\frac{1}{n}(X_1 + \cdots + X_n) - \mathbf{E}[X]\right| \geq \varepsilon\right] \leq 4e^{-2n\varepsilon^2}.$$

2.1 Contraction Matrices

We introduce contraction matrices and provide some useful properties, which can be found in [29, Chapter 6]:

▶ **Definition 2.** *A matrix $A \in \mathbb{C}^{m\times m}$ is a contraction if $\|A\| \leq 1$. Alternatively, A is a contraction if A is in the convex hull of $\mathcal{U}_m$.*

Any eigenvalue λ of a contraction must have $|\lambda| \leq 1$. If $A \in \mathbb{C}^{m\times m}$ is a contraction, then the following matrix is unitary:

$$U_A = \begin{pmatrix} A & \sqrt{\mathbf{I}_m - AA^\dagger} \\ \sqrt{\mathbf{I}_m - A^\dagger A} & -A^\dagger \end{pmatrix}$$

In particular, when $A = a$ is a real number in $[-1, 1]$, $U_a = \begin{pmatrix} a & \sqrt{1-a^2} \\ \sqrt{1-a^2} & -a \end{pmatrix}$ is a reflection. Finally, in a product of multiple contractions, individual errors will not propagate much, as we have the following lemma:

▶ **Lemma 3.** *If $A_1, \ldots, A_k \in \mathbb{C}^{m\times m}$ are contractions, and $B_1, \ldots, B_k \in \mathbb{C}^{m\times m}$ satisfy $\|A_i - B_i\| \leq \varepsilon$ for every i, then $\|A_1 \cdots A_k - B_1 \cdots B_k\| \leq (1+\varepsilon)^k - 1$. Furthermore, if $B_1, \ldots, B_k$ are also contractions, then $\|A_1 \cdots A_k - B_1 \cdots B_k\| \leq k\varepsilon$.*

2.2 Quantum Channels

A quantum channel (or operation), in its most general form, is a *completely-positive trace-preserving* (CPTP) map $\Phi : \mathcal{D}_m \to \mathcal{D}_n$ that maps a density matrix ρ to a density matrix $\Phi(\rho)$. We denote the set of such channels as $\mathcal{C}_{m,n}$. The *Kraus representation* of the quantum channel Φ is a set of matrices $\{E_1, \ldots, E_k\}$ such that $\sum_{i=1}^k E_i^\dagger E_i = \mathbf{I}_m$, and

$$\Phi(\rho) = \sum_{i=1}^k E_i \rho E_i^\dagger.$$

The *natural representation* of Φ, denoted as $K(\Phi)$, is a matrix in $\mathbb{C}^{n^2\times m^2}$ such that $\mathrm{vec}(\Phi(\rho)) = K(\Phi)\mathrm{vec}(\rho)$ for any $\rho \in \mathcal{D}_m$. Given the Kraus representation $\{E_1, \ldots, E_k\}$ of Φ, one can easily compute the natural representation $K(\Phi) = \sum_{i=1}^k \overline{E_i} \otimes E_i$.

A quantum channel Φ is *unital*, if it maps the identity to the identity of the same dimension. The Kraus representation of a unital channel is a set of square matrices $\{E_1, \ldots, E_k\}$ that additionally satisfies $\sum_{i=1}^{k} E_i E_i^\dagger = \mathbf{I}_m$. In the language of natural representation, it is known that Φ is unital if and only if $K(\Phi)$ is a contraction [20]. Notice that unitary operators and projective measurements are all unital. Our paper shows the following: one can construct in logspace a quantum circuit to simulate any arbitrary unital channel with ancillas, but without intermediate measurements.

2.3 Quantum Algorithms

A generic quantum algorithm with time T and space $S = \log m$ is specified by T quantum channels $\Phi_1, \ldots, \Phi_T \in \mathcal{C}_{m,m}$, which might depend on the inputs. We also require the channels $\Phi_1, \ldots, \Phi_T$ to be efficiently constructible, whose meaning may differ for different types of quantum algorithms, and will be specified below.

The algorithm starts from the fixed initial state $\rho_0 = |0^S\rangle\langle 0^S|$, and in the i-th step applies Φ_i on the current state, so that the state after the i-th step can be described as

$$\rho_i = \Phi_i(\rho_{i-1}) = \Phi_i \circ \Phi_{i-1} \circ \cdots \circ \Phi_1(\rho_0).$$

At the end the first qubit of the final state ρ_T is measured in the computational basis of the first qubit, where the measurement can be represented as $M_0 = |0\rangle\langle 0| \otimes \mathbf{I}_{m/2}$. The quantum algorithm outputs 0 with probability $\mathrm{Tr}[\rho_T M_0]$, and 1 with probability $1 - \mathrm{Tr}[\rho_T M_0]$. The quantum algorithm is called unitary (resp. unital), if every channel Φ_i is unitary (resp. unital).

Quantum circuit

Fix a universal quantum gate set $\mathcal{G}$, for instance Hadamard and Toffoli gates [25], and let $\mathcal{G}_S$ be the set of gates in $\mathcal{G}$ on S qubits. Let $\mathcal{M}_S$ be the set of single-qubit measurements on S qubits.

When the input of the problem is from domain X, the quantum circuit is specified by a mapping $\Phi_{\mathrm{D}} : X \times [T] \to \mathcal{G}_S \cup \mathcal{M}_S$ such that $\Phi_{i+1} = \Phi_{\mathrm{D}}(x, i)$ for every $i \in [T]$, where $x \in X$ is the input, and Φ_{D} can be computed deterministically in time $O(T)$ and space $O(S)$. The quantum algorithm decides a function $f : X \to \{0, 1\}$ with error ε if:

$$\forall x \in X, \quad |\mathrm{Tr}[\rho_T M_0] - f(x)| \leq \varepsilon.$$

Now, $\mathsf{BQ_QL}$ is the class of boolean-function families where $f_n : \{0,1\}^n \to \{0,1\}$ can be decided by a quantum circuit with time $\mathrm{poly}(n)$, space $O(\log n)$ and error $1/3$. The function is further in the class $\mathsf{BQ_UL}$ if there is no intermediate measurements, i.e. the range of Φ_{D} is $\mathcal{G}_S$. We define $\mathsf{promiseBQ_QL}$ and $\mathsf{promiseBQ_UL}$ similarly, but the domain of each f_n can be a subset of $\{0,1\}^n$.

Quantum learning algorithm

For a quantum online learning algorithm with Γ being the set of samples, there exists a mapping $\Phi_{\mathrm{L}} : \Gamma \to \mathcal{C}_{m,m}$ such that $\Phi_i = \Phi_{\mathrm{L}}(z_i)$ where $z_i \in \Gamma$ is the sample received in the i-th step, and each entry of $K(\Phi_{\mathrm{L}}(z_i))$ can be computed deterministically in time $O(T)$ and space $O(S)$.

Let $\mathcal{P}(\Gamma)$ be the collection of all probability distributions over Γ. For any distribution $D \in \mathcal{P}(\Gamma)$, let D^T be T i.i.d copies of D, so that $z \sim D^T$ means that each sample z_i is independently drawn from D. Let $\mathcal{X}, \mathcal{Y}$ be two disjoint subsets of $\mathcal{P}(\Gamma)$. The quantum learning algorithm distinguishes $\mathcal{X}$ and $\mathcal{Y}$ with error ε if:

$$\forall D \in \mathcal{X}, \quad \mathop{\mathbf{E}}_{z \sim D^T}[\mathrm{Tr}[\rho_T M_0]] \geq 1 - \varepsilon$$
$$\forall D \in \mathcal{Y}, \quad \mathop{\mathbf{E}}_{z \sim D^T}[\mathrm{Tr}[\rho_T M_0]] \leq \varepsilon.$$

And for $\varepsilon = 1/3$, we simply say that the quantum learning algorithm distinguishes $\mathcal{X}$ and $\mathcal{Y}$.

Other specifications

Notice that even in the unitary algorithms where intermediate measurements are not generally allowed, a constant number of intermediate measurements are still available because of the principle of deferred measurements (see e.g. [19, Section 4.4]), which will only increase the time and space by a constant. This means the error ε in both definitions above can be safely amplified to any constant power, and the specific constant error $1/3$ can be replaced by any constant in $[0, 1/2)$.

For constructible functions $t(n) = \Omega(n)$ and $s(n) = \Omega(\log n)$, define $\mathsf{BPTISP}(t(n), s(n))$ as the class of boolean functions families that can be decided by a classical randomized algorithm with time $O(t(n))$ and space $O(s(n))$, and $\mathsf{promiseBPTISP}(t(n), s(n))$ accordingly. The classical randomized logspace class is defined as $(\mathsf{promise})\mathsf{BPL} = (\mathsf{promise})\mathsf{BPTISP}(\mathrm{poly}(n), \log(n))$.

Phase estimation

Given the dimension m and the error parameter $\varepsilon > 0$, the *phase estimation* circuit (see e.g. [19, Section 5.2]) acts on an input register of dimension m and an estimation register of dimension $2^\ell = O(1/\varepsilon)$. The circuit is with time $O(2^\ell)$ and space $O(\ell + \log m)$, and accesses 2^ℓ oracle calls to the controlled-U gates, where $U \in \mathcal{U}_m$ is an arbitrary unitary matrix. For each $j \in [2^\ell]$, define $\lambda(j) = 2j\pi/2^\ell - \pi$, and for any $\lambda \in [-\pi, \pi]$, let $J(\lambda) = \{j \in [2^\ell] \mid |\lambda(j) - \lambda| \leq \varepsilon\}$. If v is a unit eigenvector of U with eigenvalue $e^{i\lambda}$, the circuit maps $v \otimes |0^\ell\rangle$ to

$$\sum_{j=0}^{2^\ell - 1} \alpha_j v \otimes |j\rangle,$$

so that

$$\sum_{j \in J(\lambda)} |\alpha_j|^2 \geq 1 - \varepsilon^2.$$

Given a Hermitian contraction $H \in \mathbb{C}^{m \times m}$, let P_H be the above phase estimation circuit with $U = e^{iH}$, and $P_{H,\varepsilon}$ be the above phase estimation circuit where U is replaced with the Hamiltonian simulation circuit presented in [27] which differs from e^{iH} by an error of $2^{-\ell}\varepsilon$. Notice that $P_{H,\varepsilon}$ is a unitary quantum circuit with $\mathrm{poly}(m/\varepsilon)$ and space $O(\log(m/\varepsilon))$, and by Lemma 3 we have $\|P_{H,\varepsilon} - P_H\| \leq \varepsilon$.

Since H only has eigenvalues in $[-1, 1]$, we slightly modify the definition of $\lambda(j)$ so that it's truncated at ± 1, that is

$$\lambda(j) = \begin{cases} 2j\pi/2^\ell - \pi & \text{if } 2j\pi/2^\ell - \pi \in [-1, 1] \\ \mathrm{sgn}(2j\pi/2^\ell - \pi) & \text{otherwise} \end{cases}$$

which will only make $J(\lambda)$ larger for $\lambda \in [-1, 1]$.

Every unit eigenvector v of H with eigenvalue λ is also a unit eigenvector of e^{iH} with eigenvalue $e^{i\lambda}$. Therefore for any two unit eigenvectors u, v of H, we have

$$\big(u^\dagger \otimes \langle j|\big) P_H \big(v \otimes |0^\ell\rangle\big) = \begin{cases} \alpha_j & \text{if } u = v \\ 0 & \text{if } u \perp v. \end{cases}$$

In other words, since P_H is unitary,

$$\big(u^\dagger \otimes \langle 0^\ell|\big) P_H^{-1} \big(v \otimes |j\rangle\big) = \begin{cases} \overline{\alpha_j} & \text{if } u = v \\ 0 & \text{if } u \perp v. \end{cases}$$

That means the projection of $P_H^{-1}\big(v \otimes |j\rangle\big)$ onto $\mathbb{C}^m \otimes |0^\ell\rangle$ is along $v \otimes |0^\ell\rangle$ and has amplitude $\overline{\alpha_j}$. Combing the above observations we get the following lemma:

▶ **Lemma 4.** *Given a Hermitian contraction $H \in \mathbb{C}^{m\times m}$ and $\varepsilon > 0$, there is a unitary quantum circuit $P_{H,\varepsilon}$ with time $\mathrm{poly}(m/\varepsilon)$ and space $O(\log(m/\varepsilon))$ that is ε-close to a unitary operator P_H, which satisfies the following: There is a parameter $\ell = O(\log(1/\varepsilon))$, such that if v is a unit eigenvector of H with eigenvalue $\lambda \in [-1, 1]$, then*

$$P_H(v \otimes |0^\ell\rangle) = \sum_{j=0}^{2^\ell - 1} \alpha_j v \otimes |j\rangle, \text{ where } \sum_{j \in J(\lambda)} |\alpha_j|^2 \geq 1 - \varepsilon^2.$$

Moreover, for every $j \in [2^\ell]$,

$$P_H^{-1}(v \otimes |j\rangle) = \overline{\alpha_j} v \otimes |0^\ell\rangle + |\perp\rangle,$$

where $|\perp\rangle$ is a vector orthogonal to $\mathbb{C}^m \otimes |0^\ell\rangle$.

Pure State Preparation

Our results involve the simplest form of the *quantum state preparation* problem, which is to map the initial state $|0^S\rangle$ to a given pure state. With the efficient Solovay-Kitaev Theorem in [17], we have the following:

▶ **Lemma 5.** *Given $m = 2^S$, a unit vector $v \in \mathbb{C}^m$ and $\varepsilon > 0$, there is unitary quantum circuit Q_v on S qubits with time $O(m \cdot \mathrm{polylog}(1/\varepsilon))$ and space $O(\log(m/\varepsilon))$ such that $\|Q_v|0^S\rangle - v\|_2 \leq \varepsilon$.*

3 Quantum Implementations of Contractions

▶ **Lemma 6.** *Given a contraction $A \in \mathbb{C}^{m\times m}$ and $\varepsilon > 0$, there is a unitary quantum circuit Q_A with time $\mathrm{poly}(m/\varepsilon)$ and space $O(\log(m/\varepsilon))$, and a parameter $\ell = O(\log(1/\varepsilon))$, such that for unit vector v of dimension $4m$, $\|Q_A(v \otimes |0^\ell\rangle) - (V_A v) \otimes |0^\ell\rangle\|_2 \leq \varepsilon$, where*

$$V_A = \mathrm{diag}(U_A, U_{A^\dagger}) = \begin{pmatrix} A & \sqrt{\mathbf{I}_m - AA^\dagger} & & \\ \sqrt{\mathbf{I}_m - A^\dagger A} & -A^\dagger & & \\ & & A^\dagger & \sqrt{\mathbf{I}_m - A^\dagger A} \\ & & \sqrt{\mathbf{I}_m - AA^\dagger} & -A \end{pmatrix}$$

Proof. Let H be the Hermitian contraction $\begin{pmatrix} & A \\ A^\dagger & \end{pmatrix}$. Notice that

$$U_H = \begin{pmatrix} H & \sqrt{\mathbf{I}_{2m} - H^2} \\ \sqrt{\mathbf{I}_{2m} - H^2} & -H \end{pmatrix}$$
$$= \begin{pmatrix} & A & & \sqrt{\mathbf{I}_m - AA^\dagger} & \\ A^\dagger & & & & \sqrt{\mathbf{I}_m - A^\dagger A} \\ \sqrt{\mathbf{I}_m - AA^\dagger} & & & & -A \\ & \sqrt{\mathbf{I}_m - A^\dagger A} & & -A^\dagger & \end{pmatrix}$$

which differs from V_A only by permutations:

$$V_A = \begin{pmatrix} \mathbf{I}_m & & & \\ & & & \mathbf{I}_m \\ & \mathbf{I}_m & & \\ & & \mathbf{I}_m & \end{pmatrix} \cdot U_H \cdot \begin{pmatrix} & & \mathbf{I}_m & \\ \mathbf{I}_m & & & \\ & \mathbf{I}_m & & \\ & & & \mathbf{I}_m \end{pmatrix}$$

Since the permutations are only on two qubits, it suffices to implement U_H on v up to error ε.

Let $v = \begin{pmatrix} v_1 \\ v_2 \end{pmatrix}$ where both v_1 and v_2 are of dimension $2m$. Suppose H has the eigen decomposition $H = \sum_{k=1}^{2m} \lambda_k u_k^\dagger u_k$, and v_1, v_2 are decomposed into this eigenbasis as

$$v_1 = \sum_{k=1}^{2m} \omega_k^{(0)} u_k, \quad v_2 = \sum_{k=1}^{2m} \omega_k^{(1)} u_k, \quad \text{where } \sum_{k=1}^{2m} \left|\omega_k^{(0)}\right|^2 + \sum_{k=1}^{2m} \left|\omega_k^{(1)}\right|^2 = 1.$$

Since v can be written as $|0\rangle \otimes v_1 + |1\rangle \otimes v_2$, applying the phase estimation circuit P_{H,ε_1} in Lemma 4 on $v \otimes |0^\ell\rangle$ results in:

$$\sum_{k=1}^{2m} \sum_{j=0}^{2^\ell - 1} \omega_k^{(0)} \alpha_{j,k} |0\rangle \otimes u_k \otimes |j\rangle + \sum_{k=1}^{2m} \sum_{j=0}^{2^\ell - 1} \omega_k^{(1)} \alpha_{j,k} |1\rangle \otimes u_k \otimes |j\rangle + |\epsilon_1\rangle$$
$$= \sum_{k=1}^{2m} \sum_{j \in J(\lambda_k)} \omega_k^{(0)} \alpha_{j,k} |0\rangle \otimes u_k \otimes |j\rangle + \sum_{k=1}^{2m} \sum_{j \in J(\lambda_k)} \omega_k^{(1)} \alpha_{j,k} |1\rangle \otimes u_k \otimes |j\rangle + |\epsilon_2\rangle.$$

where for each k it holds $\sum_{j \in J(\lambda_k)} |\alpha_{j,k}|^2 \geq 1 - \varepsilon_1^2$. Here ε_1 is an error parameter to be determined later, and $\ell = O(\log(1/\varepsilon_1))$. The error vector $|\epsilon_1\rangle$ is introduced due to the difference between P_{H,ε_1} and P_H, and thus $\||\epsilon_1\rangle\|_2 \leq \|P_{H,\varepsilon_1} - P_H\| \leq \varepsilon_1$. The error vector $|\epsilon_2\rangle - |\epsilon_1\rangle$ is a weighted sum of $4m$ orthogonal error vectors, with lengths at most ε_1 and weights $\omega_k^{(0)}, \omega_k^{(1)}$, and thus has length at most ε_1. Therefore $\||\epsilon_2\rangle\|_2 \leq 2\varepsilon_1$.

Now apply the following unitary transformation on the first qubit and last ℓ qubits:

$$|0\rangle|j\rangle \to \lambda(j)|0\rangle|j\rangle + \sqrt{1 - \lambda(j)^2}|1\rangle|j\rangle$$
$$|1\rangle|j\rangle \to \sqrt{1 - \lambda(j)^2}|0\rangle|j\rangle - \lambda(j)|1\rangle|j\rangle$$

which gives

$$\sum_{k=1}^{2m} \sum_{j \in J(\lambda_k)} \omega_k^{(0)} \alpha_{j,k} \left[\lambda(j)|0\rangle + \sqrt{1 - \lambda(j)^2}|1\rangle\right] \otimes u_k \otimes |j\rangle$$
$$+ \sum_{k=1}^{2m} \sum_{j \in J(\lambda_k)} \omega_k^{(1)} \alpha_{j,k} \left[\sqrt{1 - \lambda(j)^2}|0\rangle - \lambda(j)|1\rangle\right] \otimes u_k \otimes |j\rangle + |\epsilon_3\rangle$$

This unitary transformation can be implemented as a serial combination of 2^ℓ single-qubit unitaries $U_{\lambda(j)}$ controlled by the last ℓ qubits representing j. Each one of them can be constructed up to error $2^{-\ell}\varepsilon_1$ in time polylog$(1/\varepsilon_1)$ and space $O(\log(1/\varepsilon_1))$ by [17, Theorem 7]. Therefore by Lemma 3 we have $\||\epsilon_3\rangle\|_2 \le \||\epsilon_2\rangle\|_2 + \varepsilon_1 \le 3\varepsilon_1$.

Finally applying the reverse phase estimation P^{-1}_{H,ε_1} gives the following state, where $|\perp\rangle$ is orthogonal to $\mathbb{C}^2 \otimes \mathbb{C}^{2m} \otimes |0^\ell\rangle$:

$$
\begin{aligned}
&\sum_{k=1}^{2m} \sum_{j\in J(\lambda_k)} |\alpha_{j,k}|^2 \omega_k^{(0)} \left[\lambda(j)|0\rangle + \sqrt{1-\lambda(j)^2}|1\rangle\right] \otimes u_k \otimes |0^\ell\rangle \\
&\quad + \sum_{k=1}^{2m} \sum_{j\in J(\lambda_k)} |\alpha_{j,k}|^2 \omega_k^{(1)} \left[\sqrt{1-\lambda(j)^2}|0\rangle - \lambda(j)|1\rangle\right] \otimes u_k \otimes |0^\ell\rangle + |\epsilon_4\rangle + |\perp\rangle \\
=&\sum_{k=1}^{2m} \sum_{j\in J(\lambda_k)} |\alpha_{j,k}|^2 \omega_k^{(0)} \left[\lambda_k|0\rangle + \sqrt{1-\lambda_k^2}|1\rangle\right] \otimes u_k \otimes |0^\ell\rangle \\
&\quad + \sum_{k=1}^{2m} \sum_{j\in J(\lambda_k)} |\alpha_{j,k}|^2 \omega_k^{(1)} \left[\sqrt{1-\lambda_k^2}|0\rangle - \lambda_k|1\rangle\right] \otimes u_k \otimes |0^\ell\rangle + |\epsilon_5\rangle + |\perp\rangle \\
=&\sum_{k=1}^{2m} \omega_k^{(0)} \left[\lambda_k|0\rangle + \sqrt{1-\lambda_k^2}|1\rangle\right] \otimes u_k \otimes |0^\ell\rangle \\
&\quad + \sum_{k=1}^{2m} \omega_k^{(1)} \left[\sqrt{1-\lambda_k^2}|0\rangle - \lambda_k|1\rangle\right] \otimes u_k \otimes |0^\ell\rangle + |\epsilon_6\rangle + |\perp\rangle \\
=&\sum_{k=1}^{2m} \omega_k^{(0)} \big[U_H(|0\rangle \otimes u_k)\big] \otimes |0^\ell\rangle + \sum_{k=1}^{2m} \omega_k^{(1)} \big[U_H(|1\rangle \otimes u_k)\big] \otimes |0^\ell\rangle + |\epsilon_6\rangle + |\perp\rangle \\
=&\big[U_H(|0\rangle \otimes v_1)\big] \otimes |0^\ell\rangle + \big[U_H(|1\rangle \otimes v_2)\big] \otimes |0^\ell\rangle + |\epsilon_6\rangle + |\perp\rangle \\
=&(U_H v) \otimes |0^\ell\rangle + |\epsilon_6\rangle + |\perp\rangle.
\end{aligned}
$$

Here $\||\epsilon_4\rangle\|_2 \le \||\epsilon_3\rangle\|_2 + \|P^{-1}_{H,\varepsilon_1} - P^{-1}_H\| \le 4\varepsilon_1$. Also, similar to the reasoning for $|\epsilon_2\rangle - |\epsilon_1\rangle$, since for every k, $1-\varepsilon_1^2 \le \sum_{j\in J(\lambda_k)} |\alpha_{j,k}|^2 \le 1$, and for every $j \in J(\lambda_k)$, $\|U_{\lambda(j)} - U_{\lambda_k}\|_2 \le |\lambda(j) - \lambda_k| \le \varepsilon_1$, we have

$$\||\epsilon_6\rangle\|_2 \le \||\epsilon_5\rangle\|_2 + \varepsilon_1^2 \le \||\epsilon_4\rangle\|_2 + \varepsilon_1 + \varepsilon_1^2 \le 6\varepsilon_1.$$

Finally, notice that both $(U_H v) \otimes |0^\ell\rangle$ and $(U_H v) \otimes |0^\ell\rangle + |\epsilon_6\rangle + |\perp\rangle$ are unit vectors, while $|\perp\rangle$ is orthogonal to $(U_H v) \otimes |0^\ell\rangle$, so we have

$$\big|\big((U_H v)^\dagger \otimes \langle 0^\ell|\big)\big((U_H v) \otimes |0^\ell\rangle + |\epsilon_6\rangle + |\perp\rangle\big)\big| = \big|1 + \big((U_H v)^\dagger \otimes \langle 0^\ell|\big)|\epsilon_6\rangle\big| \ge 1 - \||\epsilon_6\rangle\|_2,$$

which implies that $\||\epsilon_6\rangle + |\perp\rangle\|_2 \le \sqrt{2\||\epsilon_6\rangle\|_2}$. Therefore it suffices to take $\varepsilon_1 \le \varepsilon^2/12$, and the theorem follows. ◀

As a by product, when we take the contraction A in Lemma 6 to be unitary, we get the unitary implementation of any unitary matrix, with the number of ancillas only depending on the error:

▶ **Corollary 7.** *Given a unitary matrix $U \in \mathcal{U}_m$ and $\varepsilon > 0$, there is a unitary quantum circuit Q_U with time* poly(m/ε) *and space $O(\log(m/\varepsilon))$, and a parameter $\ell = O(\log(1/\varepsilon))$, such that for any unit vector v of dimension m, $\|Q_U(v \otimes |0^\ell\rangle) - (Uv) \otimes |0^\ell\rangle\|_2 \le \varepsilon$.*

Proof. Use the exact same circuit in Lemma 6 by adding two ancilla qubits to v initialized at $|00\rangle$. Notice that $V_U = \mathrm{diag}(U, -U^\dagger, U^\dagger, -U)$, and thus the output state is ε close to $[V_U(|00\rangle \otimes v)] \otimes |0^\ell\rangle = |00\rangle \otimes (Uv) \otimes |0^\ell\rangle$. Rearranging the order of qubits and the claim follows. ◀

Finally, for permutation matrices, we present a simple unitary implementation without any ancillas by decomposing it into transpositions.

▶ **Lemma 8.** *Given a permutation $\sigma \in S_m$ and $\varepsilon > 0$, there is a unitary quantum circuit U with time $\mathrm{poly}(m/\varepsilon)$ and space $O(\log(m/\varepsilon))$, such that $\|U - P_\sigma\| \le \varepsilon$, where $P_\sigma \in \{0,1\}^{m\times m}$ is the matrix representation of σ.*

4 Contraction Powering in Quantum Logspace

▶ **Definition 9** (Contraction Powering). *Given $m = 2^S$, a contraction $A \in \mathbb{C}^{m\times m}$, a positive integer T in unary, and two vectors $v, w \in \mathbb{C}^m$ with $\|v\|_2 = \|w\|_2 = 1$ as the input, it is promised that $|w^\dagger A^T v|^2$ is either in $[0, 1/3]$ or $[2/3, 1]$, and the goal of the* Contraction-Powering *problem is to distinguish between the two cases.*

▶ **Theorem 10.** *ContractionPowering $\in$ promiseBQ$_{\mathsf{U}}$L. Moreover, given the same input (m, A, T, v, w) but without the promise on $|w^\dagger A^T v|^2$, while also given an error parameter $\varepsilon > 0$, there is a unitary quantum circuit W with time $\mathrm{poly}(mT/\varepsilon)$ and space $S' = O(\log(mT/\varepsilon))$ such that $|\langle 0^{S'}|W|0^{S'}\rangle|^2$ is ε-close to $|w^\dagger A^T v|^2$.*

Proof. First, let Q_v and Q_w be the circuits preparing states v and w with error $\varepsilon/8$ in Lemma 5 respectively. Since

$$\left||\langle 0^S|Q_w^\dagger A^T Q_v|0^S\rangle|^2 - |w^\dagger A^T v|^2\right| \le 4\|Q_v|0^S\rangle - v\|_2 + 4\|Q_w|0^S\rangle - w\|_2 \le \varepsilon/2,$$

in the rest of the proof we can safely assume that $Q_v|0^S\rangle = v$ and $Q_w|0^S\rangle = w$ while halving ε.

Let $\ell = O(\log(T/\varepsilon))$ be the one in Lemma 6 with error parameter $(2T)^{-1}\varepsilon$. The circuit works on three parts of qubits: the counter register C of dimension $2T$, the vector register of dimension m, and ℓ ancilla qubits. The circuit starts by preparing $|0\rangle_C \otimes v \otimes |0^\ell\rangle$ by applying Q_v. Then repeat the following two steps for T times:

1. Apply V_A on the last two qubits of the timer register and the entire vector register by Lemma 6;
2. Apply the permutation $|0\rangle \to |0\rangle$, $|2T-2\rangle \to |1\rangle$, $|2T-1\rangle \to |2\rangle$, $|i\rangle \to |i+2\rangle$, $\forall i = 1, \ldots, 2T-3$. on the counter register by Lemma 8.

Finally, apply $Q_w^\dagger$ on the vector register and measure with the projection onto $|0\rangle_C \otimes |0^S\rangle \otimes |0^\ell\rangle$.

To prove the correctness of the algorithm, we first assume that all the implementations in Lemma 6 and Lemma 8 are errorless, i.e. the evolution is completely within the subspace $\mathbb{C}^{2T} \otimes \mathbb{C}^m \otimes |0^\ell\rangle$. Then it suffices to notice that V_A is block-diagonal, so that step 1 acts locally on the T subspaces spanned by $|2i\rangle_C$ and $|2i+1\rangle_C$. Therefore after the i-th application of V_A, the projection of the current state onto $|j\rangle_C$ is always 0 for $j \ge 2i$, and thus before each application of V_A, the projection onto $|1\rangle_C$ is always 0. So the state after the i-th repetition is $|0\rangle_C \otimes (A^i v) + |\perp\rangle$, where $|\perp\rangle$ is orthogonal to $|0\rangle_C$. The output probability is then

$$\left|\left(\langle 0|_C \otimes \langle 0^S|\right)\left(\mathbf{I}_{2T} \otimes U_w^\dagger\right)\left(|0\rangle_C \otimes (A^T v) + |\perp\rangle\right)\right|^2 = |w^\dagger A^T v|^2.$$

Now each step in the repetition introduces an error of $(2T)^{-1}\varepsilon$. Therefore, by Lemma 3, the total error of the unitary quantum circuit W, compared to the above ideal case, is at most ε. ◀

5 Equivalence of Unital and Unitary Quantum Logspace

5.1 Simulating Unital Quantum Logspace with Constant Error

▶ **Lemma 11.** *Given a unital quantum algorithm with time T and space $S = \log m$ specified by the natural representations $K(\Phi_1), \ldots, K(\Phi_T) \in \mathbb{C}^{m^2 \times m^2}$, and an error parameter $\varepsilon > 0$, there is a unitary quantum circuit with time $\mathrm{poly}(mT/\varepsilon)$ and space $O(\log(mT/\varepsilon))$, such that if the original unital circuit outputs 0 with probability p, then the unitary circuit outputs 0 with probability $\sin^2(p+\alpha)$, where $|\alpha| \le \varepsilon$.*

Proof. We can always assume that S is an odd number and $m \ge \max(4\varepsilon^{-1}, 8)$ by adding dummy dimensions. As each $K(\Phi_i)$ is a contraction, the following matrix A of dimension m^2T is also a contraction:

$$A = \begin{pmatrix} & & & & K(\Phi_T) \\ K(\Phi_1) & & & & \\ & K(\Phi_2) & & & \\ & & \ddots & & \\ & & & K(\Phi_{T-1}) & \end{pmatrix}$$

Since the final state of the unital quantum algorithm is $\rho_T = \Phi_T \circ \cdots \circ \Phi_1(\rho_0)$, we can rewrite the output probability of the unital quantum algorithm as

$$\begin{aligned} p = \mathrm{Tr}[\rho_T M_0] &= \mathrm{vec}(M_0)^\dagger \mathrm{vec}(\rho_T) = \mathrm{vec}(M_0)^\dagger K(\Phi_T) \cdots K(\Phi_1) \mathrm{vec}(\rho_0) \\ &= \left(\mathrm{vec}(M_0)^\dagger \otimes \langle 0|\right) A^T \left(\mathrm{vec}(\rho_0) \otimes |0\rangle\right) \end{aligned}$$

Let $v = \mathrm{vec}(\rho_0) \otimes |0\rangle$ which is already a unit vector. Since $\|\mathrm{vec}(M_0)\|_2 = \sqrt{m/2}$, let $w = \sqrt{\frac{2}{m}}\mathrm{vec}(M_0) \otimes |0\rangle$. Let ε_1 be the error parameter to be determined later. Theorem 10 constructs a unitary quantum algorithm W with time $\mathrm{poly}(mT/\varepsilon_1)$ and space $S' = O(\log(mT/\varepsilon_1))$, such that $|\langle 0^{S'}|W|0^{S'}\rangle|^2$ is ε_1-close to $2p^2/m$. Therefore $|\langle 0^{S'}|W|0^{S'}\rangle|$ is $\sqrt{\varepsilon_1}$-close to $\sqrt{\frac{2}{m}}p$.

Let

$$R = \left(\mathbf{I}_{S'} - 2W|0^{S'}\rangle\langle 0^{S'}|W^\dagger\right)\left(\mathbf{I}_{S'} - 2|0^{S'}\rangle\langle 0^{S'}|\right)$$

be the rotation on the subspace spanned by $|0^{S'}\rangle$ and $W|0^{S'}\rangle$, of degree $2\cos^{-1}|\langle 0^{S'}|W|0^{S'}\rangle|$. By the estimation

$$\frac{\pi}{2} - x - \frac{1}{4}x^3 \le \cos^{-1} x \le \frac{\pi}{2} - x, \forall x \in [0,1],$$

and since $(x+y)^3 \le 4(x^3+y^3)$ for non-negative x, y, it can be calculated that the degree of the rotation R is in the range

$$\left[\pi - 2\sqrt{\frac{2}{m}}p - 4\sqrt{\varepsilon_1} - \frac{4}{m}\sqrt{\frac{2}{m}},\ \pi - 2\sqrt{\frac{2}{m}}p + 2\sqrt{\varepsilon_1}\right].$$

Since S is an odd number, $k = \sqrt{m/8}$ is an integer. Applying R for k times will rotate $|0^{S'}\rangle$ by a degree of $k\pi - p - \alpha$, where $|\alpha| \leq \sqrt{2m\varepsilon_1} + 2m^{-1}$. Therefore the projective measurement of the state $R^k|0^{S'}\rangle$ onto the subspace orthogonal to $|0^{S'}\rangle$ outputs 0 with probability $\sin^2(p+\alpha)$. Let $\varepsilon_1 = (8m)^{-1}\varepsilon^2$, and notice that $2m^{-1} \leq \varepsilon/2$, so that we have $|\alpha| < \varepsilon$, and the circuit R^k is unitary with time $\mathrm{poly}(mT/\varepsilon)$ and space $O(\log(mT/\varepsilon))$. ◀

▶ **Theorem 12.** $\mathsf{BQ_QL} = \mathsf{BQ_UL}$, *and* $\mathsf{promiseBQ_QL} = \mathsf{promiseBQ_UL}$.

Proof. Clearly $\mathsf{BQ_QL} \supseteq \mathsf{BQ_UL}$, and $\mathsf{promiseBQ_QL} \supseteq \mathsf{promiseBQ_UL}$. To prove the other direction, notice that quantum circuits are unital, therefore by Lemma 11 with $\varepsilon = 0.01$ they can be simulated by unitary quantum circuits with polynomial time and logarithmic space. Since the original output probability p is promised to be in $[0, 1/3]$ or $[2/3, 1]$, the value of $\sin^2(p+\alpha)$ is in $[0, 0.12]$ or $[0.37, 1]$ respectively, and thus it suffices to perform a constant rounds of amplification in order to bring the error down to less than $1/3$. ◀

▶ Remark 13. Though we proved Theorem 12 via the contraction powering algorithm, the unitary quantum circuit that simulates a given quantum circuit with intermediate measurements can be more simply constructed without using Lemma 6. In details, given a channel Φ in the quantum circuit, we can directly write out the natural representations $K(\Phi)$, and apply the matrix on the vectorized density matrix $\mathrm{vec}(\rho)$:

- If Φ is a unitary quantum gate U, then $K(\Phi) = \overline{U} \otimes U$ which can be implemented by applying U and then $\overline{U}$;
- If Φ is a single-qubit measurement, then $K(\Phi)$ is a diagonal matrix with diagonal entries in $\{0, 1\}$. It can be implemented using a similar "permute and throw away" technique as in Theorem 10, which after applied T times increases the dimension (instead of the space!) by a factor of T.

And the resulting circuit can be amplified in the same way as in Lemma 11.

5.2 Simulating Unital Quantum Logspace with Small Error

Now we can improve the result in Lemma 11 to arbitrarily small error (namely the probability of outputting 0 is $(p+\alpha)$ instead of $\sin^2(p+\alpha)$). Interestingly, the improvement relies on a stronger version of Theorem 10, which in turn relies on Theorem 12. In a way, we use these results to improve themselves!

We start with the stronger version of Theorem 10, which outputs the numerical value of $|w^\dagger A^T v|^2$ instead of outputting 0 with such probability. Here the quantum circuit outputs a number by a final measurement over the computational basis.

▶ **Lemma 14.** *Given $m = 2^S$, a contraction $A \in \mathbb{C}^{m\times m}$, a positive integer T, two unit vectors $v, w \in \mathbb{C}^m$ and an error parameter $\varepsilon > 0$, there is a unitary quantum circuit with time* $\mathrm{poly}(mT/\varepsilon)$ *and space* $O(\log(mT/\varepsilon))$ *such that with probability* $1 - 2^{-\mathrm{poly}(mT/\varepsilon)}$*, it outputs* $|w^\dagger A^T v|^2$ *with additive error* ε*.*

Proof. Theorem 10 provides a unitary quantum circuit W with time $\mathrm{poly}(mT/\varepsilon)$ and space $O(\log(mT/\varepsilon))$ which outputs 0 with probability p such that $\left|p - |w^\dagger A^T v|^2\right| \leq \varepsilon/2$. By Marriott-Watrous amplification [16, Theorem 3.3], there is a quantum circuit W' with time $\mathrm{poly}(mT/\varepsilon)$ and space $O(\log(mT/\varepsilon))$ *with* intermediate measurements, that uses W and W^{-1} as sub-circuits, and with probability $1 - \delta = 1 - 2^{-\mathrm{poly}(mT/\varepsilon)}$ outputs a value $\tilde{p}$ such that $|\tilde{p} - p| \leq \varepsilon/4$.

Since the resulting circuit W' is not unitary, we would like to use Theorem 12 to compute unitarily each bit in the output value $\tilde{p}$ of W'. Furthermore, using the result in [5] that $\mathsf{BQ_UL} = \mathsf{Q_UL}(1 - 2^{-\mathrm{poly}(n)}, 2^{-\mathrm{poly}(n)})$ (which stands for unitary quantum logspace with exponentially small error) the total error probability can be reduced down to $2^{-\mathrm{poly}(mT/\varepsilon)}$. Assuming that every bit in $\tilde{p}$ is 0 with probability either in $[0, 1/3]$ or $[2/3, 1]$, then for $1 \leq i \leq \lceil \log(1/\varepsilon) \rceil + 2$, we let W_i be the unitary quantum circuit that computes the i-th bit of $\tilde{p}$ with exponentially small error. Ideally, the outputs of W_i combined together would ε-approximate $|w^\dagger A^T v|^2$.

However, the value $\tilde{p}$ outputted by the Marriott-Watrous amplification might be different in each W_i, so the final approximation assembled can be totally wrong (for instance, when $p = 0.5$, the outputs $\tilde{p} = \overline{0.1000\ldots}$ and $\tilde{p} = \overline{0.0111\ldots}$ might be assembled to $\overline{0.1111\ldots}$). Moreover, the error reduction in [5] may have unpredictable results, as the promises on the distributions of the bits in $\tilde{p}$ are not guaranteed (again when $p = 0.5$, the most significant bit of $\tilde{p}$ is equally distributed on 0 and 1).

Fortunately, we can solves both problems by computing from the most significant bit to the least significant bit. We maintain a value $q \in [0, 1]$ which is initialized to 0. For each $i = 1$ to $\lceil \log(1/\varepsilon) \rceil + 2$ do the following: Run the modified circuit W_i which outputs the i-th bit of $(\tilde{p} - q)$ instead of $\tilde{p}$. To deal with case when $\tilde{p} - q$ is outside of $[0, 2^{-i+1})$, if $\tilde{p} - q < 0$ it outputs 0, and if $\tilde{p} - q \geq 2^{-i+1}$ it outputs 1. Let the output bit be b_i and update q to $q + b_i \cdot 2^{-i}$.

We claim that with probability $1 - 2^{-\mathrm{poly}(mT/\varepsilon)}$, $|q - p| \leq \varepsilon/2$. First notice that, if every bit in $\tilde{p}$ is 0 with probability in $[0, 2\delta] \cup [1 - 2\delta, 1]$, then the error reduction will work as intended, while with probability $1 - O(\delta \log(1/\varepsilon)) = 1 - 2^{-\mathrm{poly}(mT/\varepsilon)}$ the value $\tilde{p}$ is the same in each circuit W_i, so that q is also the same as $\tilde{p}$.

Now let i be the first index such that the i-th bit of $\tilde{p}$ is 0 with probability in $[2\delta, 1 - 2\delta]$. As the Marriott-Watrous amplification outputs incorrectly with probability at most δ, it means that there are two valid outputs $\tilde{p}_1$ and $\tilde{p}_2$, both are $\varepsilon/4$-close to p, and they coincide in the first $i - 1$ bits but differs at the i-th bit. Let q_i be the value of q at that step, which consists of the first $i - 1$ bits of $\tilde{p}_1$ and $\tilde{p}_2$, then $|q_i + 2^{-i} - p| \leq \varepsilon/4$. Therefore the remaining bits of q could only be $\overline{011\ldots11}$, $\overline{100\ldots00}$ or $\overline{100\ldots01}$, which means $|q_i + 2^{-i} - q| \leq \varepsilon/4$ and thus $|q - p| \leq \varepsilon/2$. Notice that on the i-th (and the last bit when $b_i = 1$) the error reduction may fail and arbitrarily output 0 or 1, but it does not matter as both 0 and 1 are viable in these cases.

As a conclusion, the value q is an ε-approximation of $|w^\dagger A^T v|^2$ with probability $1 - 2^{-\mathrm{poly}(mT/\varepsilon)}$. The above circuit that outputs q is clearly with time $\mathrm{poly}(mT/\varepsilon)$ and space $O(\log(mT/\varepsilon))$ as we use constructions in Theorem 12 and [5]. Finally, the the circuit is unitary since the $O(\log(1/\varepsilon))$ measurements that output b_i's can be deferred, and each W_i can be uncomputed by implementing the circuit in reverse. ◀

▶ **Corollary 15.** *Given $m = 2^S$, a contraction $A \in \mathbb{C}^{m \times m}$, a positive integer T, two unit vectors $v, w \in \mathbb{C}^m$ and an error parameter $\varepsilon > 0$, there is a unitary quantum circuit with time $\mathrm{poly}(mT/\varepsilon)$ and space $O(\log(mT/\varepsilon))$ such that with probability $1 - 2^{-\mathrm{poly}(mT/\varepsilon)}$, it outputs $w^\dagger A^T v$ with additive error ε.*

Proof. Let $A_1 = \begin{pmatrix} A & \\ & 1 \end{pmatrix}$, $v_1 = \begin{pmatrix} v/\sqrt{2} \\ 1/\sqrt{2} \end{pmatrix}$, $v_1' = \begin{pmatrix} v/\sqrt{2} \\ \mathrm{i}/\sqrt{2} \end{pmatrix}$ and $w_1 = \begin{pmatrix} w/\sqrt{2} \\ 1/\sqrt{2} \end{pmatrix}$. Since we have

$$w^\dagger A^T v = \frac{1}{2}\left(4|w_1^\dagger A_1^T v_1|^2 - |w^\dagger A^T v|^2 - 1\right) + \frac{\mathrm{i}}{2}\left(4|w_1^\dagger A_1^T v_1'|^2 - |w^\dagger A^T v|^2 - 1\right),$$

computing $|w^\dagger A^T v|^2$, $|w_1^\dagger A_1^T v_1|^2$ and $|w_1^\dagger A_1^T v_1'|^2$ each up to error $\varepsilon/2$ gives $w^\dagger A^T v$ with error ε. ◀

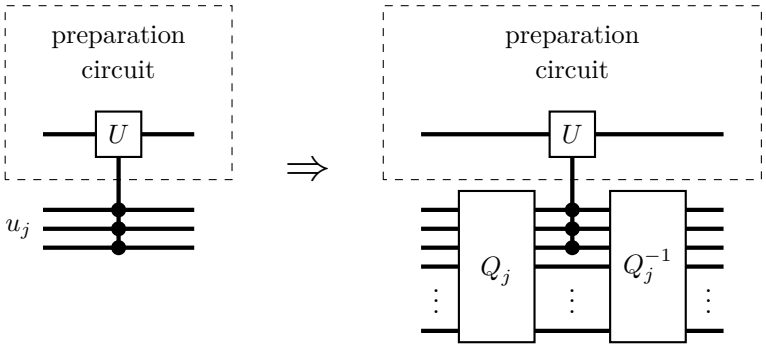


Figure 1 The quantum operator U in the preparation circuit controlled by an entry u_j of u, in binary representation with classical bits. We replace the classical control by first implementing the circuit Q_j, applying the controlled-U operator, and implementing Q_j in reverse.

Notice that one can instead achieve $1/\text{poly}(mT/\varepsilon)$ error probability without using the exponential error reduction in [5], by simply repeating the decision circuit in $\mathsf{BQ_UL}$ for $O(\log(mT/\varepsilon))$ rounds. Nevertheless, it is enough for proving the following theorem, which states that unitary quantum circuits can simulate any unital quantum algorithm by computing its output distribution with arbitrarily small error.

▶ **Theorem 16.** *Given a unital quantum algorithm with time T and space $S = \log m$ specified by the natural representations $K(\Phi_1), \ldots, K(\Phi_T) \in \mathbb{C}^{m^2 \times m^2}$, where $\rho_T = \Phi_T \circ \Phi_{T-1} \circ \cdots \circ \Phi_1(|0^S\rangle\langle 0^S|)$ is its final state, a multi-outcome measurement $\{M_0, \ldots, M_{r-1}\}$ over the computational basis, and an error parameter $\varepsilon > 0$, there is a unitary quantum circuit W with time $\text{poly}(mT/\varepsilon)$ and space $S' = O(\log(mT/\varepsilon))$ such that if $w \in \mathbb{C}^{2^{S'}}$ is the vector representation of $W|0^{S'}\rangle$ in computational basis, for every $j \in [r]$ it holds that $\left||w_j|^2 - \text{Tr}[\rho_T M_j]\right| \leq \varepsilon$.*

Proof. For every $j \in [r]$, let m_j be the dimension of the subspace that M_j projects onto. In other words, $m_j = \|\text{vec}(M_j)\|_2^2$. As in the proof of Lemma 11, we can construct a contraction $A \in \mathbb{C}^{m^2T \times m^2T}$ and unit vectors $v, w \in \mathbb{C}^{m^2T}$ such that $w^\dagger A^T v = \text{Tr}[\rho_T M_j]/\sqrt{m_j}$. By Corollary 15, for every $j \in [r]$ there is a unitary quantum circuit Q_j with time $\text{poly}(mT/\varepsilon)$ and space $O(\log(mT/\varepsilon))$ such that with probability $1 - 2^{-\text{poly}(mT/\varepsilon)}$, it gives an $(2m)^{-3}\varepsilon^2$-approximation of $\text{Tr}[\rho_T M_j]/\sqrt{m_j}$, which implies an $(2m)^{-1}\varepsilon$-approximation of $\sqrt{\text{Tr}[\rho_T M_j]}$.

Consider the preparation circuit constructed in Lemma 5 which prepares the unit vector

$$u = \left(\sqrt{\text{Tr}[\rho_T M_0]}, \sqrt{\text{Tr}[\rho_T M_1]}, \ldots, \sqrt{\text{Tr}[\rho_T M_{r-1}]}\right).$$

with error $\varepsilon/2$. By construction, the preparation circuit can be viewed as a composition of $r-1$ unitary operators, each controlled by a different entry in u. Since u is not explicitly given, we instead control these unitary operators with the output qubits of Q_j, but without measurements. Each circuit Q_j is applied in reverse after the control, so that the space can be reused.

It is clear that the entire circuit is with time $\text{poly}(mT/\varepsilon)$ and space $O(\log(mT/\varepsilon))$. The error introduced by replacing each of th $r-1$ unitary operators is at most $(2m)^{-1}\varepsilon + 2^{-\text{poly}(mT/\varepsilon)}$, therefore the total error is at most $\varepsilon/2 + (r-1)((2m)^{-1}\varepsilon + 2^{-\text{poly}(mT/\varepsilon)}) < \varepsilon$. See Figure 1 for an illustration. ◀

The measurement $\{M_0, \ldots, M_{r-1}\}$ in Theorem 16 could be over any subset of the qubits. In particular, when it is a two-outcome measurement over one qubit, we have the following direct corollary which improves Lemma 11:

▶ **Corollary 17.** *Given a unital quantum algorithm with time T and space $S = \log m$ specified by the natural representations $K(\Phi_1), \ldots, K(\Phi_T) \in \mathbb{C}^{m^2 \times m^2}$, and an error parameter $\varepsilon > 0$, there is a unitary quantum circuit with time $\mathrm{poly}(mT/\varepsilon)$ and space $O(\log(mT/\varepsilon))$, such that if the original unital circuit outputs 0 with probability p, then the unitary circuit outputs 0 with probability $p + \alpha$, where $|\alpha| \leq \varepsilon$.*

6 Powering of Non-Contraction Matrices

In this section we extend the result of Corollary 15 to matrices that may not necessarily be contractions. We state the result for general square matrices, while the additive error can be exponentially large with respect to the spectral norm:

▶ **Theorem 18.** *Given $m = 2^S$, an arbitrary matrix $A \in \mathbb{C}^{m \times m}$, a positive integer T, two unit vectors $v, w \in \mathbb{C}^m$ and an error parameter $\varepsilon > 0$, there is a unitary quantum circuit W with time $\mathrm{poly}(mT/\varepsilon)$ and space $O(\log(mT/\varepsilon))$ such that with probability $1 - 2^{-\mathrm{poly}(mT/\varepsilon)}$, it outputs $w^\dagger A^T v$ with additive error $\varepsilon \cdot \max(1, \|A\|^T)$.*

Proof. Ideally, we would like to apply the contraction powering algorithm on $A/\|A\|$ and multiply the result by $\|A\|^T$. However, the current best quantum algorithm for computing the spectral norm is [27, Theorem 5.2] which approximates $\|A\|$ with additive error ε_1 within time $\mathrm{poly}(m/\varepsilon_1)$ and space $O(\log(m/\varepsilon_1))$ and only works for contractions A. We use this algorithm to approximate $\|A\|$ for arbitrary A with multiplicative error as follows[1]: First compute $\|A\|_{\mathrm{F}}$ in $O(\log m + \log \|A\|_F) = O(\log(m\|A\|_2))$ space. Notice that $A/\|A\|_{\mathrm{F}}$ is a contraction since $\|A\|_{\mathrm{F}} \geq \|A\|$. Therefore, let σ be the approximation of $\|A/\|A\|_{\mathrm{F}}\|$ with additive error ε_1 by [27], then $\sigma\|A\|_F$ approximates $\|A\|$ since

$$\left|\sigma\|A\|_F - \|A\|\right| = \|A\|_F \cdot \left|\sigma - \left\|A/\|A\|_F\right\|\right| \leq \sqrt{m}\varepsilon_1\|A\|.$$

Let $\varepsilon_1 = (3T\sqrt{m})^{-1}$, and let $\alpha = (1 - \sqrt{m}\varepsilon_1)^{-1}\sigma\|A\|_F$. Then

$$\|A\| \leq \alpha \leq \frac{1 + \sqrt{m}\varepsilon_1}{1 - \sqrt{m}\varepsilon_1}\|A\| \leq (1 + T^{-1})\|A\|.$$

Now let $\widetilde{A} = \alpha^{-1}A$ so that $\widetilde{A}$ is always a contraction. Applying the contraction powering algorithm in Corollary 15 on $\widetilde{A}$ with error $\varepsilon/3$ results in a unitary quantum circuit with time $\mathrm{poly}(mT/\varepsilon)$ and space $O(\log(mT/\varepsilon))$ which outputs $w^\dagger \widetilde{A}^T v$ with additive error $\varepsilon/3$. Multiplying it by α^T gives the desired result, while the error is at most $\alpha^T\varepsilon/3 \leq \varepsilon \cdot \|A\|^T$. ◀

7 Classical Simulation of Quantum Learning

7.1 Equivalence of Classical Simulation in Decision and Learning

▶ **Theorem 19.** *If there are functions $t(\cdot,\cdot)$ and $s(\cdot,\cdot)$, such that every unitary quantum learning algorithm with time T and space S can be simulated classically with time $t(T,S)$ and space $s(T,S)$, then*

$$\mathsf{promiseBQ_UL} \subseteq \mathsf{promiseBPTISP}(t(\mathrm{poly}(n), O(\log n)), s(\mathrm{poly}(n), O(\log n))).$$

[1] During the analysis we assume without loss of generality that $\|A\| \geq 1$, since otherwise it can always be relaxed to 1 whenever necessary.

Specifically, if every unitary quantum learning algorithm in time T and space S can be simulated classically with time $\mathrm{poly}(2^S T)$ *and space* $O(S + \log T)$*, then* $\mathsf{promiseBQ_UL} = \mathsf{promiseBPL}$.

Proof. Suppose that we have a unitary quantum circuit with time $T(n) = \mathrm{poly}(n)$ and space $S(n) = O(\log n)$ that decides a partial function $f : X \to \{0,1\}$, where $X \subseteq \{0,1\}^n$. Let $\Phi_{\mathrm{D}}(x,i)$ be the unitary gate at the i-th step of the decision algorithm with input x, which can be constructed in time $\mathrm{poly}(n)$ and space $O(\log n)$.

We can convert the quantum circuit to a learning algorithm as follows. Use X directly as the sample space, while the samples are always constant x for some fixed $x \in X$. The learning task is to distinguish between $x \in f^{-1}(0)$ or $x \in f^{-1}(1)$. Upon receiving the sample x, the learning algorithm simply applies the following unitary operator on $\mathbb{C}^{2^{S(n)}} \otimes \mathbb{C}^{T(n)}$:

$$|\psi\rangle|i\rangle \to \big(\Phi_{\mathrm{D}}(x,i)|\psi\rangle\big)|(i+1) \bmod T(n)\rangle$$

so that after $T(n)$ steps it computes in the first register the same state as in the quantum circuit. Therefore it computes $f(x)$ and distinguishes between the two cases. Using the premises, we have a classical learning algorithm with time $t(\mathrm{poly}(n), O(\log n))$ and space $s(\mathrm{poly}(n), O(\log n))$ that accomplishes the same task. The classical learning algorithm can be viewed as a randomized decision algorithm that computes $f(x)$ by self-constructing the stochastic matrices in the same time and space. ◀

▶ **Theorem 20.** *If* $\textsc{ContractionPowering} \in \mathsf{promiseBPTISP}(t(n), s(n))$*, where* $t(n) \geq \Omega(n)$ *and* $s(n) \geq \Omega(\log n)$*, then every unital quantum learning algorithm with time T and space S can be simulated classically with time* $t(\mathrm{poly}(2^S T))$ *and space* $s(\mathrm{poly}(2^S T))$.

Proof. Suppose that we have a unital quantum learning algorithm with time T and space $S = \log m$ that distinguishes between two distribution families $\mathcal{X}$ and $\mathcal{Y}$. Let $\Phi_{\mathrm{L}}(z)$ be the unital channel applied when receiving the sample z. With the sample distribution D, let $A = \mathop{\mathbf{E}}_{z\sim D}[K(\Phi_{\mathrm{L}}(z))]$. We note that A is a contraction matrix of dimension $m^2 \times m^2$ as every $K(\Phi_{\mathrm{L}}(z))$ is a contraction. Similar to proof of Lemma 11, the probability of the learning algorithm outputting 0 is

$$\mathop{\mathbf{E}}_{z\sim D^T}\left[\mathrm{vec}(M_0)^\dagger K(\Phi_T)\cdots K(\Phi_1)\mathrm{vec}(\rho_0)\right] = \mathrm{vec}(M_0)^\dagger A^T \mathrm{vec}(\rho_0).$$

What's different from Lemma 11 is that here A is not explicitly given. Instead, by Lemma 1, each time an entry of A is requested, it takes $\mathrm{poly}(mT)$ samples z to approximate the entry to at most $O((m^{2.5}T)^{-1})$ error, so that the approximated matrix $\widetilde{A}$ differs from the actual matrix A by at most $\|\widetilde{A} - A\| \leq O((\sqrt{m}T)^{-1})$. By Lemma 3 it means that $\|\widetilde{A}^T - A^T\| \leq O(m^{-1/2})$. Therefore applying the contraction powering algorithm on $\widetilde{A}$ gives a classical learning algorithm that distinguishes $\mathcal{X}$ and $\mathcal{Y}$ in time $t(\mathrm{poly}(mT))$ and space $s(\mathrm{poly}(mT))$.

The above scheme has two problems. First, a fixed matrix $\widetilde{A}$ cannot be directly stored, and if every time the same entry is requested, the entry is approximated as the average of a different batch of samples, it may result in different requested values for the same entry (even though the difference is small with high probability), similar to the problem in Lemma 14. However, unlike the case in Lemma 14, here the classical contraction powering algorithm is not explicitly given, and may not be robust against changing inputs.

The solution to this problem is the *shift and truncate* method by Saks and Zhou[23], which has found numerous applications in space-bounded algorithms [27] and derandomizations [3, 11]. Concretely, let $P = t(\mathrm{poly}(mT))$ be the largest number of possible requests to entries

of A in the contraction powering algorithm, and take a uniform random number $\zeta \in [8L]$. For simplicity let $L = 12\sqrt{2m}T$ and $N = 24m^{2.5}T$. When the entry A_{jk} is requested, the algorithm takes $t(\mathrm{poly}(mT))$ samples z_i and calculate the average value a of the (j,k)-entries of $K(\Phi_{\mathrm{L}}(z_i))$, so that $|a - A_{jk}| < \frac{1}{8NP}$ with probability at least $1 - 2^{-P}$. The value fed back for the request is

$$\widetilde{A}_{jk} = \frac{1}{N}\left\lfloor N \cdot \mathrm{Re}(a) + \frac{\zeta}{8P} \right\rfloor + \frac{\mathrm{i}}{N}\left\lfloor N \cdot \mathrm{Im}(a) + \frac{\zeta}{8P} \right\rfloor.$$

We claim that with high probability, this value coincides with the fixed value

$$\frac{1}{N}\left\lfloor N \cdot \mathrm{Re}(A_{jk}) + \frac{\zeta}{8P} \right\rfloor + \frac{\mathrm{i}}{N}\left\lfloor N \cdot \mathrm{Im}(A_{jk}) + \frac{\zeta}{8P} \right\rfloor.$$

For the real part, as $|N \cdot \mathrm{Re}(a) - N \cdot \mathrm{Re}(A_{jk})| < \frac{1}{8P}$, there is at most one possibility for ζ such that $\left\lfloor N \cdot \mathrm{Re}(a) + \frac{\zeta}{8P} \right\rfloor \neq \left\lfloor N \cdot \mathrm{Re}(A_{jk}) + \frac{\zeta}{8P} \right\rfloor$, which is of probability $\frac{1}{8P}$, and the same holds for the imaginary part. By the union bound on the bad events during all L requests, with probability

$$1 - \left(2^{-P} + \frac{1}{4P}\right) P \geq \frac{2}{3}$$

for every (j,k) the value $\widetilde{A}_{jk}$ are always the same, and $|\widetilde{A}_{jk} - A_{jk}| \leq \frac{\sqrt{2}}{N} = \frac{1}{m^2 L}$, so $\|\widetilde{A} - A\| \leq L^{-1}$.

The second problem is that because of the approximation error, $\widetilde{A}$ might not be a contraction matrix. This is easily fixed by using the matrix $\widetilde{A}' = \frac{L}{L+1} \cdot \widetilde{A}$ as the input. Since $\|\widetilde{A} - A\| \leq L^{-1}$ with probability $2/3$, it is implied that

$$\|\widetilde{A}'\| = \frac{L}{L+1} \cdot \|\widetilde{A}\| \leq \frac{L}{L+1} \cdot \left(1 + L^{-1}\right) = 1,$$

$$\|\widetilde{A}' - A\| \leq \|\widetilde{A} - A\| + \frac{1}{L+1}\|\widetilde{A}\| \leq \frac{2}{L}.$$

Since $\|\mathrm{vec}(M_0)\|_2 = \sqrt{m/2}$, $\|\mathrm{vec}(\rho_0)\|_2 = 1$, in this case we have (by Lemma 3)

$$\left|\mathrm{vec}(M_0)^\dagger (\widetilde{A}'^T - A^T)\mathrm{vec}(\rho_0)\right| \leq \frac{\sqrt{2m}T}{L} = \frac{1}{12}.$$

Since the error of the original quantum learning algorithm can be amplified to $1/4$ so that $\mathrm{vec}(M_0)^\dagger A^T \mathrm{vec}(\rho_0)$ is in $[0, 1/4]$ or $[3/4, 1]$, we conclude that with probability $5/6$,

$$\mathrm{vec}(M_0)^\dagger \widetilde{A}'^T \mathrm{vec}(\rho_0) \in [0, 1/3] \text{ or } [2/3, 1]$$

Therefore the two cases can be distinguished by the classical contraction powering algorithms on $\widetilde{A}'$, and it can be repeated for constant rounds so that the total error rate is brought down to $1/3$. ◀

▶ **Corollary 21.** *If* ContractionPowering $\in$ promiseBPL*, then every unital quantum learning algorithm with time T and space S can be simulated classically with time* $\mathrm{poly}(2^S T)$ *and space* $O(S + \log T)$*.*

Since by Theorem 10 we already know ContractionPowering $\in$ promiseBQ$_{\mathsf{U}}$L, combined with Theorem 19, we get the equivalence between efficient simulations of decision problems and learning problems:

▶ **Theorem 22.** *Every (unital) quantum learning algorithm with time T and space S can be simulated classically with time* $\mathrm{poly}(2^S T)$ *and space* $O(S + \log T)$*, if and only if* $\mathsf{promiseBQ_UL} = \mathsf{promiseBPL}$.

Also, as we already know $\mathsf{promiseBQ_UL} \subseteq \mathsf{promiseL^2}$ [28], we have the following unconditional result:

▶ **Corollary 23.** *Every unital quantum learning algorithm with time T and space S can be simulated classically with time* $2^{O(S^2+\log^2 T)}$ *and space* $O(S^2 + \log^2 T)$.

7.2 Classical Simulation when One Family is Singleton

▶ **Theorem 24.** *If $\mathcal{Y} = \{Y\}$, then any quantum learning algorithm that distinguishes between $\mathcal{X}$ and $\mathcal{Y}$ within time T and space S can be simulated classically in time* $\mathrm{poly}(2^S T)$ *and space* $O(S + \log T)$.

References

1. Paul Beame, Shayan Oveis Gharan, and Xin Yang. Time-space tradeoffs for learning finite functions from random evaluations, with applications to polynomials. In *Conference On Learning Theory (COLT 2018)*, pages 843–856, 2018.
2. Charles Bennett. Notes on landauer's principle, reversible computation, and maxwell's demon. *Studies In History and Philosophy of Science Part B: Studies In History and Philosophy of Modern Physics*, 34:501–510, September 2003. doi:10.1016/S1355-2198(03)00039-X.
3. Jin-Yi Cai, Venkatesan T Chakaravarthy, and Dieter van Melkebeek. Time-space tradeoff in derandomizing probabilistic logspace. *Theory of Computing Systems*, 39(1):189–208, 2006.
4. Shantanav Chakraborty, András Gilyén, and Stacey Jeffery. The power of block-encoded matrix powers: Improved regression techniques via faster hamiltonian simulation. In *46th International Colloquium on Automata, Languages, and Programming (ICALP 2019)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2019.
5. Bill Fefferman, Hirotada Kobayashi, Cedric Yen-Yu Lin, Tomoyuki Morimae, and Harumichi Nishimura. Space-efficient error reduction for unitary quantum computations. In *43rd International Colloquium on Automata, Languages, and Programming (ICALP 2016)*, volume 55. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2016. doi:10.4230/LIPIcs.ICALP.2016.14.
6. Bill Fefferman and Cedric Yen-Yu Lin. A complete characterization of unitary quantum space. In *9th Innovations in Theoretical Computer Science Conference (ITCS 2018)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2018.
7. Bill Fefferman and Zachary Remscrim. Eliminating intermediate measurements in space-bounded quantum computation, 2020. arXiv:2006.03530.
8. Sumegha Garg, Ran Raz, and Avishay Tal. Extractor-based time-space lower bounds for learning. In *Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing (STOC 2018)*, pages 990–1002, 2018.
9. András Gilyén. Quantum singular value transformation & its algorithmic applications. *ILLC Dissertation Series*, 2019.
10. András Gilyén, Yuan Su, Guang Hao Low, and Nathan Wiebe. Quantum singular value transformation and beyond: exponential improvements for quantum matrix arithmetics. In *Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing*, pages 193–204, 2019.
11. Ofer Grossman and Yang P Liu. Reproducibility and pseudo-determinism in log-space. In *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 606–620. SIAM, 2019.

12 Aram W Harrow, Avinatan Hassidim, and Seth Lloyd. Quantum algorithm for linear systems of equations. *Physical review letters*, 103(15):150502, 2009.

13 Gillat Kol, Ran Raz, and Avishay Tal. Time-space hardness of learning sparse parities. In *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing (STOC 2017)*, pages 1067–1080, 2017.

14 R. Landauer. Irreversibility and heat generation in the computing process. *IBM Journal of Research and Development*, 5(3):183–191, 1961.

15 Klaus-Jörn Lange, Pierre McKenzie, and Alain Tapp. Reversible space equals deterministic space. *Journal of Computer and System Sciences*, 60(2):354–367, 2000.

16 Chris Marriott and John Watrous. Quantum arthur–merlin games. *Computational Complexity*, 14(2):122–152, 2005.

17 Dieter van Melkebeek and Thomas Watson. Time-space efficient simulations of quantum computations. *Theory of Computing*, 8(1):1–51, 2012.

18 Dana Moshkovitz and Michal Moshkovitz. Entropy samplers and strong generic lower bounds for space bounded learning. In *9th Innovations in Theoretical Computer Science Conference (ITCS 2018)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2018.

19 Michael A Nielsen and Isaac L Chuang. *Quantum Computation and Quantum Information*. Cambridge University Press, 2010.

20 David Perez-Garcia, Michael M Wolf, Denes Petz, and Mary Beth Ruskai. Contractivity of positive and trace-preserving maps under l p norms. *Journal of Mathematical Physics*, 47(8):083506, 2006.

21 Ran Raz. A time-space lower bound for a large class of learning problems. In *58th IEEE Annual Symposium on Foundations of Computer Science (FOCS 2017)*, pages 732–742, 2017.

22 Ran Raz. Fast learning requires good memory: A time-space lower bound for parity learning. *Journal of the ACM (JACM)*, 66(1):1–18, 2018.

23 Michael Saks and Shiyu Zhou. $\mathsf{BPHSPACE}(s) \subseteq \mathsf{DSPACE}(s^{3/2})$. *Journal of computer and system sciences*, 58(2):376–403, 1999.

24 Ohad Shamir. Fundamental limits of online and distributed algorithms for statistical learning and estimation. In *Advances in Neural Information Processing Systems 27 (NIPS 2014)*, pages 163–171, 2014.

25 Yaoyun Shi. Both toffoli and controlled-not need little help to do universal quantum computing. *Quantum Info. Comput.*, 3(1):84–92, 2003.

26 Jacob Steinhardt, Gregory Valiant, and Stefan Wager. Memory, communication, and statistical queries. In *Conference on Learning Theory (COLT 2016)*, pages 1490–1516, 2016.

27 Amnon Ta-Shma. Inverting well conditioned matrices in quantum logspace. In *Proceedings of the 45th Annual ACM Symposium on Theory of Computing (STOC 2013)*, pages 881–890, 2013.

28 John Watrous. Space-bounded quantum complexity. *Journal of Computer and System Sciences*, 59(2):281–326, 1999.

29 Fuzhen Zhang. *Matrix theory: basic results and techniques*. Springer Science & Business Media, 2011.

Online Stochastic Matching with Edge Arrivals

Nick Gravin
ITCS, Shanghai University of Finance and Economics, China

Zhihao Gavin Tang
ITCS, Shanghai University of Finance and Economics, China

Kangning Wang
Department of Computer Science, Duke University, Durham, NC, USA

Abstract

Online bipartite matching with edge arrivals remained a major open question for a long time until a recent negative result by Gamlath et al., who showed that no online policy is better than the straightforward greedy algorithm, i.e., no online algorithm has a worst-case competitive ratio better than 0.5. In this work, we consider the bipartite matching problem with edge arrivals in a natural stochastic framework, i.e., Bayesian setting where each edge of the graph is independently realized according to a known probability distribution.

We focus on a natural class of prune & greedy online policies motivated by practical considerations from a multitude of online matching platforms. Any prune & greedy algorithm consists of two stages: first, it decreases the probabilities of some edges in the stochastic instance and then runs greedy algorithm on the pruned graph. We propose prune & greedy algorithms that are 0.552-competitive on the instances that can be pruned to a 2-regular stochastic bipartite graph, and 0.503-competitive on arbitrary stochastic bipartite graphs. The algorithms and our analysis significantly deviate from the prior work. We first obtain analytically manageable lower bound on the size of the matching, which leads to a non-linear optimization problem. We further reduce this problem to a continuous optimization with a constant number of parameters that can be solved using standard software tools.

2012 ACM Subject Classification Theory of computation → Online algorithms; Theory of computation → Approximation algorithms analysis; Theory of computation → Mathematical optimization; Mathematics of computing → Graph algorithms

Keywords and phrases online matching, graph algorithms, prophet inequality

Digital Object Identifier 10.4230/LIPIcs.ICALP.2021.74

Category Track A: Algorithms, Complexity and Games

Related Version *Full Version*: `https://arxiv.org/abs/1911.04686`

Funding This work is supported by Science and Technology Innovation 2030 – "New Generation of Artificial Intelligence" Major Project No.(2018AAA0100903), National Natural Science Foundation of China (NSFC) grant 61932002, Program for Innovative Research Team of Shanghai University of Finance and Economics (IRTSHUFE), Fundamental Research Funds for the Central Universities.
Zhihao Gavin Tang: supported by NSFC grant 61902233.
Kangning Wang: supported by NSF grant CCF-1637397 and ONR grant N00014-19-1-2268.

1 Introduction

Matching theory is a central area in combinatorial optimization with a big range of applications [28]. Many market models for jobs, commercial products, dating, healthcare, etc., rely on matching as a fundamental mathematical primitive. These examples often aim to describe environments that evolve in real time and thus are relevant to the area of online bipartite matching initiated by a seminal paper of Karp, Vazirani and Vazirani [26]. In this work they consider the one-sided vertex-arrival model within the competitive analysis framework, i.e., vertices only on one side of a bipartite graph appear online and each new vertex reveals all its

incident edges. The algorithm immediately and irrevocably decides to which vertex (if any) the new arrival is matched. They studied the worst-case performance of online algorithms and solved the problem optimally with an elegant $(1 - 1/e)$-competitive algorithm, named RANKING. Later, the proof of the result has been simplified by a series of papers [6, 19, 13].

The interest in matching models and online bipartite matching problems in particular has been on the rise since a decade ago due to emergence of the internet advertisement industry and online market platforms [32]. With the large amount of available data on many online platforms from the day-to-day user activities, more recent literature has shifted more towards *stochastic* models, also called Bayesian in the economically oriented work. In particular, Feldman, Mehta, Mirrokni and Muthukrishnan [16] proposed a stochastic model in which online vertices are drawn i.i.d. from a known distribution and improved[1] the competitive ratio of the classic result by Karp, Vazirani and Vazirani to 0.67. The competitive ratio has been further improved by a series of papers [3, 30, 24] to 0.706. Another line of work [25, 29] studied the model in which online vertices arrive in a random order and showed that the RANKING algorithm is 0.696-competitive.

The aforementioned results and other works, e.g., [33, 7, 12, 36, 18, 22, 23, 2], have made remarkable progress on different online matching settings with vertex arrivals, i.e., models where all incident edges of a new vertex are reported to the algorithm. However, more general arrival models are much less understood. E.g., one of the most natural and nonrestrictive extensions of online bipartite matching to the model where edges appear online and must be immediately matched or discarded was not known to have a competitive ratio better than the greedy algorithm for a long time. Only a recent negative result by Gamlath et al. [18] closed this tantalizing question showing that no online algorithm can be better than 0.5-competitive in the worst case. Algorithms with better performance are only known for quite special family of graphs, e.g., bounded-degree graphs [8] and forests [32, 8], or under strong assumptions on the edge arrival order, e.g., random arrival order [21].

It might seem that the edge-arrival model is too general to allow non-trivial theoretical results without strong assumptions on the instance. Thus it is not very surprising that practically motivated models do not usually consider online setting with edge arrivals. On the other hand, most of the specific applications posses additional structure and extra information that might allow to break the theoretical barrier. The edge-arrival online model besides pure theoretical interest and clean mathematical formulation, is indeed relevant to practical problems not unlike the examples we discuss below.

Practical Motivation: Edge Arrivals

Imagine any online matching platform for job search, property market, or even online dating. All these instances can be viewed as online matching processes in bipartite graphs. They also share a common trait that the realization of any particular edge is not instantaneous, often consumes significant effort and time from one or both sides of the potential match, and may exhibit complex concurrent behavior across different parties of the market. The platform can be thought of as an online matching algorithm, if it has any degree of control to intervene in the process of edge formation at any point.[2] However, the platform does not have enough power to control the order in which edges are realized. Hence, using arbitrary edge arrival order seems to be an appropriate modeling choice in these situations.

---

[1] Their result holds under the assumption that the expected number of vertices for each type is an integer.

[2] Even if the platform cannot directly prohibit an edge formation or disallow certain matches, it usually can affect outcome indirectly by restricting access/information exchange between certain pairs of agents, so that they never consider each other as potential matches.

Another notable feature of these instances is the vast amount of historical data accumulated over time. The data enables the platform to estimate the probability of a potential match between any pair of given agents. Thus the Bayesian (stochastic) approach widely adapted in economics seems to be another reasonable modeling choice. This raises the following natural question that to the best of our knowledge has not been considered before: *Is there an online matching algorithm for stochastic bipartite graphs with edge arrivals that is better than greedy?* This question is the main focus of our work. Let us first specify the model in more details.

Our Model: Edge Arrivals in Stochastic Graphs

We call our model *online stochastic matching with edge arrivals*. It is a relaxation of the standard edge-arrival model that performs on a random bipartite graph. In particular, we assume the input graph G is stochastic. That is, each edge e exists (is realized) in G independently with probability p_e and the probabilities $(p_e)_{e\in E(G)}$ are known to the online algorithm.[3] The algorithm observes a sequence of edges arriving online in a certain (unknown) order. Upon the arrival of an edge e, we observe the realization of e and if e exists, then the algorithm immediately and irrevocably decides whether to add e to the matching. We assume that the arrival order of the edges is chosen by an *oblivious* adversary, i.e., an adversary who does not observe the realization of the edges and algorithm's decisions, which is a standard assumption in the literature on online algorithms in stochastic settings (see, e.g., [27]). We compare the expected performance of our algorithm with the maximum matching in hindsight, i.e., the expected size of a maximum matching over the randomness of all edges.

1.1 Comparison with Other Stochastic Models

Our model is closely connected to two existing theoretical lines of works on stochastic bipartite matching and prophet inequality in algorithmic game theory. Below we compare our model with the most relevant results in each of these lines of works.

Stochastic Probing Model

It has the same ingredient as our model: the underlying stochastic graph. That is, the input is also a bipartite graph with the stochastic information on existence probability of every edge e. On the other hand, it is an offline model under the query-commit framework, i.e., the algorithm can check the existence of the edges in any order. However, if an edge exists, it has to be included into the solution. For this model, an adaptation of the RANKING algorithm by Karp, Vazirani and Vazirani is $(1-1/e)$-competitive. Costello, Tetali and Tripathi [11] provided a 0.573-approximation algorithm on general (non-bipartite) graphs and showed that no algorithm can have an approximation ratio larger than 0.898. Recently, Gamlath, Kale and Svensson [17] designed a $(1-1/e)$-approximation algorithm for the edge-weighted version of this problem.

[3] Note that some independence assumption across the edges is necessary. If we allow arbitrary probability distribution over the sets of realized edges, the model would be as difficult as the worst case online setting.

Prophet Inequality for Bipartite Matching

Consider a bipartite graph, where all edges have random values independently sampled from given probability distributions. Upon the arrival of an edge, we see the realization of its value and decide immediately whether to include this edge if possible in the matching. This model was originally proposed by Kleinberg and Weinberg [27] for a more general setting of intersection of k matroids. Gravin and Wang [20] studied explicitly the setting of bipartite matching and provided a $\frac{1}{3}$-approximation. Ezra et al. [15] later improved this ratio to 0.337. Our model can be viewed as an unweighted version of this prophet setting. Indeed, we assume that each edge has value either 0 or 1 and, hence, the probability distribution is a product of Bernoulli random variables summarized by existence probabilities $(p_e)_{e\in E(G)}$. Note that the weighted case is strictly harder than the unweighted one. Gravin and Wang [20] provided a 1/2.25 hardness result for the weighted setting while our goal is to design an online algorithm with a competitive ratio strictly better than 1/2. After all, the simple greedy algorithm achieves a competitive ratio of 1/2 for unweighted graphs.

1.2 Our Results and Techniques

We study a specific family of algorithms, named *Prune & Greedy*. The algorithm consists of two steps: (i) prune the graph by removing or decreasing probabilities of certain edges in G; (ii) greedily take every edge in the pruned instance. In particular, upon the arrival of an edge, we always drop it with certain probability so that its realization probability is consistent with the pruned graph.

We argue that the family of *Prune & Greedy* algorithms is of independent interest due to their practical relevance. Indeed, in those market applications we discussed above, the online platform often cannot prevent the matching between two parties (pair of vertices) once they realized their compatibility. But the platform usually possesses all the stochastic information about the graph and thus is fully capable of implementing pruning step by restricting information to its users. After that participants naturally implement greedy matching by exploring compatibilities with the other side of the graph exposed to them by the platform in an arbitrary order.

As our first result (Theorem 5 and Theorem 14), we identify a class of graphs on which greedy algorithm performs better than the worst-case competitive ratio of 1/2. We compare the size of the matching to the total number of vertices, a stronger benchmark than the expected size of maximum matching. As the pruning step naturally decreases the expected size of the maximum matching, the change of the benchmark is indeed necessary. Specifically, we find that on log-normalized[4] c-regular graphs with small $c = 2$ the greedy algorithm matches at least 0.552 vertices. This result immediately implies that if initial stochastic graph has a 2-regular bipartite spanning subgraph, then *Prune & Greedy* algorithm is 0.552-competitive.

Second, we propose a 0.503-competitive *Prune & Greedy* algorithm for any bipartite stochastic graph (Theorem 8). This result confirms that the edge-arrival model is theoretically interesting in the stochastic framework. A complementary hardness result (Theorem 15) shows that no online algorithm can be better than 2/3-competitive.

[4] For any vertex in a log-normalized c-regular graph, the probability that it has at least an adjacent edge is $1 - e^{-c}$. The definition is given in Section 3. Informally, a log-normalized c-regular graph is a c-regular graph where all edges have weights $\varepsilon \approx 0$.

Our techniques

We first build some intuition by analyzing the greedy algorithm on log-normalized c-regular graphs. One of the main challenges is that different events such as "edge e is matched", or "vertex u is matched" may have complex dependencies. This makes it very difficult to write the performance of the greedy algorithm in an explicit analytical form. We consider simpler to analyze events: "there exists a vertex u whose first realized edge is the edge (uv)", which guarantee that vertex v is matched at the end of the algorithm. This relaxation allows us to break the analysis into independent optimization problems per each vertex. We derive a guarantee $f(c)$ on the fraction of vertices matched by the greedy algorithm for any c-regular stochastic graph, where the function $f(c)$ has a single peak around $c = 2$ with $f(2) \approx 0.532$. I.e., we develop an analytically tractable relaxation on the performance of greedy that we later generalize to non-regular case. Interestingly, the greedy algorithm may perform worse on log-normalized c-regular graphs for larger c. In particular, greedy is not better than 0.5-competitive on c-regular graphs as $c \to \infty$.

However, this relaxation alone is not sufficient for the general case of non-regular graphs, since such analysis is not tailored in any way to the expected size of optimal matching. To this end, we consider an LP relaxation (an upper bound) on the expected optimal matching in stochastic graphs proposed in [17]. This LP gives a set of values $(x_e)_{e\in E(G)}$ with the objective $\sum_{e\in E(G)} x_e$ which satisfy a set of constraints that could be conveniently added to our optimization problem. Our analysis for regular graphs prompted us to the strategy of pruning each edge of the graph to $2 \cdot x_e$ so that the pruned graph is similar to a 2-regular graph. Unfortunately, this might not be a feasible operation when p_e (realization probability of e) is smaller than $2 \cdot x_e$. For these edges, it is then natural to keep their original existence probability. Our analysis can be similarly localized to an optimization problem for individual vertices, albeit the optimization becomes more complex. The main technical challenge is to solve an unwieldy optimization problem due to the "irregular" edges. Note that even a simpler optimization problem for c-regular graphs has a continuous optimal solution (i.e., is a limit of increasing discrete instances), which required computer assisted calculations to obtain the bound.

Finally, building on top of the relaxation we discussed above, we provide a more refined analysis for the case of 2-regular graphs. Namely, we consider second-order events that also witness the matching status of a vertex. We prove that the greedy algorithm is at least 0.552-competitive on 2-regular graphs, improving on the easier $f(2) \approx 0.532$ bound. We note that the same approach could in principle be extended to general *Prune & Greedy* algorithm for arbitrary graphs with optimization part still localizable to individual vertices. However, the optimization problem becomes too complicated to solve analytically. We leave it as an interesting open question to have a better analysis of the *Prune & Greedy* algorithms. On the positive side, the improved analysis for 2-regular graphs suggests that performance of *Prune & Greedy* algorithms should be noticeably better than what we proved in this paper.

1.3 Other Related Works

The edge-arrival setting is also studied under the free-disposal assumption, i.e., the algorithm is able to dispose of previously accepted edges. McGregor [31] gave a deterministic $\frac{1}{3+2\sqrt{2}} \approx 0.171$-competitive algorithm for weighted graphs. Varadaraja [35] proved the optimality of this result among deterministic algorithms. Epstein, Levin, Segev and Weimann [14] gave a $\frac{1}{5.356} \approx 0.186$-competitive randomized algorithm later and proved a hardness result of $\frac{1}{1+\ln 2} \approx 0.591$ for unweighted graphs. Recently, the hardness bound is improved to $2 - \sqrt{2} \approx 0.585$ by Huang et al. [23]. We remark that the question of designing an algorithm that beats 0.5-competitive remains open.

One of the earlier work on stochastic matching is due to Chen et al. [9]. They proposed stochastic model with edge probing motivated by real life matching applications such as kidney exchange. This model is more complex than the stochastic probing model we discussed before, since it has an additional constraint per each vertex v on how many times edges incident to v can be queried. Another difference is that a weaker benchmark than the optimal offline matching has to be used in this setting. Chen et al. developed a $\frac{1}{4}$-approximation algorithm. Bansal et al. [4] considered the weighted version and provided a $\frac{1}{3}$-approximation and a $\frac{1}{4}$-approximation for bipartite graphs and general graphs respectively. The ratio for general graphs was further improved to $\frac{1}{3.709}$ by Adamczyk, Grandoni and Mukherjee [1], and then to $\frac{1}{3.224}$ by Baveja et al. [5].

There has been prior work in the online matching literature on regular graphs [3, 25, 34, 21, 10]. However, the notion of stochastic regular graphs we defined is very different from the notion of standard regular graphs, the previous techniques are not directly applicable.

2 Preliminaries

The bipartite graph $G = (L, R, E)$ consists of left and right sides denoted respectively L and R. The graph G is a multigraph, i.e., E is a multiset that may have multiple parallel edges between the same pair of vertices. We use E_v to denote the multiset of edges incident to vertex v and E_{uv} to denote the multiset of edges connecting u and v. We consider the Bayesian model, where each edge $e \in E$ is realized with probability $p_e \in [0, 1]$, which is known in advance. The realizations of different edges are independent. We are interested in online matching algorithms with the objective of maximizing the expected size of the matching. We assume that all edges in E arrive one by one according to some fixed unknown order (i.e., oblivious adversarial order). Upon arrival of the edge e, the algorithm observes whether or not e is realized. If the edge exists, the algorithm immediately and irrevocably decides whether to include e into the matching; the algorithm does nothing, if the edge is not realized. We compare the performance of the algorithm with the performance of the optimal offline algorithm, also known as the *prophet*, who knows the realization of the whole graph in hindsight, i.e., $\mathsf{OPT} = \mathbf{E}[\text{size of maximum matching}]$.

A natural online matching strategy is the greedy algorithm: Take every available edge $e = (u, v)$ whenever both vertices u and v have not yet been matched. Obviously, the greedy algorithm is a 0.5-approximation, since it selects a *maximal* matching in all possible realizations of the graph, which is always a 0.5-approximation to the *maximum* matching.

Paper Roadmap

In Section 3, we introduce the notion of stochastic regular graphs and establish an analytical bound on the competitive ratio of the greedy algorithm on c-regular graphs. In Section 4, we design a Prune & Greedy algorithm that is 0.503-competitive for general inputs. Section 5 provides a more refined analysis of the greedy algorithm on 2-regular graphs. In Section 6, we give a simple impossibility result showing that no online algorithm can do better than $\frac{2}{3}$ of the expected optimum. Omitted proofs are in the full version.

3 Warm-up: Regular Graphs

A regular graph is a graph whose vertices have the same degree. But how do we define vertex degrees in a stochastic graph? One standard way is to use the expected vertex degree, i.e., $\sum_{e \in E_u} p_e$ for the degree of a vertex $u \in L$. However, the expectation alone does not

contain all the important information about a degree distribution. Consider for example a vertex a having only one incident edge (a, b) with $p_{(a,b)} = 1$ and a vertex u having 2 incident parallel edges $e = (u, v)$ with probability $p_e = 0.5$ for each $e \in E_u$. Both vertices a and u have the same expected degree, but while a always has exactly one incident edge, u gets no incident edges with 0.25 probability. On the other hand, u may have 2 incident edges in some realizations, which is almost the same for our purposes as having only a single incident edge.

A good way to reconcile this difference is to substitute each edge (u, v) by multiple parallel edges $e_i = (u, v)$ with small probabilities such that $p_{(u,v)}$ matches the probability that at least one of e_i edges exists. Alternatively, we can define a *log-normalized weight* for each edge e as $w_e \stackrel{\text{def}}{=\!=} -\ln(1 - p_e)$, i.e., given an input instance $G = (L, R, E)$ we construct a one-to-one correspondence between vectors of probabilities $\mathbf{p} = (p_e)_{e\in E}$ and vectors of log-normalized weights $\mathbf{w} = (w_e)_{e\in E}$. In particular, if we split an edge with log-normalized weight w_e into two edges $e_1 = e_2 = (u, v)$ with $w_{e_1} + w_{e_2} = w_e$, then the new instance gets only harder, i.e., any online algorithm for the new instance can be easily adapted to the original instance with the same or better performance. Indeed, notice that the probability that at least one of the edges e_1, e_2 exists equals $1 - e^{-w_{e_1}} \cdot e^{-w_{e_2}} = 1 - e^{-w_e}$, the probability that e exists, i.e., there is a probability coupling between the event that e exists with the event that at least one of e_1, e_2 exists. Then, we can substitute e in any arrival order with a pair of consecutive edges e_1 and e_2 and match e whenever the online algorithm matches e_1 or e_2 in the modified instance. Thus the log-normalized weight is the correct notion for us to do additive operations over the existence probabilities and leads to the following definition of the regular stochastic graph.

▶ **Definition 1.** *A graph G is a* log-normalized c-regular graph *if for every $v \in L \cup R$, $\sum_{e\in E_v} w_e = c$.*

We restrict our attention to log-normalized regular graphs in the remainder of this section. Our goal is to analyze the performance of Greedy on log-normalized c-regular graphs for a small constant c. Remarkably, it is not easy to give a precise answer and produce a tight worst-case estimate even for a specific value $c = 1$.

We first introduce a few short hand notations for the events that will be frequently used throughout the paper.

▶ **Definition 2.** *Fix an arbitrary edge arrival order σ and an edge $e \in E_{uv}$, define the following events:*

1. *$\exists e$: the event that e is realized.*
2. *$M_u(e)$: the event that u is matched right before e arrives.*
3. *$Q_u(e)$: the event that no edge of E_u is realized before e arrives. Let $q_u(e) \stackrel{\text{def}}{=\!=} \mathbf{Pr}[Q_u(e)]$.*
4. *$F_u(e) \stackrel{\text{def}}{=\!=} Q_u(e) \cap \exists e$: the event that e is the first realized edge of vertex u.*

The following lemma gives a lower bound on the matching probability of any vertex. This analytically tractable bound will allow us to reduce the global optimization for the competitive ratio of our algorithm to the local optimization per individual vertex. The lemma will also be useful for the general case, i.e., for not necessarily regular graphs, which we discuss in Section 4. To be consistent with the notations of the next section, let $x_e = \frac{w_e}{c}$ and $y_e = 1 - e^{-w_e}$. We have the property that $\sum_{e\in E_v} x_e = 1$ for every $v \in V$ and y_e equals the probability that e is realized ($\exists e$). We state the following lemma with the weaker condition of $\sum_{e\in E_v} x_e \leqslant 1$ for more general use in later sections.

▶ **Lemma 3.** *For any $v \in R$, if $\sum_{e\in E_v} x_e \leqslant 1$, then*

$$\mathbf{Pr}\left[v \text{ is matched}\right] \geqslant \mathbf{Pr}\left[\bigcup_{e=(u,v)\in E_v} F_u\left(e\right)\right] \geqslant \sum_{e=(u,v)\in E_v} x_e \cdot \left(1-\exp\left(-\frac{q_u(e)\cdot y_e}{x_e}\right)\right). \quad (1)$$

Proof. For each edge $e \in E_v$, consider the case when edge $e = (u, v)$ arrives and the event $F_u(e) = Q_u(e) \cap \exists e$ happens. At this moment, either v is already matched, or e will be included in the matching by Greedy. Therefore, whenever $\exists u \in L$ such that there is an edge $e \in E_{uv}$ making $F_u(e)$ happen, v is covered by Greedy.

Next, the events $\left\{\bigcup_{e\in E_{uv}} F_u(e)\right\}_{u\in L}$ are mutually independent, since (i) the event $\bigcup_{e\in E_{uv}} F_u(e)$ only depends on the random realization of the edges in E_u and (ii) $E_u \cap E_{u'} = \varnothing$ when $u \neq u'$.[5]

Lastly, $F_u(e_1) \cap F_u(e_2) = \varnothing$ for any $e_1, e_2 \in E_{uv}$, and thus $\mathbf{Pr}[\bigcup_{e\in E_{uv}} F_u(e)] = \sum_{e\in E_{uv}} q_u(e) \cdot y_e$. Putting the above observations together, we have

$$\begin{aligned}
\mathbf{Pr}\left[v \text{ is matched}\right] &\geqslant \mathbf{Pr}\left[\bigcup_{e\in E_v} F_u\left(e\right)\right] = 1-\mathbf{Pr}\left[\bigcap_{e\in E_v} \overline{F_u\left(e\right)}\right] = 1-\prod_u \mathbf{Pr}\left[\bigcap_{e\in E_{uv}} \overline{F_u\left(e\right)}\right] \\
&= 1-\prod_u\left(1-\sum_{e\in E_{uv}} q_u(e)\cdot y_e\right) \geqslant 1-\prod_u \exp\left(-\sum_{e\in E_{uv}} q_u(e)\cdot y_e\right) \\
&= 1-\exp\left(-\sum_{e\in E_v} q_u(e)\cdot y_e\right) \geqslant \sum_{e\in E_v} x_e\cdot\left(1-\exp\left(-\frac{q_u(e)\cdot y_e}{x_e}\right)\right),
\end{aligned}$$

where the second inequality follows from the fact that $1 - z \leqslant e^{-z}$ and the last inequality follows from Jensen's inequality and the concavity of function $1 - \exp(-z)$ (recalling $\sum_{e\in E_v} x_e \leqslant 1$). ◀

Thus, we may think of the quantity $x_e \cdot \left(1-\exp\left(-\frac{q_u(e)\cdot y_e}{x_e}\right)\right)$ as the contribution of edge e in the algorithm.[6] Observe that this contribution depends on the event $Q_u(e)$ for $u \in L$. We sum the (1) bound over all $v \in R$ and change the order of summations.

$$\begin{aligned}
\mathsf{ALG} = \sum_{v\in R}\mathbf{Pr}\left[v \text{ is matched}\right] &\geqslant \sum_{v\in R}\sum_{e=(u,v)\in E_v} x_e\cdot\left(1-\exp\left(-\frac{q_u(e)\cdot y_e}{x_e}\right)\right) \\
&= \sum_{u\in L}\sum_{e=(u,v)\in E_u} x_e\cdot\left(1-\exp\left(-\frac{q_u(e)\cdot y_e}{x_e}\right)\right). \quad (2)
\end{aligned}$$

▶ **Lemma 4.** *For all $u \in L$,*

$$\sum_{e\in E_u} x_e\cdot\left(1-\exp\left(-\frac{q_u(e)\cdot y_e}{x_e}\right)\right) \geqslant \int_0^1 \left(1-e^{-ce^{-cz}}\right)\mathrm{d}z.$$

Proof. Let u be any fixed vertex in L and $e_1, e_2, \ldots, e_k$ be the edges of E_u enumerated according to their arrival order. For notation simplicity, we use q_i, x_i and y_i to denote $q_u(e_i), x_{e_i}$ and y_{e_i} respectively. Then we have

$$Q_u\left(e_i\right) = \bigcap_{j<i}\overline{\exists e_j} = \bigcap_{j<i}\nexists e_j; \qquad q_i = \mathbf{Pr}\left[Q_u\left(e_i\right)\right] = \prod_{j<i}(1-y_j) = \prod_{j<i} e^{-c\cdot x_j} = e^{-c\cdot\sum_{j<i} x_j}.$$

[5] It is the only place where we use that G is bipartite. Indeed, our result can be generalized to triangle-free graphs.

[6] Note that this quantity is not necessarily a lower bound of the probability that edge e is matched.

Since $\int_0^{x_i} c \cdot e^{-cz} \mathrm{d}z = 1 - e^{-cx_i} = y_i$ and $1 - \exp(-q_i \cdot z)$ is a concave function of z , we can apply Jensen's inequality to get

$$1 - e^{-\frac{q_i \cdot y_i}{x_i}} = 1 - \exp\left(-q_i \cdot \frac{1}{x_i} \int_0^{x_i} c \cdot e^{-cz} \mathrm{d}z\right) \geqslant \frac{1}{x_i} \int_0^{x_i} \left(1 - \exp\left(-q_i \cdot c \cdot e^{-cz}\right)\right) \mathrm{d}z$$
$$= \frac{1}{x_i} \int_0^{x_i} \left(1 - \exp\left(-c \cdot e^{-c \cdot \sum_{j<i} x_j} \cdot e^{-cz}\right)\right) \mathrm{d}z = \frac{1}{x_i} \int_{\sum_{j<i} x_j}^{\sum_{j \leqslant i} x_j} \left(1 - e^{-ce^{-cz}}\right) \mathrm{d}z.$$

Summing this inequality over $i \in [k]$, we have

$$\sum_{e \in E_u} x_e \cdot \left(1 - \exp\left(-\frac{q_u(e) \cdot y_e}{x_e}\right)\right) = \sum_{i=1}^{k} x_i \cdot \left(1 - \exp\left(-\frac{q_i \cdot y_i}{x_i}\right)\right)$$
$$\geqslant \sum_{i=1}^{k} \int_{\sum_{j<i} x_j}^{\sum_{j \leqslant i} x_j} \left(1 - e^{-ce^{-cz}}\right) \mathrm{d}z = \int_0^1 \left(1 - e^{-ce^{-cz}}\right) \mathrm{d}z. \quad \blacktriangleleft$$

Let us denote the lower bound in the Lemma 4 as $h_1(c) \stackrel{\text{def}}{=\!=} \int_0^1 \left(1 - e^{-ce^{-cz}}\right) \mathrm{d}z$.

▶ **Theorem 5.** *The competitive ratio of* Greedy *on c-regular graphs is at least $h_1(c)$, which is at least* 0.532 *when $c = 2$.*

Proof. By equation (2) and Lemma 4, we have

$$\mathsf{ALG} \geqslant \sum_{u \in L} \sum_{e \in E_u} x_e \cdot \left(1 - \exp\left(-\frac{q_u(e) \cdot y_e}{x_e}\right)\right) \geqslant \sum_{u \in L} h_1(c).$$

This concludes the theorem by noticing that $|L|$ is an upper bound of OPT. $\blacktriangleleft$

▶ Remark. When $c = 2$, the competitive ratio is at least $h_1(c) \geqslant 0.532$. Notice that h_1 has a peak at $c \approx 2.1$, but it gets smaller again for $c > 3$ (see the full version for a plot) and our analysis gives relatively weak results for large c. One reason is because of the relaxation from Lemma 3. On the other hand, Greedy indeed does not perform well on c-regular graphs when c is large. In particular, Greedy is no better than 0.5-competitive on c-regular graphs when c goes to infinity.

▶ **Theorem 6.** *Greedy is at most* 0.5*-competitive on log-normalized c-regular graphs when $c \to \infty$.*

Proof. Consider the graph shown in Figure 1. We use $L_1 = \{u_i\}_{i=1}^{n+1}, R_1 = \{v_j\}_{j=1}^{n+1}, L_2 = \{u'_i\}_{i=1}^{n}$ and $R_2 = \{v'_j\}_{j=1}^{n}$ to denote the vertices in the graph. The edges are defined as the following:

1. For each $i \in [n+1]$, there is a (red solid) edge (u_i, v_i) with existence probability $1 - \varepsilon$.
2. For each pair of $(u, v) \in (L_2 \times R_1) \cup (L_1 \times R_2)$, there is a (green/blue dashed) edge (u, v) with existence probability $1 - \varepsilon$.

It is easy to verify the graph is log-normalized regular. When $\varepsilon \to 0$, with high probability, the graph admits a perfect matching with size $2n + 1$. On the other hand, consider the case when the red edges arrive first. With high probability, all these edges exist and Greedy matches $n + 1$ edges. This finishes the proof since $\frac{n+1}{2n+1} \to \frac{1}{2}$ when $n \to \infty$. $\blacktriangleleft$

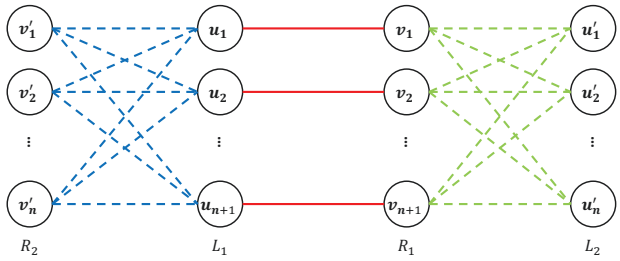


Figure 1 Hard instance for GREEDY on regular graphs.

4 Prune & Greedy: General Graphs

The fact that GREEDY beats half on log-normalized $c = 2$ regular graphs lends itself to the following natural two step adaptation for general graphs: (i) prune (remove or decrease probabilities of certain edges in G) such that the log-normalized degree of each vertex in the remaining graph is 2; (ii) greedily take every edge in the pruned instance G_c. Specifically, upon the arrival of an edge e, we first adjust its probability by dropping e so that its realization probability is consistent with e's log-normalized weight in G_c, then we match the realized edge if none of e's endpoints are currently matched. This approach would already yield the desired result for the dense graphs that can be pruned to the log-normalized 2-regular graph G_c. However, such a direct strategy fails for the graphs that have a few small degree vertices.

Before we proceed with the fix for the general graphs, let us take a closer look at the proof of Theorem 5. Note that in the theorem we actually compare our algorithm with a stronger benchmark, half the total number of vertices in G. The problem with such a benchmark, is that it may be too strong for any algorithm to approximate. To address this issue, we have to adjust our algorithm and analysis to handle low degree vertices. To this end, we can calculate x_e, the probability that e appears in the maximum matching of the random graph for every e, as the first step of our algorithm. By definition, $\mathsf{OPT} = \sum_{e\in E} x_e$ is the right benchmark to compare with. Alternatively, we can solve the following LP introduced by Gamlath, Kale and Svensson [17].[7]

$$\begin{aligned} \underset{(x_e \geqslant 0)_{e\in E}}{\text{maximize}} \quad & \sum_{e\in E} x_e \\ \text{subject to} \quad & \sum_{e\in F} x_e \leqslant 1 - \prod_{e\in F}(1-p_e), \quad \forall v \in L \cup R,\ \forall F \subseteq E_v. \end{aligned} \tag{3}$$

The constraints of the LP simply state that for each vertex v and subset $F \subseteq E_v$ of edges incident to v, the probability that an edge of F appears in the maximum matching is at most the probability that at least one edge of F is realized. Note that the value of each variable x_e in the LP (3) does not necessarily match the exact probability of e to appear in the maximum matching. However, $\sum_{e\in E} x_e$ still serves as a valid upper bound on OPT. As a matter of fact, our analysis works for either benchmark: the solution to LP (3), or for each x_e being the probability of e to appear in the optimal matching. We choose the LP (3) formulation in the description of the algorithm and the following analysis, since the LP optimal solution is a stronger benchmark and important constraints are explicitly stated in the LP.

[7] The LP is polynomial-time solvable. See [17] for the details.

A natural approach for general graphs would be to prune the graph according to the LP solution $(x_e)_{e\in E}$. To build some intuition let us consider what happens if we directly use x_es instead of p_es:

1. prune the graph by decreasing the probabilities of each edge from p_e to x_e,
2. run Greedy on the pruned instance.

Consider a special case of complete bipartite graph $K_{n,n}$ where each edge is realized with probability 1. The optimal solution to LP (3) is $x_e = \frac{1}{n}$ for all $e \in E$, as the maximum matching has size $n = \sum_{e\in E} x_e$. As $-\ln(1-x) \approx x$ when x is small, we effectively run Greedy on log-normalized 1-regular graph after pruning $K_{n,n}$. Theorem 3 from previous section gives $h_1(1) \approx 0.459 < 0.5$ in this case. Moreover, a simple computer aided simulation suggests that Greedy matches no more than 0.5 fraction of all vertices in this case. In this simulation we consider a regular complete graph G with $|L| = |R| = n$, where each edge has probability $\frac{1}{n}$; the edges arrive in random (uniformly distributed) order. In particular, we run 10^5 number of trials for $n = 3000$ and the ratio between ALG and n equals 0.50002. See the full version of this paper for more details of our experiment.

This means that pruning probabilities directly to x_e is too much and we need a more conservative pruning step. In particular, Theorem 5 suggests to prune the graph so that the log-normalized weight of edge e becomes $c \cdot x_e$. On the other hand, for some edges, p_e can be as small as x_e, in which case we have a cap on the existence probability. For those edges, it is reasonable to keep the existence probability as the original graph. Formally, we consider the following algorithm.

Algorithm 1 Prune & Greedy.

1: Solve LP (3) and let $\{x_e\}_{e\in E}$ be the optimal solution.
2: Prune the graph by decreasing the probabilities of each edge from p_e to $y_e = \min(p_e, 1 - e^{-c\cdot x_e})$.
3: Run Greedy on the pruned instance.

In an easy case when $y_e = 1 - e^{-c\cdot x_e}$ for all edges $e \in E$, we can adapt our analysis for c-regular graphs with a similar performance guarantee. On the other hand, if $y_e = x_e$ for all edges, then the algorithm might not be better than 0.5-competitive according to the previous discussion. However, the constraints from LP (3) guarantee that this cannot happen for all e.

Note that Lemma 3 and equation (2) apply to our Prune & Greedy algorithm with the $\{x_e, y_e\}$ defined in this section. We shall prove Lemma 7 an analog of Lemma 4 to conclude Theorem 8, which is the main result of this section.).

▶ **Lemma 7.** *For all $u \in L$, when $c = 1.7$,*

$$\sum_{e\in E_u} x_e \cdot \left(1 - \exp\left(-\frac{q_u(e)\cdot y_e}{x_e}\right)\right) \geqslant 0.503 \cdot \sum_{e\in E_u} x_e$$

▶ **Theorem 8.** *Prune & Greedy is 0.503-competitive when $c = 1.7$.*

Proof. We write the lower bound on the performance of Greedy using Lemma 3.

$$\begin{aligned}
\mathsf{ALG} &\geqslant \sum_{u\in L}\sum_{e\in E_u} x_e \cdot \left(1 - \exp\left(-\frac{q_u(e)\cdot y_e}{x_e}\right)\right) && \text{(by Equation (2))}\\
&\geqslant \sum_{u\in L} 0.503 \cdot \sum_{e\in E_u} x_e = 0.503 \cdot \sum_{e\in E} x_e \geqslant 0.503 \cdot \mathsf{OPT}. && \text{(by Lemma 7)}
\end{aligned}$$

◀

4.1 Proof of Lemma 7

The proof of Lemma 7 is highly technical and requires us to solve a rather non-trivial optimization. Let $\{e_1, e_2, \cdots, e_k\}$ be all edges incident to u and enumerated in their arrival order. To simplify notations, let p_i, x_i, y_i and q_i denote $p_{e_i}, x_{e_i}, y_{e_i}$ and $q_u(e_i)$ respectively.

The optimization problem (4) captures the ratio that we want to study.

$$\min_{(p_i \leqslant 1),(x_i \geqslant 0)} \sum_{i=1}^{k} x_i \cdot \left(1 - e^{-\frac{q_i \cdot y_i}{x_i}}\right) \Big/ \sum_{i=1}^{k} x_i \tag{4}$$
$$\text{s.t. } y_i = \min(p_i, 1 - e^{-c \cdot x_i}), \quad \forall i \in [k]$$
$$q_i = \prod_{j<i}(1 - y_j), \quad \forall i \in [k]$$
$$\sum_{i \in S} x_i \leqslant 1 - \prod_{i \in S}(1 - p_i), \ \forall S \subseteq [k]$$

$$\min_{(p_i \leqslant 1),x \geqslant 0} \sum_{i=1}^{k} x \cdot \left(1 - e^{-\frac{q_i \cdot y_i}{x}}\right) \Big/ (kx) \tag{5}$$
$$\text{s.t. } y_i = \min(p_i, 1 - e^{-cx}), \quad \forall i \in [k]$$
$$q_i = \prod_{j<i}(1 - y_j), \quad \forall i \in [k]$$
$$|S| \cdot x \leqslant 1 - \prod_{i \in S}(1 - p_i), \ \forall S \subseteq [k]$$

The first family of constraints in (4) comes from the design of our algorithm. The second family of constraints characterizes the probability that u has no realized edge before e_i. The last family of constraints follows from LP (3).

We first decrease the value of optimization (4) by increasing the size k of the instance and get a simpler optimization problem (5). The fact that we can construct more regular instance with all $x_i = x$ and smaller or equal objective value follows from the "subdivision" Lemma 9 below.

▶ **Lemma 9.** *Let $(x_j, p_j)_{j \in [k]}$ be any feasible solution to (4). Let e_i be any edge $i \in [k]$ which we subdivide into two consecutive parallel edges e' and e''. Then there is a feasible solution to the new instance of (4) with the same $(x_j, p_j)_{j \neq i}$ and $x_{e'} + x_{e''} = x_i$, and smaller objective value. Moreover, $x_{e'} \geqslant 0$ and $x_{e''} \geqslant 0$ can be set to have any values subject to $x_{e'} + x_{e''} = x_i$.*

Proof. We fix feasible solution $(x_j, p_j)_{j \in [k]}$ to (4), edge e_i, and particular $x_{e'} \geqslant 0$ and $x_{e''} \geqslant 0$ such that $x_{e'} + x_{e''} = x_i$. Let $\beta = \frac{x_{e'}}{x_i}$. Note that $0 \leqslant \beta \leqslant 1$. We need to define $p_{e'}$,$p_{e''}$. Consider two cases depending on the value of y_i:

Case 1. If $y_i = 1 - e^{-c \cdot x_i}$, then we let $p_{e'} = p_{e''} = 1$.

Case 2. If $y_i = p_i$, then we let $p_{e'} = 1 - (1 - p_i)^{\beta}$ and $p_{e''} = 1 - (1 - p_i)^{1-\beta}$.

In the first case, $y_{e'} = 1 - e^{-cx_{e'}}$ and $y_{e''} = 1 - e^{-cx_{e''}}$. In the second case, we verify that

$$p_{e'} \leqslant 1 - e^{-cx_{e'}} \iff 1 - (1 - p_i)^{\beta} \leqslant 1 - e^{-cx_{e'}} \iff p_i \leqslant 1 - e^{-cx_i}.$$

Similarly, $p_{e''} \leqslant 1 - e^{-cx_{e''}}$ and we have that $y_{e'} = p_{e'}$ and $y_{e''} = p_{e''}$. In both cases, we have that $(1 - y_{e'})(1 - y_{e''}) = 1 - y_i$. Consequently, the value of q_j does not change by our subdivision for all $j \neq i$. Next, we examine the change to the numerator of the objective function.

$$
\begin{aligned}
&x_{e'} \cdot \left(1 - e^{-q_i \cdot \frac{y_{e'}}{x_{e'}}}\right) + x_{e''} \cdot \left(1 - e^{-q_i \cdot \frac{(1-y_{e'})y_{e''}}{x_{e''}}}\right) - x_i \cdot \left(1 - e^{-q_i \cdot \frac{y_i}{x_i}}\right) \\
=&x_i \cdot \left(\frac{x_{e'}}{x_i} \cdot \left(1 - e^{-q_i \cdot \frac{y_{e'}}{x_{e'}}}\right) + \frac{x_{e''}}{x_i} \cdot \left(1 - e^{-q_i \cdot \frac{(1-y_{e'})y_{e''}}{x_{e''}}}\right)\right) - x_i \cdot \left(1 - e^{-q_i \cdot \frac{y_i}{x_i}}\right) \\
\leqslant&x_i \cdot \left(1 - e^{-\frac{q_i}{x_i} \cdot (y_{e'} + (1-y_{e'})y_{e''})}\right) - x_i \cdot \left(1 - e^{-q_i \cdot \frac{y_i}{x_i}}\right) = 0,
\end{aligned}
$$

where the inequality holds by Jensen's inequality for the concave function $1 - e^{-x}$, and the last equality is due to the fact that $(1-y_{e'})(1-y_{e''}) = 1-y_i$. Thus, the subdivision decreases the objective function. We are left to verify that all inequality constraints in (4) for $S \subseteq [k]$ are satisfied.

First, we consider **Case 1**, when $y_i = 1 - e^{-c \cdot x_i}$. We have $p_{e'} = p_{e''} = 1$. Then

- if $e', e'' \notin S$, the constraint trivially holds (none of x_j, p_j change);
- if $e' \in S$ or $e'' \in S$, the right-hand side of the constraint becomes 1.

Next, in **Case 2**, $y_i = p_i$, $y_{e'} = p_{e'}$, $y_{e''} = p_{e''}$, and $(1 - y_{e'})(1 - y_{e''}) = 1 - y_i$. Then

- if $e', e'' \notin S$, the constraint still holds (none of x_j, p_j change);
- if both $e', e'' \in S$, the inequality holds since $x_{e'} + x_{e''} = x_i$ and $(1 - p_{e'})(1 - p_{e''}) = 1 - p_i$;
- if exactly one of $e', e'' \in S$, we may assume w.l.o.g. that $e' \in S, e'' \notin S$ (as the other case $e'' \in S, e' \notin S$ is symmetric). Let $T \subseteq [k] - \{i\}$ be any set of indexes. Then

$$
\begin{aligned}
&1 - (1 - p_{e'}) \prod_{j \in T}(1 - p_j) = 1 - (1 - p_i)^{\beta} \prod_{j \in T}(1 - p_j) \geqslant 1 - (1 - \beta p_i) \prod_{j \in T}(1 - p_j) \\
&= \beta \left(1 - (1 - p_i) \prod_{j \in T}(1 - p_j)\right) + (1 - \beta) \left(1 - \prod_{j \in T}(1 - p_j)\right) \\
&\qquad \geqslant \beta \left(x_i + \sum_{j \in T} x_j\right) + (1 - \beta) \sum_{j \in T} x_j = \beta x_i + \sum_{j \in T} x_j = x_{e'} + \sum_{j \in T} x_j,
\end{aligned}
$$

where to get the first inequality we used the fact $(1 - x)^{\beta} \leqslant 1 - x \cdot \beta$, for any $x > -1$, $0 \leqslant \beta \leqslant 1$; the second inequality holds due to the original constraints in (4) for $S = T \cup \{i\}$ and $S = T$.

This concludes our proof. ◀

To get (5), we can start with the optimal solution to (4) for any given k, then apply multiple times Lemma 9 to every edge e_i, $i \in [k]$ getting an instance with $k' \gg k$ edges and a feasible solution with the same value, where almost all $x_j = x$ and at most k edges have $x_e < x$ (x may depend on k'). Finally, we can remove all edges with $x_e < x$, keep the rest x_j and p_j untouched and redefine (q_j) according to the recurrent formula. The impact of the change to q_j's before the removal of edges $x_e < x$ can be made vanishingly small as $k' \to \infty$. At the end, we get a feasible solution to (4) of the form (5) (for bigger k) with almost the same value as the optimum of (4) for the initial k. Thus we can analyze (5) without loss of generality instead of (4).

We prove the following lemma that describes the optimal solution to problem (5).

▶ **Lemma 10.** *For an optimal solution to* (5)*: (i)* $(y_i)_{i \in [k]}$ *are decreasing; (ii)* $\exists$ *cut-off point* $\ell \in [k]$ *such that* $y_i = cx$ *for* $i \leqslant \ell$ *and* $y_i = p_i$ *for* $i > \ell$*; (iii) constraints* $|S| \cdot x \leqslant 1 - \prod_{i \in S}(1 - p_i)$ *are tight for all* $S = [j..k]$*, where* $j > \ell$*.*

Proof. We prove the three statements sequentially. Let (p_i) be the optimal solution. If $y_i < y_{i+1}$ for an $i \in [k]$. Consider swapping p_i and p_{i+1} in the instance and the other $(p_j)_{j\neq i,i+1}$ remain the same. Note that the last family of constraints are preserved since the constraints are invariant under any permutation of p_j's. It is easy to see that q_j's are not changed for all $j \neq i+1$ and $q'_{i+1} = q_i(1-y_{i+1})$. Moreover, $y'_i = y_{i+1}$ and $y'_{i+1} = y_i$. Therefore, to prove that the swap decreases the objective, it suffices to show

$$1 - e^{-\frac{q_i \cdot y_i}{x}} + 1 - e^{-\frac{q_i(1-y_i)\cdot y_{i+1}}{x}} < 1 - e^{-\frac{q_i \cdot y_{i+1}}{x}} + 1 - e^{-\frac{q_i(1-y_{i+1})\cdot y_i}{x}}.$$

Observe that $q_i y_{i+1} > q_i y_i$, $q_i y_{i+1} > q_i(1-y_i)y_{i+1}$ and $q_i y_i + q_i(1-y_i)y_{i+1} = q_i y_{i+1} + q_i(1-y_{i+1})y_i$. The above inequality is true due to the convexity of the function $\exp(-z)$. A contradiction that concludes the proof of (i), the monotonicity of y_i's.

The second statement follows immediately from (i) according to our definition of $y_i = \min(p_i, cx)$ which is a monotone function with respect to p_i.[8] Let ℓ be the cut-off point such that $y_i = cx$ for $i \leqslant \ell$ and $y_i = p_i$ for $i > \ell$.

We are left to prove (iii). Given the statement (ii), we safely assume that $p_i = 1$ for all $i \leqslant \ell$ since this would not affect all y_i, q_i's and only trivialize the last family of constraints when $S \cap [\ell] \neq \varnothing$, since in this case the right-hand side of the constraint equals 1. Since $p_i = 1$ for $i \in [\ell]$ and $1 \geqslant p_i = y_i$ for $i > \ell$, we can assume that $(p_i)_{i\in[k]}$ are decreasing as well.

Given the monotonicity of p_i's, we note that "critical" inequality constraints $|S| \cdot x \leqslant 1 - \prod_{i\in S}(1-p_i)$ are those where $S = \{j, j+1, \cdots, k\}$, i.e., the remaining (non-critical) inequality constrains for other sets S are automatically satisfied, if the constraints for $S = \{j, j+1, \cdots, k\}$ hold. Indeed, when restricting to S with a fixed cardinality s, the left-hand side of each constraint is the same $|S| \cdot x$, while the right-hand side is minimized when S consists of the s smallest p_i, i.e., $\{p_j, p_{j+1} \ldots, p_k\}$. We are going to prove (iii), that the critical constraints for $j > \ell$ are tight.

Now, suppose to the contrary that a critical constraint is not tight for an $S = \{\bar{i}, \bar{i}+1, \ldots, k\}$ for $\bar{i} > \ell$, while all critical constraints for each $S = \{i, i+1, \ldots, k\}$ where $i > \bar{i}$ are tight (if $\bar{i} = k$, we don't require any constraints to be tight). We first consider a non-degenerate case when $1 > p_{\bar{i}-1}$, which also means that $\bar{i} - 1 > \ell$ (otherwise $y_{\bar{i}-1} = cx$ and we would set $p_{\bar{i}-1} = 1$). Before that we prove the following fact.

▷ Claim 11. If $1 > p_i = p_{i+1}$ for $i \in (\ell..k)$, then inequality $|S| \cdot x < 1 - \prod_{j\in S}(1-p_j)$ for $S = \{i+1, \ldots, k\}$ is strict.

Proof. Suppose to the contrary that the inequality is an equality, that is

$$(1-p_{i+1}) = \frac{\prod_{j>i}(1-p_j)}{\prod_{j>i+1}(1-p_j)} = \frac{1-(k-i)x}{\prod_{t>i+1}(1-p_j)}.$$

The inequality constraint for $S = \{i, \ldots, k\}$ gives:

$$(1-p_i)(1-p_{i+1}) = \frac{\prod_{j\geqslant i}(1-p_j)}{\prod_{j>i+1}(1-p_j)} \leqslant \frac{1-(k-i+1)x}{\prod_{j>i+1}(1-p_j)}.$$

[8] Notice that $cx \approx 1 - e^{-cx}$ when $x \approx 0$ in (5).

Putting the two equations together and by the assumption that $p_i = p_{i+1}$, we have

$$\left(\frac{1-(k-i)x}{\prod_{j>i+1}(1-p_j)}\right)^2 \leqslant \frac{1-(k-i+1)x}{\prod_{j>i+1}(1-p_j)}$$
$$\Longrightarrow \frac{(1-(k-i)x)^2}{(1-(k-i+1)x)} \leqslant \prod_{j>i+1}(1-p_j) \leqslant 1-(k-i-1)x,$$

where the last inequality follows from the constraint for $S = \{i+2, \dots, k\}$ (if $i+2 > k$, the inequality still holds, as $i = k-1$, $1-(k-i-1)x = 1$, and $\prod_{j>i+1}(1-p_j) = 1$).

Thus $(1-(k-i+1)x)(1-(k-i-1)x) = (1-(k-i)x)^2 - x^2 \geqslant (1-(k-i)x)^2$, a contradiction. ◁

Case 1 ($1 > p_{\bar{i}-1}$). We will get a contradiction by providing an instance with a strictly smaller objective's value. Let $\underline{i}$ be the smallest index so that $p_{\underline{i}} = p_{\underline{i}+1} = \cdots = p_{\bar{i}-1}$. So $p_{\underline{i}-1} > p_{\underline{i}}$, if $\underline{i} > 1$. Recall that we consider the case $1 > p_{\bar{i}-1} = p_{\underline{i}}$ and thus $\ell < \underline{i}$ (otherwise $y_{\underline{i}} = cx$ and we should have set $p_{\underline{i}} = 1$). By Claim 11, each inequality in (5) for $S = \{i, i+1, \dots, k\}$ with $\underline{i}+1 \leqslant i \leqslant \bar{i}-1$ must be strict. On the other hand, a contra-positive statement to Claim 11 gives us that $p_{\bar{i}}$ cannot be equal to $p_{\bar{i}+1}$ (if $\bar{i} = k$, this also is true). Thus $p_{\bar{i}} > p_{\bar{i}+1}$ (if $\bar{i} < k$). If $\bar{i} = k$, then $p_{\bar{i}} > 0$ (otherwise, we can decrease k in (5)).

We consider the following modification $(\mathbf{p}', x)$ of (5)'s feasible solution: slightly increase $p_{\underline{i}}$ and decrease $p_{\bar{i}}$ so that $(1-p_{\underline{i}})(1-p_{\bar{i}})$ remains the same; all other p_i for $i \neq \underline{i}, \bar{i}$ and x are the same in $(\mathbf{p}', x)$ and original optimum $(\mathbf{p}, x)$; $\mathbf{y}', \mathbf{q}'$ are redefined according to the formula in (5). Note that we can always do such modification when $1 > p_{\underline{i}} > p_{\bar{i}} > 0$.

For any sufficiently small such perturbation of $p_{\bar{i}}$ and $p_{\underline{i}}$, $(p'_i)_{i\in[k]}$ remain monotone and all constraints in (5) are satisfied. Indeed, we only need to check the critical constraints in (5) for monotone $\mathbf{p}'$: $\mathbf{p}'$ and $\mathbf{p}$ are the same for $S = \{i, i+1, \dots, k\}$ for $i \in (\bar{i}..k]$; all inequalities for $S = \{i, i+1, \dots, k\}$ where $i \in [\underline{i}+1, \bar{i}]$ are strict and, therefore, for sufficiently small perturbation of $p_{\bar{i}}$ and $p_{\underline{i}}$ they still hold; for $S = \{i, i+1, \dots, k\}$ where $i \in (\ell..\underline{i}]$, the right-hand side of each critical constraint does not change, because $(1-p_{\bar{i}})(1-p_{\underline{i}}) = (1-p'_{\bar{i}})(1-p'_{\underline{i}})$.

Now we examine the changes to q_i. Observe that each $q'_i = q_i$ and $y'_i = y_i$ for any $i < \underline{i}$, as $(p'_i)_{i<\underline{i}}$ and $(p_i)_{i<\underline{i}}$ are the same. For $i \geqslant \bar{i}$, we also have $q'_i = q_i$ and $y'_i = y_i$ since $(1-y_{\underline{i}})(1-y_{\bar{i}}) = (1-y'_{\underline{i}})(1-y'_{\bar{i}})$. Moreover, we notice that

$$\sum_{i\in[\underline{i}..\bar{i}]} q_i y_i = q_{\underline{i}}\left(1 - \prod_{i\in[\underline{i}..\bar{i}]}(1-y_i)\right) = \sum_{i\in[\underline{i}..\bar{i}]} q'_i y'_i.$$

In the interval $i \in [\underline{i}, \bar{i}]$, we notice that by increasing $p_{\underline{i}}$ and decreasing $p_{\bar{i}}$ we increase $q_{\underline{i}} y_{\underline{i}}$ and decrease each $q_i y_i$ for $i \in (\underline{i}..\bar{i})$, since each q_i deceases. Moreover,

$$q'_{\bar{i}} y'_{\bar{i}} = \left[\prod_{\substack{j<\bar{i},\\ j\neq\underline{i}}}(1-y_j)\right]\cdot(1-y'_{\underline{i}})\cdot y'_{\bar{i}} = \left[\prod_{\substack{j<\bar{i},\\ j\neq\underline{i}}}(1-y_j)\right]\cdot(1-y'_{\underline{i}}-(1-y'_{\bar{i}})(1-y'_{\underline{i}}))$$
$$< \left[\prod_{\substack{j<\bar{i},\\ j\neq\underline{i}}}(1-y_j)\right]\cdot(1-y_{\underline{i}}-(1-y_{\bar{i}})(1-y_{\underline{i}})) = q_{\bar{i}} y_{\bar{i}},$$

since $y'_{\underline{i}} > y_{\underline{i}}$ while $(1-y'_{\underline{i}})(1-y'_{\bar{i}}) = (1-y_{\underline{i}})(1-y_{\bar{i}})$. Hence, due to convexity of the function $\exp(-z)$, we conclude that the objective $\sum_i(1-\exp(-\frac{q_i y_i}{x}))$ decreases when we substitute $(\mathbf{q},\mathbf{y})$ with $(\mathbf{q}',\mathbf{y}')$. Indeed, $q_{\underline{i}}y_{\underline{i}}$, the largest number among $\{q_i y_i\}_{i=\underline{i}}^{\bar{i}}$, increases, while all other affected $q_i y_i$ in $[\underline{i},\bar{i}]$ decrease.

Case 2 ($p_{\bar{i}-1} = 1$). Now we consider a degenerate case when $p_{\bar{i}-1} = 1$. In this case, $p_{\bar{i}}$ only appears in the critical constraint for $S = \{\bar{i}, \bar{i}+1, \dots, k\}$, which we assume to be not tight. Note that $p_{\bar{i}} > p_{\bar{i}+1}$ if $\bar{i} < k$ by Claim 11 and also that $p_{\bar{i}} > 0$ if $\bar{i} = k$ (otherwise, we can decrease k in (5)). Thus, sightly decreasing $p_{\bar{i}}$ shall not violate any constraint. Furthermore, since $\bar{i} > \ell$, $p_{\bar{i}} < cx$, we can also slightly increase $p_{\bar{i}}$ without violating any constraints. Now, we fix all p_i for $i \neq \bar{i}$ and consider $y_{\bar{i}} = p_{\bar{i}}$ as a locally free variable that we can slight increase or decrease. We study the objective of (5) as a function of $y_{\bar{i}} = p_{\bar{i}}$.

Observe that any change to $y_{\bar{i}}$ only affects the terms $(1-e^{-\frac{q_i y_i}{x}})$ for $i \geqslant \bar{i}$. Furthermore, for $i = \bar{i}$, $(1-e^{-\frac{q_{\bar{i}} y_{\bar{i}}}{x}})$ is a strictly concave function of $y_{\bar{i}}$; and for $i > \bar{i}$

$$1-\exp\left(-\frac{q_i y_i}{x}\right) = 1-\exp\left(-(1-y_{\bar{i}})\cdot\frac{\prod_{j<i, j\neq\bar{i}}(1-y_j)\cdot y_i}{x}\right)$$

is also a concave function of $y_{\bar{i}}$.

Thus, the objective function of (5) is a strictly concave function of $y_{\bar{i}} = p_{\bar{i}}$, at least in some neighborhood of $p_{\bar{i}}$. Note that the minimum of a strictly concave function is always achieved on the boundary of its domain. Therefore, some small perturbation of $p_{\bar{i}}$ and consequently $y_{\bar{i}} = p_{\bar{i}}$ (either slightly increase or decrease $p_{\bar{i}}$ such that all constraint in (5) are still satisfied) would strictly decrease the objective. This contradicts the optimality of $\mathbf{p}$. ◀

Now we can write explicit formula for p_i for $i > \ell$. By (iii) of Lemma 10, we have $\prod_{j=i+1}^{k}(1-p_j) = 1-(k-i)x$ and $\prod_{j=i}^{k}(1-p_j) = 1-(k-i+1)x$. Thus, if $i > \ell$, then $y_i = p_i = \frac{x}{1-(k-i)x}$ and

$$\begin{aligned} q_i = \prod_{j<i}(1-y_j) = (1-cx)^{\ell}\cdot\prod_{j=\ell+1}^{i-1}(1-p_i) &= (1-cx)^{\ell}\cdot\frac{\prod_{j=\ell+1}^{k}(1-p_i)}{\prod_{j=i}^{k}(1-p_i)} \\ &= (1-cx)^{\ell}\cdot\frac{1-(k-\ell)x}{1-(k-i+1)x}. \end{aligned}$$

Let $t = (k-\ell)x$ and $s = \ell x$ for notation simplicity. We have, for small x

$$\frac{q_i y_i}{x} = \begin{cases} \frac{1-e^{-cx}}{x}(1-cx)^{i-1} \approx c\cdot e^{-c\cdot(i-1)x}, & i \leqslant \ell \\ (1-cx)^{\ell}\cdot\frac{(1-t)}{(1-(k-i)x)\cdot(1-(k-i+1)x)} \approx e^{-c\cdot s}\frac{(1-t)}{(1-t+(i-\ell)x)^2}, & i > \ell \end{cases}$$

Consequently, we have that for small x

$$\begin{aligned} \sum_{i=1}^{k} x\cdot\left(1-e^{-\frac{q_i y_i}{x}}\right) &= \sum_{i=1}^{\ell} x\cdot\left(1-e^{-\frac{q_i y_i}{x}}\right) + \sum_{i=\ell+1}^{k} x\cdot\left(1-e^{-\frac{q_i y_i}{x}}\right) \\ &= \sum_{i=1}^{\ell} x\cdot\left(1-e^{-c\cdot e^{-c\cdot(i-1)x}}\right) + \sum_{i=\ell+1}^{k} x\cdot\left(1-e^{-e^{-c\cdot s}\frac{(1-t)}{(1-t+(i-\ell)x)^2}}\right) \\ &\geqslant \int_0^s 1-e^{-ce^{-cz}}\,\mathrm{d}z + \int_0^t 1-e^{-e^{-cs}\cdot\frac{1-t}{(1-t+z)^2}}\,\mathrm{d}z \end{aligned}$$

Furthermore, since $\frac{x}{1-t} \approx \frac{x}{1-(k-\ell-1)x} = p_{\ell+1} \leqslant 1-e^{-cx} \approx cx$, we have $t \leqslant 1-\frac{1}{c}$.

To finish the proof, it suffices to lower bound the following function

$$h_2(s,t) \stackrel{\text{def}}{=\!=} \left(\int_0^s 1 - e^{-ce^{-cz}}\,\mathrm{d}z + \int_0^t 1 - e^{-e^{-cs}\cdot\frac{1-t}{(1-t+z)^2}}\,\mathrm{d}z\right) / (s+t)\,,$$

subject to $s \in [0,1], t \in \left[0, 1-\frac{1}{c}\right], s+t \in (0,1]$.

We use numerical methods to show $h_2(s,t) \geqslant h_2\left(\frac{1}{c}, 1-\frac{1}{c}\right) > 0.503$ when $c = 1.7$. The details are in the full version.

5 Improved Analysis: Regular Graphs

In this section, we prove a stronger performance guarantee of Greedy for regular graphs. According to the definition of log-normalized regular graph, each edge with log-normalized weight w_e can be substituted by a set of consecutive "small" edges with the same total log-normalized weight. For the ease of presentation, we assume that all edges are infinitesimal within this section. Let $x_e = \frac{w_e}{c}$ and $y_e = 1 - e^{-w_e}$ for all $e \in E$ as defined in Section 3.

Define the following two types of contributions of each edge $e \in E_{vu}$, where $u \in L$ and $v \in R$:

$$\mathrm{I}(e) \stackrel{\text{def}}{=\!=} x_e \cdot \left(1 - \exp\left(-\frac{q_u(e)\cdot y_e}{x_e}\right)\right);$$

and $\mathrm{II}(e) \stackrel{\text{def}}{=\!=} y_e \cdot \left(\mathbf{Pr}\left[\overline{M_u}(e)\right] - \mathbf{Pr}\left[Q_u\left(e\right)\right]\right).$

In Section 3, we estimated performance of Greedy with a lower bound of $\sum_{e\in E} \mathrm{I}(e)$. Recall that this bound corresponds to the event that edge e is the first realized edge of u. It turns out that we can add an extra term $\mathrm{II}(e)$ on top of $\mathrm{I}(e)$ to have a better bound on the probability that edge e is matched. The term $\mathrm{II}(e)$ corresponds to the event that e is not the first realized edge of u, but u is still unmatched before e. Formally, we have the following Lemma 12, where coefficient $e^{-c-ce^{-c}}$ in front of $\mathrm{II}(e)$ ensures that the event from which we get extra gain is disjoint with the events from which we obtain the contribution of the first kind. I.e., we avoid double counting.

Within this section, we use computer assisted calculations in several places. We state all our lemmas in the case when $c = 2$ to highlight the improvement of our analysis over the competitive ratio of 0.532. We remark that our analysis generalizes for a wide range of the parameter c. The proofs of Lemmas 12 and 13 can be found in the full version.

▶ **Lemma 12.** *When $c = 2$,*

$$\mathsf{ALG} \geqslant \sum_{e\in E}\left(I(e) + e^{-c-ce^{-c}}\cdot II(e)\right). \tag{6}$$

By Lemma 4, the $\mathrm{I}(e)$ term alone is sufficient to show that Greedy is 0.532-competitive for 2-regular graphs. Next, we study the $\mathrm{II}(e)$ term.

Let $\delta_u = \mathbf{Pr}[\overline{M_u}] - \mathbf{Pr}[Q_u]$ for each vertex $u \in L$ at the end of algorithm's execution. Note that $\mathbf{Pr}[\overline{M_u}(e)] \geqslant \mathbf{Pr}[Q_u(e)]$ for all $e \in E_u$, because u cannot be matched at the moment of edge e arrival, if u had no realized edges. Thus $\delta_u \geqslant 0$. Similar to Lemma 3, we fix a vertex $u \in L$ and study the sum of $\mathrm{II}(e)$ for all edges $e \in E_u$.

▶ **Lemma 13.** *For any vertex $u \in L$, when $c = 2$,*

$$\sum_{e\in E_u} II(e) = \sum_{e\in E_u} y_e \cdot \left(\mathbf{Pr}\left[\overline{M_u}(e)\right] - \mathbf{Pr}\left[Q_u(e)\right]\right) \geqslant 1.98\cdot\delta_u^2.$$

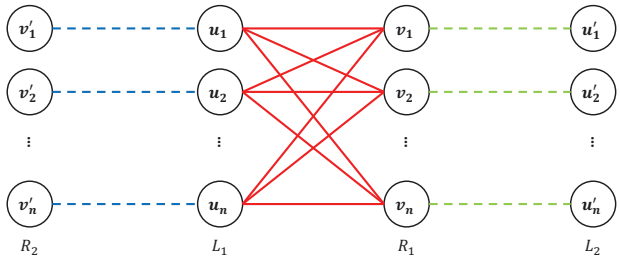


Figure 2 Hard instance for any algorithm.

▶ **Theorem 14.** *GREEDY is 0.552-competitive on 2-regular graphs.*

Proof. We have that

$$\mathsf{ALG} = \sum_{u\in L} \mathbf{Pr}\left[M_u\right] = \sum_{u\in L} \left(1 - \mathbf{Pr}\left[\overline{M_u}\right]\right) = \sum_{u\in L} \left(1 - e^{-c} - \delta_u\right).$$

We recall definition of $h_1(c) \stackrel{\text{def}}{=\!=} \int_0^1 \left(1 - e^{-ce^{-cz}}\right) \mathrm{d}z$ from Section 3. Then,

$$\begin{aligned}
&|L| \cdot (1 - e^{-c}) - \sum_{u\in L} \delta_u \\
\geqslant &\sum_{e\in E} \left(\mathrm{I}(e) + e^{-c-ce^{-c}} \cdot \mathrm{II}(e)\right) && \text{(by Lemma 12)} \\
\geqslant &\sum_{u\in L} \left(h_1(c) + e^{-c-ce^{-c}} \cdot 1.98 \cdot \delta_u^2\right) && \text{(by Lemma 4, 13)} \\
\geqslant &h_1(c) \cdot |L| + 1.98 \cdot e^{-c-ce^{-c}} \cdot \frac{\left(\sum_{u\in L} \delta_u\right)^2}{|L|}. && \text{(by Cauchy-Schwarz inequality)}
\end{aligned}$$

Let $\Delta = \frac{\sum_{u\in L} \delta_u}{|L|}$. We rearrange the above inequality and get the following for $c = 2$.

$$(1 - e^{-2} - h_1(2)) \geqslant \Delta + 1.98 \cdot e^{-2-2e^{-2}} \cdot \Delta^2.$$

Solving the inequality numerically, we have that $\Delta \leqslant 0.312$. Therefore,

$$\mathsf{ALG} = (1 - e^{-2} - \Delta) \cdot |L| \geqslant (1 - e^{-2} - 0.312) \cdot |L| \geqslant 0.552 \cdot |L|.$$ ◀

6 Problem Hardness

In this section, we present an upper bound of $\frac{2}{3} \approx 0.667$ for all online algorithms. Consider the graph shown in Figure 2. We use $L_1 = \{u_i\}_{i=1}^n, R_1 = \{v_j\}_{j=1}^n, L_2 = \{u'_i\}_{i=1}^n$ and $R_2 = \{v'_j\}_{j=1}^n$ to denote the vertices in the graph. The edges are defined as the following:

1. For each pair of $(u, v) \in L_1 \times R_1$, let there be an edge (u, v) with existence probability 1. We call them type-1 edges (red solid edges).
2. For each $i \in [n]$, let there be an edge (u_i, v'_i) with existence probability $\frac{1}{2}$. We call them type-2 edges (blue dashed edges).
3. For each $i \in [n]$, let there be an edge (u'_i, v_i) with existence probability $\frac{1}{2}$. We call them type-3 edges (green dashed edges).

Let the type-1 edges arrive first and then type-2 and type-3 edges.

▶ **Theorem 15.** *No algorithm is better than* $\frac{2}{3}$*-competitive.*

Proof. Note that there is no randomness for type-1 edges. If an algorithm matches k of them, there will be $\frac{n-k}{2}$ possible type-2 edges and $\frac{n-k}{2}$ type-3 edges in expectation. Thus any online algorithm matches no more than $k + \frac{n-k}{2} + \frac{n-k}{2} = n$ in expectation.

On the other hand, with high probability, there are at least $(0.5 - o(1)) \cdot n$ realized type-2 edges and at least $(0.5 - o(1)) \cdot n$ realized type-3 edges. In this case, the prophet can match $(0.5 - o(1)) \cdot n$ type-2 and type-3 edges respectively and then $0.5 \cdot n$ type-1 edges. In total, the prophet matches $(1.5 - o(1)) \cdot n$ edges with high probability. That is, $\mathsf{OPT} \geqslant (1.5 - o(1)) \cdot n$ when $n \to \infty$. ◀

References

1. Marek Adamczyk, Fabrizio Grandoni, and Joydeep Mukherjee. Improved approximation algorithms for stochastic matching. In *ESA*, volume 9294 of *Lecture Notes in Computer Science*, pages 1–12. Springer, 2015.
2. Itai Ashlagi, Maximilien Burq, Chinmoy Dutta, Patrick Jaillet, Amin Saberi, and Chris Sholley. Edge weighted online windowed matching. In *EC*, pages 729–742. ACM, 2019.
3. Bahman Bahmani and Michael Kapralov. Improved bounds for online stochastic matching. In *ESA (1)*, volume 6346 of *Lecture Notes in Computer Science*, pages 170–181. Springer, 2010.
4. Nikhil Bansal, Anupam Gupta, Jian Li, Julián Mestre, Viswanath Nagarajan, and Atri Rudra. When LP is the cure for your matching woes: Improved bounds for stochastic matchings. *Algorithmica*, 63(4):733–762, 2012.
5. Alok Baveja, Amit Chavan, Andrei Nikiforov, Aravind Srinivasan, and Pan Xu. Improved bounds in stochastic matching and optimization. *Algorithmica*, 80(11):3225–3252, 2018.
6. Benjamin Birnbaum and Claire Mathieu. On-line bipartite matching made simple. *ACM SIGACT News*, 39(1):80–87, 2008.
7. Niv Buchbinder, Kamal Jain, and Joseph Naor. Online primal-dual algorithms for maximizing ad-auctions revenue. In *ESA*, volume 4698 of *Lecture Notes in Computer Science*, pages 253–264. Springer, 2007.
8. Niv Buchbinder, Danny Segev, and Yevgeny Tkach. Online algorithms for maximum cardinality matching with edge arrivals. *Algorithmica*, 81(5):1781–1799, 2019.
9. Ning Chen, Nicole Immorlica, Anna R. Karlin, Mohammad Mahdian, and Atri Rudra. Approximating matches made in heaven. In *ICALP (1)*, volume 5555 of *Lecture Notes in Computer Science*, pages 266–278. Springer, 2009.
10. Ilan Reuven Cohen and David Wajc. Randomized online matching in regular graphs. In *SODA '18*, pages 960–979. SIAM, 2018.
11. Kevin P. Costello, Prasad Tetali, and Pushkar Tripathi. Stochastic matching with commitment. In *ICALP (1)*, volume 7391 of *Lecture Notes in Computer Science*, pages 822–833. Springer, 2012.
12. Nikhil R. Devanur and Kamal Jain. Online matching with concave returns. In *STOC*, pages 137–144. ACM, 2012.
13. Nikhil R. Devanur, Kamal Jain, and Robert D. Kleinberg. Randomized primal-dual analysis of RANKING for online bipartite matching. In *SODA*, pages 101–107. SIAM, 2013.
14. Leah Epstein, Asaf Levin, Danny Segev, and Oren Weimann. Improved bounds for online preemptive matching. In *STACS*, volume 20 of *LIPIcs*, pages 389–399. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2013.
15. Tomer Ezra, Michal Feldman, Nick Gravin, and Zhihao Gavin Tang. Online stochastic max-weight matching: Prophet inequality for vertex and edge arrival models. In *EC '20*, pages 769–787. ACM, 2020.
16. Jon Feldman, Aranyak Mehta, Vahab S. Mirrokni, and S. Muthukrishnan. Online stochastic matching: Beating $1 - 1/e$. In *FOCS*, pages 117–126. IEEE Computer Society, 2009.

17 Buddhima Gamlath, Sagar Kale, and Ola Svensson. Beating greedy for stochastic bipartite matching. In *SODA*, pages 2841–2854. SIAM, 2019.

18 Buddhima Gamlath, Michael Kapralov, Andreas Maggiori, Ola Svensson, and David Wajc. Online matching with general arrivals. In *FOCS*, pages 26–37. IEEE Computer Society, 2019.

19 Gagan Goel and Aranyak Mehta. Online budgeted matching in random input models with applications to adwords. In *SODA*, pages 982–991, 2008. URL: `http://dl.acm.org/citation.cfm?id=1347082.1347189`.

20 Nikolai Gravin and Hongao Wang. Prophet inequality for bipartite matching: Merits of being simple and non adaptive. In *EC*, pages 93–109. ACM, 2019.

21 Guru Prashanth Guruganesh and Sahil Singla. Online matroid intersection: Beating half for random arrival. In *IPCO*, volume 10328 of *Lecture Notes in Computer Science*, pages 241–253. Springer, 2017.

22 Zhiyi Huang, Ning Kang, Zhihao Gavin Tang, Xiaowei Wu, Yuhao Zhang, and Xue Zhu. How to match when all vertices arrive online. In *STOC*, pages 17–29. ACM, 2018.

23 Zhiyi Huang, Binghui Peng, Zhihao Gavin Tang, Runzhou Tao, Xiaowei Wu, and Yuhao Zhang. Tight competitive ratios of classic matching algorithms in the fully online model. In *SODA*, pages 2875–2886. SIAM, 2019.

24 Patrick Jaillet and Xin Lu. Online stochastic matching: New algorithms with better bounds. *Math. Oper. Res.*, 39(3):624–646, 2014.

25 Chinmay Karande, Aranyak Mehta, and Pushkar Tripathi. Online bipartite matching with unknown distributions. In *STOC*, pages 587–596, 2011.

26 Richard M. Karp, Umesh V. Vazirani, and Vijay V. Vazirani. An optimal algorithm for on-line bipartite matching. In *STOC*, pages 352–358, 1990.

27 Robert Kleinberg and S. Matthew Weinberg. Matroid prophet inequalities and applications to multi-dimensional mechanism design. *Games and Economic Behavior*, 113:97–115, 2019.

28 László Lovász and Michael D Plummer. *Matching theory*. Providence, R.I. : AMS Chelsea Pub, 2009. Originally published: Amsterdam; New York : North-Holland, 1986. URL: `https://ebookcentral.proquest.com/lib/qut/detail.action?docID=4832190`.

29 Mohammad Mahdian and Qiqi Yan. Online bipartite matching with random arrivals: an approach based on strongly factor-revealing LPs. In *STOC*, pages 597–606, 2011.

30 Vahideh H. Manshadi, Shayan Oveis Gharan, and Amin Saberi. Online stochastic matching: Online actions based on offline statistics. *Math. Oper. Res.*, 37(4):559–573, 2012.

31 Andrew McGregor. Finding graph matchings in data streams. In *APPROX-RANDOM*, volume 3624 of *Lecture Notes in Computer Science*, pages 170–181. Springer, 2005.

32 Aranyak Mehta. Online matching and ad allocation. *Foundations and Trends in Theoretical Computer Science*, 8(4):265–368, 2013.

33 Aranyak Mehta, Amin Saberi, Umesh V. Vazirani, and Vijay V. Vazirani. Adwords and generalized online matching. *J. ACM*, 54(5):22, 2007.

34 Joseph (Seffi) Naor and David Wajc. Near-optimum online ad allocation for targeted advertising. *ACM Trans. Economics and Comput.*, 6(3–4):16:1–16:20, 2018.

35 Ashwinkumar Badanidiyuru Varadaraja. Buyback problem – approximate matroid intersection with cancellation costs. In *ICALP (1)*, volume 6755 of *Lecture Notes in Computer Science*, pages 379–390. Springer, 2011.

36 Yajun Wang and Sam Chiu-wai Wong. Two-sided online bipartite matching and vertex cover: Beating the greedy algorithm. In *ICALP (1)*, volume 9134 of *Lecture Notes in Computer Science*, pages 1070–1081. Springer, 2015.

Faster Monotone Min-Plus Product, Range Mode, and Single Source Replacement Paths

Yuzhou Gu
MIT, Cambridge, MA, USA

Adam Polak

École Polytechnique Fédérale de Lausanne, Switzerland

Virginia Vassilevska Williams
MIT, Cambridge, MA, USA

Yinzhan Xu
MIT, Cambridge, MA, USA

Abstract

One of the most basic graph problems, All-Pairs Shortest Paths (APSP) is known to be solvable in $n^{3-o(1)}$ time, and it is widely open whether it has an $O(n^{3-\epsilon})$ time algorithm for $\epsilon > 0$. To better understand APSP, one often strives to obtain subcubic time algorithms for structured instances of APSP and problems equivalent to it, such as the Min-Plus matrix product.

A natural structured version of Min-Plus product is Monotone Min-Plus product which has been studied in the context of the Batch Range Mode [SODA'20] and Dynamic Range Mode [ICALP'20] problems. This paper improves the known algorithms for Monotone Min-Plus Product and for Batch and Dynamic Range Mode, and establishes a connection between Monotone Min-Plus Product and the Single Source Replacement Paths (SSRP) problem on an n-vertex graph with potentially negative edge weights in $\{-M, \ldots, M\}$.

SSRP with positive integer edge weights bounded by M can be solved in $\tilde{O}(Mn^{\omega})$ time, whereas the prior fastest algorithm for graphs with possibly negative weights [FOCS'12] runs in $O(M^{0.7519}n^{2.5286})$ time, the current best running time for directed APSP with small integer weights. Using Monotone Min-Plus Product, we obtain an improved $O(M^{0.8043}n^{2.4957})$ time SSRP algorithm, showing that SSRP with constant negative integer weights is likely easier than directed unweighted APSP, a problem that is believed to require $n^{2.5-o(1)}$ time.

Complementing our algorithm for SSRP, we give a reduction from the Bounded-Difference Min-Plus Product problem studied by Bringmann et al. [FOCS'16] to negative weight SSRP. This reduction shows that it might be difficult to obtain an $\tilde{O}(Mn^{\omega})$ time algorithm for SSRP with negative weight edges, thus separating the problem from SSRP with only positive weight edges.

2012 ACM Subject Classification Theory of computation → Shortest paths

Keywords and phrases APSP, Min-Plus Product, Range Mode, Single-Source Replacement Paths

Digital Object Identifier 10.4230/LIPIcs.ICALP.2021.75

Category Track A: Algorithms, Complexity and Games

Related Version *Full Version*: `https://arxiv.org/abs/2105.02806`

Funding *Adam Polak*: Supported by the Swiss National Science Foundation within the project *Lattice Algorithms and Integer Programming* (185030). Part of this work was done at Jagiellonian University, supported by Polish National Science Center grant 2017/27/N/ST6/01334.
Virginia Vassilevska Williams: Supported by an NSF CAREER Award, NSF Grants CCF-1528078, CCF-1514339 and CCF-1909429, a BSF Grant BSF:2012338, a Google Research Fellowship and a Sloan Research Fellowship.
Yinzhan Xu: Supported by NSF Grant CCF-1528078.

Acknowledgements This project was started as part of an open problem session held alongside MIT subject 6.890 taught by the third author. The authors would like to acknowledge the rest of the participants in the open problems session including but not limited to Angelos Pelecanos, Nicole Wein and Yuancheng Yu.

1 Introduction

The All-Pairs Shortest Paths problem (APSP) is one of the most well-known problems in graph algorithms. Following the classical Floyd–Warshall algorithm that solves APSP in $O(n^3)$ time in n-vertex edge-weighted graphs, a long list of papers have been dedicated to improving the APSP running time. The current best algorithm by Williams [26] runs in $n^3/2^{\Theta(\sqrt{\log n})}$ time. It is a big open problem whether a truly sub-cubic, $O(n^{3-\epsilon})$-time for $\epsilon > 0$, algorithm for APSP exists. In fact, the popular APSP hypothesis from Fine-Grained Complexity [23] asserts that this is not the case.

In the Min-Plus Product problem, one is given two $n \times n$ integer matrices A, B and is required to compute an $n \times n$ matrix $C = A \star B$ such that $C_{i,j} = \min_k\{A_{i,k} + B_{k,j}\}$. Fischer and Meyer [12] showed that Min-Plus Product is equivalent to APSP, in the sense that a $T(n)$ time algorithm for either of the problems immediately implies an $O(T(n))$ time algorithm for the other. The APSP hypothesis thus states that computing the min-plus product of $n \times n$ integer matrices requires $n^{3-o(1)}$ time[1].

Structured Min-Plus Products. To better understand the complexity of APSP, much research focuses on improving the running time for Min-Plus Product when one or both of the matrices have some structure, with the hope that eventually all instances can be handled. As a fundamental problem, Min-Plus Product can be used to solve many other problems. It turns out that in many cases a structured version of Min-Plus Product suffices [25, 4]. Thus, studying structured instances of Min-Plus Product has the potential to speed up the running times for many applications.

Alon, Galil and Margalit [3] first studied the Min-Plus Product of structured matrices. They showed, following ideas of Yuval [30], that if all entries of two $n \times n$ matrices A, B are integers in $\{-M, \ldots, M\} \cup \{\infty\}$, then one can compute the min-plus product of A and B in $\tilde{O}(Mn^\omega)$ time[2], where $\omega \in [2, 2.373)$ denotes the best possible exponent of square matrix multiplication [22, 17, 1].

Yuster [28] considered Min-Plus Product when one of the matrices has a small number of distinct entries in each row, generalizing [3]. Bringmann et al. [4] studied Min-Plus Product of bounded-difference matrices, generalizing [3, 28]. An integer matrix is called to have *bounded differences* if all pairs of adjacent entries (both horizontally and vertically) differ by at most $O(1)$. Bringmann et al. [4] gave an $O(n^{2.8244})$ time algorithm for computing the Min-Plus Product between two bounded-difference matrices. When $\omega = 2$, their algorithm runs in $O(n^{2.7554})$ time. They also studied variants of this problem including the case when only one matrix is guaranteed to have bounded differences, and the bounded-differences are only in the rows or only in the columns.

[1] In fine-grained complexity one needs to fix the model of computation for each hardness hypothesis, and the APSP hypothesis is typically stated for a word RAM with $O(\log n)$ bit words, which is the model the algorithms in our paper are in.

[2] Throughout the paper the $\tilde{O}$ notation hides subpolynomial factors.

Chan [5] gave a truly sub-cubic time algorithm for the min-plus product between *geometrically weighted* matrices using a geometric tool called the *partition theorem*. Recently, Vassilevska Williams and Xu [25] combined the approach of Bringmann et al. [4] and a geometric data structure to give a truly subcubic Min-Plus Product algorithm for integer matrices where one of the matrices has constant $O(1)$-approximate rank, further generalizing the results of [4] and partially [5].

Our contribution. In this work, we study the Min-Plus Product of a *monotone* integer matrix with an arbitrary[3] integer matrix. We defer the general definition of *Monotone Min-Plus Product* to Section 2. For now let us focus on an interesting special case: We are given an arbitrary $n \times n$ integer matrix A and an $n \times n$ matrix B whose entries are positive integers bounded by $O(n)$, and such that each row of B is non-decreasing.

The above special case already subsumes the Min-Plus Product of bounded-difference matrices studied by Bringmann et al.: Suppose we are asked to compute the min-plus product of matrices A and B where B has bounded differences. In other words, all pairs of adjacent entries (both horizontally and vertically) differ by at most some constant M. We can create a matrix B' so that $B'_{k,j} = B_{k,j} - B_{1,1} + jM$. It is easy to check that the matrix B' satisfies the above-mentioned special case definition of monotone matrix. Thus, we can use our algorithm to compute the min-plus product $C' = A \star B'$. Then it is easy to recover $C = A \star B$ by setting $C_{i,j} = C'_{i,j} + B_{1,1} - jM$. Therefore, Monotone Min-Plus Product is more general than the Min-Plus Product of an arbitrary matrix with a bounded-difference matrix.

Monotone Min-Plus Product was first studied by Vassilevska Williams and Xu [25] as a tool to give a fast algorithm for the *Batch Range Mode* problem. In their work, the authors devise a black-box reduction from Monotone Min-Plus Product to their Min-Plus Product algorithm for matrices with a small $O(1)$-approximate rank. Their algorithm runs in $\tilde{O}(n^{(15+\omega)/6}) = O(n^{2.8955})$ time for the above-mentioned special case. We improve and generalize their algorithm. Below is a special case of our main theorem, which will be introduced in Section 3.

▶ **Theorem 1.** *The min-plus product $A \star B$ of two $n \times n$ matrices where entries of B are non-negative integers bounded by $O(n)$ and each row of B is non-decreasing can be computed deterministically in $\tilde{O}(n^{\frac{12+\omega}{5}})$ time. Using the current best bound on fast rectangular matrix multiplication the running time improves to $O(n^{2.8653})$.*

If $\omega = 2$, our improvement is from $\tilde{O}(n^{17/6}) \leq O(n^{2.8334})$ time to $\tilde{O}(n^{14/5}) = \tilde{O}(n^{2.8})$ time. We provide several interesting applications of our improved algorithm for Monotone Min-Plus Product.

1.1 Applications

Single Source Replacement Paths. The main contribution of this paper is establishing a relationship between Monotone Min-Plus Product and the Single-Source Replacement Paths (SSRP) problem. In the SSRP problem, one is given a directed edge-weighted graph G and a source vertex s, and is asked to compute for each edge e, $d_G(s, v, e)$'s, the shortest path distances from s to each vertex v in $G \setminus \{e\}$. Note that the interesting case is when e belongs to a shortest paths tree rooted at s, so that there are only $O(n^2)$ distances to report.

[3] Throughout the paper we assume the entries of the matrices are polylog(n)-bit integers or ∞ unless otherwise stated.

The trivial algorithm for SSRP runs in $\tilde{O}(n^3)$ time: For each edge e on the shortest path tree rooted at s, run Dijkstra's algorithm on the graph with e removed. Negative edge weights can be handled with Johnson's trick [16], without increasing the asymptotic complexity. Vassilevska Williams and Williams [24] showed that APSP and SSRP are sub-cubically equivalent. Hence, assuming the APSP hypothesis, there is no $O(n^{3-\epsilon})$ time algorithm for SSRP in graphs with arbitrary integer weights, for any $\epsilon > 0$. There seems to be little hope to improve upon the trivial algorithm for the general case.

Grandoni and Vassilevska Williams [14] studied SSRP in graphs with integer edge weights of small absolute value. They gave an algorithm that solves SSRP in directed n-vertex graphs with edge weights in $\{-M, \ldots, M\}$ in $\tilde{O}(M^{\frac{1}{4-\omega}} n^{2+\frac{1}{4-\omega}})$ time, which would become $\tilde{O}(M^{0.5} n^{2.5})$ if $\omega = 2$. For positive weights only, they reduce the runtime to $\tilde{O}(Mn^{\omega})$.

Let us consider the special case $M = 1$. Here, the algorithms of [14] solve SSRP with positive weights 1 in $\tilde{O}(n^{\omega})$ time, while the $\tilde{O}(n^{2+\frac{1}{4-\omega}})$ runtime for SSRP with weights in $\{-1, 0, 1\}$ is the same as the runtime for APSP with weights in $\{-1, 0, 1\}$.

As APSP in graphs with arbitrary integer weights is fine-grained equivalent to SSRP with arbitrary integer weights [24], it is possible that APSP with weights in $\{-1, 0, 1\}$ could be fine-grained equivalent to SSRP with weights in $\{-1, 0, 1\}$.

It is believed that APSP in directed graphs with weights in $\{-1, 0, 1\}$ (and even for unweighted graphs) requires $n^{2.5-o(1)}$ time [19, 27], as the best known algorithm by Zwick [31] would run in $\Omega(n^{2.5})$ time even if $\omega = 2$. As SSRP with small *positive* weights is in $O(n^{2.5-\varepsilon})$ time for $\varepsilon > 0$ [14], it is likely not fine-grained equivalent to directed unweighted APSP. Beating the APSP runtime for SSRP with *negative* weights is an open problem.

This leads to the following interesting questions.

(1) Is SSRP with negative weights inherently harder than SSRP with only positive weights?
(2) Or, is it possible to improve the running time of SSRP with negative weights, possibly below $n^{2.5}$, thus showing that it is likely not as hard as directed unweighted APSP?

Quite surprisingly, we give positive answers to both of these questions. First, we improve over the running time of [14] for negative weights.

▶ **Theorem 2.** *There is a randomized algorithm that solves SSRP in a directed n-vertex graph with edge weights in $\{-M, \ldots, M\}$ in $\tilde{O}(M^{\frac{5}{17-4\omega}} n^{\frac{36-7\omega}{17-4\omega}})$ time, with high probability. Using the current best bound on fast rectangular matrix multiplication the running time improves to $O(M^{0.8043} n^{2.4957})$.*

Notably, when M is small enough, the running time $O(M^{0.8043} n^{2.4957})$ is polynomially faster than $n^{2.5}$, and hence faster than the best known running time of APSP in directed unweighted graphs which is $\Omega(n^{2.5})$ even if $\omega = 2$. This answers our question (2) above. If $\omega = 2$, our running time for SSRP with negative weights is $\tilde{O}(M^{5/9} n^{22/9}) \leq O(M^{0.556} n^{2.445})$.

APSP in directed graphs with edge weights in $\{-1, 0, 1\}$ is one of long list of so-called *intermediate* graph and matrix problems [19, 27], whose running time lies between $\tilde{O}(n^{\omega})$ and $\tilde{O}(n^3)$ and becomes $\tilde{O}(n^{2.5})$ when $\omega = 2$. Our result shows that SSRP with bounded negative integer weights is not an intermediate problem. We remark that recently Grandoni et al. [13] showed that another (ex-)candidate intermediate problem, All-Pairs LCA in DAGs, can actually be solved faster than $O(n^{2.5})$ time.

We prove Theorem 2 by improving the runtime of the so-called *subpath problem*, which is the bottleneck in the algorithm of [14]. Grandoni and Vassilevska Williams solve it by reducing to APSP in directed graphs with edge weights in $\{-M, \ldots, M\}$, and applying Zwick's APSP algorithm [31]. We show that the APSP computation can be rearranged so that certain min-plus products that appear throughout involve monotone matrices.

Next, we identify an obstacle to obtaining a $\tilde{O}(Mn^{\omega})$ time algorithm for SSRP with negative weights, addressing our question (1).

▶ **Theorem 3.** *If there exists a $T(n)$ time algorithm for SSRP in n-vertex graphs with edge weights in $\{-1, 0, 1\}$, then there exists an $O(T(n)\sqrt{n})$ time algorithm for the Bounded-Difference Min-Plus Product of $n \times n$ matrices.*

Theorem 3 gives the following argument why SSRP with negative weights might be hard. The current best algorithm for Bounded-Difference Min-Plus Product runs in $O(n^{2.7554})$ time even if $\omega = 2$. If SSRP with weights $\{-1, 0, 1\}$ could be solved in $\tilde{O}(n^2)$ time (when $\omega = 2$), then Bounded-Difference Min-Plus Product could be solved in $\tilde{O}(n^{2.5})$ time, which would be a breakthrough in structured Min-Plus Product algorithms.

Recently replacement paths problems have received increased attention [8, 2, 9, 10]. None of these works is directly related to ours, because they focus either on the s-t Replacement Paths problem (with both source and target nodes fixed), or on combinatorial algorithms (i.e. without fast matrix multiplication) for sparse graphs.

Range Mode. Given an array a of elements, a range mode query asks for the most frequent element in a contiguous interval of a. In the Batch Range Mode problem the array a is fixed and all range mode queries are given in advance. In the Dynamic Range Mode problem one starts with an empty array and has to support insertions and deletions, and handle queries in an online fashion.

Vassilevska Williams and Xu [25] were the first to use structured Min-Plus Product in range mode algorithms. They reduced Batch Range Mode to a Min-Plus Product instance where both matrices have some monotone structures. Their techniques give an $O(n^{1.4854})$ time algorithm for Batch Range Mode on an array of size n and n queries. Since we improve over their Monotone Min-Plus Product algorithm, we naturally obtain a faster Batch Range Mode algorithm.

▶ **Theorem 4.** *The Batch Range Mode problem can be solved deterministically in time $\tilde{O}(n^{\frac{21+2\omega}{15+\omega}})$. Using the current best bound on fast rectangular matrix multiplication the running time improves to $O(n^{1.4805})$.*

There are multiple algorithms that solve Dynamic Range Mode in $\tilde{O}(n^{2/3})$ time per update and query on an array of size bounded by n [6, 11]. Recently, Sandlund and Xu [21] improved both update and query time to $O(n^{0.6560})$ by using a so-called *Min-Plus-Query-Witness* problem. During the preprocessing phase of the Min-Plus-Query-Witness problem, one is given two matrices A, B. During the query phase, given two indices i, j and a set S, one is asked to compute $\arg\min_{k \notin S}\{A_{i,k} + B_{k,j}\}$, where the set S can be viewed as the set of elements recently deleted in the array. In the Min-Plus-Query-Witness instances reduced from Dynamic Range Mode, the matrices A, B have the monotone property, so our techniques for Monotone Min-Plus Product can also apply to these Min-Plus-Query-Witness instances, leading to a faster Dynamic Range Mode algorithm.

▶ **Theorem 5.** *The Dynamic Range Mode problem can be solved deterministically in $\tilde{O}(n^{\frac{\omega+9}{\omega+15}})$ worst-case time per query with $\tilde{O}(n^{\frac{3\omega+39}{2\omega+30}})$ space. Using the current best bound on fast rectangular matrix multiplication improves the running time to $O(n^{0.6524})$ and the space complexity to $O(n^{1.3262})$.*

1.2 Overview of the Monotone Min-Plus Product Algorithm

Our improvement is achieved by extending Vassilevska Williams and Xu's [25] framework so that it can handle the more general monotone matrices. The algorithm has three phases. Say we would like to compute the min-plus product $C = A \star B$, where B is a monotone matrix.

In Phase 1, we compute a matrix $\tilde{C}$ which is close in ℓ_∞ norm to the desired output C. This can be done by, e.g., computing the min-plus product of $\lfloor \frac{A}{W} \rfloor$ and $\lfloor \frac{B}{W} \rfloor$, the downscaled versions of A and B, for some small parameter W.

In Phase 2, we repeatedly sample columns of B, and create new matrices A^r and B^r (for $r = 1, 2, \ldots$) whose entries are simple linear combinations of A, B and $\tilde{C}$. We replace large-magnitude entries of A with ∞, so that all finite entries of A^r are of small absolute values, and that the min-plus product $A^r \star B^r$ can be computed efficiently. The results are collected in a way such that by the end of Phase 2, we have found, for every pair (i, j),

$$\min_k \{A_{i,k} + B_{k,j} : A^r_{i,k} \neq \infty \text{ for some } r\}.$$

In Phase 3, we deal with (i, k) such that $A^r_{i,k} = \infty$ for all r. Such (i, k) are called *uncovered*. It can be shown that the number of *relevant* triples (i, k, j) with (i, k) uncovered is small, and we can afford enumerating all such triples. The enumeration is done by performing a witness-listing version of min-plus product of the downscaled versions of A and B.

We base on the fact that, when a monotone matrix B has very small entries, the number of *changes*, i.e. pairs (k, j) for which $B_{k,j} \neq B_{k,j+1}$, can be upper-bounded. Then for each fixed i we let j iterate through its range, and we maintain the set $\{A_{i,k} + B_{k,j}\}$ during the iteration. The total number of updates to the set is exactly the number of changes. This gives us an efficient way to compute min-plus product (and its witness-listing version) of the downscaled matrices, and leads to an efficient running time for Phases 1 and 3.

What makes our improvement possible is that we focus directly on the structure of monotone matrices, instead of going through a lossy black-box reduction to the Min-Plus Product of bounded-difference matrices, like the previous work [25] did.

2 Preliminaries

▶ **Definition 6** (Rectangular Matrix Multiplication Exponent). *Let α, β, γ be non-negative real numbers. Define $\omega(\alpha, \beta, \gamma)$ to be the smallest number such that the product of an $n^\alpha \times n^\beta$ matrix by an $n^\beta \times n^\gamma$ matrix can be computed in $\tilde{O}(n^{\omega(\alpha,\beta,\gamma)})$ time.*

▶ **Definition 7.** *Let α, β, γ, θ be non-negative real numbers. Define $g(\alpha, \beta, \gamma, \theta)$ to be the smallest number such that the min-plus product of an $n^\alpha \times n^\beta$ matrix whose entries are in $\{-n^\theta, \ldots, n^\theta\} \cup \{\infty\}$ by an arbitrary $n^\beta \times n^\gamma$ matrix can be computed in $\tilde{O}(n^{g(\alpha,\beta,\gamma,\theta)})$ time.*

▶ **Definition 8** (Bounded-Difference Matrix). *An $n \times m$ matrix A is called a bounded-difference matrix if $|A_{i,j} - A_{i,j+1}| \leq 1$ for every $1 \leq i \leq n, 1 \leq j < m$ and $|A_{i,j} - A_{i+1,j}| \leq 1$ for every $1 \leq i < n, 1 \leq j \leq m$.*

▶ **Problem 9** (Bounded-Difference Min-Plus Product). *Given two bounded-difference integer matrices A and B, compute $A \star B$.*

▶ **Definition 10** (Monotone Matrix). *An $n^\beta \times n^\gamma$ matrix B is called monotone if for every $k \in [n^\beta]$, $B_{k,j}$ is non-decreasing in $j \in [n^\gamma]$. For a monotone matrix B, we define its total range as $\sum_{k \in [n^\beta]} (\max_{j \in [n^\gamma], B_{k,j} \neq \infty} B_{k,j} - B_{k,1} + 1)$.*

▶ **Problem 11** (Monotone Min-Plus Product). *Given an $n^\alpha \times n^\beta$ matrix A and an $n^\beta \times n^\gamma$ matrix B where B is monotone and has total range $O(n^{\beta+\eta})$, where α, β, γ, η are non-negative real numbers, compute the min-plus product $A \star B$.*

▶ **Definition 12.** *Define $m(\alpha, \beta, \gamma, \eta)$ to be the smallest number such that Monotone Min-Plus Product with parameters α, β, γ, η, can be computed in $\tilde{O}(n^{m(\alpha,\beta,\gamma,\eta)})$ time.*

In our applications, we only need the case $\alpha = \gamma$. Because $m(c\alpha, c\beta, c\gamma, c\eta) = cm(\alpha, \beta, \gamma, \eta)$ for any $c \geq 0$, it suffices to consider only the case $\alpha = \gamma = 1$.

2.1 Upper bounds for g

We prove some useful upper bound for the function $g(\cdot, \cdot, \cdot, \cdot)$ introduced in Definition 7. We use the following bound from [7], which is a straightforward generalization of Theorem 1.2 from [25] to the case of rectangular matrices.

▶ **Lemma 13.** *For any non-negative real numbers $\alpha, \beta, \gamma, \theta$,*

$$g(\alpha, \beta, \gamma, \theta) \leq \min_{0 \leq \delta \leq \beta} \max\{\omega(\alpha, \beta, \beta + \gamma - \delta) + \theta, \delta + \alpha + \gamma\}. \tag{1}$$

We only need the following special cases of (1).

▶ **Corollary 14.** *For any non-negative real numbers β and θ,*

$$g(1, \beta, 1, \theta) \leq \frac{1}{2}(2 + \beta + \omega(1, \beta, 1) + \theta), \tag{2}$$

$$g(1, 1, 1, \theta) \leq \min_{0 \leq \delta \leq 1} \max\{\omega(1, 1, 2 - \delta) + \theta, 2 + \delta\}. \tag{3}$$

Proof. Consider the term $\omega(\alpha, \beta, \beta + \gamma - \theta)$ in (1). By splitting matrix B into $n^{\beta-\delta}$ matrices along its second dimension and computing $n^{\beta-\delta}$ independent instances of matrix multiplications, we get

$$\omega(\alpha, \beta, \beta + \gamma - \delta) \leq \omega(\alpha, \beta, \gamma) + \beta - \delta. \tag{4}$$

We plug (4) into (1), and take $\delta = \min\{\beta, \frac{1}{2}(\omega(\alpha, \beta, \gamma) + \beta + \theta - \alpha - \gamma)\}$, to get

$$g(\alpha, \beta, \gamma, \theta) \leq \max\{\omega(\alpha, \beta, \gamma) + \theta, \frac{1}{2}(\alpha + \beta + \gamma + \omega(\alpha, \beta, \gamma) + \theta)\}. \tag{5}$$

However, if $\omega(\alpha, \beta, \gamma) + \theta \geq \frac{1}{2}(\alpha + \beta + \gamma + \omega(\alpha, \beta, \gamma) + \theta)$, then both sides of (5) are at least $\alpha + \beta + \gamma$, and we can compute the min-plus product in $O(n^{\alpha+\beta+\gamma})$ time using a trivial algorithm. Therefore we get (2).

(3) is a simple substitution $\alpha = \beta = \gamma = 1$ to (1). ◀

▶ Remark 15. In the rectangular case $\alpha = \gamma = 1$, we use (4) so that in the final expression (2) we only need to deal with terms of the form $\omega(1, \beta, 1)$, whose value can be bounded by [18]. We know of no handy upper bounds for $\omega(\alpha, \beta, \gamma)$ when all three parameters are distinct.

In the square case $\alpha = \beta = \gamma = 1$, we do not need to use the simplification (4). This is because the upper bound of $\omega(1, \beta, 1)$ in [18] is by a bilinear algorithm. Thus by [15], the same upper bound works for $\omega(1, 1, \beta)$.

3 Monotone Min-Plus Product

▶ **Theorem 16.** *The min-plus product of an $n \times n^\beta$ matrix A and an $n^\beta \times n$ matrix B where B is monotone with total range $O(n^{\beta+\eta})$ can be deterministically computed, for any $\theta \in [0, \eta]$, in time $\tilde{O}(n^{\max\{1+\beta+\eta-\theta, \frac{1}{2}(2+\beta+g(1,\beta,1,\theta))\}})$. In other words,*

$$m(1,\beta,1,\eta) \le \min_{0\le\theta\le\eta} \max\{1+\beta+\eta-\theta, \frac{1}{2}(2+\beta+g(1,\beta,1,\theta))\}.$$

Proof. Proof of Theorem 16 follows the three-phase framework of [4, 25], which we have briefly described in Section 1.2. Here we present the full algorithm.

Phase 1. Let $\theta \in [0, \eta]$ be a parameter, and let $W = \lfloor n^\theta \rfloor$. We define two matrices $\tilde{A}$ and $\tilde{B}$ as $\tilde{A}_{i,k} = \lfloor \frac{A_{i,k}}{W} \rfloor$ and $\tilde{B}_{k,j} = \lfloor \frac{B_{i,k}}{W} \rfloor$. We compute the min-plus product $\tilde{A} \star \tilde{B}$ and let $\tilde{C}_{i,j} = (\tilde{A} \star \tilde{B})_{i,j} W$. Then $||\tilde{C} - C||_\infty \le 2W$.

We compute $\tilde{A} \star \tilde{B}$ using the following lemma, which is a simple algorithm that works fast when the total range is very small.

▶ **Lemma 17.** $m(\alpha, \beta, \gamma, \eta) \le \max\{\alpha+\gamma, \beta+\gamma, \alpha+\beta+\eta\}$.

Proof. Say we would like to compute $C = A \star B$. For a fixed row $i \in [n^\alpha]$, we iterate through columns $j \in [n^\gamma]$, maintaining the multi-set $\{A_{i,k} + B_{k,j} : k \in [n^\beta]\}$.

Each time j increases, we need to update the multi-set for those k where $B_{k,j} \ne B_{k,j-1}$. The total number of (k, j) satisfying $B_{k,j} \ne B_{k,j-1}$ is $O(n^{\beta+\eta})$ by monotonicity and the bound on total range.

For each $i \in [n^\alpha]$, we need to make $O(n^{\beta+\eta})$ updates and $O(n^\gamma)$ queries for the minimum number in the multi-set. We can use a balanced BST to maintain the multi-set so that each update and query costs $\tilde{O}(1)$ time. The total running time is $\tilde{O}(n^{\max\{\alpha+\gamma, \beta+\gamma, \alpha+\beta+\eta\}})$. ◀

Total range of $\tilde{B}$ is $O(n^{\beta+\eta-\theta})$, so running time of Phase 1 is $\tilde{O}(n^{\max\{2, 1+\beta+\eta-\theta\}})$.

Phase 2. In Phase 2 we compute a matrix $\hat{C}$ which upper bounds $C = A \star B$ and agrees with it on most entries. Initially, let $\hat{C}_{i,j} \leftarrow \infty$ for all $i, j \in [n]$.

Phase 2 consists of $(10+\beta)n^\rho \log n$ *rounds*, for a parameter $\rho \ge 0$ to be chosen later. In the r-th round, we choose $j^r \in [n]$ uniformly at random[4]. Define matrix A^r and B^r as

$$A^r_{i,k} = \begin{cases} A_{i,k} + B_{k,j^r} - \tilde{C}_{i,j^r} & \text{if } A_{i,k} + B_{k,j^r} - \tilde{C}_{i,j^r} \le 3W \text{ and } A^{r'}_{i,k} = \infty \forall r' < r, \\ \infty & \text{otherwise,} \end{cases}$$

$$B^r_{k,j} = \begin{cases} B_{k,j} - B_{k,j^r} & \text{if } B_{k,j^r} \ne \infty, \\ 0 & \text{otherwise.} \end{cases}$$

We compute $C^r = A^r \star B^r$ using Corollary 14 because A^r has bounded entries. Finally, for all $i, j \in [n]$, we make the update $\hat{C}_{i,j} \leftarrow \min\{\hat{C}_{i,j}, C^r_{i,j} + \tilde{C}_{i,j^r}\}$.

In other words, in the end we have $\hat{C}_{i,j} = \min_r\{C^r_{i,j} + \tilde{C}_{i,j^r}\}$. If $A^r_{i,k} \ne \infty$, then for all j, we have $\hat{C}_{i,j} \le C^r_{i,j} + \tilde{C}_{i,j^r} \le A^r_{i,k} + B^r_{i,k} + \tilde{C}_{i,j^r} = A_{i,k} + B_{k,j}$. Thus in this case we have effectively updated $\hat{C}_{i,j}$'s using $A_{i,k} + B_{k,j}$ for all j.

[4] For simplicity of presentation, we use randomness here. The derandomization is deferred to the full version of the paper.

Following [25], we make the following definitions: a triple $(i,k,j) \in [n] \times [n^\beta] \times [n]$ is *strongly relevant*, if $A_{i,k} + B_{k,j} = C_{i,j}$; *weakly relevant*, if $A_{i,k} + B_{k,j} - \tilde{C}_{i,j} \leq 3W$; *covered*, if $A^r_{i,k} \neq \infty$ for some r; *uncovered*, if it is not covered. We use the following lemma from [25].

▶ **Lemma 18** ([25, Lemma 4.3]). *With probability $1 - n^{-9}$, the number of triples that are weakly relevant and uncovered is at most $n^{2+\beta-\rho}$.*

Proof. Fix some pair of (i,k). If the number of j such that (i,k,j) is weakly relevant is at least $n^{1-\rho}$, then with probability at least $1 - (1-n^{-\rho})^{(10+\beta)n^\rho \log n} \geq 1 - n^{-10-\beta}$, we will sample a j^r such that (i,k,j^r) is weakly relevant. If so, $A^r_{i,k} \neq \infty$ for some r and thus (i,k,j) will be covered for all j. Therefore, with probability at least $1 - n^{-9}$, all (i,k) that are in at least $n^{1-\rho}$ weakly relevant triples will be covered. The number of remaining weakly relevant triples is at most $n^{2+\beta-\rho}$. ◀

Each round costs $\tilde{O}(n^{g(1,\beta,1,\theta)})$ time. Phase 2 costs $\tilde{O}(n^{\rho+g(1,\beta,1,\theta)})$ time in total.

Phase 3. We define a triple (i,k,j) to be *moderately relevant* if $\tilde{A}_{i,k} + \tilde{B}_{k,j} \leq (\tilde{A} \star \tilde{B})_{i,j} + 1$. In Phase 3, we enumerate over moderately relevant and uncovered triples to complete matrix C.

▶ **Lemma 19.** *Every strongly relevant triple is also moderately relevant.*

Proof. Suppose (i,k,j) is strongly relevant. If $\tilde{A}_{i,k'} + \tilde{B}_{k',j} = (\tilde{A} \star \tilde{B})_{i,j}$ for some k', then because $A_{i,k} + B_{k,j} \leq A_{i,k'} + B_{k',j}$, we have

$$\tilde{A}_{i,k} + \tilde{B}_{k,j} \leq \tilde{A}_{i,k'} + \tilde{B}_{k',j} + 1 = (\tilde{A} \star \tilde{B})_{i,j} + 1.$$

Hence (i,k,j) is moderately relevant. ◀

▶ **Lemma 20.** *Every moderately relevant triple is also weakly relevant.*

Proof. Suppose (i,k,j) is moderately relevant. Then

$$\begin{aligned} A_{i,k} + B_{k,j} - \tilde{C}_{i,j} &\leq (\tilde{A}_{i,k} + 1)W + (\tilde{B}_{k,j} + 1)W - (\tilde{A} \star \tilde{B})_{i,j}W \\ &\leq (\tilde{A}_{i,k} + \tilde{B}_{k,j} - (\tilde{A} \star \tilde{B})_{i,j})W + 2W \leq 3W. \end{aligned}$$

Hence (i,k,j) is weakly relevant. ◀

By Lemma 19, it suffices to enumerate over moderately relevant and uncovered triples to recover all of C. By Lemmas 18 and 20, the number of moderately relevant and uncovered triples is at most $O(n^{2+\beta-\rho})$, with high probability.

▶ **Lemma 21.** *With high probability, it takes time $\tilde{O}(n^{\max\{2,1+\beta+\eta-\theta,2+\beta-\rho\}})$ to enumerate all moderately relevant and uncovered triples.*

Proof. Define matrix $\check{A}$ as $\check{A}_{i,k} = \tilde{A}_{i,k}$ if (i,k) is uncovered; and $\check{A}_{i,k} = \infty$ otherwise.

We proceed on computing $\check{A} \star \tilde{B}$ in a way similar to Lemma 17 from Phase 1. For each row i, we maintain the set $\{(\check{A}_{i,k} + \tilde{B}_{k,j}, k) : k \in [n^\beta]\}$ as j iterates over $[n]$. Each time j increases, we need to update the multi-set for those k where $\tilde{B}_{k,j} \neq \tilde{B}_{k,j-1}$. The total number of (k,j) satisfying $\tilde{B}_{k,j} \neq \tilde{B}_{k,j-1}$ is $O(n^{\beta+\eta-\theta})$.

For each (i,j), we enumerate the elements in the multi-set in the increasing order, and stop as soon as we observe a k where $\check{A}_{i,k} + \tilde{B}_{k,j} > (\tilde{A} \star \tilde{B})_{i,j} + 1$. Therefore we enumerate exactly the moderately relevant uncovered triples. The running time is the running time from Lemma 17, plus the number of triples emitted, which, with high probability, is at most $O(n^{2+\beta-\rho})$, by Lemma 18. ◀

Phase 3 runs in time $\tilde{O}(n^{\max\{2,1+\beta+\eta-\theta,2+\beta-\rho\}})$.

Summary. Overall running time of our algorithm is $\tilde{O}(n^{\max\{2,1+\beta+\eta-\theta,\rho+g(1,\beta,1,\theta),2+\beta-\rho\}})$. Note that $g(1,\beta,1,\theta) \leq 2+\beta$. So we can take $\rho = \frac{1}{2}(2+\beta-g(1,\beta,1,\theta))$. Also note that $\rho + g(1,\beta,1,\theta) \geq 2$, so the 2 in the max expression can be ignored. In the end we get $\tilde{O}(n^{\max\{1+\beta+\eta-\theta,\frac{1}{2}(2+\beta+g(1,\beta,1,\theta))\}})$ as claimed. ◀

As a benchmark, let us consider the case $\alpha = \beta = \gamma = \eta = 1$.

▶ **Theorem 1.** *The min-plus product $A \star B$ of two $n \times n$ matrices where entries of B are non-negative integers bounded by $O(n)$ and each row of B is non-decreasing can be computed deterministically in $\tilde{O}(n^{\frac{12+\omega}{5}})$ time. Using the current best bound on fast rectangular matrix multiplication the running time improves to $O(n^{2.8653})$.*

Proof. In this case, Theorem 16 simplifies (via (3)) to

$$\begin{aligned} m(1,1,1,1) &\leq \min_{0\leq\theta\leq 1} \max\{3-\theta, \frac{1}{2}(3+g(1,1,1,\theta))\} \\ &\leq \min_{0\leq\theta\leq 1} \max\{3-\theta, \frac{1}{2}(3+\min_{0\leq\delta\leq 1} \max\{\omega(1,1,2-\delta)+\theta, 2+\delta\})\}. \end{aligned} \tag{6}$$

Without using rectangular matrix multiplication, we can use $\omega(1,1,2-\delta) \leq 1-\delta+\omega$ and take $\theta = \frac{3-\omega}{5}$ and $\delta = \frac{2\omega-1}{5}$, so (6) takes value $\frac{12+\omega}{5}$.

Using the rectangular matrix multiplication upper bounds in [18] (see also Remark 15), we find that when $\theta = 0.1348$, $\delta = 0.7305$, expression (6) takes value ≤ 2.8653. ◀

4 Single Source Replacement Paths

We show our algorithm and lower bound for SSRP in this section. We use $d_G(u,v)$ to denote the length of a shortest path from u to v in a graph G, and we use $d_G(u,v,e)$ as a shorthand for $d_{G\setminus\{e\}}(u,v)$. When it is clear from the context, we sometimes omit G.

4.1 Algorithm

In this section we present our improved algorithm for SSRP, proving Theorem 2.

▶ **Theorem 2.** *There is a randomized algorithm that solves SSRP in a directed n-vertex graph with edge weights in $\{-M,\ldots,M\}$ in $\tilde{O}(M^{\frac{5}{17-4\omega}} n^{\frac{36-7\omega}{17-4\omega}})$ time, with high probability. Using the current best bound on fast rectangular matrix multiplication the running time improves to $O(M^{0.8043}n^{2.4957})$.*

To this end we improve the bottleneck in Grandoni-Vassilevska Williams algorithm [14], hence let us begin with a high level overview of that algorithm. This is however just to give a context and intuition, and our formal proof of Theorem 2 follows from a black-box[5] application of Lemmas 23 and 27.

[5] See end of the proof of Lemma 27 for a discussion why the $\tilde{O}(Mn^{\omega})$ component from Lemma 23 can be omitted.

Algorithm overview and the subpath problem. The algorithm first computes a shortest paths tree (from the source vertex s), and splits it into a subpolynomial number of subtrees. By using balanced separators, the subtrees can be roughly of the same size. Then, for each such subtree T, values $d_G(s, v, e)$ such that both vertex v and edge e belong to T are deferred to a recursive call on a graph obtained from G by carefully compressing its parts outside T. The only remaining interesting values $d_G(s, v, e)$ (i.e. such that they might be different from $d_G(s, v)$) are such that vertex v belongs to subtree T and edge e lies on the path from s to the root of T in the shortest paths tree. The problem of computing those remaining values is called *subpath problem*, which we now define formally.

▶ **Definition 22** (Subpath problem). *Given an n-vertex directed graph G with edge weights in $\{-M, \ldots, M\}$, a source vertex s, and a tree T which is a subtree of a shortest paths tree from s, compute $d_G(s, v, e)$ for every $v \in T$ and every e on the path from s to the root t of T in the shortest paths tree.*

Using the ideas outlined above, Grandoni and Vassilevska Williams formally reduce SSRP to the subpath problem.

▶ **Lemma 23** (Lemma 5.1 in [14]). *Given an algorithm that solves the subpath problem in time $\tilde{O}(M^\alpha n^\beta)$, with high probability, for constants $\beta \geq \alpha + 1 \geq 1$, there is an algorithm that solves SSRP in time $\tilde{O}(Mn^\omega + M^\alpha n^\beta)$, with high probability.*

Jumping paths and departing paths. We proceed to show how to solve the subpath problem. Let $P = (s = s_1 \to s_2 \to \cdots \to s_{|P|} = t)$ be the s-t path in the shortest paths tree. A replacement path witnessing $d_G(s, v, e)$ has to depart from P somewhere before e and then can either (1) join P back somewhere after e, and thus reach v through t, or (2) never use any other edge of P after departing. Paths of the first type are called *jumping paths*, and of the second type – *departing paths*. Grandoni and Vassilevska Williams [14] use the fact that jumping paths can be found by solving the *s-t replacement paths* problem, i.e. computing all $d_G(s, t, e)$'s for fixed t, which can be computed in $\tilde{O}(Mn^\omega)$ time (see Lemma 24). We just follow their approach in that regard.

▶ **Lemma 24** (Theorem 1.1 in [14]). *There is a randomized algorithm that solves s-t replacement paths problem in $\tilde{O}(Mn^\omega)$ time, with high probability.*

Improved algorithm for departing paths. Let $\widetilde{G}$ denote the graph obtained from G by removing all edges on path P. Note that the length of a shortest departing replacement path to v avoiding $e = (s_i, s_{i+1})$ equals to $\min_{j \leq i} d_G(s, s_j) + d_{\widetilde{G}}(s_j, v)$. Grandoni and Vassilevska Williams [14] simply feed $\widetilde{G}$ to Zwick's APSP algorithm [31], running in time $\tilde{O}(M^{\frac{1}{4-\omega}} n^{2+\frac{1}{4-\omega}})$, to compute all $d_{\widetilde{G}}(s_j, v)$'s. We take a different approach and employ our truly subcubic algorithm for Monotone Min-Plus Product. We remark that any truly subcubic algorithm would yield an improvement.

Let $\zeta \in [0, 1]$ be a parameter to be determined later. We say that a departing replacement path is *hop-long* if it visits at least n^ζ nodes after departing P, otherwise it is *hop-short*. We handle the two types of paths separately.

Hop-short paths. To find hop-short paths we use a modification of Zwick's APSP algorithm [31], already described in [14]. Zwick's algorithm consists of $O(\log n)$ iterations, and in the i-th iteration it computes the shortest paths which use at most $(3/2)^i$ nodes. By running only first few iterations we can compute all hop-short shortest paths in time $\tilde{O}(Mn^{\zeta+\omega(1,1-\zeta,1)})$, which is faster than it would take to compute all shortest paths (given that ζ is small enough). For a formal proof of this statement we refer to [14].

▶ **Lemma 25** (Corollary 3.1 in [14]). *The distances between all pairs of nodes that have shortest paths on at most n^ζ nodes can be computed in time $\tilde{O}(Mn^{\zeta+\omega(1,1-\zeta,1)})$, with high probability.*

Hop-long paths. To find hop-long paths, first we sample (with replacement) $c \cdot n^{1-\zeta} \ln n$ nodes, for a large enough constant c. Let $B \subseteq V$ denote the set of sampled nodes. For the sake of analysis let us fix a set $\mathcal{S}$ of shortest hop-long departing replacement for all nodes $v \in T$ and all edges $e \in P$. When there is more than one such path of the smallest length for a given pair (v, e), we choose an arbitrary one. Note that for paths in $\mathcal{S}$, we only include the portions after they depart P so that they only contain edges in $\widetilde{G}$. Since the definition of hop-long paths only concerns the length of the part of a path after it departs P, all paths in $\mathcal{S}$ have length at least n^ζ. By a standard proof, with high probability, every path in $\mathcal{S}$ contains a node from B which lies in that path's middle third part.

Then we construct Yuster-Zwick distance oracle for graph $\widetilde{G}$ (see Lemma 26 below) and use it compute all $\tilde{O}(n \cdot n^{1-\zeta})$ shortest paths to and from B using at least $(1/3) \cdot n^\zeta$ nodes. In total it takes time $\tilde{O}(Mn^\omega + n^{3-2\zeta})$.

▶ **Lemma 26** (implicit in [29], Lemma 2.3 in [14]). *Given an n-vertex directed graph G, with edge weights in $\{-M, \ldots, M\}$, one can compute in $\tilde{O}(Mn^\omega)$ time an $n \times n$ matrix D, so that the (i, j)-th entry of the min-plus product $D \star D$ is the distance from node i to j in G. Furthermore, by the properties of D, the length of a shortest $i \rightsquigarrow j$ path containing at least n^ζ nodes can be computed in $\tilde{O}(n^{1-\zeta})$ extra time, with high probability.*

Let $d^{YZ}(\cdot, \cdot)$ denote such computed distances. The length of a shortest hop-long departing replacement path to v avoiding $e = (s_i, s_{i+1})$ equals $\min_{j \le i} \min_{b \in B} d_G(s, s_j) + d^{YZ}(s_j, b) + d^{YZ}(b, v)$.

We create two matrices A and B, of dimensions at most $n \times |B|$ and $|B| \times n$, respectively, such that

$$A_{i,b} = \min_{j \le i} d_G(s, s_j) + d^{YZ}(s_j, b), \quad \text{and} \quad B_{b,v} = d^{YZ}(b, v).$$

We need to compute $A \star B$. Note that $A_{i+1,b} \le A_{i,b}$, i.e. columns of A are monotone. Moreover, finite entries of A are of absolute value at most $2 \cdot nM$, so we can compute $A \star B = (B^T \star A^T)^T$ in time $\tilde{O}(n^{m(1,1-\zeta,1,1+\log_n M)})$.

Wrap-up and runtime analysis. Now we can sum up the running time and then balance the terms. The proof of the following lemma is deferred to the full version of the paper.

▶ **Lemma 27.** *There is a randomized algorithm that solves the subpath problem in a directed n-vertex graph with edge weights in $\{-M, \ldots, M\}$ in $\tilde{O}(M^{\frac{5}{17-4\omega}} n^{\frac{36-7\omega}{17-4\omega}})$ time, with high probability. Using fast rectangular matrix multiplication improves the running time to $O(M^{0.8043} n^{2.4957})$.*

4.2 Lower Bound

In this section, we prove our conditional lower bound for SSRP.

▶ **Theorem 3.** *If there exists a $T(n)$ time algorithm for SSRP in n-vertex graphs with edge weights in $\{-1, 0, 1\}$, then there exists an $O(T(n)\sqrt{n})$ time algorithm for the Bounded-Difference Min-Plus Product of $n \times n$ matrices.*

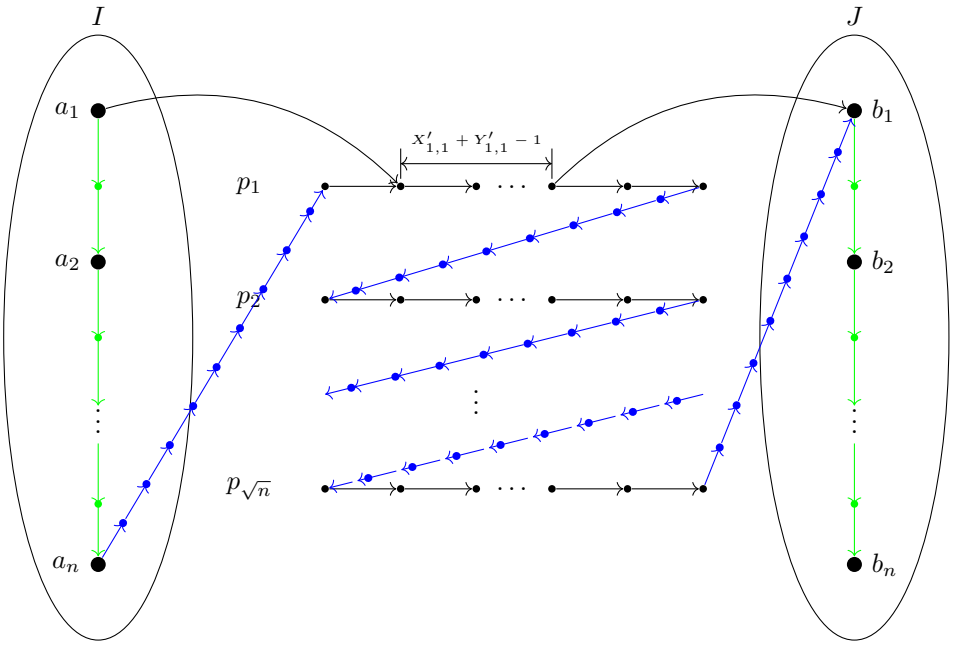


Figure 1 Reduction from Bounded-Difference Min-Plus Product to Ham-APSP (Lemma 29). Most edges between the parts I, J and the middle paths $p_1, \ldots, p_{\sqrt{n}}$ are omitted for clarity. The green portions are the first type paths, and the blue portions are the second type paths.

We first reduce Bounded-Difference Min-Plus Product to a problem called Ham-APSP, then we further reduce Ham-APSP to SSRP.

▶ **Problem 28** (Ham-APSP). *Given a directed unweighted graph G with vertex set $\{v_1, \ldots, v_n\}$ and a Hamiltonian path $v_1 \to \cdots \to v_n$ of G, compute all pairs shortest path distances in G.*

The key idea in the following reduction was used by Chan et al. [7] for a reduction from Min-Plus Product with small integer weights to unweighted directed APSP.

▶ **Lemma 29.** *If there exists a $T(n)$ time algorithm for Ham-APSP in a graph with n vertices, then there exists an $O(T(n)\sqrt{n})$ time algorithm for Bounded-Difference Min-Plus Product of $n \times n$ matrices.*

Proof. Given two $n \times n$ bounded-difference matrices A and B, we first split the columns of A and rows of B to $O(\sqrt{n})$ pieces. For each pair of pieces, we need to compute the min-plus product of an $n \times \sqrt{n}$ bounded-difference matrix X and a $\sqrt{n} \times n$ bounded-difference matrix Y. We will use a single call of the assumed $T(n)$ time algorithm for Ham-APSP to compute the min-plus product between each pair of pieces, yielding an $O(T(n)\sqrt{n})$ overall running time.

We create a new matrix X' such that $X'_{i,k} = X_{i,k} - X_{i,1}$. Since $|X_{i,k} - X_{i,k+1}| \leq 1$ for any i, k, all entries of X' are bounded by $\sqrt{n}$. We can create Y' similarly by setting $Y'_{k,j} = Y_{k,j} - Y_{1,j}$ so that all entries of Y' are bounded by $\sqrt{n}$ as well. We will later use the Ham-APSP algorithm to compute $X' \star Y'$, which immediately gives $X \star Y$ via the relation $(X \star Y)_{i,j} = (X' \star Y')_{i,j} + X_{i,1} + Y_{1,j}$.

In [7], Min-Plus Product of an $n \times \sqrt{n}$ and a $\sqrt{n} \times n$ matrices with weights up to $\sqrt{n}$ is reduced to unweighted directed APSP on n-node graphs. Here is a description of that reduction. We create a vertex set I of n vertices $\{a_1, \ldots, a_n\}$ and a vertex set J of n vertices $\{b_1, \ldots, b_n\}$. We also create $\sqrt{n}$ paths $p_1, \ldots, p_{\sqrt{n}}$ each of length $2\sqrt{n}$. From each a_i to p_k,

we add a directed edge from a_i to the $(\sqrt{n} - X'_{i,k})$-th node on p_k; similarly, from each p_k to b_j, we add a directed edge from the $(\sqrt{n} + Y'_{k,j})$-th node on p_k to b_j. Then we can see that the distance from a_i to b_j equals $(X' \star Y')_{i,j} + 2$.

In order to have a Hamiltonian path in the graph, we need to add two types of additional paths.

For the first type, we add paths of length 2 from a_i to a_{i+1} and from b_i to b_{i+1} for every $1 \le i < n$. Clearly, we only add $O(n)$ vertices and $O(n)$ edges. Now consider the shortest path from a_i to b_j for some i, j. The shortest path has the option to go to some $a_{i'}$ for $i' \ge i$, then choose some path p_k in the middle, then go to $b_{j'}$ for $j' \le j$, and finally reach b_j. The cost of this path would be $2(i' - i) + 1 + X'_{i',k} + Y'_{k,j'} + 1 + 2(j - j')$. Because X and Y have bound differences, we have that $2(i' - i) + X'_{i',k} \ge X'_{i,k}$ and $Y'_{k,j'} + 2(j - j') \ge Y'_{k,j}$. Therefore, in one of the shortest paths from a_i to b_j, we have $i' = i$ and $j' = j$. Thus, the distance from a_i to b_j is exactly $2 + (X' \star Y')_{i,j}$, so we can recover $X' \star Y'$ by computing all the pairwise distances.

For the second type of paths, we add $O(\sqrt{n})$ paths of lengths $3\sqrt{n}$ to connect I, J and each p_k, as shown in Figure 1. The total number of vertices and number of edges added are both $O(n)$. Since all distances we care about are at most $2\sqrt{n} + O(1)$, adding those paths won't affect these distances.

This graph now has a Hamiltonian path: we can travel from a_1 to a_n via the first type of paths. Then we use the second type of paths to travel from a_n to the beginning of p_1 and then we can easily travel to the end of p_1 by using edges of p_1. Similarly, we can go through all vertices in $p_2, \ldots, p_{\sqrt{n}}$. Finally, we travel from the end of $p_{\sqrt{n}}$ to b_1 via the second type of paths, and then use the first type of paths to travel to b_n. ◀

In the following lemma, we further reduce Ham-APSP to SSRP.

▶ **Lemma 30.** *If there exists a $T(n)$ time algorithm for SSRP in a graph with n vertices whose edge weights are in $\{-1, 0, 1\}$, then there exists an $O(T(n))$ time algorithm for Ham-APSP in a graph with n vertices.*

Proof. Let G be an instance of the Ham-APSP problem. We first create a graph G' whose vertex set is $\{v'_0, v'_1, \ldots, v'_n, v'_{n+1}\} \cup \{v_1, \ldots, v_n\}$. Then we add the following three types of edges to G, as depicted in Figure 2:

1. We add an edge from v'_i to v'_{i-1} of weight -1 for every $1 \le i \le n + 1$;
2. We add an edge from v'_i to v_i of weight 0 for every $1 \le i \le n$;
3. We add an edge from v_i to v_j of weight 1 for every $(v_i, v_j) \in E(G)$. This part essentially pastes a copy of G to G'.

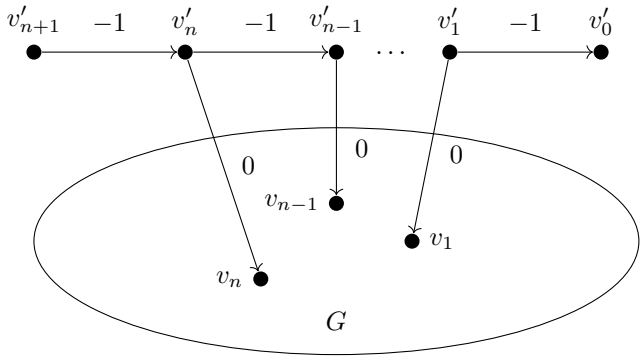


Figure 2 Reduction from Ham-APSP to SSRP with weights in $\{-1, 0, 1\}$ (Lemma 30).

If we cut the edge (v'_i, v'_{i-1}) in the graph G', then the shortest path from v'_{n+1} to v_j for some $1 \le j \le n$ must move from v'_{n+1} to v'_k for some $i \le k \le n$, then take the weight 0 edge to v_k, and finally move to v_j in the copy of graph G. Therefore, $d_{G'}(v'_{n+1}, v_i, (v'_i, v'_{i-1})) = \min_{i \le k \le n}(k - n - 1) + d_G(v_k, v_j)$. We show that $\min_{i \le k \le n}(k - n - 1) + d_G(v_k, v_j) = (i - n - 1) + d_G(v_i, v_j)$. Clearly, $\min_{i \le k \le n}(k - n - 1) + d_G(v_k, v_j) \le (i - n - 1) + d_G(v_i, v_j)$ since the right hand side is one of the terms we are minimizing over.

To show the other direction, we fix an arbitrary $k \in [i, n]$. By triangle inequality, $d_G(v_i, v_k) + d_G(v_k, v_j) \ge d_G(v_i, v_j)$. Since G has a Hamiltonian path $v_1 \to \cdots \to v_n$, it holds that $d_G(v_i, v_k) \le k - i$. Hence,

$$(k - n - 1) + d_G(v_k, v_j) \ge (k - n - 1) + d_G(v_i, v_j) - d_G(v_i, v_k) \ge (i - n - 1) + d_G(v_i, v_j).$$

We have shown that $d_{G'}(v'_{n+1}, v_i, (v'_i, v'_{i-1})) = (i - n - 1) + d_G(v_i, v_j)$. Thus, we can infer the pairwise distances in G by querying the assumed $T(n)$ time SSRP algorithm on graph G' since $d_G(v_i, v_j) = d_{G'}(v'_{n+1}, v_i, (v'_i, v'_{i-1})) - (i - n - 1)$. ◀

5 Range Mode

In this section, we show our improved algorithms for Batch Range Mode and Dynamic Range Mode.

5.1 Batch Range Mode

▶ **Theorem 4.** *The Batch Range Mode problem can be solved deterministically in time* $\tilde{O}(n^{\frac{21+2\omega}{15+\omega}})$. *Using the current best bound on fast rectangular matrix multiplication the running time improves to* $O(n^{1.4805})$.

Proof. Via a binary search, Sandlund and Xu [21] showed that the Batch Range Mode problem can be reduced to finding the frequencies of the most frequent elements for all queries (with an $\tilde{O}(1)$ factor overhead), which is in turn reduced to Monotone Min-Plus Product in [21, Theorem 6.1], leading to a deterministic

$$\tilde{O}(n^{\min_{0 \le \tau \le 1} \max\{m(1-\tau, 1-\tau, 1-\tau, \tau), 1+\tau\}})$$

time algorithm for Batch Range Mode.

Expanding the expression using Theorem 16 and (3), we have

$$\begin{aligned}
&\min_{0 \le \tau \le 1} \max\{m(1-\tau, 1-\tau, 1-\tau, \tau), 1+\tau\} \\
&= \min_{0 \le \tau \le 1} \max\{(1-\tau)m(1, 1, 1, \frac{\tau}{1-\tau}), 1+\tau\} \\
&\le \min_{0 \le \tau \le 1} \max\{(1-\tau) \min_{0 \le \theta \le \frac{\tau}{1-\tau}} \max\{2 + \frac{\tau}{1-\tau} - \theta, \frac{1}{2}(3 + g(1, 1, 1, \theta))\}, 1+\tau\} \\
&= \min_{\substack{0 \le \tau \le 1 \\ 0 \le \theta \le \frac{\tau}{1-\tau}}} \max\{\tau + (1-\tau)(2-\theta), \frac{1-\tau}{2}(3 + g(1, 1, 1, \theta)), 1+\tau\} \\
&\le \min_{\substack{0 \le \tau \le 1 \\ 0 \le \theta \le \frac{\tau}{1-\tau} \\ 0 \le \delta \le 1}} \max\{\tau + (1-\tau)(2-\theta), \frac{1-\tau}{2}(3 + \max\{\omega(1, 1, 2-\delta) + \theta, 2 + \delta\}), 1+\tau\}.
\end{aligned} \tag{7}$$

By using $\omega(1, 1, 2-\delta) \le 1 - \delta + \omega$ and taking $\delta = \frac{4\omega-3}{9}, \theta = \frac{3-\omega}{9}, \tau = \frac{6+\omega}{15+\omega}$, we can upper bound (7) by $\frac{21+2\omega}{15+\omega}$. Using the upper bounds in [18], we find that when $\tau = 0.4804$, $\theta = 0.0754$, $\delta = 0.6984$, expression (7) takes value ≤ 1.4805. ◀

5.2 Dynamic Range Mode

▶ **Theorem 5.** *The Dynamic Range Mode problem can be solved deterministically in $\tilde{O}(n^{\frac{\omega+9}{\omega+15}})$ worst-case time per query with $\tilde{O}(n^{\frac{3\omega+39}{2\omega+30}})$ space. Using the current best bound on fast rectangular matrix multiplication improves the running time to $O(n^{0.6524})$ and the space complexity to $O(n^{1.3262})$.*

Our strategy is to improve [21, Lemma 11]. The following Min-Plus-Query-Witness problem defined in [21, Problem 7] plays a key role in the algorithm for Dynamic Range Mode.

▶ **Problem 31** (Min-Plus-Query-Witness problem)**.** *We are given two matrices A and B and are able to perform preprocessing before the first query. For each query, we are given two indices i, j and a set S of indices, and we must output an index $k^* \notin S$ such that*

$$A_{i,k^*} + B_{k^*,j} = \min_{k \notin S}\{A_{i,k} + B_{k,j}\}.$$

We recall the following lemma from [21].

▶ **Lemma 32** ([21, Lemma 9])**.** *Let β and θ be non-negative real numbers. The Min-Plus-Query-Witness problem where A is an $n \times n^\beta$ matrix whose entries are in $\{-n^\theta, \ldots, n^\theta\} \cup \{\infty\}$ and B is an $n^\beta \times n$ matrix can be solved with*

- *$\tilde{O}(n^{\theta+\omega(1,\beta,1)+\beta-\sigma})$ preprocessing time,*
- *$\tilde{O}(|S| + n^\sigma)$ worst-case time per query,*
- *and $\tilde{O}(n^{\max\{2+\beta+\theta, 1+2\beta\}-\sigma})$ space,*

for every $0 \le \sigma \le \beta$.

The following is our key lemma for the dynamic range mode algorithm.

▶ **Lemma 33.** *The Min-Plus-Query-Witness problem where A is an $n \times n^\beta$ matrix, B is an $n^\beta \times n$ monotone matrix with total range $O(n^{\beta+\eta})$, and the size of S for each query is $O(n^\lambda)$, can be solved with*

- *$\tilde{O}(n^{\max\{1+\beta+\eta-\theta, \rho+\theta+\omega(1,\beta,1)+\beta-\sigma, 2+\beta-\rho\}})$ preprocessing time,*
- *$\tilde{O}(n^\lambda + n^{\rho+\sigma})$ worst-case time per query,*
- *and $\tilde{O}(n^{\max\{1+\beta+\eta-\theta, \rho+\max\{2+\beta+\theta, 1+2\beta\}-\sigma, 2+\beta-\rho\}})$ space,*

for any constants $0 \le \theta \le \eta$, $\rho \ge 0$ and $0 \le \sigma \le \beta$.

Proof. Preprocessing is in three steps, corresponding to the three phases of the algorithm for Theorem 16.

Preprocessing Step 1. This step is slightly more complicated than Phase 1 of Theorem 16. Let $0 \le \theta \le \eta$ be a parameter to be chosen. Let $W = \lfloor n^\theta \rfloor$. Define two matrices $\tilde{A}$ and $\tilde{B}$ as $\tilde{A}_{i,k} = \lfloor \frac{A_{i,k}}{W} \rfloor$ and $\tilde{B}_{k,j} = \lfloor \frac{B_{k,j}}{W} \rfloor$.

We iterate through $j \in [n]$, and in each iteration j, maintain for each $i \in [n]$ the set $L_{i,j} := \{(\tilde{A}_{i,k} + \tilde{B}_{k,j}, k) : k \in [n^\beta]\}$ using a *persistent* balanced BST.

Using the maintained information, we can compute a matrix $\tilde{C}'$, where each entry $\tilde{C}'_{i,j}$ is defined as W times the $(n^\lambda + 1)$-th smallest element in $\{\tilde{A}_{i,k} + \tilde{B}_{k,j} : k \in [n^\beta]\}$. Furthermore, for each (i, j) and for any t, we can enumerate the t smallest elements in $L_{i,j}$ in $\tilde{O}(t)$ time.

Time complexity and space complexity of this step are both $\tilde{O}(n^{\max\{2, 1+\beta+\eta-\theta\}})$.

Preprocessing Step 2. As in Phase 2 of Theorem 16, we sample $j^r \in [n]$ for $r \in [\tilde{O}(n^\rho)]$ for some parameter ρ to be chosen, and compute matrices A^r and B^r as

$$A^r_{i,k} = \begin{cases} A_{i,k} + B_{k,j^r} - \tilde{C}'_{i,j^r} & \text{if } |A_{i,k} + B_{k,j^r} - \tilde{C}'_{i,j^r}| \le 3W \text{ and } A^{r'}_{i,k} = \infty \forall r' < r, \\ \infty & \text{otherwise,} \end{cases}$$

$$B^r_{k,j} = \begin{cases} B_{k,j} - B_{k,j^r} & \text{if } B_{k,j^r} \neq \infty, \\ 0 & \text{otherwise.} \end{cases}$$

Then for each r, we apply the data structure in Lemma 32 to A^r and B^r. Running time for this step is $\tilde{O}(n^{\rho+\theta+\omega(1,\beta,1)+\beta-\sigma})$. Space complexity for this step is $\tilde{O}(n^{\rho+\max\{2+\beta+\theta,1+2\beta\}-\sigma})$.

We note that this part can be derandomized as well.

Preprocessing Step 3. For a pair (i,k) if $A^r_{i,k} \neq \infty$ for some r, we call (i,k) *covered*; otherwise we call it *uncovered*. We call a triple (i,k,j) *almost relevant* if $0 \le \tilde{A}_{i,k} + \tilde{B}_{k,j} - \frac{1}{W}\tilde{C}'_{i,j} \le 1$. Note that for such triples, we must have $|A_{i,k} + B_{k,j} - \tilde{C}'_{i,j}| \le 3W$. So the number of uncovered and almost relevant triples is $\tilde{O}(n^{2+\beta-\rho})$.

We run an algorithm similar to Lemma 21 to enumerate all uncovered and almost relevant triples (i,k,j). Then for each $i,j \in [n]$, we use a balanced BST to store the set $T_{i,j} := \{(A_{i,k} + B_{k,j}, k) : (i,k,j) \text{ is uncovered and almost relevant}\}$.

Running time for this step is $\tilde{O}(n^{\max\{2,1+\beta+\eta-\theta,2+\beta-\rho\}})$. Space complexity for this step is $\tilde{O}(n^{2+\beta-\rho})$.

Query. Now let us describe how to handle a query. Let (S,i,j) be a query, and k^* be an optimal index, i.e., $A_{i,k^*} + B_{k^*,j} = \min_{k \notin S}\{A_{i,k} + B_{k,j}\}$. There are three cases.

Case 1: (i,k^*) is covered. For each r, we query the data structure (Lemma 32) for A^r and B^r with (S_r,i,j) where $S_r = S \cap \{k : A^r_{i,k} \neq \infty\}$. Because finite entries of A^r are disjoint for different r, we have $\sum_r |S_r| = |S|$. So the total query time for this case is $\tilde{O}(|S| + n^{\rho+\sigma})$.

Case 2: (i,k^*) is uncovered and almost relevant. In this case k^* must be among the $(|S|+1)$ smallest elements in $T_{i,j}$. We deal with this case by enumerating the $(|S|+1)$ smallest elements in $T_{i,j}$. The query time for this case is $\tilde{O}(|S|)$.

Case 3: (i,k^*) is uncovered and not almost relevant. Note that by definition of $\tilde{C}'_{i,j}$, in this case we must have $\tilde{A}_{i,k^*} + \tilde{B}_{k^*,j} - \frac{1}{W}\tilde{C}'_{i,j} \le -1$. Therefore k^* must be among the $(n^\lambda + 1)$ smallest elements in $L_{i,j}$. We deal with this case by enumerating the $(n^\lambda+1)$ smallest elements in $L_{i,j}$. The query time for this case is $\tilde{O}(n^\lambda)$.

Summary. Total preprocessing time is $\tilde{O}(n^{\max\{1+\beta+\eta-\theta,\rho+\theta+\omega(1,\beta,1)+\beta-\sigma,2+\beta-\rho\}})$. Space complexity is $\tilde{O}(n^{\max\{1+\beta+\eta-\theta,\rho+\max\{2+\beta+\theta,1+2\beta\}-\sigma,2+\beta-\rho\}})$. Each query costs $\tilde{O}(n^\lambda + n^{\rho+\sigma})$ time. ◀

Proof of Theorem 5. The algorithm is exactly the same as in [21], except for replacing [21, Lemma 11] with Lemma 33. We skip most of the algorithm description and focus on analyzing the time and space complexity.

For the algorithm we need to choose three constants $t_1, t_2, t_3 \in [0,1]$. The algorithm has three parts.

Part 1: Infrequent values. In the first part, we handle values that appear at most n^{1-t_1} times. By maintaining n^{1-t_1} balanced BSTs, this part can be done in $\tilde{O}(n^{2-2t_1})$ time per operation. Space complexity is $\tilde{O}(n^{2-t_1})$.

Part 2: Newly modified values. We maintain a data structure holding frequent values that rebuilds every n^{t_2} operations. The data structure is discussed in Part 3. In Part 2, we deal with values that have been modified in the last n^{t_2} operations. This part can be done in $\tilde{O}(n^{t_2})$ time and $\tilde{O}(n^{t_2})$ space.

Part 3: Data structure. In Part 3, we build the data structure. For this part, we call Lemma 33 where A is an $n^{1-t_3} \times n^{t_1}$ matrix, B is an $n^{t_1} \times n^{1-t_3}$ monotone matrix with total range $O(n)$, and in the queries we have $|S| = O(n^{t_2})$. The dimensions $\alpha = \gamma = 1 - t_3$ come from choosing n^{1-t_3} evenly spaced points in $[n]$, and $\beta = t_1$ comes from the $O(n^{t_1})$ values which appear more than n^{1-t_1} times. In a query we pick the maximum interval whose endpoints are chosen points inside the query interval, and S is the set of values modified in the last n^{t_2} operations.

Rebuilding costs $\tilde{O}(n^{(1-t_3)\max\{1+\beta+\eta-\theta,\rho+\theta+\omega(1,\beta,1)+\beta-\sigma,2+\beta-\rho\}-t_2})$ time per operation[6], where $\beta = \frac{t_1}{1-t_3}$, $\eta = \frac{1-t_1}{1-t_3}$, and $0 \le \theta \le \eta$, $\rho \ge 0$, $0 \le \sigma \le \beta$ are constants to be chosen. One query in Lemma 33 costs $\tilde{O}(n^{t_2} + n^{(1-t_3)(\rho+\sigma)})$ time. Besides the above, we also need $\tilde{O}(n^{t_3})$ time per operation to deal with elements not covered by the maximum interval of chosen points inside the query interval.

Space cost of the data structure is $\tilde{O}(n^{(1-t_3)\max\{\rho+\max\{2+\beta+\theta,1+2\beta\}-\sigma,2+\beta-\rho\}})$.

Summary. Running time per operation is

$$\tilde{O}(n^{2-2t_1} + n^{t_2} + n^{(1-t_3)\max\{1+\beta+\eta-\theta,\rho+\theta+\omega(1,\beta,1)+\beta-\sigma,2+\beta-\rho\}-t_2} + n^{(1-t_3)(\rho+\sigma)} + n^{t_3}),$$

subject to $t_1, t_2, t_3 \in [0,1]$, $\beta = \frac{t_1}{1-t_3}$, $\eta = \frac{1-t_1}{1-t_3}$, $0 \le \theta \le \eta$, $\rho \ge 0$, $0 \le \sigma \le \beta$.

Space cost is $\tilde{O}(n^{2-t_1} + n^{t_2} + n^{(1-t_3)\max\{\rho+\max\{2+\beta+\theta,1+2\beta\}-\sigma,2+\beta-\rho\}})$.

As an observation, in the optimum case we have $\beta \ge 1$, so we may use $\omega(1,\beta,1) \le \omega+\beta-1$. As a result, we can set $t_1 = \frac{\omega+21}{2\omega+30}, t_2 = t_3 = \frac{\omega+9}{\omega+15}, \theta = \frac{3-\omega}{6}, \rho = \frac{3-\omega}{4}$ and $\sigma = \frac{5\omega+9}{12}$ to upper bound the running time by $\tilde{O}(n^{\frac{\omega+9}{\omega+15}})$. The space complexity is dominated by Part 1, which is $\tilde{O}(n^{2-t_1}) = \tilde{O}(n^{\frac{3\omega+39}{2\omega+30}})$.

Taking $t_1 = 0.67385$, $t_2 = t_3 = 0.6523$, $\theta = 0.1239$, $\rho = 0.1859$, $\sigma = 1.6902$ and using fast rectangular matrix multiplication we get $O(n^{0.6524})$ running time per operation. The space complexity is again dominated by Part 1, which is $\tilde{O}(n^{2-t_1}) = O(n^{1.3262})$. ◀

References

1 Josh Alman and Virginia Vassilevska Williams. A refined laser method and faster matrix multiplication. In *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 522–539, 2021. doi:10.1137/1.9781611976465.32.

2 Noga Alon, Shiri Chechik, and Sarel Cohen. Deterministic combinatorial replacement paths and distance sensitivity oracles. In *46th International Colloquium on Automata, Languages, and Programming, ICALP 2019, July 9-12, 2019, Patras, Greece*, volume 132 of *LIPIcs*, pages 12:1–12:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019. doi:10.4230/LIPIcs.ICALP.2019.12.

[6] The rebuilding is done every n^{t_2} operations, so a priori we get an amortized time complexity. However, as pointed out in [21], we can use the global rebuilding technique of [20] to achieve a worst-case time bound.

3 Noga Alon, Zvi Galil, and Oded Margalit. On the exponent of the all pairs shortest path problem. *J. Comput. Syst. Sci.*, 54(2):255–262, 1997.

4 Karl Bringmann, Fabrizio Grandoni, Barna Saha, and Virginia Vassilevska Williams. Truly subcubic algorithms for language edit distance and RNA folding via fast bounded-difference min-plus product. *SIAM Journal on Computing*, 48(2):481–512, 2019.

5 Timothy M. Chan. More algorithms for all-pairs shortest paths in weighted graphs. *SIAM Journal on Computing*, 39(5):2075–2089, 2010.

6 Timothy M. Chan, Stephane Durocher, Kasper Green Larsen, Jason Morrison, and Bryan T. Wilkinson. Linear-space data structures for range mode query in arrays. *Theory of Computing Systems*, 55:719–741, 2014.

7 Timothy M. Chan, Virginia Vassilevska Williams, and Yinzhan Xu. Algorithms, reductions and equivalences for small weight variants of all-pairs shortest paths. In *48th International Colloquium on Automata, Languages, and Programming, ICALP 2021*, page to appear. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021.

8 Shiri Chechik and Sarel Cohen. Near optimal algorithms for the single source replacement paths problem. In *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2019, San Diego, California, USA, January 6-9, 2019*, pages 2090–2109. SIAM, 2019. doi:10.1137/1.9781611975482.126.

9 Shiri Chechik and Ofer Magen. Near optimal algorithm for the directed single source replacement paths problem. In *47th International Colloquium on Automata, Languages, and Programming, ICALP 2020, July 8-11, 2020, Saarbrücken, Germany (Virtual Conference)*, volume 168 of *LIPIcs*, pages 81:1–81:17. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPIcs.ICALP.2020.81.

10 Shiri Chechik and Moran Nechushtan. Simplifying and unifying replacement paths algorithms in weighted directed graphs. In *47th International Colloquium on Automata, Languages, and Programming, ICALP 2020, July 8-11, 2020, Saarbrücken, Germany (Virtual Conference)*, volume 168 of *LIPIcs*, pages 29:1–29:12. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPIcs.ICALP.2020.29.

11 Hicham El-Zein, Meng He, J. Ian Munro, and Bryce Sandlund. Improved time and space bounds for dynamic range mode. In *Proceedings of the 26th Annual European Symposium on Algorithms*, pages 25:1–25:13, 2018.

12 Michael J. Fischer and Albert R. Meyer. Boolean matrix multiplication and transitive closure. In *12th Annual Symposium on Switching and Automata Theory, SWAT 1971*, pages 129–131. IEEE, 1971.

13 Fabrizio Grandoni, Giuseppe F. Italiano, Aleksander Łukasiewicz, Nikos Parotsidis, and Przemysław Uznański. All-Pairs LCA in DAGs: Breaking through the $O(n^{2.5})$ barrier. In *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 273–289, 2021. doi:10.1137/1.9781611976465.18.

14 Fabrizio Grandoni and Virginia Vassilevska Williams. Faster replacement paths and distance sensitivity oracles. *ACM Transactions on Algorithms*, 16(1):1–25, 2019.

15 Xiaohan Huang and Victor Y. Pan. Fast rectangular matrix multiplication and applications. *Journal of complexity*, 14(2):257–299, 1998.

16 Donald B. Johnson. Efficient algorithms for shortest paths in sparse networks. *J. ACM*, 24(1):1–13, 1977. doi:10.1145/321992.321993.

17 François Le Gall. Powers of tensors and fast matrix multiplication. In *Proceedings of the 39th International Symposium on Symbolic and Algebraic Computation*, ISSAC '14, pages 296–303, New York, NY, USA, 2014. ACM. doi:10.1145/2608628.2608664.

18 François Le Gall and Florent Urrutia. Improved rectangular matrix multiplication using powers of the Coppersmith-Winograd tensor. In *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1029–1046. SIAM, 2018.

19 Andrea Lincoln, Adam Polak, and Virginia Vassilevska Williams. Monochromatic triangles, intermediate matrix products, and convolutions. In *11th Innovations in Theoretical Computer Science Conference, ITCS 2020, January 12-14, 2020, Seattle, Washington, USA*, volume 151 of *LIPIcs*, pages 53:1–53:18. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPIcs.ITCS.2020.53.

20 Mark H. Overmars. *The design of dynamic data structures*, volume 156. Springer Science & Business Media, 1987.

21 Bryce Sandlund and Yinzhan Xu. Faster Dynamic Range Mode. In *47th International Colloquium on Automata, Languages, and Programming (ICALP 2020)*, volume 168 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 94:1–94:14, Dagstuhl, Germany, 2020. Schloss Dagstuhl–Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.ICALP.2020.94.

22 Virginia Vassilevska Williams. Multiplying matrices faster than Coppersmith-Winograd. In *Proceedings of the Forty-fourth Annual ACM Symposium on Theory of Computing*, STOC '12, pages 887–898, 2012. doi:10.1145/2213977.2214056.

23 Virginia Vassilevska Williams. On some fine-grained questions in algorithms and complexity. In *Proceedings of the ICM*, volume 3, pages 3431–3472. World Scientific, 2018.

24 Virginia Vassilevska Williams and Ryan Williams. Subcubic equivalences between path, matrix, and triangle problems. *J. ACM*, 65(5):1–38, 2018.

25 Virginia Vassilevska Williams and Yinzhan Xu. Truly subcubic min-plus product for less structured matrices, with applications. In *Proceedings of the Fourteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 12–29. SIAM, 2020.

26 Ryan Williams. Faster all-pairs shortest paths via circuit complexity. In *Proceedings of the forty-sixth annual ACM symposium on Theory of computing*, pages 664–673. ACM, 2014.

27 Virginia Vassilevska Williams and Yinzhan Xu. Monochromatic triangles, triangle listing and apsp. In *2020 IEEE 61st Annual Symposium on Foundations of Computer Science (FOCS)*, pages 786–797, 2020. doi:10.1109/FOCS46700.2020.00078.

28 Raphael Yuster. Efficient algorithms on sets of permutations, dominance, and real-weighted APSP. In Claire Mathieu, editor, *Proceedings of the Twentieth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2009, New York, NY, USA, January 4-6, 2009*, pages 950–957. SIAM, 2009.

29 Raphael Yuster and Uri Zwick. Answering distance queries in directed graphs using fast matrix multiplication. In *Proceedings of the 46th Annual IEEE Symposium on Foundations of Computer Science*, FOCS '05, pages 389–396, USA, 2005. IEEE Computer Society. doi:10.1109/SFCS.2005.20.

30 Gideon Yuval. An algorithm for finding all shortest paths using $N^{2.81}$ infinite-precision multiplications. *Information Processing Letters*, 4(6):155–156, 1976. doi:10.1016/0020-0190(76)90085-5.

31 Uri Zwick. All pairs shortest paths using bridging sets and rectangular matrix multiplication. *J. ACM*, 49(3):289–317, 2002. doi:10.1145/567112.567114.

Constructing a Distance Sensitivity Oracle in $O(n^{2.5794}M)$ Time

Yong Gu
Institute for Interdisciplinary Information Sciences, Tsinghua University, Beijing, China

Hanlin Ren
Institute for Interdisciplinary Information Sciences, Tsinghua University, Beijing, China

Abstract

We continue the study of *distance sensitivity oracles* (DSOs). Given a directed graph G with n vertices and edge weights in $\{1, 2, \ldots, M\}$, we want to build a data structure such that given any source vertex u, any target vertex v, and any failure f (which is either a vertex or an edge), it outputs the length of the shortest path from u to v not going through f. Our main result is a DSO with preprocessing time $O(n^{2.5794}M)$ and constant query time. Previously, the best preprocessing time of DSOs for directed graphs is $O(n^{2.7233}M)$, and even in the easier case of undirected graphs, the best preprocessing time is $O(n^{2.6865}M)$ [Ren, ESA 2020]. One drawback of our DSOs, though, is that it only supports distance queries but not path queries.

Our main technical ingredient is an algorithm that computes the inverse of a degree-d polynomial matrix (i.e. a matrix whose entries are degree-d univariate polynomials) modulo x^r. The algorithm is adapted from [Zhou, Labahn and Storjohann, *Journal of Complexity*, 2015], and we replace some of its intermediate steps with faster rectangular matrix multiplication algorithms.

We also show how to compute *unique* shortest paths in a directed graph with edge weights in $\{1, 2, \ldots, M\}$, in $O(n^{2.5286}M)$ time. This algorithm is crucial in the preprocessing algorithm of our DSO. Our solution improves the $O(n^{2.6865}M)$ time bound in [Ren, ESA 2020], and matches the current best time bound for computing all-pairs shortest paths.

2012 ACM Subject Classification Theory of computation → Shortest paths; Theory of computation → Design and analysis of algorithms

Keywords and phrases graph theory, shortest paths, distance sensitivity oracles

Digital Object Identifier 10.4230/LIPIcs.ICALP.2021.76

Category Track A: Algorithms, Complexity and Games

Related Version *Full Version*: https://arxiv.org/abs/2102.08569

Acknowledgements We would like to thank Ran Duan and Tianyi Zhang for helpful discussions during the initial stage of this research. We are grateful to anonymous reviewers for helpful comments. We would also like to thank an anonymous reviewer for suggesting the title of Section 5, and another anonymous reviewer for pointing out a subtle issue regarding the invertibility of polynomial matrices (and fixing the issue).

1 Introduction

In this paper, we consider the problem of constructing a *distance sensitivity oracle* (DSO). A DSO is a data structure that preprocesses a directed graph $G = (V, E)$ with n vertices and m edges, and supports queries of the following form: Given a source vertex u, a target vertex v, and a failure f (which can be either a vertex or an edge), output the length of the shortest path from u to v that does not go through f.

One motivation for constructing DSOs is the fact that real-life networks often suffer from failures. Consider a communication network among n servers. When a server u wants to send a message to another server v, the most efficient way would be to send the message along the

shortest path from u to v. However, if a failure happens in a server or a link between two servers, we would need to recompute the shortest path with the failure taken into account. It may be too slow to compute the shortest path from scratch each time a failure happens. A better solution is to construct a DSO for the communication network, and invoke the query algorithm of the DSO whenever a failure happens.

1.1 Related Work

The problem of constructing DSOs has received a lot of attention in the literature. A naïve solution is to precompute the answers for every possible query (u, v, f), but it requires $\Omega(n^2 m)$ space to store this DSO. Demetrescu et al. [11] constructed a DSO with $O(n^2 \log n)$ space that answers a query in constant time. However, the preprocessing time of the DSO in [11] is $O(mn^2 + n^3 \log n)$, which is inefficient for large networks. Subsequently, Bernstein and Karger improved the preprocessing time to $\tilde{O}(n^2\sqrt{m})$ [5], and finally $\tilde{O}(mn)$ [6].[1] The preprocessing time $\tilde{O}(mn)$ matches the current best time bound for the easier problem of computing *all-pairs shortest paths* (APSP), and it is conjectured that APSP requires $mn^{1-o(1)}$ time [20]. In this sense, the $\tilde{O}(mn)$ time bound of [6] is optimal. Duan and Zhang [14] improved the space complexity of the DSO to $O(n^2)$, eliminating the last $\log n$ factor, while preserving constant query time and $\tilde{O}(mn)$ preprocessing time.

However, for dense graphs (i.e. $m = \Theta(n^2)$) with edge weights in $[-M, M]$, it is possible to compute APSP in time faster than $\tilde{O}(mn) = \tilde{O}(n^3)$. The best APSP algorithm for undirected graphs runs in $\tilde{O}(n^\omega M)$ time [27, 30], and the best APSP algorithm for directed graphs runs in $O(n^{2.5286}M)$ time [4, 42]. (Here $\omega < 2.3728596$ is the exponent of matrix multiplication [2, 9, 19, 32, 37].) Therefore, it is natural to ask whether one can beat $\tilde{O}(n^3)$ preprocessing time for DSOs in this regime.

The answer turned out to be *yes*. Weimann and Yuster [36] showed that for any constant $0 < \alpha < 1$, there is a DSO with $\tilde{O}(n^{1-\alpha+\omega}M)$ preprocessing time and $\tilde{O}(n^{1+\alpha})$ query time. Subsequently, Grandoni and Williams [16] showed that for any constant $0 < \alpha < 1$, there is a DSO with $\tilde{O}(n^{\omega+1/2}M + n^{\omega+\alpha(4-\omega)}M)$ preprocessing time and $\tilde{O}(n^{1-\alpha})$ query time. Recently, Chechik and Cohen [8] constructed the first DSO that achieves both sub-cubic ($O(n^{2.873}M)$) preprocessing time and poly-logarithmic query time simultaneously. For the case that edge weights are positive, Ren [22] improved the previous results by presenting a much simpler DSO with $\tilde{O}(n^{2.7233}M)$ preprocessing time and constant query time.

Note that most DSOs mentioned above are randomized. Recently, there are also some efforts on derandomizing these DSOs, see e.g. [3, 23].

1.2 Our Results

Our main result is an improved DSO for directed graphs with integer edge weights in $[1, M]$. In particular, our DSO has preprocessing time $O(n^{2.5794}M)$ and constant query time.

▶ **Theorem 1.1** (Main). *Given as input a directed graph $G = (V, E)$ with edge weights in $\{1, 2, \ldots, M\}$, we can construct a DSO with $O(n^{2.5794}M)$ preprocessing time and constant query time. With high probability over the randomized preprocessing algorithm, the DSO answers every possible query correctly.*

[1] $\tilde{O}$ hides polylog(n) factors.

▶ Remark 1.2. Our preprocessing algorithm uses fast *rectangular* matrix multiplication algorithms. To express our time bound as a function of ω, we could also simulate rectangular matrix multiplications by square matrix multiplications, e.g. multiply an $n \times m$ matrix and an $m \times n$ matrix by $\lceil m/n \rceil$ square matrix multiplications of dimension n. In this case, the preprocessing time becomes $\tilde{O}(n^{2+1/(4-\omega)}M) < O(n^{2.6146}M)$.

▶ Remark 1.3 (Comparison with Prior Works). The biggest advantage of our DSO is, of course, its fast preprocessing algorithm. In fact, the preprocessing time bound is only an $O(n^{0.051})$ factor away from the current best time bound for APSP. Our DSO is also the first one to break a barrier of $\tilde{\Omega}(n^{8/3})$ preprocessing time, while keeping constant query time.[2] However, our DSO has two drawbacks. First, it can only return the length of the shortest path. It does not suggest an efficient way to produce this path. Second, it does not support negative edge weights.

We highlight two technical ingredients that are crucial for the preprocessing algorithm of our DSO.

Inverting a polynomial matrix modulo x^r

Let r be an integer parameter, and $\mathbf{F}$ be a polynomial matrix of degree d (i.e. each entry of $\mathbf{F}$ is a degree-d polynomial over some formal variable x) that is invertible. We show how to compute $\mathbf{F}^{-1} \bmod x^r$ in time

$$\tilde{O}(dn^\omega) + (r^2/d) \cdot \mathsf{MM}(n, nd/r, nd/r) \cdot n^{o(1)}.$$

(That is, we only preserve the monomials in $\mathbf{F}^{-1}$ with degrees at most $r-1$.) Here, $\mathsf{MM}(n_1, n_2, n_3)$ is the time complexity of multiplying an $n_1 \times n_2$ matrix and an $n_2 \times n_3$ matrix.

It is shown in [40] that we can compute the full $\mathbf{F}^{-1}$ (instead of $\mathbf{F}^{-1} \bmod x^r$) in $\tilde{O}(n^3 d)$ time. We examine their algorithm carefully, and adapt it to our case where we only want to compute $\mathbf{F}^{-1} \bmod x^r$. We modulo each polynomial in the intermediate steps of the algorithm by x^r, and use fast rectangular matrix multiplication to speed up the algorithm.

▶ **Theorem 1.4.** *Let r be an integer, $\mathbb{F}$ be a finite field. Let $\mathbf{F} \in \mathbb{F}[x]^{n \times n}$ be an $n \times n$ matrix over the ring of univariate polynomials $\mathbb{F}[x]$, and let $d \geq 1$ be an upper bound on the degrees of entries of $\mathbf{F}$. If $\mathbf{F}$ is invertible over $(\mathbb{F}[x]/\langle x^r \rangle)^{n \times n}$, the number of field operations to compute $\mathbf{F}^{-1} \bmod x^r$ is at most*

$$\tilde{O}(dn^\omega) + (r^2/d) \cdot \mathsf{MM}(n, nd/r, nd/r) \cdot n^{o(1)}.$$

▶ Remark 1.5. The idea of using polynomial matrices to capture distances is a common technique in graph algorithms. It has found many applications in static algorithms [24], fault-tolerant algorithms [35], and dynamic algorithms [25, 33, 34].

[2] There are three previous DSOs with both sub-cubic preprocessing time and constant query time: [16], [8], and [22]. (The query time of the first two DSOs can be brought down to constant using Observation 2.1 of [22]. In the case of [16], this increases the preprocessing time by an additive factor of $\tilde{O}(n^{3-\alpha})$.) Even when $\omega = 2$, the preprocessing time bounds of these DSOs are $\tilde{O}(n^{8/3})$ (setting α appropriately), $\tilde{O}(n^{14/5})$, and $\tilde{O}(n^{8/3})$ respectively.

Computing consistent shortest path trees

Our DSO needs to invoke [22, Observation 2.1] (see also [6]), which needs a *consistent* set of (incoming and outgoing) shortest path trees rooted at each vertex. Here, by *consistent*, we mean that for every pair of vertices u, v and any two shortest path trees T_1 and T_2 (from the $2n$ trees; recall they are *directed* rooted trees), if u can reach v in both T_1 and T_2, then the $u \rightsquigarrow v$ paths in T_1 and T_2 are the same path. In other words, we want to specify a *unique* shortest path between each pair of vertices, such that for every vertex v, the shortest paths starting from v (or ending at v, respectively) form a tree.

Note that this problem is quite nontrivial in small-weighted graphs. There may be many shortest paths between two vertices, and it is not obvious how to pick one shortest path for each vertex pair, while guaranteeing consistency. Also, we cannot randomly perturb the edge weights by small values, as that would break the property that edge weights are small integers. It is also unclear how to construct such a set of shortest path trees from the APSP algorithm in [42]. Previously, combining ideas in [10, Section 3.4] and an algorithm in [13], [22] showed how to compute such shortest path trees in $\tilde{O}(n^{(3+\omega)/2}M) \leq O(n^{2.6865}M)$ time; unfortunately, this time bound is worse than our claimed time bound $O(n^{2.5794}M)$ in Theorem 1.1.

In this paper, we show how to construct consistent shortest paths trees in $O(n^{2.5286}M)$ time, matching the currently best time bound for APSP [42]. Below is an informal statement, see Theorem 5.1 for the precise version.

▶ **Theorem 1.6** (Informal Version). *Given a directed graph $G = (V, E)$ with edge weights in $\{1, 2, \ldots, M\}$, we can compute a set of incoming and outgoing shortest path trees rooted at each vertex that are consistent, in $O(n^{2.5286}M)$ time.*

1.3 Warm-Up: DSO in $\tilde{O}(n^{(3+\omega)/2}M)$ Preprocessing Time

Actually, the ideas in [35] of maintaining the *adjoint* of the *symbolic adjacency matrix* (see Section 3), together with ideas in [22], already give us a DSO with $\tilde{O}(n^{(3+\omega)/2}M)$ preprocessing time and constant query time. As a warm-up, we briefly describe this DSO before we proceed into the details of Theorem 1.1.

An *r-truncated DSO* [22] is a DSO that only needs to be correct for the queries (u, v, f) whose answer (i.e. length of the corresponding shortest path) is at most r. If the answer is greater than r, it should return r instead. In what follows, we will describe how to construct an r-truncated DSO in $\tilde{O}(rn^{\omega})$ preprocessing time and $\tilde{O}(r)$ query time. Using techniques in [22] (see also Section 3.3), this implies a DSO with $\tilde{O}(n^{(3+\omega)/2}M)$ preprocessing time and constant query time.

Let $\mathbb{F}$ be a sufficiently large finite field, and $\mathbf{A}$ be the following matrix. For every vertices u, v, if there is an edge from u to v with weight l, then let $\mathbf{A}_{u,v} = a_{u,v}x^l$, where $a_{u,v}$ is a random element in $\mathbb{F}$, and x is an indeterminate. Furthermore, for every vertex v, let $\mathbf{A}_{v,v} = 1$. It is well-known [24] that with high probability over the choices of $a_{u,v}$, the *adjoint* matrix of $\mathbf{A}$ encodes the shortest path information of the input graph, as follows. Let $\mathrm{adj}(\mathbf{A})$ be the adjoint matrix of $\mathbf{A}$, and u, v be two vertices, then the lowest degree of $\mathrm{adj}(\mathbf{A})_{u,v}$ is exactly the distance from u to v. For example, if $\mathrm{adj}(\mathbf{A})_{u,v} = 7x^8 + 6x^5 - 9x^4$, then the distance from u to v is 4.

A big advantage of the adjoint matrix, exploited in [35] and also this work, is that it is easy to perform *low-rank* updates, by the Sherman-Morrison-Woodbury formula (see Theorem 3.2). Given a matrix $\mathbf{A}$, its adjoint $\mathrm{adj}(\mathbf{A})$, and a low-rank matrix $\mathbf{B}$, we can compute a specific element of $\mathrm{adj}(\mathbf{A}+\mathbf{B})_{u,v}$, in time much faster than brute force. Therefore,

we answer a query (u, v, f) as follows: We first express the failure as a *rank-one* matrix $\mathbf{F}$, such that $\mathbf{A} + \mathbf{F}$ is the matrix corresponding to the graph with f removed. Then we can compute $\text{adj}(\mathbf{A} + \mathbf{F})_{u,v}$ quickly. Given this element (a polynomial over $\mathbb{F}$), we can easily compute the answer to the query.

What is the time complexity of this DSO? Recall that we only want to construct an r-truncated DSO, so we can modulo every entry in the process of computing $\text{adj}(\mathbf{A})$ by the polynomial x^r. Every arithmetic operation in the commutative ring $\mathbb{F}[x]/x^r$ only takes $\tilde{O}(r)$ time. Computing the adjoint of a matrix reduces to inverting that matrix, which takes $\tilde{O}(n^\omega)$ arithmetic operations [7]. Therefore it takes $\tilde{O}(rn^\omega)$ time to compute $\text{adj}(\mathbf{A}) \bmod x^r$. A close inspection of the Sherman-Morrison-Woodbury formula shows that each query can be completed in $O(1)$ arithmetic operations, i.e. $\tilde{O}(r)$ time.

The $\tilde{O}(rn^\omega)$-time algorithm for inverting a polynomial matrix modulo x^r is not optimal; the time bound in Theorem 1.4 is better. In Section 4, we use fast rectangular matrix multiplication algorithms to speed up the algorithm in [40], obtaining a faster algorithm for inverting polynomial matrices modulo x^r.

2 Preliminaries

In this paper, we say an event happens *with high probability* (w.h.p.) if it happens with probability at least $1 - 1/n^c$, for a constant c that can be made arbitrarily large. Our DSOs (or r-truncated DSOs) will have a randomized preprocessing algorithm and a deterministic query algorithm. We say a DSO is *correct with high probability* if w.h.p. over its (randomized) preprocessing algorithm, it answers every possible query (u, v, f) correctly.

Notation

We use the following notation in [12, 22].

- Let p be a path, we use $|p|$ to denote the number of edges in p, and use $\|p\|$ to denote the length of p (i.e. total weight of edges in p).
- Let u, v be two vertices, we define $\|uv\|$ as the length of the shortest path from u to v. Furthermore, let f be a failure (which is either an edge or a vertex), we define $\|uv \diamond f\|$ as the length of the shortest path from u to v that does not go through f.
- Let u, v be two vertices, we define $|uv|$ as the number of edges in the shortest path from u to v. In the case that there are many shortest paths from u to v, it turns out that the following definition will be convenient in Section 5: We define $|uv|$ as the *largest* number of edges in any shortest path from u to v.

Fast matrix multiplication

Let ω be the exponent of matrix multiplication; the current best upper bound is $\omega \leq 2.3728596$ [2]. For positive integers n_1, n_2, n_3, let $\mathsf{MM}(n_1, n_2, n_3)$ denote the minimum number of arithmetic operations needed to multiply an $n_1 \times n_2$ matrix and an $n_2 \times n_3$ matrix. We define $\omega(a, b, c)$ to be the exponent of multiplying an $n^a \times n^b$ matrix and an $n^b \times n^c$ matrix, i.e.

$$\omega(a, b, c) = \inf\{w : \mathsf{MM}(n^a, n^b, n^c) = O(n^w)\}.$$

It is a classical result that $\omega(1, 1, \lambda) = \omega(1, \lambda, 1) = \omega(\lambda, 1, 1)$ for any real number $\lambda > 0$ [21]; we denote $\omega(\lambda) = \omega(1, 1, \lambda)$.

We will need the following lemmas about the exponent of rectangular matrix multiplication. We refer the reader to the full version of this paper for their proofs.

▶ **Lemma 2.1.** *Let a, b, c, r be positive real numbers, then $r + \omega(a, b, c) \leq \omega(a, b + r, c + r)$.*

▶ **Lemma 2.2.** *Consider the function $f(\tau) = \omega(1, 1 - \tau, 1 - \tau)$, where $\tau \in [0, 1]$. Then $\tau + f(\tau)$ is monotonically non-increasing in τ, and $2\tau + f(\tau)$ is monotonically non-decreasing in τ.*

Polynomial operations

Let $p, q \in \mathbb{F}[x]$ be two polynomials of degree d. It is easy to compute $p + q$ or $p - q$ in $O(d)$ field operations. We can also compute $p \cdot q$ in $\tilde{O}(d)$ field operations using fast Fourier transform. (Here, $\tilde{O}$ hides polylog(d) factors.) When p is invertible, it is also possible to compute $p^{-1} \bmod x^d$ in $\tilde{O}(d)$ field operations [1, Section 8.3].

3 Constructing a DSO in $O(n^{2.5794}M)$ Time

In this section, we show how to preprocess a distance sensitivity oracle in $O(n^{2.5794}M)$ time, such that every query can be answered in constant time. Our preprocessing algorithm is randomized; with high probability over the preprocessing algorithm, the query algorithm always returns the correct answer.

3.1 Preliminaries

First, our preprocessing algorithm will use the following algorithm for inverting a polynomial matrix. A sketch of this algorithm will be given in Section 4.

▶ **Theorem 1.4.** *Let r be an integer, $\mathbb{F}$ be a finite field. Let $\mathbf{F} \in \mathbb{F}[x]^{n \times n}$ be an $n \times n$ matrix over the ring of univariate polynomials $\mathbb{F}[x]$, and let $d \geq 1$ be an upper bound on the degrees of entries of $\mathbf{F}$. If $\mathbf{F}$ is invertible over $(\mathbb{F}[x]/\langle x^r \rangle)^{n \times n}$, the number of field operations to compute $\mathbf{F}^{-1} \bmod x^r$ is at most*

$$\tilde{O}(dn^\omega) + (r^2/d) \cdot \mathsf{MM}(n, nd/r, nd/r) \cdot n^{o(1)}.$$

Let G be a directed graph whose edge weights are integers in $[1, M]$. We define its *symbolic adjacency matrix* $\mathsf{SA}(G)$ as (see [24])

$$\mathsf{SA}(G)_{i,j} = \begin{cases} 1 & \text{if } i = j, \\ z_{i,j}x^l & \text{if there is an edge from } i \text{ to } j \text{ with weight } l \text{ in } G, \\ 0 & \text{otherwise,} \end{cases}$$

where $z_{i,j}$ are unique variables corresponding to edges of G.

It will be inefficient to deal with these variables $z_{i,j}$, therefore we will pick a suitably large field $\mathbb{F}$, and substitute each variable $z_{i,j}$ by a random element in $\mathbb{F}$. However, we still keep the indeterminate x. Now, let $\mathbf{Z}$ be a matrix where each $\mathbf{Z}_{i,j} \in \mathbb{F}$, we will use $\mathsf{SA}_{\mathbf{Z}}(G)$ to denote the matrix $\mathsf{SA}(G)$ with each formal variable $z_{i,j}$ substituted by the field element $\mathbf{Z}_{i,j}$. Note that $\mathsf{SA}_{\mathbf{Z}}(G)$ is a polynomial matrix where every entry is a polynomial over x with degree at most M.

We recall the definition of *adjoint* matrix that will be crucial to our algorithm. Let $\mathbf{A}$ be an $n \times n$ matrix, $i, j \in [n]$. We denote by $\mathbf{A}^{i,j}$ the matrix $\mathbf{A}$ with every element in the i-th row and the j-th column set to zero, except that $(\mathbf{A}^{i,j})_{i,j} = 1$. The adjoint matrix of $\mathbf{A}$, denoted as $\mathrm{adj}(\mathbf{A})$, is an $n \times n$ matrix such that $\mathrm{adj}(\mathbf{A})_{i,j} = \det(\mathbf{A}^{j,i})$ for every $i, j \in [n]$. A basic fact about $\mathrm{adj}(\mathbf{A})$ is that if $\det(\mathbf{A}) \neq 0$, then $\mathrm{adj}(\mathbf{A}) = \det(\mathbf{A}) \cdot \mathbf{A}^{-1}$.

There is a close relationship between the distances in the graph G, and the entries in the adjoint of $\mathsf{SA}(G)$. Let p be a multivariate polynomial, we define $\deg_x^*(p)$ as the lowest degree of the variable x in any monomial of p. If $p = 0$, then we define $\deg_x^*(p) := +\infty$. We have:

▶ **Theorem 3.1** ([24, Lemma 4]). *Let G be a directed graph with positive integer weights, i, j be two vertices. Then the distance from i to j in G is $\deg_x^*(\mathrm{adj}(\mathsf{SA}(G))_{i,j})$.*

We need the following theorem that allows us to maintain the adjoint of a matrix under *rank*-1 queries. (This theorem is a special case of [35, Lemma 1.6].)

▶ **Theorem 3.2.** *Let $\mathcal{R}$ be an arbitrary commutative ring, $\mathbf{A} \in \mathcal{R}^{n\times n}$ be an invertible matrix, $\mathbf{u}, \mathbf{v} \in \mathcal{R}^n$ be column vectors, and $\gamma = 1 + \mathbf{v}^\mathsf{T}\mathbf{A}^{-1}\mathbf{u}$. Suppose γ is invertible, then $\mathbf{A} + \mathbf{u}\mathbf{v}^\mathsf{T}$ is also invertible, and*

$$\mathrm{adj}(\mathbf{A} + \mathbf{u}\mathbf{v}^\mathsf{T}) = \det(\mathbf{A})(\gamma\mathbf{A}^{-1} - (\mathbf{A}^{-1}\mathbf{u}\mathbf{v}^\mathsf{T}\mathbf{A}^{-1})).$$

Proof Sketch. By the matrix determinant lemma, we have

$$\det(\mathbf{A} + \mathbf{u}\mathbf{v}^\mathsf{T}) = \gamma \cdot \det(\mathbf{A}).$$

Since γ is invertible, we can use the Sherman-Morrison-Woodbury formula [29, 38]:

$$(\mathbf{A} + \mathbf{u}\mathbf{v}^\mathsf{T})^{-1} = \mathbf{A}^{-1} - \gamma^{-1}(\mathbf{A}^{-1}\mathbf{u}\mathbf{v}^\mathsf{T}\mathbf{A}^{-1}).$$

The theorem is proved by multiplying the above two formulas together. ◀

We need the Schwartz-Zippel lemma that guarantees the correctness of our randomized algorithm.

▶ **Theorem 3.3** (Schwartz-Zippel Lemma, [26, 41]). *Let $p(x_1, x_2, \ldots, x_m)$ be a non-zero polynomial of (total) degree d over a field $\mathbb{F}$. Let S be a finite subset of $\mathbb{F}$, and $r_1, r_2, \ldots, r_m$ be independently and uniformly sampled from S. Then*

$$\Pr[p(r_1, r_2, \ldots, r_m) = 0] \leq \frac{d}{|S|}.$$

We also need the following algorithm that computes the determinant of a polynomial matrix.

▶ **Theorem 3.4** ([18, 31]). *Let $\mathbf{B} \in \mathbb{F}[x]^{n\times n}$ be a matrix of degree at most d, then we can compute $\det(\mathbf{B})$ in $\tilde{O}(dn^\omega)$ field operations.*

3.2 Constructing an r-Truncated DSO

Recall that for a failure f (which is either a vertex or an edge), $\|uv \diamond f\|$ denotes the length of the shortest path from u to v that avoids f. An *r-truncated* DSO, as defined in [22], is a DSO that given a query (u, v, f), outputs the value $\min\{\|uv \diamond f\|, r\}$. The main result of this subsection is that given an integer r and an input graph G, an r-truncated DSO can be constructed in time

$$\tilde{O}(n^\omega M) + r^2/M \cdot \mathsf{MM}(n, nM/r, nM/r) \cdot n^{o(1)}.$$

Preprocessing algorithm

Let C be a large enough constant. First, we choose a prime $p \in [n^C, 2n^C]$ and let $\mathbb{F} = \mathbb{Z}_p$. Then we let $\mathbf{Z}$ be an $n \times n$ matrix over $\mathbb{F}$, where every $\mathbf{Z}_{i,j}$ is sampled independently from $\mathbb{F}$ uniformly at random. We substitute $\mathbf{Z}$ into $\mathsf{SA}(G)$ to obtain the matrix $\mathsf{SA}_{\mathbf{Z}}(G)$. Recall that each element of $\mathsf{SA}_{\mathbf{Z}}(G)$ is a polynomial over x with coefficients in $\mathbb{F}$, whose degree is at most M. Then we compute $\mathsf{SA}_{\mathbf{Z}}(G)^{-1}$ and $\det(\mathsf{SA}_{\mathbf{Z}}(G))$ using Theorem 1.4 and Theorem 3.4 respectively.

Since we only want an r-truncated DSO, we only need to compute $\mathsf{SA}_{\mathbf{Z}}(G)^{-1}$ modulo x^r, i.e. we only preserve the monomials with degree less than r in every entry of $\mathsf{SA}_{\mathbf{Z}}(G)^{-1}$. Note that $\mathsf{SA}_{\mathbf{Z}}(G)$ is of the form $\mathbf{I} + x\mathbf{M}$ for some matrix $\mathbf{M} \in \mathbb{F}[x]^{n\times n}$, therefore its determinant is of the form $1 + x \cdot p(x)$ for some polynomial $p(x)$. As the determinant is invertible modulo x^r, $\mathsf{SA}_{\mathbf{Z}}(G)$ is also invertible modulo x^r. By Theorem 1.4, we can compute $\mathsf{SA}_{\mathbf{Z}}(G)^{-1} \bmod x^r$ in time

$$\tilde{O}(n^\omega M) + (r^2/M) \cdot \mathsf{MM}(n, nM/r, nM/r) \cdot n^{o(1)}.$$

By Theorem 3.4, we can compute $\det(\mathsf{SA}_{\mathbf{Z}}(G))$ in $\tilde{O}(n^\omega M)$ time. Again, we only need to store the polynomial $\det(\mathsf{SA}_{\mathbf{Z}}(G)) \bmod x^r$. This concludes the preprocessing algorithm.

For the following query algorithms, we use $\mathbf{e}_i$ to denote the i-th standard unit vector, i.e. $(\mathbf{e}_i)_i = 1$, and $(\mathbf{e}_i)_j = 0$ for every index $j \neq i$.

Query algorithm for an edge failure

A query consists of vertices $u, v \in V$ and a failed edge e. We assume that e goes from vertex a to vertex b, and has weight l. Let G' be the graph obtained by removing e from G, then we have $\mathsf{SA}(G') = \mathsf{SA}(G) + \mathbf{u}\mathbf{v}^\mathsf{T}$, where $\mathbf{u} = \mathbf{e}_a$ and $\mathbf{v} = -z_{a,b}x^l\mathbf{e}_b$. Let

- $\gamma = 1 + \mathbf{v}^\mathsf{T}\mathsf{SA}(G)^{-1}\mathbf{u} = 1 - z_{a,b}x^l\mathsf{SA}(G)^{-1}_{b,a}$,
- $\beta = (\mathsf{SA}(G)^{-1}\mathbf{u}\mathbf{v}^\mathsf{T}\mathsf{SA}(G)^{-1})_{u,v} = -\mathsf{SA}(G)^{-1}_{u,a}z_{a,b}\mathsf{SA}(G)^{-1}_{b,v}x^l$, and
- $\alpha = \det(\mathsf{SA}(G))(\gamma \cdot \mathsf{SA}(G)^{-1}_{u,v} - \beta)$,

then by Theorem 3.2, we have $\alpha = \mathrm{adj}(\mathsf{SA}(G'))_{u,v}$. (Note that since $l \geq 1$, γ is always invertible.)

Query algorithm for a vertex failure

A query consists of vertices $u, v \in V$ and a failed vertex $f \in V$. It suffices to remove every outgoing edge from f (and we do not need to also remove incoming edges to f), as f already cannot appear as an intermediate vertex in every path from u to v. Therefore, we need to compute $\mathrm{adj}(\mathsf{SA}(G'))_{u,v}$, where G' is obtained by removing all outgoing edges from f in G. Let $\mathbf{u} = \mathbf{e}_f$, and $\mathbf{v}$ be the negation of the transpose of the f-th row of $\mathsf{SA}(G)$, except that $\mathbf{v}_f = 0$, i.e.,

$$\mathbf{v}_j = \begin{cases} -z_{f,j}x^l & \text{if there is an edge from } f \text{ to } j \text{ with weight } l \text{ in } G, \\ 0 & \text{otherwise,} \end{cases}$$

It is easy to see $\mathsf{SA}(G') = \mathsf{SA}(G) + \mathbf{u}\mathbf{v}^\mathsf{T}$. To compute $\mathrm{adj}(\mathsf{SA}(G'))_{u,v}$ using Theorem 3.2, we let

- $\gamma = 1 + \mathbf{v}^\mathsf{T}\mathsf{SA}(G)^{-1}\mathbf{u}$. Note that $(\mathbf{e}_f - \mathbf{v})^\mathsf{T}$ is exactly the f-th row of $\mathsf{SA}(G)$, so $(\mathbf{e}_f - \mathbf{v})^\mathsf{T}\mathsf{SA}(G)^{-1} = \mathbf{e}_f^\mathsf{T}$, and $\mathbf{v}^\mathsf{T}\mathsf{SA}(G)^{-1} = \mathbf{e}_f^\mathsf{T}\mathsf{SA}(G)^{-1} - \mathbf{e}_f^\mathsf{T}$. We have $\gamma = 1 + \mathbf{e}_f^\mathsf{T}\mathsf{SA}(G)^{-1}\mathbf{u} - \mathbf{e}_f^\mathsf{T}\mathbf{u} = \mathsf{SA}(G)^{-1}_{f,f}$;

- $\beta = (\mathsf{SA}(G)^{-1}\mathbf{u}\mathbf{v}^{\mathsf{T}}\mathsf{SA}(G)^{-1})_{u,v} = (\mathbf{e}_u^{\mathsf{T}}\mathsf{SA}(G)^{-1}\mathbf{u})(\mathbf{v}^{\mathsf{T}}\mathsf{SA}(G)^{-1}\mathbf{e}_v)$
  $= \mathsf{SA}(G)^{-1}_{u,f}(\mathbf{e}_f^{\mathsf{T}}\mathsf{SA}(G)^{-1}\mathbf{e}_v) = \mathsf{SA}(G)^{-1}_{u,f}\mathsf{SA}(G)^{-1}_{f,v}$;
- and $\alpha = \det(\mathsf{SA}(G))(\gamma \cdot \mathsf{SA}(G)^{-1}_{u,v} - \beta)$,

then we have $\alpha = \mathrm{adj}(\mathsf{SA}(G'))_{u,v}$. (Note that γ is always invertible since the constant term of $\mathsf{SA}(G)^{-1}_{f,f}$ must be 1.)

In the actual query algorithm, we will substitute each formal variable $z_{i,j}$ by $\mathbf{Z}_{i,j}$. Let $\gamma_{\mathbf{Z}}$ denote the resulting polynomial after this substitution. Note that $\gamma_{\mathbf{Z}}$ is a polynomial in $\mathbb{F}[x]$. Similarly we can define $\beta_{\mathbf{Z}}$ and $\alpha_{\mathbf{Z}}$. If $\alpha_{\mathbf{Z}} \not\equiv 0 \pmod{x^r}$, then our query algorithm outputs $\deg^*_x(\alpha_{\mathbf{Z}})$; otherwise it outputs r.

From the above formulas, we can compute $\gamma_{\mathbf{Z}}$, $\beta_{\mathbf{Z}}$, and $\alpha_{\mathbf{Z}}$ in $O(1)$ arithmetic operations over polynomials. Note that we only need to compute these polynomials modulo x^r, so each such arithmetic operation takes $\tilde{O}(r)$ time. The total query time is thus $\tilde{O}(r)$.

▶ Remark 3.5 (Query Algorithm for Undirected Graphs). Our r-truncated DSO can also deal with undirected graphs, but the details are a bit different from the case of directed graphs. To remove an undirected edge, we need to update two entries in $\mathsf{SA}(G)$, which corresponds to a rank-2 update to $\mathsf{SA}(G)$. To remove a vertex, we need to update one row and one column in $\mathsf{SA}(G)$, which is also a rank-2 update to $\mathsf{SA}(G)$. Therefore, we need to use the rank-2 version of Theorem 3.2 (see [35, Lemma 1.6]). Actually, our r-truncated DSOs also support deleting f failures, and the query time is $\tilde{O}(f^\omega r)$. We omit the details here and refer the interested readers to [35].

▶ **Theorem 3.6.** *For every integer r, we can construct an r-truncated DSO with preprocessing time*

$$\tilde{O}(n^\omega M) + r^2/M \cdot \mathsf{MM}(n, nM/r, nM/r) \cdot n^{o(1)},$$

and query time $\tilde{O}(r)$. Our r-truncated DSO is correct w.h.p.

(Recall that by saying our r-truncated DSO is correct w.h.p, we mean that w.h.p. over its randomized preprocessing algorithm, it answers every query correctly.)

Proof of Theorem 3.6. We only need to prove the correctness of our r-truncated DSO. Consider a query (u, v, f) where f is an edge or a vertex, and let G' be the graph obtained by removing f from G. By Theorem 3.2, we have $\alpha_{\mathbf{Z}} = \mathrm{adj}(\mathsf{SA}_{\mathbf{Z}}(G'))_{u,v}$. (Note that the constant term of $\gamma_{\mathbf{Z}}$ is always 1, so $\gamma_{\mathbf{Z}}$ is always invertible.)

If $\|uv \diamond f\| \geq r$, then by Theorem 3.1, $\mathrm{adj}(\mathsf{SA}(G'))_{u,v}$ must be a polynomial whose minimum degree over x is at least r. In this case, we have $\alpha_{\mathbf{Z}} \equiv 0 \pmod{x^r}$ for every $\mathbf{Z}$. Therefore, our algorithm returns r, which is correct.

If $\|uv \diamond f\| = k < r$, then by Theorem 3.1, $\mathrm{adj}(\mathsf{SA}(G'))_{u,v}$ must be a polynomial whose minimum degree is exactly k. In this case, the coefficient of x^k in α is a polynomial of $z_{i,j}$ with (total) degree at most n. (This is because $\mathrm{adj}(\mathsf{SA}(G'))_{u,v}$ is the determinant of a certain $n \times n$ matrix in which every entry has total degree at most one in the variables $z_{i,j}$.) If this polynomial is nonzero at $\mathbf{Z}$, then $\deg^*_x(\alpha_{\mathbf{Z}}) = k$ and our query algorithm is correct. By Theorem 3.3, this polynomial is 0 with probability at most $1/n^{C-1}$. Therefore, our query algorithm returns the correct answer k with probability at least $1 - 1/n^{C-1}$.

In conclusion, for every fixed query (u, v, f), our query algorithm is correct with probability $1-1/n^{C-1}$ over the choice of $\mathbf{Z}$. By a union bound over $O(n^4)$ possible queries, the probability (over our randomized preprocessing algorithm) that every query is answered correctly is at least $1 - 1/\Theta(n^{C-5})$, which is a high probability. ◀

3.3 Constructing the Full DSO

Now we have constructed an r-truncated DSO, which we denote by $\mathcal{D}^{\mathsf{start}}$. In this subsection, we will extend it to a *full* DSO using the techniques in [22]. Specifically, we use the following two algorithms from [22].

The first algorithm transforms an (r-truncated) DSO with a possibly large query time into an (r-truncated) DSO with query time $O(1)$. More precisely:

▶ **Lemma 3.7** ([22, Observation 2.1]). *Given an r-truncated DSO $\mathcal{D}$ with preprocessing time P and query time Q, we can build an r-truncated DSO $\mathsf{Fast}(\mathcal{D})$ with query time $O(1)$ which is correct w.h.p. The preprocessing algorithm of $\mathsf{Fast}(\mathcal{D})$ is as follows:*

- *It needs the all-pairs distance matrix of the input graph G, as well as the set of* consistent *(incoming and outgoing) shortest path trees rooted at each vertex in G. By Theorem 1.6, these shortest path trees can be computed in $O(n^{2.5286}M)$ time. For details, see Section 5.*
- *It invokes the preprocessing algorithm of $\mathcal{D}$ on the input graph G once, and makes $\tilde{O}(n^2)$ queries to $\mathcal{D}$. The preprocessing time is $P + \tilde{O}(n^2)Q$.*

The second algorithm we use is implicit in the argument of [22, Section 2.3]. We formalize it as the following lemma.

▶ **Lemma 3.8.** *Given an r-truncated DSO $\mathcal{D}$ with preprocessing time P and query time $O(1)$, we can build a $(3/2)r$-truncated DSO $\mathsf{Extend}(\mathcal{D})$ with preprocessing time $P + O(n^2)$ and query time $\tilde{O}(nM/r)$. The new DSO is correct w.h.p.*

Now, we are ready to explain our algorithm to build a full DSO. Given an r-truncated DSO $\mathcal{D}^{\mathsf{start}}$, we first obtain an r-truncated DSO $\mathcal{D}_0$ with query time $O(1)$ by applying Lemma 3.7.

Let $i^\star = \lfloor \log_{3/2}(nM/r) \rfloor$. For every $0 \le i \le i^\star$, we construct an $r(3/2)^{i+1}$-truncated DSO $\mathcal{D}_{i+1}$ by applying Lemma 3.8 and Lemma 3.7 sequentially on $\mathcal{D}_i$, i.e. $\mathcal{D}_{i+1} = \mathsf{Fast}(\mathsf{Extend}(\mathcal{D}_i))$. Let the resulting DSO be $\mathcal{D}^{\mathsf{final}} = \mathcal{D}_{i^\star+1}$, since $r(3/2)^{i^\star+1} \ge nM$, $\mathcal{D}^{\mathsf{final}}$ is a full DSO.

We can also summarize our construction algorithm in one formula:

$$\mathcal{D}^{\mathsf{final}} = \underbrace{\mathsf{Fast}(\mathsf{Extend}(\mathsf{Fast}(\mathsf{Extend}(\cdots \mathsf{Fast}(\mathcal{D}^{\mathsf{start}})))))}_{O(\log(nM/r)) \text{ times}}.$$

Complexity of our DSO

Let $r = Mn^\alpha$, where $\alpha \in [0, 1]$ is a parameter to be determined. By Theorem 3.6, the preprocessing time of $\mathcal{D}^{\mathsf{start}}$ is

$$\tilde{O}(n^\omega M) + r^2/M \cdot \mathsf{MM}(n, nM/r, nM/r) \cdot n^{o(1)} \le \tilde{O}(n^\omega M) + n^{2\alpha+\omega(1,1-\alpha,1-\alpha)+o(1)}M,$$

and the query time of $\mathcal{D}^{\mathsf{start}}$ is $\tilde{O}(r) = \tilde{O}(n^\alpha M)$. By Lemma 3.7, the preprocessing time of $\mathcal{D}_0$ is

$$\tilde{O}(n^{2+\alpha}M + n^\omega M) + n^{2\alpha+\omega(1,1-\alpha,1-\alpha)+o(1)}M.$$

Now consider the preprocessing algorithm of $\mathcal{D}^{\mathsf{final}}$. We need to compute the all-pairs distance matrix and in/out shortest path trees of G as required by Lemma 3.7, which takes $\tilde{O}(n^{2+\mu}M)$ time by Theorem 1.6. We also need to run the preprocessing algorithm of $\mathcal{D}_0$. Also, for every $0 \le i \le i^\star$, we need to preprocess the oracle $\mathcal{D}_{i+1}$, which takes $n^2 \cdot \tilde{O}(nM/(r(3/2)^{i+1})) = \tilde{O}\left(\frac{n^{3-\alpha}M}{(3/2)^i}\right)$ time.

Therefore, the preprocessing time of $\mathcal{D}^{\mathsf{final}}$ is:

$$\tilde{O}(n^{2+\alpha}M + n^{\omega}M + n^{2+\mu}M) + n^{2\alpha+\omega(1,1-\alpha,1-\alpha)+o(1)}M + \sum_{i=0}^{\lfloor \log_{3/2}(nM/r)\rfloor} \tilde{O}\left(\frac{n^{3-\alpha}M}{(3/2)^i}\right)$$
$$\leq n^{\max\{2+\alpha,2+\mu,3-\alpha,2\alpha+\omega(1,1-\alpha,1-\alpha)\}+o(1)}M.$$

Let $\alpha = 0.420645$, $\beta = \frac{1}{1-\alpha}$, then $1.5 < \beta < 1.75$. Recall that for any real number λ, $\omega(\lambda)$ is a shorthand for $\omega(1,1,\lambda)$. We have

$$\begin{aligned}
\omega(1,1-\alpha,1-\alpha) &= (1-\alpha)\omega(\beta) \\
&\leq (1-\alpha)\cdot\frac{(1.75-\beta)\omega(1.5)+(\beta-1.5)\omega(1.75)}{1.75-1.5} && (1)\\
&\leq 0.579355\cdot 4\cdot(0.023943\cdot\omega(1.5)+0.226058\cdot\omega(1.75)) \\
&\leq 1.738094. && (2)
\end{aligned}$$

Here, Equation (1) uses the convexity of the $\omega(\cdot)$ function [21], and Equation (2) uses the recent bounds in [15] that $\omega(1.5) \leq 2.796537$ and $\omega(1.75) \leq 3.021591$. We can see that

$$\max\{2+\alpha, 2+\mu, 3-\alpha, 2\alpha+\omega(1,1-\alpha,1-\alpha)\} = 2\alpha+\omega(1,1-\alpha,1-\alpha) \leq 2.579384.$$

By Lemma 3.7, the query time of $\mathcal{D}^{\mathsf{final}}$ is $O(1)$. Therefore, we can construct a DSO with $O(n^{2.5794}M)$ preprocessing time and $O(1)$ query time.

As the DSOs constructed in Lemma 3.7 always have size $\tilde{O}(n^2)$, our final DSO only occupies $\tilde{O}(n^2)$ space. However, we remark that the preprocessing algorithm of our DSO requires $\tilde{O}(rn^2) = O(n^{2.4207})$ space (in particular, to store $\mathsf{SA}_{\mathbf{Z}}(G)^{-1} \bmod x^r$).

4 Inverting a Polynomial Matrix Modulo x^r

As we see in Section 3, the algorithm in Theorem 1.4 for inverting a polynomial matrix modulo x^r is very crucial for our results.

▶ **Theorem 1.4.** *Let r be an integer, $\mathbb{F}$ be a finite field. Let $\mathbf{F} \in \mathbb{F}[x]^{n\times n}$ be an $n\times n$ matrix over the ring of univariate polynomials $\mathbb{F}[x]$, and let $d \geq 1$ be an upper bound on the degrees of entries of $\mathbf{F}$. If $\mathbf{F}$ is invertible over $(\mathbb{F}[x]/\langle x^r\rangle)^{n\times n}$, the number of field operations to compute $\mathbf{F}^{-1} \bmod x^r$ is at most*

$$\tilde{O}(dn^{\omega}) + (r^2/d)\cdot \mathsf{MM}(n, nd/r, nd/r)\cdot n^{o(1)}.$$

Our algorithm is essentially the algorithm in [40]. In fact, the only difference is that we only consider polynomials modulo x^r. This allows us to invert the polynomial matrix in time faster than $\tilde{O}(n^3 d)$ using fast rectangular matrix multiplication algorithms.

In this section, we provide a very brief exposition of the algorithm in [40], and justify the time bound in Theorem 1.4. A detailed description of the algorithm can be found in the full version.

Let $\mathbf{F}$ be an input polynomial matrix where each entry has degree at most d. Suppose $\mathbf{F}$ is invertible over $(\mathbf{F}[x]/\langle x^r\rangle)^{n\times n}$. We will compute a *kernel basis decomposition* of $\mathbf{F}$, which is a chain of matrices $\mathbf{A}_1, \mathbf{A}_2, \ldots, \mathbf{A}_{\log n}$ and a diagonal matrix $\mathbf{B}$, such that

$$\mathbf{F}^{-1} = \mathbf{A}_1\mathbf{A}_2\ldots\mathbf{A}_{\log n}\mathbf{B}^{-1}. \quad (3)$$

Then, to compute $\mathbf{F}^{-1}$, we simply multiply the above matrices. Note that $\mathbf{B}$ is a diagonal matrix, so its inverse is easy to compute.[3]

To start, we write $\mathbf{F} = \begin{bmatrix}\mathbf{F}_{\mathsf{U}}\\ \mathbf{F}_{\mathsf{D}}\end{bmatrix}$, where each $\mathbf{F}_{\mathsf{U}}$ or $\mathbf{F}_{\mathsf{D}}$ is an $(n/2) \times n$ matrix. Then we compute two $n \times (n/2)$ matrices $\mathbf{N}_{\mathsf{R}}$ and $\mathbf{N}_{\mathsf{L}}$ with full rank, such that $\mathbf{F}_{\mathsf{U}}\mathbf{N}_{\mathsf{R}} = \mathbf{0}$, and $\mathbf{F}_{\mathsf{D}}\mathbf{N}_{\mathsf{L}} = \mathbf{0}$. (This can be done by [39, Theorem 4.2].) Let $\mathbf{A}_1 = \begin{bmatrix}\mathbf{N}_{\mathsf{L}} & \mathbf{N}_{\mathsf{R}}\end{bmatrix}$, then $\mathbf{A}_1$ has full rank, and

$$\mathbf{F} \cdot \mathbf{A}_1 = \begin{bmatrix}\mathbf{F}_{\mathsf{U}}\mathbf{N}_{\mathsf{L}} & \mathbf{F}_{\mathsf{U}}\mathbf{N}_{\mathsf{R}}\\ \mathbf{F}_{\mathsf{D}}\mathbf{N}_{\mathsf{L}} & \mathbf{F}_{\mathsf{D}}\mathbf{N}_{\mathsf{R}}\end{bmatrix} = \begin{bmatrix}\mathbf{F}_{\mathsf{U}}\mathbf{N}_{\mathsf{L}} & \\ & \mathbf{F}_{\mathsf{D}}\mathbf{N}_{\mathsf{R}}\end{bmatrix}.$$

Therefore, $\mathbf{F} \cdot \mathbf{A}_1$ is a block diagonal matrix with two blocks, each of size $(n/2) \times (n/2)$. We can then recursively invoke the kernel basis decomposition of these two blocks, and form the matrices $\mathbf{A}_2, \ldots, \mathbf{A}_{\log n}$. The diagonal matrix $\mathbf{B}$ is created at the base case of the recursion, where the diagonal blocks of $\mathbf{F} \cdot \mathbf{A}_1 \cdot \cdots \cdot \mathbf{A}_{\log n}$ are of size 1×1. It is shown in [40] that the kernel basis decomposition takes only $\tilde{O}(dn^{\omega})$ time to compute.

We still need to compute Equation (3). From the above algorithm, we can see that each $\mathbf{A}_i$ is a block-diagonal matrix, which consists of 2^{i-1} blocks of size $(n/2^{i-1}) \times (n/2^{i-1})$. Now we *assume* that each entry in $\mathbf{A}_i$ also has degree at most $d \cdot 2^{i-1}$. (In reality, the behavior of degrees in $\mathbf{A}_i$ may be complicated, and we need the notion of *shifted column degree* to control them; see the full version of this paper for more details.)

To compute Equation (3), we define $\mathbf{M}_i = \mathbf{A}_1\mathbf{A}_2 \ldots \mathbf{A}_i$, and compute each $\mathbf{M}_i$ by the formula

$$\mathbf{M}_{i+1} = \mathbf{M}_i\mathbf{A}_{i+1}. \tag{4}$$

The degree of each entry in $\mathbf{M}_i$ will be at most $O(2^i \cdot d)$. As we only need the results modulo x^r, we can assume the degrees are actually $O\big(\min\{r, 2^i \cdot d\}\big)$. Note that $\mathbf{A}_{i+1}$ consists of 2^i blocks, each of size $(n/2^i) \times (n/2^i)$, and the degree of each (nonempty) entry in $\mathbf{A}_{i+1}$ is also $O\big(\min\{r, 2^i \cdot d\}\big)$. Therefore, we can compute Equation (4) in

$$O\big(\min\{r, 2^i \cdot d\}\big) \cdot 2^i \cdot \mathsf{MM}(n, n/2^i, n/2^i) \tag{5}$$

time. (It is basically 2^i matrix products of size $n \times (n/2^i)$ and $(n/2^i) \times (n/2^i)$; we need to multiply another factor of $\min\{r, 2^i \cdot d\}$ which is the degree of polynomials in these matrices.)

Now, it is easy to see that the bottleneck of this algorithm occurs when $r = 2^i \cdot d$, and the time for computing Equation (4) is:

$$(5) = (r^2/d) \cdot \mathsf{MM}(n, nd/r, nd/r).$$

5 Computing Unique Shortest Paths in Directed Graphs

In this section, we show how to compute *unique* shortest paths in a directed graph in $\tilde{O}(n^{2+\mu}M)$ time, matching the current best time bound for computing the all-pairs distances [42]. Here $\mu < 0.5286$ is the solution of $\omega(1, 1, \mu) = 1 + 2\mu$ [15]. This algorithm is needed before we use Lemma 3.7.

[3] Every diagonal element of $\mathbf{B}$ is a divisor of the *largest invariant factor* of $\mathbf{F}$ (see [40, Section 5.1]), which is (again) a divisor of $\det(\mathbf{F})$. Since $\det(\mathbf{F})$ is invertible modulo x^r, every diagonal element of $\mathbf{B}$ is also invertible modulo x^r.

We may assume that before we proceed, we have already computed the all-pairs distances $\|uv\|$ for every $u, v \in V$, using the APSP algorithm in [42].

Our tie-breaking method requires a (random) permutation π of all vertices, or equivalently a bijection between the vertex set V and $[n]$, i.e. $\pi : V \to [n]$. According to π, for every graph G on V and every $u, v \in V$, we will specify a shortest path $\rho_G(u, v)$ in G from u to v in a certain way. These shortest paths will be *consistent* and *easy to compute*, which is captured by the following theorem. (See also [22, Theorem 1.3 and 1.4].)

▶ **Theorem 5.1.** *Given a graph G on V, a representation of the set of shortest paths $\{\rho_G(u, v)\}_{u,v\in V}$ can be computed in $\tilde{O}(n^{2+\mu}M)$ time, with high probability over the random choice of permutation π, such that the following hold.*

(Property a) *Let G be a graph on V. For every $u', v' \in \rho_G(u, v)$ such that u' appears before v', the portion of $u' \rightsquigarrow v'$ in $\rho_G(u, v)$ coincides with the path $\rho_G(u', v')$.*

(Property b) *Let G be a graph on V, $u, v \in V$, and G' be a subgraph of G. Suppose $\rho_G(u, v)$ is completely contained in G', then $\rho_{G'}(u, v) = \rho_G(u, v)$.*

From (Property a), for every vertex u, the shortest paths from u to every other vertex in G form a tree, and we call this tree the *outgoing shortest path tree* rooted at u, denoted as $T^{\mathsf{out}}(u)$. Similarly, the shortest paths to u from every other vertex in G also form a tree, and we call this tree the *incoming shortest path tree* rooted at u, denoted as $T^{\mathsf{in}}(u)$. Actually, the "representation" computed is exactly the set of n outgoing shortest path trees $\{T^{\mathsf{out}}(u)\}_{u\in V}$ and the set of n incoming shortest path trees $\{T^{\mathsf{in}}(u)\}_{u\in V}$.

The rest of this section

We first define the paths $\rho_G(u, v)$ in Section 5.1. Then we explain how to compute them efficiently in Section 5.2, by presenting an algorithm that computes the incoming and outgoing shortest path trees in $\tilde{O}(Mn^{2+\mu})$ time. Finally, we prove (Property a) and (Property b) in Section 5.3.

5.1 Defining $\rho_G(u, v)$

Let G be an input graph, and $\pi : V \to [n]$ be a (random) bijection. Let $u, v \in V$, P be a path from u to v, we will say that any vertex on P that is neither u nor v is an *internal vertex of P*.

Recall that we defined $|uv|$ as the *largest* number of edges in any shortest path from u to v. In particular:

- $|uv| = 0$ if and only if $u = v$;
- $|uv| = 1$ if and only if the edge (u, v) is the *only* shortest path from u to v;
- $|uv| = \infty$ if there is no path from u to v in G;
- otherwise, we have $2 \le |uv| < \infty$.

We claim that the set of vertices mapped to small values by π is a good "hitting set" w.h.p:

▷ Claim 5.2. Fix the graph G. For some large constant C, with high probability over the choice of π, the following holds. For every pair of vertices $u, v \in V$ such that $2 \le |uv| < \infty$, there is a shortest path $\rho'(u, v)$ from u to v, and an internal vertex z on $\rho'(u, v)$, such that $\pi(z) \le CMn \ln n/\|uv\|$.

Proof. Fix two vertices $u, v \in V$, and any shortest path $\rho'(u, v)$ from u to v. Denote $r = \|uv\|$, if $r \leq M \ln n$ then the claim is trivial. Otherwise, there are at least $r/1.1M$ vertices on $\rho'(u, v)$. Therefore, the probability over a random bijection $\pi : V \to [n]$ that π maps every vertex on $\rho'(u, v)$ to an integer greater than $CMn \ln n/r$ is at most

$$(1 - CM \ln n/r)^{r/1.1M} \leq 1/n^{C/1.1}.$$

Thus by a union bound, the probability that the above condition holds (for every u, v) is at least $1 - 1/n^{C/1.1-2}$, which is a high probability. ◁

Let $u, v \in V$ such that $2 \leq |uv| < \infty$. Define $w(u, v)$ as the intermediate vertex with the smallest label in any shortest path from u to v, i.e.

$$w(u, v) = \arg_w \min\{\pi(w) : \|uv\| = \|uw\| + \|wv\|, w \neq u \text{ and } w \neq v\}. \tag{6}$$

Claim 5.2 states that w.h.p. for every vertices $u, v \in V$ such that $2 \leq |uv| < \infty$, we have that

$$\pi(w(u, v)) \leq CMn \ln n/\|uv\|. \tag{7}$$

In the rest of this section, we assume that Equation (7) holds for every vertices $u, v \in V$ such that $2 \leq |uv| < \infty$. Now we define the paths $\rho_G(u, v)$.

▶ **Definition 5.3.** *Let $u, v \in V$ such that $|uv| \neq \infty$. The path $\rho_G(u, v)$ is recursively defined as follows.*

- *If $u = v$, then $\rho_G(u, v)$ is the empty path that starts and ends at u.*
- *If $|uv| = 1$, then $\rho_G(u, v)$ consists of a single edge, i.e. the edge from u to v.*
- *Otherwise, let $w = w(u, v)$, then $\rho_G(u, v)$ is the concatenation of $\rho_G(u, w)$ and $\rho_G(w, v)$.*

For every u, v such that $2 \leq |uv| < \infty$, since w is an intermediate vertex on some shortest path from u to v, it is easy to see that $|uw| < |uv|$ and $|wv| < |uv|$. Therefore $\rho_G(u, v)$ is well defined – it is inductively defined in the nondecreasing order of $|uv|$.

5.2 Computing Shortest Path Trees in $\tilde{O}(Mn^{2+\mu})$ Time

We will need the following classical algorithm for computing distance products:

▶ **Lemma 5.4** ([42]). *Let A be an $n \times m$ matrix, and B be an $m \times n$ matrix. Suppose every entry in A or B is either $+\infty$ or an integer with absolute value at most M. Then the distance product of A and B can be computed in $\tilde{O}(M \cdot \mathsf{MM}(n, m, n))$ time.*

Computing $w(u, v)$

We first show how to compute $w(u, v)$ for every $u, v \in V$ such that $2 \leq |uv| < \infty$ in $\tilde{O}(Mn^{2+\mu})$ time. Then we use the values of all $w(u, v)$ to compute the incoming and outgoing shortest path trees in $\tilde{O}(n^2)$ additional time. Our strategy for computing $w(u, v)$ is to mimic the algorithm in [17, 28] for computing maximum witness of Boolean matrix multiplication. In particular, we divide the possible witnesses into blocks, and use fast matrix multiplication algorithms to find the block containing $w(u, v)$, for every u, v. After that, we use brute force to find $w(u, v)$ inside that block. Details follow.

Let $r = 2^k$ be a parameter, we show how to compute $w(u, v)$ for every pair of vertices $u, v \in V$ such that $r \leq \|uv\| < 2r$. Let

$$\mathcal{H}_r = \{z \in V : \pi(z) \le CMn \ln n/r\}.$$

By Claim 5.2, for every vertices u, v such that $\|uv\| \in [r, 2r)$, we have $w(u, v) \in \mathcal{H}_r$.

We define an $n \times |\mathcal{H}_r|$ matrix A and an $|\mathcal{H}_r| \times n$ matrix B as follows. For every $u \in V$ and $z \in \mathcal{H}_r$, we define

$$A[u, z] = \begin{cases} \|uz\| & \text{if } \|uz\| \le 2r \text{ and } u \ne z \\ +\infty & \text{otherwise} \end{cases}, \text{ and } B[z, u] = \begin{cases} \|zu\| & \text{if } \|zu\| \le 2r \text{ and } u \ne z \\ +\infty & \text{otherwise} \end{cases}.$$

Then we compute the *minimum witness* of the distance product $A \star B$. To be more precise, we compute the matrix $W[\cdot, \cdot]$ such that for every $u, v \in V$,

$$W[u, v] = \arg_z \min\{\pi(z) : \|uv\| = A[u, z] + B[z, v]\}.$$

Correctness. Fix $u, v \in V$, where $\|uv\| \in [r, 2r)$. We will show that if $|uv| = 1$, then $W[u, v]$ does not exist; otherwise $W[u, v]$ coincides with $w(u, v)$ defined in Equation (6).

First, suppose $|uv| = 1$, then there are no intermediate vertex z such that $\|uv\| = \|uz\| + \|zv\|$, which means $W[u, v]$ does not exist.

Now we assume $|uv| \ge 2$. Since $\|uv\| \ge r$, by Claim 5.2, there is an intermediate vertex $z \in \mathcal{H}_r$ such that $\|uz\| + \|zv\| = \|uv\|$. Since $\|uz\|, \|zv\| \le \|uv\| < 2r$, we can see that $\|uv\| = A[u, z] + B[z, v]$, therefore $W[u, v]$ exists. Let $z = W[u, v]$, then by Equation (6), $\pi(w(u, v)) \le \pi(z)$. On the other hand, Claim 5.2 shows that $w(u, v) \in \mathcal{H}_r$, so by the definition of $z = W[u, v]$, we have $\pi(z) \le \pi(w(u, v))$. Therefore $z = w(u, v)$ and we have established the correctness of $W[\cdot, \cdot]$.

Time complexity. Now we show how to compute the matrix $W[\cdot, \cdot]$ efficiently.

Let $s = n^\mu$, where $\mu \in (0, 1)$ is a parameter to be determined later. If $|\mathcal{H}_r| < s$, then we can compute the matrix W by brute force in $\tilde{O}(n^2 s)$ time. Otherwise, we partition $\mathcal{H}_r$ into blocks of size s, where the i-th block contains vertices that are mapped by π to values between $(i-1) \cdot s + 1$ and $i \cdot s$. For every block i, we compute the distance product of A and B where only vertices in block i are allowed as witnesses. In other words, we compute the following matrix

$$D^i[u, v] = \min\{A[u, z] + B[z, v] : (i-1) \cdot s + 1 \le \pi(z) \le i \cdot s\}.$$

By Lemma 5.4, this matrix can be computed in $\tilde{O}(r \cdot \mathsf{MM}(n, s, n))$ time. There are $O(|\mathcal{H}_r|/s) = \tilde{O}(Mn/(rs))$ blocks, and we need to compute a distance product D^i for each block i. Therefore the total time for computing all these distance products is

$$\tilde{O}(r \cdot \mathsf{MM}(n, s, n) \cdot Mn/(rs)) = \tilde{O}(M \cdot (n/s) \cdot \mathsf{MM}(n, s, n)).$$

Now for every $u, v \in V$ such that $\|uv\| \in [r, 2r)$ and $|uv| \ge 2$, we want to compute $W[u, v]$, which is the vertex $z \in \mathcal{H}_r$ with the minimum $\pi(z)$, such that $\|uv\| = A[u, z] + B[z, v]$. First, we find the smallest i such that $D^i[u, v] = \|uv\|$, and we know that $W[u, v]$ is in the i-th block. (If such i does not exist, then $W[u, v]$ does not exist either, and $|uv| = 1$.) This step takes $\tilde{O}(Mn/(rs))$ time. Then we iterate through the vertices in this block, and find the vertex z with the smallest $\pi(z)$ such that $A[u, z] + B[z, v] = \|uv\|$. This step takes $O(s)$ time.

It follows that the time complexity for computing every $w(u,v)$ where $\|uv\| \in [r, 2r)$ is

$$\begin{aligned}&\tilde{O}(M \cdot \mathsf{MM}(n,s,n) \cdot (n/s) + n^2 \cdot Mn/(rs) + n^2 s)\\ \leq\ &\tilde{O}(M \cdot \mathsf{MM}(n,s,n) \cdot (n/s) + n^2 s) \qquad (8)\\ \leq\ &\tilde{O}(M \cdot n^{\omega(1,\mu,1)+1-\mu} + n^{2+\mu}).\end{aligned}$$

Here, Equation (8) is because $n^2 \cdot Mn/(rs) \leq n^2 \cdot M \cdot (n/s) \leq M \cdot \mathsf{MM}(n,s,n) \cdot (n/s)$.

Let μ be the solution to $\omega(1,\mu,1) = 1 + 2\mu$, then $\mu < 0.5286$ ([15,42]). It follows that the time complexity for computing every $w(u,v)$, where $r \leq \|uv\| < 2r$, is at most $\tilde{O}(Mn^{2+\mu})$.

Putting it together. We run the above algorithm for k from 0 to $\lfloor \log(nW) \rfloor$, and for each k, we update the values $w(u,v)$ where $\|uv\| \in [2^k, 2^{k+1})$. The total time to compute $w(u,v)$ for all u, v is thus $\tilde{O}(Mn^{2+\mu})$.

From $w(u,v)$ to unique shortest paths

For every $u, v \in V$, we will compute the parent of u in the tree $T^{\mathsf{in}}(v)$, denoted as $\mathsf{parent}_v(u)$. In other words, $\mathsf{parent}_v(u)$ is the second vertex in the path $\rho_G(u,v)$ (the first being u). After computing $\mathsf{parent}_v(u)$ for every $u, v \in V$, it is easy to construct $T^{\mathsf{in}}(v)$ for every vertex v. We can compute every $T^{\mathsf{out}}(u)$ in a symmetric fashion.

We proceed by nondecreasing order of $\|uv\|$. Suppose that for every (u', v') such that $\|u'v'\| < \|uv\|$, we have already computed $\mathsf{parent}_{v'}(u')$. Now we compute $\mathsf{parent}_v(u)$ as follows. Let $w = w(u,v)$. If w does not exist, let $\mathsf{parent}_v(u) = v$; otherwise $\mathsf{parent}_v(u) = \mathsf{parent}_w(u)$.

This algorithm (that given every $w(u,v)$, computes every $\mathsf{parent}_v(u)$) clearly runs in $\tilde{O}(n^2)$ time. Notice that if w exists, then w is an intermediate vertex in $\rho_G(u,v)$, thus $\|uw\| < \|uv\|$, and the second vertex in the path $\rho_G(u,v)$ coincides with the second vertex in the path $\rho_G(u,w)$. Hence, the correctness of the algorithm can be easily proved by induction on $\|uv\|$.

5.3 Proof of Theorem 5.1

▶ **Theorem 5.1.** *Given a graph G on V, a representation of the set of shortest paths $\{\rho_G(u,v)\}_{u,v\in V}$ can be computed in $\tilde{O}(n^{2+\mu}M)$ time, with high probability over the random choice of permutation π, such that the following hold.*

(Property a) *Let G be a graph on V. For every $u', v' \in \rho_G(u,v)$ such that u' appears before v', the portion of $u' \rightsquigarrow v'$ in $\rho_G(u,v)$ coincides with the path $\rho_G(u',v')$.*

(Property b) *Let G be a graph on V, $u, v \in V$, and G' be a subgraph of G. Suppose $\rho_G(u,v)$ is completely contained in G', then $\rho_{G'}(u,v) = \rho_G(u,v)$.*

In this subsection, for any path P and vertices $u', v' \in P$ such that u' appears before v' on P, we use $P[u', v']$ to denote the portion of $u' \rightsquigarrow v'$ on the path P.

Proof of (Property a). We prove it by induction on the number of edges of $\rho_G(u,v)$. Let $P = \rho_G(u,v)$. If $u = v$ or P has only one edge, (Property a) is trivial. Now suppose P has k edges where $k > 1$. Let $w = w(u,v)$, then w must lie on P. Consider the following three cases:

- Suppose u' appears after (or coincides with) w on P. By definition, $P[w,v] = \rho_G(w,v)$. Then $P[u',v'] = \rho_G(u',v')$ by induction hypothesis on $\rho_G(w,v)$ since it has fewer edges than $\rho_G(u,v)$.
- Suppose v' appears before (or coincides with) w. This case is symmetric to the above case.

- Otherwise, w lies between u' and v' on P.

  First, we claim that $w = w(u', v')$. As w lies on some shortest path from u' to v' (i.e. $P[u', v']$), we have $\pi(w(u', v')) \leq \pi(w)$. On the other hand, suppose there exists w' such that $\pi(w') < \pi(w)$ and w' is on some shortest path from u' to v'. Then w' also lies on some shortest path from u to v, so it is a better candidate for $w(u, v)$, contradicting the definition of w.

  Second, by induction hypothesis on $\rho_G(u, w)$, which has fewer edges than $\rho_G(u, v)$, we have $P[u', w] = \rho_G(u', w)$. Similarly, $P[w, v'] = \rho_G(w, v')$. Therefore, by definition, $P[u', v'] = P[u', w] \circ P[w, v'] = \rho_G(u', v')$. ◀

Proof of (Property b). We prove it by induction on the number of edges of $\rho_G(u, v)$. Let $P = \rho_G(u, v)$. If $u = v$ or P has only one edge, (Property b) is trivial.

Now suppose P has more than one edge. Let $w = w_G(u, v)$ (i.e. the vertex $w(u, v)$ defined in Equation (6) in graph G), we claim that w coincides with $w_{G'}(u, v)$ (i.e. the vertex $w(u, v)$ defined in Equation (6) in graph G'). Since P is also a shortest path from u to v in G', we have $\pi(w_{G'}(u, v)) \leq \pi(w)$. On the other hand, suppose there exists w' such that $\pi(w') < \pi(w)$ and w' is on some shortest path from u to v in G'. Then w' also lies on some shortest path from u to v in G, so it is a better candidate for $w_G(u, v)$, contradicting the definition of w.

Since $\rho_G(u, w)$ has fewer edges than $\rho_G(u, v)$, and $\rho_G(u, w)$ is completely contained in G', we can use induction hypothesis on $\rho_G(u, w)$ to conclude that $P[u, w] = \rho_{G'}(u, w)$. Similarly, we can use the induction hypothesis on $\rho_G(w, v)$ to conclude that $P[w, v] = \rho_{G'}(w, v)$. Therefore, by definition, $\rho_{G'}(u, v) = \rho_{G'}(u, w) \circ \rho_{G'}(w, v) = P$. ◀

6 Conclusions and Open Problems

We presented an improved DSO for directed graphs with integer weights in $[1, M]$. The preprocessing time is $O(n^{2.5794}M)$ and the query time is $O(1)$. However, there is still a small gap between the preprocessing time of our DSO and the current best time bound for the APSP problem in directed graphs, which is $\tilde{O}(n^{2+\mu}M) \leq O(n^{2.5286}M)$ [42]. Can we improve the preprocessing time to $\tilde{O}(n^{2+\mu}M)$, matching the latter time bound? Another interesting problem is to investigate the complexity of preprocessing a DSO in undirected graphs – here, the best time bound for APSP is $\tilde{O}(n^{\omega}M)$ [27, 30]. Can we preprocess a DSO in $\tilde{O}(n^{\omega}M)$ time on undirected graphs?

Compared to other DSOs [8, 16, 36], our oracle has two drawbacks. First, our query algorithm only outputs the shortest distance, but we do not know how to find the actual shortest paths. So another open problem is whether we can find the actual shortest path with additional $O(l)$ query time, where l is the number of edges in the returned shortest path. Second, since we used [22, Observation 2.1], our oracle can only deal with positive edge weights. Can we extend our oracle to also deal with negative edge weights?

For every parameter f, the r-truncated DSO in Section 3.2 can actually handle f edge/vertex deletions in $\tilde{O}(f^{\omega}r)$ query time. (See also [35].) However, as far as we know, [22, Observation 2.1] only works for one failure. It would be exciting to extend [22, Observation 2.1] or our (full) DSO to also handle f failures.

References

1 Alfred V. Aho, John E. Hopcroft, and Jeffrey D. Ullman. *The Design and Analysis of Computer Algorithms*. Addison-Wesley, 1974.

2 Josh Alman and Virginia Vassilevska Williams. A refined laser method and faster matrix multiplication. In *Proc. 32nd Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 522–539, 2021. doi:10.1137/1.9781611976465.32.

3 Noga Alon, Shiri Chechik, and Sarel Cohen. Deterministic combinatorial replacement paths and distance sensitivity oracles. In *Proc. 46th International Colloquium on Automata, Languages and Programming (ICALP)*, volume 132 of *LIPIcs*, pages 12:1–12:14, 2019. doi:10.4230/LIPIcs.ICALP.2019.12.

4 Noga Alon, Zvi Galil, and Oded Margalit. On the exponent of the all pairs shortest path problem. *Journal of Computer and System Sciences*, 54(2):255–262, 1997. doi:10.1006/jcss.1997.1388.

5 Aaron Bernstein and David R. Karger. Improved distance sensitivity oracles via random sampling. In *Proc. 19th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 34–43, 2008. URL: http://dl.acm.org/citation.cfm?id=1347082.1347087.

6 Aaron Bernstein and David R. Karger. A nearly optimal oracle for avoiding failed vertices and edges. In *Proc. 41st Annual ACM Symposium on Theory of Computing (STOC)*, pages 101–110, 2009. doi:10.1145/1536414.1536431.

7 James R. Bunch and John E. Hopcroft. Triangular factorization and inversion by fast matrix multiplication. *Mathematics of Computation*, 28(125):231–236, 1974. doi:10.2307/2005828.

8 Shiri Chechik and Sarel Cohen. Distance sensitivity oracles with subcubic preprocessing time and fast query time. In *Proc. 52nd Annual ACM Symposium on Theory of Computing (STOC)*, pages 1375–1388, 2020. doi:10.1145/3357713.3384253.

9 Don Coppersmith and Shmuel Winograd. Matrix multiplication via arithmetic progressions. *Journal of Symbolic Computation*, 9(3):251–280, 1990. doi:10.1016/S0747-7171(08)80013-2.

10 Camil Demetrescu and Giuseppe F. Italiano. A new approach to dynamic all pairs shortest paths. *Journal of the ACM*, 51(6):968–992, 2004. doi:10.1145/1039488.1039492.

11 Camil Demetrescu, Mikkel Thorup, Rezaul Alam Chowdhury, and Vijaya Ramachandran. Oracles for distances avoiding a failed node or link. *SIAM Journal of Computing*, 37(5):1299–1318, 2008. doi:10.1137/S0097539705429847.

12 Ran Duan and Seth Pettie. Dual-failure distance and connectivity oracles. In *Proc. 20th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 506–515, 2009. doi:10.1137/1.9781611973068.56.

13 Ran Duan and Seth Pettie. Fast algorithms for (max, min)-matrix multiplication and bottleneck shortest paths. In *Proc. 20th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 384–391, 2009. doi:10.1137/1.9781611973068.43.

14 Ran Duan and Tianyi Zhang. Improved distance sensitivity oracles via tree partitioning. In *Proc. 15th International Symposium on Algorithms and Data Structures (WADS)*, volume 10389 of *LNCS*, pages 349–360, 2017. doi:10.1007/978-3-319-62127-2_30.

15 Francois Le Gall and Florent Urrutia. Improved rectangular matrix multiplication using powers of the Coppersmith-Winograd tensor. In *Proc. 29th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1029–1046, 2018. doi:10.1137/1.9781611975031.67.

16 Fabrizio Grandoni and Virginia Vassilevska Williams. Faster replacement paths and distance sensitivity oracles. *ACM Transactions on Algorithms*, 16(1):15:1–15:25, 2020. doi:10.1145/3365835.

17 Miroslaw Kowaluk and Andrzej Lingas. LCA queries in directed acyclic graphs. In *Proc. 32nd International Colloquium on Automata, Languages and Programming (ICALP)*, volume 3580 of *LNCS*, pages 241–248, 2005. doi:10.1007/11523468_20.

18 George Labahn, Vincent Neiger, and Wei Zhou. Fast, deterministic computation of the Hermite normal form and determinant of a polynomial matrix. *Journal of Complexity*, 42:44–71, 2017. doi:10.1016/j.jco.2017.03.003.

19 François Le Gall. Powers of tensors and fast matrix multiplication. In *Proc. 39th International Symposium on Symbolic and Algebraic Computation, (ISSAC)*, pages 296–303, 2014. doi:10.1145/2608628.2608664.

20 Andrea Lincoln, Virginia Vassilevska Williams, and R. Ryan Williams. Tight hardness for shortest cycles and paths in sparse graphs. In *Proc. 29th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1236–1252, 2018. doi:10.1137/1.9781611975031.80.

21 Grazia Lotti and Francesco Romani. On the asymptotic complexity of rectangular matrix multiplication. *Theoretical Computer Science*, 23:171–185, 1983. doi:10.1016/0304-3975(83)90054-3.

22 Hanlin Ren. Improved distance sensitivity oracles with subcubic preprocessing time. In *Proc. 28th European Symposium on Algorithms (ESA)*, volume 173 of *LIPIcs*, pages 79:1–79:13, 2020. doi:10.4230/LIPIcs.ESA.2020.79.

23 Karthik C. S. and Merav Parter. Deterministic replacement path covering. In *Proc. 32nd Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 704–723, 2021. doi:10.1137/1.9781611976465.44.

24 Piotr Sankowski. Shortest paths in matrix multiplication time. In *Proc. 13th European Symposium on Algorithms (ESA)*, volume 3669 of *LNCS*, pages 770–778, 2005. doi:10.1007/11561071_68.

25 Piotr Sankowski. Subquadratic algorithm for dynamic shortest distances. In *Proc. 11th International Computing and Combinatorics Conference (COCOON)*, volume 3595 of *LNCS*, pages 461–470, 2005. doi:10.1007/11533719_47.

26 Jacob T. Schwartz. Fast probabilistic algorithms for verification of polynomial identities. *Journal of the ACM*, 27(4):701–717, 1980. doi:10.1145/322217.322225.

27 Raimund Seidel. On the all-pairs-shortest-path problem in unweighted undirected graphs. *Journal of Computer and System Sciences*, 51(3):400–403, 1995. doi:10.1006/jcss.1995.1078.

28 Asaf Shapira, Raphael Yuster, and Uri Zwick. All-pairs bottleneck paths in vertex weighted graphs. *Algorithmica*, 59(4):621–633, 2011. doi:10.1007/s00453-009-9328-x.

29 Jack Sherman and Winifred J. Morrison. Adjustment of an inverse matrix corresponding to a change in one element of a given matrix. *The Annals of Mathematical Statistics*, 21(1):124–127, 1950. URL: http://www.jstor.org/stable/2236561.

30 Avi Shoshan and Uri Zwick. All pairs shortest paths in undirected graphs with integer weights. In *Proc. 40th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 605–615, 1999. doi:10.1109/SFFCS.1999.814635.

31 Arne Storjohann. High-order lifting and integrality certification. *Journal of Symbolic Computation*, 36(3-4):613–648, 2003. doi:10.1016/S0747-7171(03)00097-X.

32 Andrew James Stothers. *On the complexity of matrix multiplication*. PhD thesis, The University of Edinburgh, 2010.

33 Jan van den Brand and Danupon Nanongkai. Dynamic approximate shortest paths and beyond: Subquadratic and worst-case update time. In *Proc. 60th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 436–455, 2019. doi:10.1109/FOCS.2019.00035.

34 Jan van den Brand, Danupon Nanongkai, and Thatchaphol Saranurak. Dynamic matrix inverse: Improved algorithms and matching conditional lower bounds. In *Proc. 60th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 456–480, 2019. doi:10.1109/FOCS.2019.00036.

35 Jan van den Brand and Thatchaphol Saranurak. Sensitive distance and reachability oracles for large batch updates. In *Proc. 60th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 424–435, 2019. doi:10.1109/FOCS.2019.00034.

36 Oren Weimann and Raphael Yuster. Replacement paths and distance sensitivity oracles via fast matrix multiplication. *ACM Transactions on Algorithms*, 9(2):14:1–14:13, 2013. doi:10.1145/2438645.2438646.

37 Virginia Vassilevska Williams. Multiplying matrices faster than Coppersmith-Winograd. In *Proc. 44th Annual ACM Symposium on Theory of Computing (STOC)*, pages 887–898, 2012. doi:10.1145/2213977.2214056.

38 Max A Woodbury. Inverting modified matrices. *Memorandum report*, 42(106):336, 1950.

39 Wei Zhou, George Labahn, and Arne Storjohann. Computing minimal nullspace bases. In *Proc. 37th International Symposium on Symbolic and Algebraic Computation, (ISSAC)*, pages 366–373, 2012. doi:10.1145/2442829.2442881.

40 Wei Zhou, George Labahn, and Arne Storjohann. A deterministic algorithm for inverting a polynomial matrix. *Journal of Complexity*, 31(2):162–173, 2015. doi:10.1016/j.jco.2014.09.004.

41 Richard Zippel. Probabilistic algorithms for sparse polynomials. In *Symbolic and Algebraic Computation, EUROSAM '79*, volume 72 of *LNCS*, pages 216–226, 1979. doi:10.1007/3-540-09519-5_73.

42 Uri Zwick. All pairs shortest paths using bridging sets and rectangular matrix multiplication. *Journal of the ACM*, 49(3):289–317, 2002. doi:10.1145/567112.567114.

Structural Iterative Rounding for Generalized k-Median Problems

Anupam Gupta
Computer Science Department, Carnegie Mellon University, Pittsburgh, PA, USA

Benjamin Moseley
Tepper School of Business, Carnegie Mellon University, Pittsburgh, PA, USA

Rudy Zhou
Tepper School of Business, Carnegie Mellon University, Pittsburgh, PA, USA

Abstract

This paper considers approximation algorithms for generalized k-median problems. This class of problems can be informally described as k-median with a constant number of extra constraints, and includes k-median with outliers, and knapsack median. Our first contribution is a pseudo-approximation algorithm for generalized k-median that outputs a 6.387-approximate solution with a constant number of fractional variables. The algorithm is based on iteratively rounding linear programs, and the main technical innovation comes from understanding the rich structure of the resulting extreme points.

Using our pseudo-approximation algorithm, we give improved approximation algorithms for k-median with outliers and knapsack median. This involves combining our pseudo-approximation with pre- and post-processing steps to round a constant number of fractional variables at a small increase in cost. Our algorithms achieve approximation ratios $6.994 + \epsilon$ and $6.387 + \epsilon$ for k-median with outliers and knapsack median, respectively. These both improve on the best known approximations.

2012 ACM Subject Classification Theory of computation → Facility location and clustering

Keywords and phrases approximation algorithms, clustering, linear programming

Digital Object Identifier 10.4230/LIPIcs.ICALP.2021.77

Category Track A: Algorithms, Complexity and Games

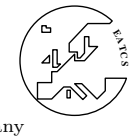
Related Version *Full Version*: https://arxiv.org/abs/2009.00808

Funding *Anupam Gupta*: Supported in part by NSF awards CCF-1907820, CCF-1955785, and CCF-2006953.
Benjamin Moseley: Supported in part by a Google Research Award, an Infor Research Award, a Carnegie Bosch Junior Faculty Chair and NSF grants CCF-1824303, CCF-1845146, CCF-1733873 and CMMI-1938909.

1 Introduction

Clustering is a fundamental problem in combinatorial optimization, where we wish to partition a set of data points into *clusters* such that points within the same cluster are more similar than points across different clusters. In this paper, we focus on generalizations of the *k-median* problem. Recall that in this problem, we are given a set F of facilities, a set C of clients, a metric d on $F \cup C$, and a parameter $k \in \mathbb{N}$. The goal is to choose a set $S \subset F$ of k facilities to open to minimize the sum of *connection costs* of each client to its closest open facility. That is, to minimize the objective $\sum_{j \in C} d(j, S)$, where we define $d(j, S) = \min_{i \in S} d(i, j)$.

The k-median problem is well-studied from the perspective of approximation algorithms, and many new algorithmic techniques have been discovered while studying it. Examples include linear program rounding [3, 13], primal-dual algorithms [10], and local search [1].



Recently, there has been significant interest in generalizations of the k-median problem [4, 11]. One such generalization is the *knapsack median* problem. In knapsack median, each facility has a non-negative weight, and we are given budget $B \geq 0$. The goal is to choose a set of open facilities of total weight at most B (instead of having cardinality at most k) to minimize the same objective function. That is, the open facilities must satisfy a knapsack constraint. Another commonly-studied generalization is *k-median with outliers*, also known as *robust k-median*. Here we open k facilities S, as in basic k-median, but we no longer have to serve all clients; now, we are only required to serve at least m clients $C' \subset C$ of our choice. Formally, the objective function is now $\sum_{j \in C'} d(j, S)$.

Both knapsack median and k-median with outliers have proven to be much more difficult than the standard k-median problem. Algorithmic techniques that have been successful in approximating k-median often lead to only a pseudo-approximation for these generalizations – that is, they violate the knapsack constraint or serve fewer than m clients [2, 4, 6, 9]. Obtaining "true" approximation algorithms requires new ideas beyond those of k-median. Currently the best approximation ratio for both problems is $7.081 + \epsilon$ due to the beautiful iterative rounding framework of Krishnaswamy, Li, and Sandeep [12]. The first and only other true approximation for k-median with outliers is a local search algorithm due to Ke Chen [5].

Generalized k-Median

Both knapsack median and k-median with outliers maintain the salient features of k-median; that is, the goal is to open facilities to minimize the connection costs of served clients. These variants differ in the way we put constraints on the open facilities and served clients. For example, in k-median with outliers, we are constrained to open at most k facilities, and serve at least m clients.

In this paper, we consider a further generalization of k-median that we call *generalized k-median (GKM)*. As in k-median, our goal is to open facilities to minimize the connection costs of served clients. In GKM, the open facilities must satisfy r_1 given knapsack constraints, and the served clients must satisfy r_2 given coverage constraints. We define $r = r_1 + r_2$ to be the number of side constraints overall.

For each knapsack constraint, we have a unique non-negative budget and each facility has a non-negative cost with respect to that budget. The open facilities satisfy all budgets. Similarly, for each coverage constraint, we have a unique non-negative quota and each client has a non-negative value with respect to that quota. Then the served clients must satisfy all quotas.

1.1 Our Results

The main contribution of this paper is a refined iterative rounding algorithm for GKM. Specifically, we show how to round the natural linear program (LP) relaxation of GKM to ensure all except $O(r)$ of the variables are integral, and the objective function is increased by at most a 6.387-factor. It is not difficult to show that the iterative rounding framework in [12] can be extended to show a similar result. Indeed, a 7.081-approximation for GKM with at most $O(r)$ fractional facilities is implicit in their work. The improvement in this work is the smaller loss in the objective value.

▶ **Theorem 1** (Pseudo-Approximation for GKM). *There exists a poly-time pseudo-approximation for GKM that outputs a solution of cost at most $6.387 \cdot Opt$ with at most $O(r)$ fractional facilities.*

Our improvement relies on analyzing the extreme points of certain set-cover-like LPs. These extreme points arise at the intermediate steps of our iterative rounding, and by using their structural properties, we obtain our improved pseudo-approximation for GKM. This work reveals some of the structure of such extreme points, and it shows how this structure can lead to improvements.

Our second contribution is improved "true" approximation algorithms for two special cases of GKM: knapsack median and k-median with outliers. For both problems, applying the pseudo-approximation algorithm for GKM gives a solution with $O(1)$ fractional facilities. Thus, the remaining work is to round a constant number of fractional facilities to obtain an integral solution. To achieve this goal, we apply known sparsification techniques [12] to pre-process the instance, and then develop new post-processing algorithms to round the final $O(1)$ fractional facilities.

We show how to round these remaining variables for knapsack median at arbitrarily small loss, giving a $6.387 + \epsilon$-approximation, improving on the best $7.081 + \epsilon$-approximation. For k-median with outliers, a more sophisticated post-processing is needed to round the $O(1)$ fractional facilities. This procedure loses more in the approximation ratio. In the end, we obtain a $6.994 + \epsilon$-approximation, modestly improving on the best known $7.081 + \epsilon$-approximation.

▶ **Theorem 2** (Approximation for Knapsack Median). *For any $\epsilon > 0$, there exists a $n^{(1/\epsilon)}$-time $(6.387 + \epsilon)$-approximation for knapsack median.*

▶ **Theorem 3** (Approximation for k-Median with Outliers). *For any $\epsilon > 0$, there exists a $n^{(1/\epsilon)}$-time $(6.994 + \epsilon)$-approximation for k-median with outliers.*

Organization

In this paper, we develop and analyze the pseudo-approximation algorithm for GKM guaranteed by Theorem 1. We defer the "true" approximation algorithms guaranteed by Theorems 2 and 3 to the full version of this paper [8], §6.

1.2 Overview of Techniques

To illustrate our techniques, we first introduce a natural LP relaxations for GKM. The problem admits an integer program formulation, with variables $\{x_{ij}\}_{i\in F, j\in C}$ and $\{y_i\}_{i\in F}$, where x_{ij} indicates that *client j connects to facility i* and y_i indicates that *facility i is open*. Relaxing the integrality constraints gives the linear program relaxation LP_1. We focus on only LP_1 for now.

$$
\begin{array}{lll|lll}
(LP_1)\ \min_{x,y} & \sum_{i\in F}\sum_{j\in C} d(i,j)\, x_{ij} & & (LP_2): \quad \min_{y} & \sum_{j\in C}\sum_{i\in F_j} d(i,j)\, y_i & \\
& \sum_{i\in F} x_{ij} \le 1 & \forall j \in C & & y(F_j) \le 1 & \forall j \in C \\
& x_{ij} \le y_i & \forall i \in F, j \in C & & & \\
& Wy \le b & & & Wy \le b & \\
& \sum_{j\in C} a_j(\sum_{i\in F} x_{ij}) \ge c & & & \sum_{j\in C} a_j y(F_j) \ge c & \\
& x_{ij}, y_i \in [0,1] & \forall i \in F, j \in C & & y_i \in [0,1] & \forall i \in F
\end{array}
$$

The linear program LP_1 is the standard k-median LP with the extra side constraints. Note that $\sum_{i\in F} x_{ij} \le 1$ may seem opposite to the intuition that we want clients to get "enough" coverage from the facilities, but that will be guaranteed by the coverage constraints below.

The constraint $Wy \le b$ corresponds to the r_1 knapsack constraints on the facilities y, where $W \in \mathbb{R}_+^{r_1 \times F}$ and $b \in \mathbb{R}_+^{r_1}$. These r_1 packing constraints can be thought of as a multidimensional knapsack constraint over the facilities, and ensure that "few" facilities

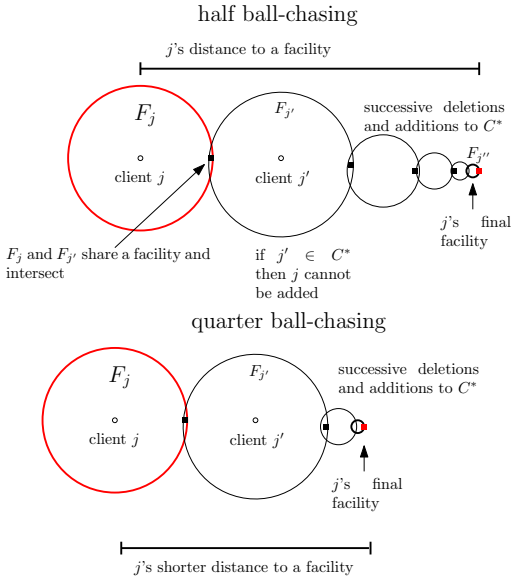


Figure 1 Half and quarter ball chasing.

are opened. Next, $\sum_{j\in C} a_j(\sum_i x_{ij}) \geq c$ corresponds to the r_2 coverage constraints on the clients, where $a_j \in \mathbb{R}_+^{r_2}$ for all $j \in C$ and $c \in \mathbb{R}_+^{r_2}$. These coverage constraints ensure that "enough" clients are served. E.g., having one packing constraint $\sum_{i\in F} y_i \leq k$ and one covering constraint $\sum_{j\in C}\sum_{i\in F} x_{ij} \geq m$ ensures that at least m clients are covered by at most k facilities; this is the k-median with outliers problem.

1.2.1 Constructing LP_2

The idea from [12] is to prescribe a set $F_j \subseteq F$ of permissible facilities for each client j such that x_{ij} is implicitly set to $y_i\mathbf{1}(i \in F_j)$. The procedure to construct these F_j's is given in Proposition 4. Using this procedure, LP_2 is also a relaxation for GKM. Note that in LP_2, we use the notation $y(F') = \sum_{i\in F'} y_i$ for $F' \subset F$.

Now consider solving LP_2 to obtain an optimal extreme point $\bar{y}$. There must be $|F|$ linearly independent tight constraints at $\bar{y}$. The tight constraints of interest are the $y(F_j) \leq 1$ constraints; in general, there are at most $|C|$ such tight constraints, and we have little structural understanding of the F_j-sets.

1.2.2 Prior Iterative Rounding Framework

Consider the family of F_j-sets corresponding to tight constraints, so $\mathcal{F} = \{F_j \mid j \in C, \bar{y}(F_j) = 1\}$. If $\mathcal{F}$ is a family of disjoint sets , then the tight constraints of LP_2 form a face of a partition matroid polytope intersected with at most r side constraints (the knapsack and coverage constraints.) Using ideas from, e.g., [12, 7], we can show that $\bar{y}$ has at most $O(r)$ fractional variables.

Indeed, the goal of the iterative rounding framework in [12] is to control the set family $\mathcal{F}$ to obtain an optimal extreme point where $\mathcal{F}$ is a disjoint family. To achieve this goal, they iteratively round an auxiliary LP based on LP_2, where they have the constraint $y(F_j) = 1$ for all clients j in a special set $C^* \subset C$. Roughly, they regulate what clients are added to C^* and delete constraints $y(F_j) \leq 1$ for some clients. The idea is that a client j whose constraint is deleted must be close to some client j' in C^*. Since $y(F_{j'}) = 1$ we can serve j with the facility for j'; the cost is small if j''s facility is close to j.

To get intuition, assume each client j can pay the farthest distance to a facility in F_j, and call this the *radius* of F_j. (Precisely, clients may not be able to afford this distance, but we use this assumption to highlight the ideas behind our algorithmic decisions.) For simplicity, assume every F_j-set is a ball whose radius is a power of two. Over time, this radius shrinks if some y-variables in F_j are set to zero. Consider applying the following iterative steps until none are applicable, in which case C^* corresponds to the tight constraints: (1) delete a constraint for $j \notin C^*$ if the radius of F_j is at least that of some $F_{j'}$ for $j' \in C^*$ and $F_j \cap F_{j'} \neq \emptyset$. (2) add $j \notin C^*$ to C^* if $y(F_j) \leq 1$ is tight and for every $j' \in C^*$ such that $F_j \cap F_{j'} \neq \emptyset$ it is the case that $F_{j'}$ has a radius strictly larger than F_j. If added then remove all j' from C^* where j's radius is *half* or less of the radius of j' and $F_j \cap F_{j'} \neq \emptyset$.

The approximation ratio is bounded by how much a client j with a deleted constraint pays to get to a facility serving a client in C^*. After removing j's constraint, the case to worry about is if j's closest client $j' \in C^*$ is later removed from C^*. This happens only if j'' is added to C^*, with $F_{j''}$ having half the radius of $F_{j'}$. Thus every time we remove j's closest client in C^*, we guarantee that j's cost only increases geometrically. The approximation ratio is proportional to the total distance that j must travel and can be directly related to the distance of "ball-chasing" though these F_j sets. When we remove a client j from C^* due to $j' \in C^*$ such that $F_{j'} \cap F_j \neq \emptyset$ and j' has radius at most half of j, we call this a *half-chasing step*. See Figure 1.

1.2.3 New Framework via Structured Extreme Points

The target of our framework is to ensure that the radius decreases in the ball-chasing at a faster rate, in particular *one-quarter*. This gives closer facilities for clients whose constraints are deleted. See Figure 1. To achieve this *quarter-chasing step*, we can simply change half to one-quarter in step (2) above.

Making this change immediately decreases the approximation ratio; however, the challenge is that $\mathcal{F}$ is no longer disjoint. Indeed, it can be the case that $j, j' \in C^*$ such that $F_j \cap F_{j'} \neq \emptyset$ if their radii differ by only a one half factor. Instead, our quarter ball-chasing algorithm maintains that $\mathcal{F}$ is not disjoint, but has a *bipartite intersection graph*.

The main technical challenge is obtaining an extreme point with $O(r)$ fractional variables, which is no longer guaranteed as when $\mathcal{F}$ was disjoint. Indeed, if $\mathcal{F}$ has bipartite intersection graph, then the tight constraints form a face of the intersection of two partition matroid polytopes intersected with at most r side constraints. In general, we *cannot upper bound the number of fractional variables* arising in the extreme points of such polytopes. However, such extreme points have a nice combinatorial structure: the intersection graph can be decomposed into $O(r)$ disjoint paths. We exploit this "chain decomposition" of extreme points arising in our iterative rounding to discover clients j that can be removed from C^* even if there is not a $j' \in C^*$ where $F_{j'}$ has one quarter of the radius of F_j. We continue this procedure until we are left with only $O(r)$ fractional variables.

The main technical contribution of this work is showing how the problem can be reduced to structural characterization of extreme points corresponding to bipartite matching. This illustrates some of the structural properties of polytopes defined by k-median-type problems. We hope that this helps lead to other structural characterizations of these polytopes and ultimately improved algorithms.

2 Auxiliary LP for Iterative Rounding

In this section, we construct the auxiliary LP, LP_{iter}. We note that we use the same relaxation used in [12]. Recall the two goals of iterative rounding, outlined in the technical overview; we want to maintain a set of clients $C^* \subset C$ such that $\{F_j \mid j \in C^*\}$ has bipartite intersection graph, and C^* should provide a good set of open facilities for the clients that are not in C^*. Thus, we want to define LP_{iter} to accommodate moving clients in and out of C^*, while having the LP faithfully capture how much we think the clients outside of C^* should pay in connection costs.

2.1 Defining F-balls

Our starting point is LP_2, so we assume that we have sets $F_j \subset F$ for all $j \in C$. The next proposition states that such sets can be found efficiently so that LP_2 is a relaxation of GKM.

▶ **Proposition 4.** *There is a poly-time algorithm that given GKM instance $\mathcal{I}$ outputs sets $F_j \subseteq F$ for $j \in C$ such that $Opt(LP_2) \leq Opt(\mathcal{I})$.*

Proof. Let $\mathcal{I}$ be the given instance of GKM and (x^*, y^*) be an optimal solution to LP_1.

Observe that if $x^*_{ij} \in \{0, y^*_i\}$ for all $i \in F, j \in C$, then we can define $F_j = \{i \in F \mid x^*_{ij} > 0\}$ for all $j \in C$. It is easy to verify in this case that y^* is feasible for LP_2 and achieves the same objective value in LP_2 as (x^*, y^*) achieves in LP_1, which completes the proof.

Thus our goal is to duplicate facilities in F and re-allocate the x- and y-values appropriately until $x^*_{ij} \in \{0, y^*_i\}$ for all $i \in F, j \in C$. To prevent confusion, let F denote the original set of facilities, and let F' denote the modified set of facilities, where make $n = |C|$ copies of each facility in F, so for each $i \in F$, we have copies $i_1, \ldots, i_n \in F'$.

Now we define $x' \in [0,1]^{F' \times C}$ and $y' \in [0,1]^{F'}$ with the desired properties. For each $i \in F$, we assume without loss of generality that $0 \leq x_{i1} \leq x_{i2} \leq \cdots \leq x_{in} \leq y_i$. We define $x'_{i_1 1}, \ldots, x'_{i_n n}$ and $y'_{i_1}, \ldots, y'_{i_n}$ recursively:

Let $y'_{i_1} = x_{i1}$ and $x'_{i_1 j} = x_{ij}$ for all $j \in [n]$.

Now for $k > 1$, let $y'_{i_k} = x_{ik} - x_{i(k-1)}$ and $x'_{i_k j} = \begin{cases} 0 & , j < k \\ y'_{i_k} & , j \geq k \end{cases}$ for all $j \in [n]$.

It is easy to verify that (x', y') is feasible for LP_1 (after duplicating facilities) and $x'_{ij} \in \{0, y'_i\}$ for all $i \in F', j \in C$, as required. Further, it is clear that this algorithm is polynomial time. ◀

In the technical overview, we assumed the radii of the F_j sets were powers of two. To formalize this idea, we discretize the distances to powers of $\tau > 1$ (up to some random offset.) The choice of τ is to optimize the final approximation ratio. The main ideas of the algorithm remain the same if we discretize to powers of, say 2, with no random offset. Our discretization procedure is the following:

Fix some $\tau > 1$ and sample the random offset $\alpha \in [1, \tau)$ such that $\log_e \alpha$ is uniformly distributed in $[0, \log_e \tau)$. Without loss of generality, we may assume that the smallest non-zero inter-point distance is 1. Then we define the possible discretized distances, $L(-2) = -1, L(-1) = 0, \ldots, L(\ell) = \alpha\tau^\ell$ for all $\ell \in \mathbb{N}$. For each $p, q \in F \cup C$, we round $d(p,q)$ *up* to the next largest discretized distance. Let $d'(p,q)$ denote the rounded distances. Observe that $d(p,q) \leq d'(p,q)$ for all $p, q \in F \cup C$. The next proposition bounds the cost of discretization.

▶ **Proposition 5.** *For all $p, q \in F \cup C$, we have $\mathbb{E}[d'(p,q)] = \frac{\tau - 1}{\log_e \tau} d(p,q)$*

Proof. If $d(p,q) = 0$, then the claim is trivial. Suppose $d(p,q) \geq 1$. We can rewrite $d(p,q) = \tau^{\ell+f}$ for some $\ell \in \mathbb{N}, f \in [0,1)$. Also, for convenience we define $\beta = \log_\tau \alpha$. Because $\log_e \alpha$ is uniformly distributed in $[0, \log_e \tau)$, it follows that β is uniformly distributed in $[0,1)$.

It follows, $d(p,q)$ is rounded to $\alpha\tau^\ell = \tau^{\ell+\beta}$ exactly when $\beta \geq f$, and otherwise $d(p,q)$ is rounded to $\tau^{\ell+\beta+1}$ when $\beta < f$. Thus we compute:

$$\begin{aligned}
\mathbb{E}[d'(p,q)] &= \int_{\beta=0}^{f} \tau^{\ell+\beta+1}\, d\beta + \int_{\beta=f}^{1} \tau^{\ell+\beta}\, d\beta \\
&= \frac{1}{\log_e \tau}(\tau^{\ell+\beta+1}|_{\beta=0}^{f} + \tau^{\ell+\beta}|_{\beta=f}^{1}) \\
&= \frac{1}{\log_e \tau}(\tau^{\ell+f+1} - \tau^{\ell+1} + \tau^{\ell+1} - \tau^{\ell+f}) \\
&= \frac{1}{\log_e \tau}(\tau^{\ell+f+1} - \tau^{\ell+f}) \\
&= \frac{\tau-1}{\log_e \tau} d(p,q).
\end{aligned}$$

◀

Now using the discretized distances, we can define the *radius level* of F_j for all $j \in C$ by:

$$\ell_j = \min_{\ell \geq -1} \{\ell \mid d'(j,i) \leq L(\ell) \quad \forall i \in F_j\}.$$

One should imagine that F_j is a ball of radius $L(\ell_j)$ in terms of the d'-distances. Thus, we will often refer to F_j as the *F-ball of client j*. Further, to accommodate "shrinking" the F_j sets, we define the *inner ball of F_j* by:

$$B_j = \{i \in F_j \mid d'(j,i) \leq L(\ell_j - 1)\}.$$

Note that we defined $L(-2) = -1$ so that if $\ell_j = -1$, then $B_j = \emptyset$.

2.2 Constructing LP_{iter}

Our auxiliary LP will maintain three sets of clients: C_{part}, C_{full}, and C^*. C_{part} consists of all clients, whom we have not yet decided whether we should serve them or not. Then for all clients in C_{full} and C^*, we decide to serve them fully. The difference between the clients in C_{full} and C^* is that for the former, we remove the constraint $y(F_j) = 1$ from the LP, while for the latter we still require $y(F_j) = 1$. Thus although we commit to serving C_{full}, such clients rely on C^* to find an open facility to connect to. Using the discretized distances, radius levels, inner balls, and these three sets of clients, we are ready to define LP_{iter}:

$$\begin{aligned}
\min_{y} \quad & \sum_{j \in C_{part}} \sum_{i \in F_j} d'(i,j) y_i + \sum_{j \in C_{full} \cup C^*} \Big(\sum_{i \in B_j} d'(i,j) y_i + (1 - y(B_j)) L(\ell_j) \Big) && (LP_{iter}) \\
\text{s.t.} \quad & y(F_j) \leq 1 \quad \forall j \in C_{part} \\
& y(B_j) \leq 1 \quad \forall j \in C_{full} \\
& y(F_j) = 1 \quad \forall j \in C^* \\
& Wy \leq b \\
& \sum_{j \in C_{part}} a_j y(F_j) \geq c - \sum_{j \in C_{full} \cup C^*} a_j \\
& 0 \leq y \leq 1
\end{aligned}$$

This completes the construction of LP_{iter}. Note that we use the *rounded* distances in the definition of LP_{iter} rather than the original distances. Keeping this in mind, if $C_{part} = C$ and $C_{full}, C^* = \emptyset$, then LP_{iter} is the same as LP_2 up to the discretized distances, so the following lemma is immediate. The algorithm described by the lemma is exactly the steps we took in this section.

▶ **Lemma 6.** *There exists a poly-time algorithm that takes as input a GKM instance $\mathcal{I}$ and outputs LP_{iter} such that $\mathbb{E}[Opt(LP_{iter})] \leq \frac{\tau-1}{\log_e \tau} Opt(\mathcal{I})$.*

The remainder of the paper shows how to iterative round LP_{iter} to obtain our pseudo-approximation for GKM.

2.3 Understanding LP_{iter}

Initially, all clients are in C_{part}. For clients in C_{part}, we are not sure yet whether we should serve them or not. Thus for these clients, we simply require $y(F_j) \leq 1$, so they can be served any amount, and in the objective, the contribution of a client from C_{part} is exactly its connection cost (up to discretization) to F_j.

The clients in C_{full} correspond to the "deleted" constraints in the technical overview. Importantly, for $j \in C_{full}$, we do not require that $y(F_j) = 1$; rather, we relax this condition to $y(B_j) \leq 1$. Recall that we made the assumption that every client can pay the radius of its F_j set. To realize this idea, we require that each $j \in C_{full}$ pays its connection costs to B_j in the objective. Then, to serve j fully, j must find $(1 - y(B_j))$ units of open facility to connect to beyond B_j. Now j truly pays its radius, $L(\ell_j)$, for this $(1 - y(B_j))$ units of connections in LP_{iter}, so we can do ball-chasing to C^* to find these facilities. In this case, we say that we *re-route* the client j to some *destination*.

Note that using the discretized distances, a half-chasing step corresponds to intersecting a neighboring ball of one radius level smaller, and a quarter-chasing step is analogously defined.

For clients in C^*, we require $y(F_j) = 1$. Note that the contribution of a $j \in C^*$ to the objective of LP_{iter} is exactly its connection cost to F_j. The purpose of C^* is to provide destinations for C_{full}.

Finally, because we have decided to fully serve all clients in C_{full} and C^*, regardless of how much they are actually served in their F-balls, we imagine that they every $j \in C_{full} \cup C^*$ contributes a_j to the coverage constraints, which is reflected in LP_{iter}.

3 Basic Iterative Rounding Phase

In this section, we describe the iterative rounding phase of our algorithm. This phase has two main goals: (a) to simplify the constraint set of LP_{iter}, and (b) to decide which clients to serve and how to serve them. To make these two decisions, we repeatedly solve LP_{iter} to obtain an optimal extreme point, and then use the structure of tight constraints to update LP_{iter}, and reroute clients accordingly.

Throughout our algorithm, we will modify the data of LP_{iter} - we will move clients between C_{part}, C_{full}, and C^* and modify the F-balls and radius levels. The key property that we wish to maintain is the *Distinct Neighbors Property*.

▶ **Definition 7** (Distinct Neighbors Property). *For all $j_1, j_2 \in C^*$, if $F_{j_1} \cap F_{j_2} \neq \emptyset$, then $|\ell_{j_1} - \ell_{j_2}| = 1$. In words, if the F-balls of two clients in C^* intersect, then they differ by exactly one radius level.*

This simple property will enable quarter-chasing and a structural characterization of the extreme points of LP_{iter} - both of which are crucial to our improved algorithm.

3.1 The Algorithm

Our algorithm repeatedly solves LP_{iter} to obtain an optimal extreme point $\bar{y}$, and then performs one of the following three possible updates, based on the tight constraints:

1. If some facility i is set to zero in $\bar{y}$, we delete it from the instance.
2. If constraint $\bar{y}(F_j) \leq 1$ is tight for some $j \in C_{part}$, then we decide to fully serve client j by moving j to either C_{full} or C^*. Initially, we add j to C_{full} then run Algorithm 2 to decide if j should be in C^* instead.
3. If constraint $\bar{y}(B_j) \leq 1$ is tight for some $j \in C_{full}$, we shrink F_j by one radius level (so j's new F-ball is exactly B_j.) Then we possibly move j to C^* by running Algorithm 2 for j.

These steps are made formal in Algorithms 1 (IterativeRound) and 2 (ReRoute). IterativeRound relies on the subroutine ReRoute, which gives our criterion for moving a client to C^*. This criterion for adding clients to C^* is the key way in which our algorithm differs from that of [12]. In [12], the criterion used ensures that $\{F_j \mid j \in C^*\}$ is a family of disjoint sets. In contrast, we allow F-balls for clients in C^* to intersect, as long as they satisfy the Distinct Neighbors Property. Thus, our algorithm allows for rich structures in the set system $\{F_j \mid j \in C^*\}$.

Algorithm 1 IterativeRound.

Input: LP_{iter}
Result: Modifies LP_{iter} and outputs an optimal extreme point of LP_{iter}

repeat
  Solve LP_{iter} to obtain optimal extreme point $\bar{y}$.
  if *there exists a facility* $i \in F$ *such that* $\bar{y}_i \geq 0$ *is tight* **then**
    Delete i from F.
  else if *there exists a client* $j \in C_{part}$ *such that* $y(F_j) \leq 1$ *is tight* **then**
    Move j from C_{part} to C_{full}.
    ReRoute(j)
  else if *there exists a client* $j \in C_{full}$ *such that* $\bar{y}(B_j) \leq 1$ *is tight* **then**
    Update $F_j \leftarrow B_j$ and decrement ℓ_j by 1.
    Update $B_j \leftarrow \{i \in F_j \mid d'(j,i) \leq L(\ell_j - 1)\}$.
    ReRoute(j)
  else
    Output $\bar{y}$ and Terminate.
until *termination*

The modifications made by IterativeRound do not increase $Opt(LP_{iter})$, so upon termination of our algorithm, we have an optimal extreme point $\bar{y}$ to LP_{iter} such that LP_{iter} is still a relaxation of GKM and no non-negativity constraint, C_{part}-constraint, or C_{full}-constraint is tight for $\bar{y}$. Further, it is easy to check that the Distinct Neighbors Property is maintained.

Algorithm 2 ReRoute.

Input: Client $j \in C_{full}$
Result: Decide whether to move j to C^* or not
if $\ell_j \leq \ell_{j'} - 1$ *for all* $j' \in C^*$ *such that* $F_j \cap F_{j'} \neq \emptyset$ **then**
  Move j from C_{full} to C^*.
  For all $j' \in C^*$ such that $F_j \cap F_{j'} \neq \emptyset$ and $\ell_{j'} \geq \ell_j + 2$, move j' from C^* to C_{full}.

3.2 Sketch of Analysis

Recall the goals from the beginning of the section: procedure IterativeRound achieves goal (a) of making $\{F_j \mid j \in C^*\}$ simpler while maintaining the Distinct Neighbors Property. Since we moved facilities between C^* and C_{full}, achieving goal (b) means deciding which facilities to open, and guaranteeing that each client has a "close-by" open facility. (Recall from §2 that C^* is the set of clients such that their F_j-balls are guaranteed to contain an open facility, and C_{full} are the clients which are guaranteed to be served but using facilities opened in C^*.)

To achieve goal (b), we observe that ReRoute always gives quarter-chasing steps. That is, if we move a client j from C^* to C_{full}, then we are guaranteed a neighboring client $j' \in C^*$ with radius level at least two smaller than j. Thus, each time we re-route j to a further destination (i.e. if j' is subject to another quarter-chasing step), the extra distance j must travel decreases geometrically. In the end, we can show that j will have an open facility within $O(1)$ times its radius.

4 Iterative Operation for Structured Extreme Points

In this section, we achieve two goals: (a) we show that the structure of the extreme points of LP_{iter} obtained from IterativeRound are highly structured, and admit a *chain decomposition*. Then, (b) we exploit this chain decomposition to define a *new* iterative operation that is applicable whenever $\bar{y}$ has "many" (i.e., more than $O(r)$) fractional variables. We emphasize that this characterization of the extreme points is what enables the new iterative rounding algorithm.

4.1 Chain Decomposition

A chain is a sequence of clients in C^* where the F-ball of each client j contains exactly two facilities – one shared with the previous ball and other with the next.

▶ **Definition 8** (Chain). *A* chain *is a sequence of clients* $(j_1, \ldots, j_p) \subseteq C^*$ *satisfying:*
- $|F_{j_q}| = 2$ *for all* $q \in [p]$, *and*
- $F_{j_q} \cap F_{j_{q+1}} \neq \emptyset$ *for all* $q \in [p-1]$.

Our chain decomposition is a partition of the *fractional* C^*-clients given in the next theorem, which is our main structural characterization of the extreme points of LP_{iter}. (We say a client j is fractional if all facilities in F_j are fractional; we denote the fractional clients in C^* by $C^*_{<1}$.) We defer the proof of the next structural theorem to the full version of this paper [8] §8, and instead focus on how to apply it.

▶ **Theorem 9** (Chain Decomposition). *Upon termination of* IterativeRound, *there exists a partition of* $C^*_{<1}$ *into at most* $3r$ *chains, along with a set of at most* $2r$ *violating clients (clients that are not in any chain.)*

The proof relies on analyzing the extreme points of LP_{iter} satisfying the Distinct Neighbors Property. We show that this boils down to analyzing a bipartite matching polytope with r side constraints.

4.2 Iterative Operation for Chain Decompositions

Leveraging Theorem 9, consider an optimal extreme point $\bar{y}$ of LP_{iter}, and its chain decomposition. We show that if the number of fractional variables in $\bar{y}$ is sufficiently large, there exists a useful structure in the chain decomposition, which we call a *candidate configuration*.

▶ **Definition 10** (Candidate Config). *Let $\bar{y}$ be an optimal extreme point of LP_{iter}. A candidate configuration is a pair of two clients $(j, j') \subset C^*_{<1}$ such that:*

1. $F_j \cap F_{j'} \neq \emptyset$
2. $\ell_{j'} \leq \ell_j - 1$
3. *Every facility in F_j and $F_{j'}$ is in at exactly two F-balls for clients in C^**
4. $|F_j| = 2$ *and* $|F_{j'}| = 2$

One should imagine that a candidate configuration is two neighboring balls on a sufficiently long chain.

▶ **Lemma 11.** *If* IterativeRound *outputs an extreme point that has at least $15r$ fractional facilities, then there exist a candidate configuration in $C^*_{<1}$.*

To prove Lemma 11, which bounds the number of fractional facilities needed to have a candidate configuration, we first prove a bound on the number of factional clients needed. The bound on the number of facilities will follow by a dimension argument.

▶ **Proposition 12.** *Suppose LP_{iter} satisfies the Distinct Neighbors Property. Then each facility is in at most two F-balls for clients in C^*.*

Proof. Assume for contradiction that there exists a facility i such that $i \in F_{j_1} \cap F_{j_2} \cap F_{j_3}$ for distinct clients $j_1, j_2, j_3 \in C^*$. Then j_1 and j_2 differ by one radius level, and j_2 and j_3 differ by one radius level. However, now it cannot be the case that j_1 and j_3 also differ by one radius level. This contradicts the Distinct Neighbors Property. ◀

▶ **Lemma 13.** *Suppose LP_{iter} satisfies all Basic Invariants, and let $\bar{y}$ be an optimal extreme point of LP_{iter} such that no C_{part}-, C_{full}-, or non-negativity constraint is tight. If $|C^*_{<1}| \geq 14r$, then there exist a candidate configuration in $C^*_{<1}$.*

Proof. We claim that in order for $C^*_{<1}$ to have a candidate configuration, it suffices to have a chain of length at least four in $C^*_{<1}$. To see this, let $(j_1, j_2, j_3, j_4, \dots) \subset C^*_{<1}$ be a chain of length at least four. Then $F_{j_2} \cap F_{j_3} \neq \emptyset$, and by the Distinct Neighbors Property, either $\ell_{j_3} = \ell_{j_2} - 1$ or $\ell_{j_2} = \ell_{j_3} - 1$.

We only consider the former case, because both cases are analogous. Thus, if $\ell_{j_3} = \ell_{j_2} - 1$, then we claim that (j_2, j_3) forms a candidate configuration. We already have the first two properties of a candidate configuration. Now we verify the last two. Because j_2 and j_3 are part of a chain, we have $|F_{j_2}| = 2$ and $|F_{j_3}| = 2$. Further, j_2 has neighbors j_1 and j_3 along the chain. By Proposition 12, each facility in F_{j_2} is in at most two F-balls for clients in C^*. In particular, one of the facilities in F_{j_2} is shared by F_{j_1} and F_{j_2}, and the other must be shared by F_{j_2} and F_{j_3}. Thus, each facility in F_{j_2} is in exactly two F-balls for clients in C^*. An analogous argument holds for F_{j_3}, so (j_2, j_3) satisfies all properties of a candidate configuration, as required.

Now suppose $|C^*_{<1}| \geq 14r$. By Theorem 9, $C^*_{<1}$ admits a chain decomposition into at most $3r$ chains and a set of at most $2r$ violating clients. Then at least $12r$ of the clients in $C^*_{<1}$ belong to the $3r$ chains. By averaging, there must exist a chain with size at least $\frac{12r}{3r} = 4$, as required. ◀

Now we relate the number of fractional facilities with the number of fractional C^*-clients by a dimension argument.

▶ **Lemma 14.** *Let $\bar{y}$ be an extreme point of LP_{iter} such that no C_{part}-, C_{full}-, or non-negativity constraint is tight. Then the number of fractional facilities in $\bar{y}$ satisfies $|F_{<1}| \leq |C^*_{<1}| + r$ (recall that r is the number of side constraints.)*

Proof of Lemma 14. We construct a basis $\bar{y}$. First, for each integral facility $i \in F_{=1}$, we add the integrality constraint $\bar{y}_i \leq 1$ to our basis. Thus we currently have $|F_{=1}|$ constraints in our basis.

It remains to choose $|F_{<1}|$ further linearly independent constraints to add to our basis. Note that we have already added all tight integrality constraints to our basis, and no non-negativity constraint is tight. Then the only remaining tight constraints we can add are the C^*-constraints and the r side constraint.

We claim that we cannot add any $C^*_{=1}$-constraints, because every $C^*_{=1}$-constraint is of the form $y(F_j) = y_{i_j} = 1$ for the unique integral facility $i_j \in F_1$. Note that here we used the fact that there is no facility that is set to zero. Thus every $C^*_{=1}$-constraint is linearly dependent with the tight integrality constraints, which we already chose.

It follows, the only possible constraints we can choose are the $C^*_{<1}$-constraints and the r side constraints so:

$$|F_{<1}| \leq |C^*_{<1}| + r.$$ ◀

Lemma 11 is immediate by composing the above two lemmas.

Proof of Lemma 11. By Lemma 13, it suffices to show that $|F_{<1}| \geq 15r$ implies that $|C^*_{<1}| \geq 14r$. Applying Lemma 14, we have:

$$15r \leq |F_{<1}| \leq |C^*_{<1}| + r.$$ ◀

Our new iterative operation is easy to state. Find a candidate configuration (j, j') and move j from C^* to C_{full}.

Algorithm 3 ConfigReRoute.

Input: An optimal extreme point $\bar{y}$ to LP_{iter} s.t. there exists a candidate configuration
Result: Modify LP_{iter}
1 Let $(j, j') \subset C^*_{<1}$ be any candidate configuration.
2 Move j from C^* to C_{full}.

It is easy to check that ConfigReRoute maintains the Distinct Neighbors Property and weakly decreases $Opt(LP_{iter})$.

4.3 Sketch of Analysis

The first two properties of candidate configurations are used to re-route j to j'. Observe a key difference between ReRoute and ConfigReRoute: In the former, we always guarantee quarter-chasing steps. On the other hand, in ConfigReRoute, we only guarantee a neighboring client of at least one radius level smaller, which corresponds to a half-chasing step. This raises the worry that if all re-routings are due to ConfigReRoute, any potential gains by ReRoute are not realized in the worst case. However we show that, roughly speaking, the last two properties of candidate configurations guarantee that the half-chasing steps of ConfigReRoute happen at most half the time.

In particular, suppose client j is re-routed via ConfigReRoute to j' , which is exactly one radius level smaller. If j' is later re-routed via ReRoute, then we can re-route j to j' and then this new destination. This gives one half- and one quarter-chasing step. The concern is if j' is later re-routed via ConfigReRoute, which would give j two half-chasing steps in a row. By analyzing the interactions between ReRoute and ConfigReRoute, we show that there must exist a j'' that gives j a quarter-chasing step. See Figure 2.

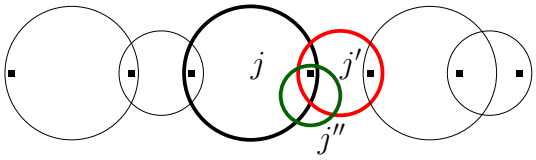


Figure 2 A chain of balls in C^*, where squares indicate facilities. First j is removed from C^* as part of candidate configuration (j, j'), so j' has strictly smaller radius than j. Then j'' is added to C^*, which has strictly smaller radius than j'. This gives j a destination that is at least two radius levels smaller.

5 Pseudo-Approximation Algorithm for GKM

The pseudo-approximation algorithm for GKM combine the iterative rounding algorithm IterativeRound from §3 with the re-routing operation ConfigReRoute from §4 to construct a solution to LP_{iter}. PseudoApproximation is the algorithm guaranteed by Theorem 1.

Algorithm 4 PseudoApproximation.

Input: LP_{iter}
Result: Modifies LP_{iter} and outputs an optimal extreme point of LP_{iter}

```
repeat
    Run IterativeRound to obtain an optimal extreme point ȳ of LP_iter
    if there exists a candidate configuration then
        Run ConfigReRoute
    else
        Output ȳ and Terminate
until Termination
```

5.1 Sketch of Analysis

There are two main components to analyzing PseudoApproximation. First, we show that the output extreme point has $O(r)$ fractional variables, which follows from Lemma 11. Second, we bound the re-routing cost, which follows from the sketches in §3 and §4. In particular, for each client, we can charge each of its half-chasing steps to a quarter-chasing step. This improves on [12], where every re-routing is via half-chasing steps. Optimizing the choice of τ (the discretization factor) gives our final approximation ratio.

5.2 Analysis of PseudoApproximation

In this section, we prove that PseudoApproximation satisfies the guarantees of Theorem 1. We begin by analyzing the runtime and number of fractional facilities.

▶ **Lemma 15.** *PseudoApproximation is a polynomial time algorithm that maintains the Distinct Neighbors Property, weakly decreases $Opt(LP_{iter})$, and outputs an optimal extreme point of LP_{iter} with at most $15r$ fractional variables.*

Proof of Lemma 15. We first show that both IterativeRound and ReRoute are polynomial time. It is clear that the latter runs in polynomial time. For IterativeRound, it suffices to show that the number of iterations of IterativeRound is polynomial. In each iteration, we make one of three actions. We either delete a facility from F, move a client from C_{part} to C_{full} or shrink a F-ball by one radius level for a client in $j \in C_{full}$.

We can delete each facility from F at most once, so we make at most $|F|$ deletions. Each client can move from C_{part} to C_{full} at most once, because we never move clients back from C_{full} to C_{part}, so we do this operations at most $|C|$ times. Finally, observe that $\ell_j \geq -1$ for all $j \in C$ over all iterations. We conclude that we can shrink each F-ball only polynomially many times.

For the runtime of PseudoApproximation, it suffices to show that the number of calls to IterativeRound and ConfigReRoute is polynomial.

In every iteration of PseudoApproximation, either we terminate or we are guaranteed to move a client from C^* to C_{full} in ConfigReRoute. Each client can be removed from C^* only polynomially many times, because each time a client is removed, in order to be re-added to C^*, it must be the case that we shrunk the F-ball of that client. However, again because $\ell_j \geq -1$ for all $j \in C$, we can shrink each F-ball only polynomially many times.

It is easy to check that both IterativeRound and ReRoute maintain the Distinct Neighbors Property and weakly decrease $Opt(LP_{iter})$.

Finally, upon termination of PseudoApproximation, there is no candidate configuration, so Lemma 11 implies that $\bar{y}$ has at most $15r$ fractional variables. ◀

5.3 Analysis of Re-Routing Cost

We now bound the re-routing cost by analyzing how C^* evolves throughout PseudoApproximation. This is one of the main technical contributions of our paper, and it is where our richer C^*-set and relaxed re-routing rules are used. [12] prove an analogous result about the re-routing cost of their algorithm. In the language of the following theorem statement, they show that $\alpha = \frac{\tau+1}{\tau-1}$ for the case $\beta = 1$. We improve on this factor by analyzing the interactions between ReRoute and ConfigReRoute. Interestingly, analyzing each of ReRoute and ConfigReRoute separately would not yield any improvement over [12] in the worst case, even with our richer set C^*. It is only by using the properties of candidate configurations and analyzing sequences of calls to ReRoute and ConfigReRoute that we get an improvement.

▶ **Theorem 16** (Re-Routing Cost). *Upon termination of* PseudoApproximation*, let $S \subset F$ be a set of open facilities and $\beta \geq 1$ such that $d(j,S) \leq \beta L(\ell_j)$ for all $j \in C^*$. Then for all $j \in C_{full} \cup C^*$, $d(j,S) \leq (2+\alpha)L(\ell_j)$, where $\alpha = \max(\beta, 1+\frac{1+\beta}{\tau}, \frac{\tau^3+2\tau^2+1}{\tau^3-1})$.*

We will need the following discretized version of the triangle inequality.

▶ **Proposition 17.** *Let $j, j' \in C$ such that F_j and $F_{j'}$ intersect. Then $d(j,j') \leq L(\ell_j)+L(\ell_{j'})$.*

Proof. Let $i \in F_j \cap F_{j'}$. Then using the triangle inequality we can bound:

$$d(j,j') \leq d(j,i) + d(i,j') \leq d'(j,i) + d'(i,j') \leq L(\ell_j) + L(\ell_{j'}).$$ ◀

The next lemma analyzes the life-cycle of a client that enters C^* at some point in PseudoApproximation. Our improvement over [12] comes from this lemma.

▶ **Lemma 18.** *Upon termination of* PseudoApproximation*, let $S \subset F$ be a set of open facilities and $\beta \geq 1$ such that $d(j,S) \leq \beta L(\ell_j)$ for all $j \in C^*$. Suppose client j is added to C^* at radius level ℓ during* PseudoApproximation *(it may be removed later.) Then upon termination of* PseudoApproximation*, we have $d(j,S) \leq \alpha L(\ell)$, where $\alpha = \max(\beta, 1+\frac{1+\beta}{\tau}, \frac{\tau^3+2\tau^2+1}{\tau^3-1})$.*

Proof. Consider a client j added to C^* with radius level ℓ. If j remains in C^* until termination, the lemma holds for j because $\alpha \geq \beta$. Thus, consider the case where j is later removed from C^* in PseudoApproximation. Note that the only two operations that can possibly cause this removal are ReRoute and ConfigReRoute. We prove the lemma by induction on $\ell = -1, 0, \ldots$. If $\ell = -1$, then j remains in C^* until termination because it has the smallest possible radius level and both ReRoute and ConfigReRoute remove a client from C^* only if there exists another client with strictly smaller radius level.

Similarly, if $\ell = 0$, we note that ReRoute removes a client from C^* only if there exists another client with radius level at least two smaller, which is not possible for j. Thus, if j does not remain in C^* until termination, there must exist some j' that is later added to C^* with radius level at most $\ell - 1 = -1$ such that $F_j \cap F_{j'} \neq \emptyset$. We know that j' remains in C^* until termination since it is of the lowest radius level. Thus:

$$d(j,S) \leq d(j,j') + d(j',S) \leq L(0) + L(-1) + \beta L(-1) = L(0).$$

Now consider $\ell > 0$ where j can possibly be removed from C^* by either ReRoute or ConfigReRoute. In the first case, j is removed by ReRoute, so there exists j' that is added to C^* such that $\ell_{j'} \leq \ell - 2$ and $F_j \cap F_{j'} \neq \emptyset$. Applying the inductive hypothesis to j', we can bound:

$$d(j,S) \leq d(j,j') + d(j',S) \leq L(\ell) + L(\ell-2) + \alpha L(\ell-2) \leq (1 + \frac{1+\alpha}{\tau^2})L(\ell).$$

It is easy to verify by routine calculations that $1 + \frac{1+\alpha}{\tau^2} \leq \alpha$ given that $\alpha \geq \frac{\tau^3+2\tau^2+1}{\tau^3-1}$.

For our final case, suppose j is removed by ConfigReRoute. Then there exists $j' \in C^*$ such that $F_j \cap F_{j'} \neq \emptyset$ and $\ell_{j'} \leq \ell - 1$. Further, $|F_{j'}| = 2$. If j' remains in C^* until termination, then:

$$d(j,S) \leq d(j,j') \leq L(\ell) + L(\ell-1) + \beta L(\ell-1) \leq (1 + \frac{1+\beta}{\tau})L(\ell).$$

Otherwise, j' is removed by ReRoute at an even later time because some j'' is added to C^* such that $\ell_{j''} \leq \ell_{j'} - 2$ and $F_{j'} \cap F_{j''} \neq \emptyset$. Applying the inductive hypothesis to j'', we can bound:

$$d(j,S) \leq d(j,j') + d(j',j'') + d(j'',S) \leq (1 + \frac{2}{\tau} + \frac{1+\alpha}{\tau^3})L(\ell).$$

where $\alpha \geq \frac{\tau^3+2\tau^2+1}{\tau^3-1}$ implies $1 + \frac{2}{\tau} + \frac{1+\alpha}{\tau^3} \leq \alpha$.

Now, we consider the case where j' is later removed by ConfigReRoute. To analyze this case, consider when j was removed by ConfigReRoute. At this time, we have $|F_{j'}| = 2$ by definition of Candidate Configuration. Because $F_j \cap F_{j'} \neq \emptyset$, consider any facility $i \in F_j \cap F_{j'}$. When j is removed from C^* by ConfigReRoute, we have that i is in exactly two F-balls for clients in C^*, exactly F_j and $F_{j'}$. However, after removing j from C^*, i is only in one F-ball for clients in C^* - namely $F_{j'}$.

Later, at the time j' is removed by ConfigReRoute, it must be the case that $|F_{j'}| = 2$ still, so $F_{j'}$ is unchanged between the time that j is removed and the time that j' is removed. Thus the facility i that was previously in $F_j \cap F_{j'}$ must still be present in $F_{j'}$. Then this facility must be in exactly two F-balls for clients in C^*, one of which is j'. It must be the case that the other F-ball containing i, say $F_{j''}$, was added to C^* between the removal of j and j'.

Note that the only operation that adds clients to C^* is ReRoute, so we consider the time between the removal of j and j' when j'' is added to C^*. Refer to Figure 2. At this time, we have $j' \in C^*$, and $F_{j'} \cap F_{j''} \neq \emptyset$ because of the facility i. Then it must be the case that j'' has strictly smaller radius level than j', so $\ell_{j''} \leq \ell_{j'} - 1 \leq \ell - 2$. To conclude the proof, we note that $F_j \cap F_{j''} \neq \emptyset$ due to the facility i, and apply the inductive hypothesis to j'':

$$d(j, S) \leq d(j, j'') + d(j'', S) \leq (1 + \frac{1+\alpha}{\tau^2})L(\ell,)$$

which is at most $\alpha L(\ell)$. ◀

Now using the above lemma, we can prove Theorem 16.

Proof of Theorem 16. Consider any client j that is in $C_{full} \cup C^*$ upon termination of PseudoApproximation. It must be the case that ReRoute(j) was called at least once during PseudoApproximation. Consider the time of the last such call to ReRoute(j). If j is added to C^* at this time, note that its radius level from now until termination remains unchanged, so applying Lemma 18 gives that $d(j, S) \leq \alpha L(\ell_j)$, as required. Otherwise, if j is not added to C^* at this time, then there must exist some $j' \in C^*$ such that $F_j \cap F_{j'} \neq \emptyset$ and $\ell_{j'} \leq \ell_j$. Then applying Lemma 18 to j', we have:

$$d(j, S) \leq d(j, j') + d(j', S) \leq L(\ell_j) + L(\ell_{j'}) + \alpha L(\ell_{j'}) \leq (2 + \alpha)L(\ell_j).$$ ◀

5.4 Putting it all Together: Pseudo-Approximation for GKM

In this section, we prove Theorem 1. In particular, we use the output of PseudoApproximation to construct a setting of the x-variables with the desired properties.

Proof of Theorem 1. Given as input an instance $\mathcal{I}$ of GKM, our algorithm is first to run the algorithm guaranteed by Lemma 6 to construct LP_{iter} from LP_1 such that $\mathbb{E}[Opt(LP_{iter})] \leq \frac{\tau-1}{\log_e \tau} Opt(\mathcal{I})$. Note that we will choose $\tau > 1$ later to optimize our final approximation ratio. Then we run PseudoApproximation on LP_{iter}, so by Theorem 15, PseudoApproximation outputs in polynomial time LP_{iter} along with an optimal solution $\bar{y}$ with $O(r)$ fractional variables.

Given $\bar{y}$, we define a setting $\bar{x}$ for the x-variables: for all $j \in C_{part}$, connect j to all facilities in F_j by setting $\bar{x}_{ij} = \bar{y}_i$ for all $i \in F_j$. For all $j \in C^*$, we have $\bar{y}(F_j) = 1$, so connect j to all facilities in F_j. Finally, to connect every $j \in C_{full}$ to one unit of open facilities, we use the following modification of Theorem 16:

▶ **Proposition 19.** *When* PseudoApproximation *terminates, for all* $j \in C_{full} \cup C^*$*, there exists one unit of open facilities with respect to* $\bar{y}$ *within distance* $(2+\alpha)L(\ell_j)$ *of* j*, where* $\alpha = \max(1, 1+\frac{2}{\tau}, \frac{\tau^3+2\tau^2+1}{\tau^3-1})$.

The proof of the above proposition is analogous to that of Theorem 16 in the case $\beta = 1$, so we omit it. To see this, note that for all $j \in C^*$, we have $\bar{y}(F_j) = 1$. This implies that each $j \in C^*$ has one unit of fractional facility within distance $L(\ell_j)$. Following an analogous inductive argument as in Lemma 18 gives the desired result.

By routine calculations, it is easy to see that $\alpha = \frac{\tau^3+2\tau^2+1}{\tau^3-1}$ for all $\tau > 1$. Now, for all $j \in C_{full}$, we connect j to all facilities in B_j. We want to connect j to one unit of open facilities, so to find the remaining $1 - \bar{y}(B_j)$ units, we connect j to an arbitrary $1 - \bar{y}(B_j)$ units of open facilities within distance $(2+\alpha)L(\ell_j)$ of j, whose existence is guaranteed by Proposition 19. This completes the description of $\bar{x}$.

It is easy to verify that $(\bar{x}, \bar{y})$ is feasible for LP_1, because $\bar{y}$ satisfies all knapsack constraints, and every client's contribution to the coverage constraints in LP_1 is exactly its contribution in LP_{iter}. Thus it remains to bound the cost of this solution. We claim that $LP_1(\bar{x}, \bar{y}) \leq (2+\alpha)Opt(LP_{iter})$, because each client in C_{part} and C^* contributes the same amount to LP_1 and LP_{iter} (up to discretization), and each client $j \in C_{full}$ has connection cost at most $2+\alpha$ times its contribution to LP_{iter}.

In conclusion, the expect cost of the solution $(\bar{x}, \bar{y})$ to LP_1 is at most:

$$(2+\alpha)\,\mathbb{E}[Opt(LP_{iter})] \leq \frac{\tau-1}{\log_e \tau}\left(2 + \frac{\tau^3+2\tau^2+1}{\tau^3-1}\right) Opt(\mathcal{I}).$$

Choosing $\tau > 1$ to minimize $\frac{\tau-1}{\log_e \tau}(2 + \frac{\tau^3+2\tau^2+1}{\tau^3-1})$ gives $\tau = 2.046$ and $\frac{\tau-1}{\log_e \tau}(2 + \frac{\tau^3+2\tau^2+1}{\tau^3-1}) = 6.387$. ◀

5.5 From Pseudo-Approximation to True Approximation

To extend PseudoApproximation to a true approximation algorithm for the special cases of knapsack median and k-median with outliers, we need to round the final $O(1)$ fractional facilities from the output of PseudoApproximation. To do so, we wrap PseudoApproximation with pre-processing and post-processing algorithms. The pre-processing involves enumeration to overcome the unbounded integrality gap, and the post-processing rounds the final $O(1)$ fractional facilities. See [8], §6 for details.

References

1. Vijay Arya, Naveen Garg, Rohit Khandekar, Adam Meyerson, Kamesh Munagala, and Vinayaka Pandit. Local search heuristics for k-median and facility location problems. *SIAM J. Comput.*, 33(3):544–562, 2004. doi:10.1137/S0097539702416402.
2. Jaroslaw Byrka, Thomas W. Pensyl, Bartosz Rybicki, Joachim Spoerhase, Aravind Srinivasan, and Khoa Trinh. An improved approximation algorithm for knapsack median using sparsification. *Algorithmica*, 80(4):1093–1114, 2018. doi:10.1007/s00453-017-0294-4.
3. Jaroslaw Byrka, Thomas W. Pensyl, Bartosz Rybicki, Aravind Srinivasan, and Khoa Trinh. An improved approximation for k-median and positive correlation in budgeted optimization. *ACM Trans. Algorithms*, 13(2):23:1–23:31, 2017. doi:10.1145/2981561.
4. Moses Charikar, Samir Khuller, David M. Mount, and Giri Narasimhan. Algorithms for facility location problems with outliers. In *Proceedings of the Twelfth Annual Symposium on Discrete Algorithms, January 7-9, 2001, Washington, DC, USA*, pages 642–651. ACM/SIAM, 2001.
5. Ke Chen. A constant factor approximation algorithm for k-median clustering with outliers. In *SODA*, pages 826–835, 2008.

6 Zachary Friggstad, Kamyar Khodamoradi, Mohsen Rezapour, and Mohammad R. Salavatipour. Approximation schemes for clustering with outliers. *ACM Trans. Algorithms*, 15(2):26:1–26:26, 2019. doi:10.1145/3301446.

7 Fabrizio Grandoni, R. Ravi, Mohit Singh, and Rico Zenklusen. New approaches to multi-objective optimization. *Math. Program.*, 146(1-2):525–554, 2014. doi:10.1007/s10107-013-0703-7.

8 Anupam Gupta, Benjamin Moseley, and Rudy Zhou. Structural Iterative Rounding for Generalized k-Median Problems. *arXiv e-prints*, page arXiv:2009.00808, 2020. arXiv:2009.00808.

9 Sungjin Im, Mahshid Montazer Qaem, Benjamin Moseley, Xiaorui Sun, and Rudy Zhou. Fast noise removal for k-means clustering. In *The 23rd International Conference on Artificial Intelligence and Statistics, AISTATS 2020, 26-28 August 2020, Online [Palermo, Sicily, Italy]*, pages 456–466, 2020.

10 Kamal Jain and Vijay V. Vazirani. Approximation algorithms for metric facility location and k-median problems using the primal-dual schema and lagrangian relaxation. *J. ACM*, 48(2):274–296, 2001. doi:10.1145/375827.375845.

11 Ravishankar Krishnaswamy, Amit Kumar, Viswanath Nagarajan, Yogish Sabharwal, and Barna Saha. Facility location with matroid or knapsack constraints. *Math. Oper. Res.*, 40(2):446–459, 2015. doi:10.1287/moor.2014.0678.

12 Ravishankar Krishnaswamy, Shi Li, and Sai Sandeep. Constant approximation for k-median and k-means with outliers via iterative rounding. In *Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2018, Los Angeles, CA, USA, June 25-29, 2018*, pages 646–659. ACM, 2018. doi:10.1145/3188745.3188882.

13 Shi Li and Ola Svensson. Approximating k-median via pseudo-approximation. *SIAM J. Comput.*, 45(2):530–547, 2016. doi:10.1137/130938645.

Near-Optimal Schedules for Simultaneous Multicasts

Bernhard Haeupler
Carnegie Mellon University, Pittsburgh, PA, USA
ETH Zürich, Switzerland

D. Ellis Hershkowitz
Carnegie Mellon University, Pittsburgh, PA, USA

David Wajc
Stanford University, CA, USA

Abstract

We study the store-and-forward packet routing problem for simultaneous multicasts, in which multiple packets have to be forwarded along given trees as fast as possible.

This is a natural generalization of the seminal work of Leighton, Maggs and Rao, which solved this problem for unicasts, i.e. the case where all trees are paths. They showed the existence of asymptotically optimal $O(C + D)$-length schedules, where the congestion C is the maximum number of packets sent over an edge and the dilation D is the maximum depth of a tree. This improves over the trivial $O(CD)$ length schedules.

We prove a lower bound for multicasts, which shows that there do not always exist schedules of non-trivial length, $o(CD)$. On the positive side, we construct $O(C + D + \log^2 n)$-length schedules in any n-node network. These schedules are near-optimal, since our lower bound shows that this length cannot be improved to $O(C + D) + o(\log n)$.

2012 ACM Subject Classification Theory of computation → Distributed algorithms; Theory of computation → Graph algorithms analysis; Theory of computation → Routing and network design problems

Keywords and phrases Packet routing, multicast, scheduling algorithms

Digital Object Identifier 10.4230/LIPIcs.ICALP.2021.78

Category Track A: Algorithms, Complexity and Games

Related Version *Full Version*: `https://arxiv.org/pdf/2001.00072.pdf`

Funding Supported in part by NSF grants CCF-1527110, CCF-1618280, CCF-1814603, CCF-1910588, NSF CAREER award CCF-1750808, a Sloan Research Fellowship, and funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation program (ERC grant agreement 949272).
D. Ellis Hershkowitz: Supported by the Air Force Office of Scientific Research under award number FA9550-20-1-0080.
David Wajc: Supported by NSF award 1812919, ONR award N000141912550, and a gift from Cisco Research.

1 Introduction

We study how to efficiently schedule multiple simultaneous multicasts in the store-and-forward model.

Unicasts and multicasts are two of the most basic and important information dissemination primitives in modern communication networks. In a unicast a source sends information to a receiver and in a multicast a source sends information to several receivers. Typically, many such primitives are run simultaneously, causing these primitives to contend for the same resources, most notably the bandwidth of communication links.


The store-and-forward model has been the classic model for developing a clean theoretical understanding of how to most efficiently schedule many such primitives contending for the same link bandwidth. In the store-and-forward model, a network is modeled as a simple undirected graph $G = (V, E)$ with n nodes. Time proceeds in synchronous rounds during which nodes trade packets. In each round a node can send packets it holds to neighbors in G, but at most one packet is allowed to be sent along an edge in each round. Nodes can copy packets and send duplicate packets to neighbors, again subject to the constraint that at most one packet crosses an edge each round.

The store-and-forward model, in turn, enables a formal definition of the problem of scheduling many simultaneous multicasts or unicasts. A simultaneous multicast instance is given by a set of rooted trees $\mathcal{T}$ – one for each multicast – on a store-and-forward network G. Each $T_i \in \mathcal{T}$ has root r_i and leaves L_i along with a packet (a.k.a. message) m_i, initially only known to r_i. A schedule instructs nodes what packets to send in which rounds, subject to the constraint that m_i can only be sent over edges in T_i. The quality of a schedule is its *length*; i.e., the number of rounds until all nodes in L_i have received m_i for every i. A simultaneous unicast instance is the simple case of a simultaneous multicast where all T_i are paths. The goal of past work and this work is to understand the length of the shortest schedule.

The most important parameters in understanding the length of the shortest schedule has been the *congestion* $C = \max_e |\{T_i \ni e\}|$, i.e., the maximum number of packets that need to be routed over any edge in G and the *dilation* $D = \max_i \text{depth}(T_i)$, i.e., the maximum depth of any multicast-tree or the maximum length of any path in the case of simultaneous unicast. It is easy to see that any schedule requires at least $\max(C, D) = \Omega(C + D)$ rounds: a tree with depth D requires at least D rounds to deliver its message and any edge with congestion C requires at least C rounds to forward all packets that need to be sent over it. On the other hand, any instance can easily be scheduled in $O(CD)$ rounds in a greedy manner: in each round and for each edge $e = (u, v)$, forward m_i from u to v where T_i is an arbitrary tree such that $e \in T_i$ and u knows m_i but v does not; it is easy to verify that this schedule takes $O(CD)$ rounds.

Classic results of Leighton, Maggs, and Rao [30] improve upon this trivial $O(CD)$ bound for the case of simultaneous unicast. They showed that introducing a simple independent random delay for each packet at its source suffices to obtain schedules of length $O(C + D \cdot \log n)$ or $O((C + D) \cdot \frac{\log n}{\log \log n})$. A similar strategy can be shown to also work for simultaneous multicasts [12]. More surprisingly, Leighton et al. show how an intricate repeated application of the Lovász Local Lemma [1] proves the existence of length $O(C + D)$ for any simultaneous unicast instance. This seminal paper initiated a long line of followup work [39, 36, 43, 5, 28, 6, 35, 38, 34, 2, 12, 33], some of which even showed these $O(C + D)$-length schedules are efficiently computable [31], even by *distributed* algorithms [34, 38].

In contrast, essentially nothing beyond the above trivial $O(CD)$ and simple random delay bounds of $O(C+D\cdot\log n)$ and $O((C+D)\cdot\frac{\log n}{\log\log n})$ is known for simultaneous multicast, despite ample practical and theoretical motivation. In particular, simultaneous multicast forms an important component of practical content-delivery systems [25, 8], as well as numerous recent theoretical advances in distributed computing [9, 16, 13, 14, 18, 19, 20, 17, 21, 29, 27, 15].

The length of the optimal simultaneous multicast schedule is made all the more intriguing by the work of Ghaffari [12]. This work studied a natural generalization of simultaneous multicast, namely how to schedule many simultaneous distributed algorithms, which corresponds to scheduling the routing of messages on directed acyclic graphs (DAGs). Ghaffari showed that in this setting no $O(C + D)$ schedules exist and, in fact, (up to $O(\log\log n)$ factors) the random delay upper bound of $O(C + D \cdot \log n)$ is the closest that one can get to

an $O(C + D)$ bound. Given that multicasts are more general than unicasts but less general than DAGs, it has remained an interesting open question whether an $O(C + D)$ schedule comparable to those for unicasts is also possible for multicasts or whether, like for DAGs, a multiplicative $O(\log n)$ overhead is required.

1.1 Our Contributions

We show that, unlike in the unicast setting where $O(C + D)$ schedules are possible, for multicasts the trivial $O(CD)$ upper bound cannot be improved without introducing a dependence on the number of nodes, n.

▶ **Theorem 1.1.** *For any $C, D, n \in \mathbb{Z}_+$ such that $C2^{D+1} \leq \log n$ there exists a simultaneous multicast instance on an n-node graph with congestion C and dilation D whose optimal schedule requires at least $\frac{CD}{2}$ rounds.*

We note that our lower bound also implies a new lower bound of $\Omega(CD)$ for the DAGs case studied by Ghaffari [12] since the DAGs case generalizes simultaneous multicasts.

On the positive side, we show that if one allows a schedule's length to depend on n then, unlike in the DAGs case where $O(C + D \cdot \log n)$ is the closest one can get to $O(C + D)$, one can get $O(C + D)$ with a mere *additive* $O(\log^2 n)$.

▶ **Theorem 1.2.** *Each simultaneous multicast instance with congestion C and dilation D in an n-node network admits a schedule of length at most $O(C + D + \log^2 n)$.*

We also verify that these schedules are efficiently computable both by a deterministic, centralized polynomial-time algorithm and by a randomized distributed algorithm in the CONGEST model. Our centralized algorithms are a straightforward extension of our constructions while our distributed algorithms will be based on exponentially decreasing the number of messages that must be sent by using a "rank-decomposition" idea from the union find data structure; we defer the details of our algorithms to Section 7.

Complementing our proof that shows the existence of $O(C + D + \log^2 n)$ schedules, we extend our lower bound to show that any schedule with purely additive dependence on C, D and any function of n incurs at least an additive $\Omega(\log n)$ term. This implies that the additive $\log^2 n$ in Theorem 1.2 is essentially optimal.

▶ **Theorem 1.3.** *Suppose there is a function f such that for any simultaneous multicast instance with congestion C and dilation D, there is a schedule delivering all packets in $O(C + D) + f(n)$ steps. Then $f(n) = \Omega(\log n)$.*

In summary, our results give an essentially optimal characterization of what simultaneous multicast schedules are possible and cleanly separate the complexity of simultaneous multicast schedules from those of simultaneous unicasts and DAGs.

2 Related Work

While we have already mentioned the most relevant previous work, we give some additional related work below.

The seminal work of Leighton et al. [30] initiated a series of works aimed at showing short simultaneous unicast schedules exist. For example, [40, 36] improved the constants in the $O(C + D)$ schedules of Leighton et al., with [36] also generalizing this result to edges with non-unit transit times and bandwidth. Rothvoss [39] presented a simplified proof compared to that of [30] by way of the "method of conditional expectations", and also increased the constant in the $\Omega(C + D)$ lower bound.

In addition to the mentioned work of Ghaffari [12], there is a variety of work in scheduling of specific distributed algorithms. A classic result of Topkis [44] shows that h-hop broadcast of k messages from different sources can be done in $O(k+h)$ rounds. This is a special case of simultaneous multicast, where k multicast instances are to be scheduled along edges of trees with congestion $C \leq k$ and depth D. So, for this special case of simultaneous multicast a $O(C+D)$-length schedule always exists. More recently, Holzer and Wattenhofer [22] showed that n BFSs can be performed from different nodes in $O(n)$ rounds. This was generalized by Lenzen and Peleg [32] who showed that k many h-hop BFSs from different sources can be done in $O(k+h)$ rounds.

Another line of work on simultaneous unicast and related problem focused on *computing* optimal or near-optimal schedules efficiently, starting with work of Leighton et al. [31]. There has been work on simultaneous unicast focused on "local-control" or distributed algorithms, where at each step each node makes decisions on which packets to move forward along their paths, based only on the routing information that the packets carry and on the local history of execution. The $O(C + D \cdot \log n)$ algorithm of Leighton et al. [30], for example, is such a distributed simultaneous unicast algorithm. Rabani and Tardos [38] improved this bound to $O(C) + D \cdot (\log^* n)^{O(\log^* n)}$ rounds, which was then further improved by Ostrovsky and Rabani [34] to $O(C + D + \log^{1+\epsilon} n)$ rounds for any constant $\epsilon > 0$. Another series of works also studied centralized algorithms for simultaneous unicast where the source and sink pairs are fixed but the algorithm is free to choose what paths it uses to deliver packets from sources to sinks. Notably, Srinivasan and Teo [43] gave a constant approximation for this problem. Bertsimas and Gamarnik [5] then provided an asymptotically-optimal algorithm, outputting a schedule of length $OPT + (\sqrt{n \cdot OPT})$; i.e., $OPT(1 + o(1))$ for sufficiently large OPT. Lastly, there has been work in computing schedules for single multicasts [4, 10] and even simultaneous multicasts [23, 24] in models fundamentally different from the store-and-forward model we study.

3 Intuition and Overview of Techniques

We now give an overview of and intuition for the techniques we use in proving the impossibility of good simultaneous multicast schedules and our nearly matching upper bound.

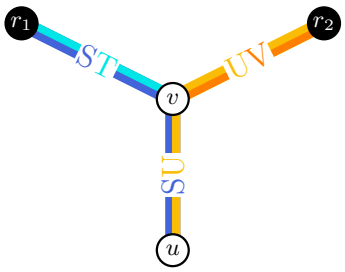
3.1 $\Omega(CD)$ Lower Bound

The goal of our lower $\Omega(CD)$ lower bound construction is to repeatedly "accumulate" delays by combining together already delayed multicast trees. Here, we build some intuition for this strategy .

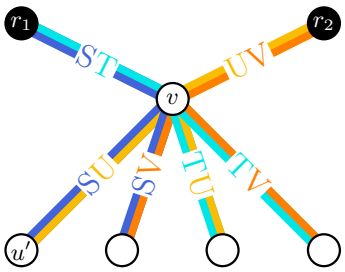
Consider a simultaneous multicast instance consisting of two trees S and T using a single edge as in Figure 1. Since at most one message crosses this edge each round, we know that after one round at least one of our trees' messages will be delayed by 1 round, i.e., will not have crossed the edge. More generally, if C trees all use a single edge e then for any fixed schedule one of these trees will require at least C rounds until its message crosses e.

If we knew, a priori, for any C-congested edge which multicast tree was delayed by C, producing a hard multicast instance would be easy as we could repeatedly combine together the multicast trees delayed by C in each congested edge. For instance, consider the following example, illustrated in Figure 2a where $C = 2$. We have four multicast trees S, T, U and V where S and T have root r_1 and U and V have root r_2. Both roots connect to a vertex v where (r_1, v) is used by S and T and (r_2, v) is used by U and V. If we knew that after a single round T and V used edges (r_1, v) and (r_2, v) respectively, then we could "combine" S

Figure 1 A congested edge example on multicast trees S and T. Root of both trees given by black node. Each multicast tree given in a different color and edges labeled by which multicast trees use them.



(a) Accumulating delays if S and U delayed.



(b) Guessing the delayed trees.

Figure 2 Illustration of how one can "guess" which trees are delayed. Roots given by black nodes. Each multicast tree given in a different color and edges labeled by which multicast trees use them.

and U into a new edge (v, u). Then, the messages for S and U wouldn't arrive at v until at least two rounds have passed and since both S and U use the edge (v, u), one of the messages of either S or U wouldn't arrive at u until four rounds have passed, despite the fact that u is only two hops from the root of each tree. We might hope, then, to recursively repeat this strategy, combining together such gadgets to accumulate larger and larger delays.

However, we, of course, do not always know which trees are delayed and so combining together the most delayed tree is not a feasible strategy. That is, we must provide a construction which requires many rounds for *every possible* simultaneous multicast schedule, not many rounds for one fixed schedule.

We overcome this challenge by using the fact that trees, unlike paths, branch. In particular, we will use the branching of trees to "guess" which tree was delayed for every congested edge. As a concrete example of this strategy consider the simultaneous multicast instance given in Figure 2b. We have the instance as in Figure 2a but now instead of vertex u, we have four vertices, one for each possible guess of which pair of elements in $\{S, T\} \times \{U, V\}$ are delayed at (r_1, v) and (r_2, v). Now notice that for any fixed simultaneous multicast schedule for this instance we know that after one round only one of S and T's messages will cross (r_1, v) and only one of U and V's message will cross (r_2, v). Without loss of generality suppose S and U do not cross (r_1, v) and (r_2, v) respectively in the first round. We then know that one of the edges (v, u') corresponding to one of our guesses – in this case the edge used by S and U – is such that the trees which use this edge will not deliver the their messages to v until two rounds have passed. Similarly, we know that at most one of S and U's messages arrive at u' by the third round – without loss of generality U's message. Thus, S will not successfully deliver its message to all leaves until at least four rounds have passed, despite the fact that all leaves of S are only two hops from S's root. More generally, if we repeated this construction with a larger congestion C we would have that some multicast tree requires at least $2C$ rounds to deliver its message to all leaves, despite the fact that $C + D = C + 2$.

Our lower bound construction will recursively stack trees like those in Figure 2b to guess which multicast trees a schedule chooses to delay and accumulate a larger and larger delay by combining together these delayed trees. We will guarantee that some sub-graph is always correct in its guesses. We will also make use of the observation that if C trees all use a single edge e then by Markov's inequality at least $\frac{C}{2}$ of these trees will require $\frac{C}{2}$ rounds until their message crosses e to reduce the amount of guessing we must do; this will allow us to expand the possible values of C and D we can use when constructing our lower bound graph which will aid in the proof of Theorem 1.3. We elaborate on our $\Omega(CD)$ construction in Section 4 and then extend it in Section 6 to show that additive $\Omega(\log n)$ is necessary for length $O(C + D)$ schedules.

3.2 Existence of $O(C + D + \log^2 n)$ Simultaneous Multicast Schedules

The main intuition underlying our $O(C+D+\log^2 n)$-length simultaneous multicast schedules is that every instance of multicast can be reduced to a series of unicast instances and, as Leighton et al. [30] showed, unicast instances admit schedules of length linear in their congestion and dilation. Our goal then is to gracefully reduce a simultaneous multicast to a series of simultaneous unicasts.

Here, we discuss two natural approaches for such a reduction, argue that they fail and extract intuition for our upper bound from this failure. In the first approach, for each multicast tree T_i we define $|L_i|$ unicast instances, where for each leaf $l \in L_i$ we have a unicast instance on the root-to-leaf path from r_i to l. While this simultaneous unicast instance has dilation $D' = D$, it also has congestion potentially as high as $C' = \Omega(n)$: unicasts corresponding to the same tree are run independently, and each edge in T_i is contained in every root-to-leaf unicast path. Relying on the existence of schedules of length $O(C' + D')$ guaranteed by [30], then, could yield schedules of length as bad as $\Omega(D + n)$. In the second approach, we define a separate unicast instance for each edge in each T_i. We then run a simultaneous unicast schedule for all edges from roots of multicast trees to their children, then from roots' children to their children, and so on and so forth. Here we have at least obtained a sequence of simultaneous unicast instances with lower dilation – $D' = 1$ – and congestion no larger than what we started with – $C' \leq C$. [30] guarantees the existence of schedules of length $O(C' + D')$ for each such simultaneous unicast instance. Unfortunately, we must concatenate together the schedules of D such simultaneous unicast instances to solve the simultaneous multicast instance, which would yield schedules of length $\Omega(D(C' + D')) = \Omega(CD)$; i.e., no better than the trivial schedule.

Thus, the challenge in reducing simultaneous multicast to simultaneous unicast is finding a suitable way of balancing between these two approaches. In the first reduction, we were able to solve a single simultaneous unicast problem with dilation D but one whose congestion was much larger than the congestion of the simultaneous multicast problem with which we started. In the second extreme, we were able to solve simultaneous unicast instances with dilation and congestion only 1 and C but we had to solve many such problems.

Our goal, then, is to find a way of reducing simultaneous multicast to simultaneous unicast in a way that keeps the dilation and congestion of the resulting simultaneous unicast instances small but does not require solving too many simultaneous unicasts. We strike such a balance by computing what we call a $(\log n, \log n)$-*short path decomposition* of each multicast tree. This decomposition is based on subdividing paths in the heavy path decompositions of [42]. By using such a decomposition on each multicast tree along with random delays determining when to schedule each path in the decomposition, we obtain a sequence of $\frac{C}{\log n} + \frac{D}{\log n} + \log n$ many simultaneous unicast instance whose congestion C' and dilation

D' are both at most $O(\log n)$ with high probability. Relying on the $O(C' + D')$ schedules guaranteed by [30] for these simultaneous unicast instances, we find that every simultaneous multicast instance admits a schedules of length $O(C + D + \log^2 n)$. We elaborate on this in Section 5. We also provide centralized and distributed algorithms for the computation of these schedules in Section 7.

4 $\Omega(CD)$ Lower Bound

This section is dedicated to the proof of our $\Omega(CD)$ lower bound. We begin this section by providing the family of instances we use to show this lower bound. We proceed to show how this family requires $\Omega(CD)$ rounds, showing that $O(C+D)$ simultaneous multicast schedules are generally impossible and that the trivial $O(CD)$ schedule is the best simultaneous multicast schedule without a dependence on n. Specifically, we prove the following.

▶ **Theorem 1.1.** *For any $C, D, n \in \mathbb{Z}_+$ such that $C2^{D+1} \leq \log n$ there exists a simultaneous multicast instance on an n-node graph with congestion C and dilation D whose optimal schedule requires at least $\frac{CD}{2}$ rounds.*

4.1 Multicast Instance

We will describe how our instance is constructed in a top-down manner. For the remainder of this section we fix a desired congestion C and dilation D. We will recursively construct a graph in which every edge receives C "labels" where the graph induced by each label is a distinct multicast tree.[1] As each label corresponds to a multicast tree, each label will also have a root corresponding to it which will be the root of the corresponding multicast tree. Ultimately, our instance corresponding to a fixed C and D will contain $C \cdot 2^{D-1}$ multicast trees and so throughout this section we will imagine we have $C \cdot 2^{D-1}$ distinct labels. Throughout this section we will also let capital letters correspond to labels; e.g. $\{S, T, U, V, W, X, Y, Z\}$ is a set of 8 labels. Before moving onto specific details, we refer the reader to Figure 5 for a visual preview of our lower bound construction.

4.1.1 Interleaving Labels

In order to rigorously define what it means to guess which multicast trees are delayed, we introduce the idea of "interleaving" the sets of labels corresponding to our multicast trees.

Given sets S_1 and S_2, each consisting of C labels, we let the interleaving of S_1 and S_2 be $I(S_1, S_2) := \{S_1' \cup S_2' : S_i' \subseteq S_i, |S_i'| = C/2\}$ be all subsets which take $C/2$ labels from S_1 and $C/2$ labels from S_2. For example, if $C = 2$ and $S_1 = \{S, T\}$ and $S_2 = \{U, V\}$ then $I(S_1, S_2) = \{\{S, U\}, \{S, V\}, \{T, U\}, \{T, V\}\}$. S_1 and S_2 will correspond to two adjacent edges, each in a disjoint set of C multicast trees each and so $I(S_1, S_2)$ will correspond to all ways of guessing which C trees, taking $C/2$ trees from one edge and $C/2$ trees from the other edge, are delayed among the $2C$ multicast trees which use one of the two edges.

Let $\mathcal{S} = (S_i)_{i=1}^{2^{D-1}}$ be a tuple partitioning our $C \cdot 2^{D-1}$ distinct labels into sets of size C. That is, each S_i is a set (with associated index i) containing C distinct labels and $S_i \cap S_j = \emptyset$ for $i \neq j$. We call two sets in $\mathcal{S}$ adjacent if the index of one is $2i - 1$ and the index of the other is $2i$ for some $i \in \mathbb{Z}_{\geq 1}$. Finally, we let

$$I(\mathcal{S}) := \bigtimes_{i=1}^{|\mathcal{S}|/2} I(S_{2i-1}, S_{2i})$$

[1] The graph induced by a label χ on graph G is the subgraph of edges from G labeled χ.

be all possible interleavings of adjacent sets in $\mathcal{S}$ where $\bigtimes$ denotes an $|S|/2$-wise Cartesian product. This tuple $\mathcal{S}$ will correspond to all edges of our construction at height D while a pair of adjacent sets S_{2i-1} and S_{2i} will correspond to two adjacent edges, each in disjoint sets of C multicast trees; thus $\mathcal{I}(\mathcal{S})$ will correspond to all possible ways to guess how, among all pairs of adjacent edges at height D, which C trees, taking $C/2$ trees from one edge and $C/2$ trees from the other edge, are delayed in each pair.

We give a concrete example of our notation where $C = 2$ and $D = 3$. Let $\mathcal{S} = (S_1, S_2, S_3, S_4) = (\{S,T\}, \{U,V\}, \{W,X\}, \{Y,Z\})$. Then $I(\mathcal{S})$ corresponds to all ways of combining S_1 and S_2 by taking one element from each and all ways of combining S_3 and S_4 by taking one element from each. In particular, we have

$$\begin{aligned} I(\mathcal{S}) &= I(S_1, S_2) \times I(S_3, S_4) \\ &= \big\{\{S,U\}, \{S,V\}, \{T,U\}, \{T,V\}\big\} \times \big\{\{W,Y\}, \{W,Z\}, \{X,Y\}, \{X,Z\}\big\}, \end{aligned}$$

Notice that each $\mathcal{S}' \in I(\mathcal{S})$ is a tuple of sets, each of C labels, and the number of distinct labels across all sets in $\mathcal{S}'$ is exactly half of the number of labels across all sets in $\mathcal{S}$. Each $\mathcal{S}' \in I(\mathcal{S})$ will correspond to a single recursive call in our construction.

4.1.2 Our Instance

With the above notation in hand, we now describe how we recursively construct our multicast instance for a fixed C and D. Let $\mathcal{S} = (S_i)_{i=1}^{2^{D-1}}$ be an arbitrary partition of our $C \cdot 2^{D-1}$ distinct labels into sets of size C as above. Our recursion will be on D; that is, we will recursively construct several instances of simultaneous multicast with dilation $D-1$ and congestion C and then combine together these instances into a single instance with congestion C and dilation D (in fact, every edge will have congestion exactly C and every tree will have depth exactly D). We let $M_{\mathcal{S}}$ be the simultaneous multicast instance we construct from $\mathcal{S}$ and let $G_{\mathcal{S}}$ be its corresponding graph. As can easily be seen by induction on our recursive depth, each root will inductively only be incident on a single edge and so we let $\chi(r)$ be the set of labels of the one edge incident to root r.

- **Base case ($D = 1$):** In this case we have $\mathcal{S} = (S_1)$. We let $M_{\mathcal{S}}$ consist of one edge (r, v) which receives every label in S_1 and let r be the root of every label/tree.
- **Inductive case ($D > 1$):** We construct $M_{\mathcal{S}}$ in three steps. See Figures 3, 4 and 5 for an illustration of the results of steps one, two and three respectively; we defer the former two to the appendix. In the example in these figures $C = 2$, $D = 3$ and $\mathcal{S} = (S_1, S_2, S_3, S_4) = (\{S,T\}, \{U,V\}, \{W,X\}, \{Y,Z\})$; roots of multicast trees are indicated by black nodes and edges are colored according to their label/tree.
  1. First, for each pair of adjacent sets S_{2i-1} and S_{2i} in $\mathcal{S}$ we introduce vertices r_{2i-1}, r_{2i} and v_i and edges $e_{2i-1} = (r_{2i-1}, v_i)$ and $e_{2i} = (r_{2i}, v_i)$. We let r_j be the root of all trees with a label in S_j and e_j receive all labels in S_j for every j (Figure 3).
  2. Next, we "guess" which trees will be delayed on the e_js. In particular, we add to our graph the disjoint union of $G_{\mathcal{S}'}$ for each $\mathcal{S}' \in \mathcal{I}(\mathcal{S})$; each of the new connected components in our graph at this point corresponds to some instance $M_{\mathcal{S}'}$; each edge inherits the C labels it received in $M_{\mathcal{S}'}$ (Figure 4).
  3. Finally, we connect up our guesses to the corresponding parents. In particular, for each vertex r that was a root in $M'_{\mathcal{S}}$ if $\chi(r) \in I(S_{2i-1}, S_{2i})$, we identify r and v_i as the same vertex (Figure 5).

It is easy to verify by induction on our recursive depth that, indeed, each label and its root induce a tree in the returned graph and so $M_{\mathcal{S}}$ is an instance of simultaneous multicast.

Figure 3 The result of step one of our lower bound construction. Notice that we have an edge for each set S_i and each pair of adjacent S_i are joined together at a vertex. We outline in red and green v_1 and v_2 respectively. Left-to-right black nodes are r_1, r_2, r_3 and r_4.

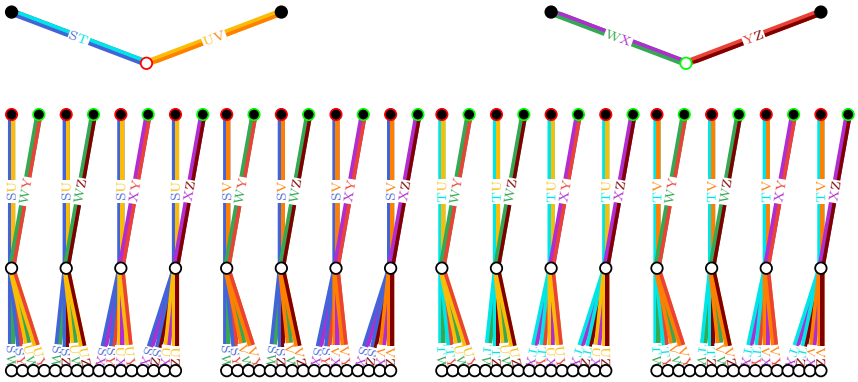


Figure 4 The result of step two of our construction. Notice that we now have a new connected component for each $\mathcal{S}' \in I(\mathcal{S})$, each of which corresponds to a guess for which trees will be delayed at v_1 and v_2. We outline in red and green the vertices which in step three we identify with v_1 and v_2 respectively.

▶ **Lemma 4.1.** *Let $\mathcal{S} = (S_i)_i^{2^{D-1}}$ be a partition of $C \cdot 2^{D-1}$ distinct labels as above. Then each label in $\bigcup_i S_i$ induces a rooted tree in $G_{\mathcal{S}}$.*

Proof. We prove this by induction on D. Let T_χ be the graph induced by label χ and let r_χ be χ's root. We will prove the slightly stronger claim that T_χ is a tree containing r_χ and for any two labels $\chi \neq \chi'$ we have $r_\chi \in T_{\chi'}$ only if $r_\chi = r_{\chi'}$. Call this latter property ⊛.

As a base case suppose that $D = 1$. We then have that $\mathcal{S} = (S_1)$ is a single set of labels and so by definition of $M_{\mathcal{S}}$ we have that our graph will consist of a single edge (r, v) where r is the root for every label and (r, v) is labeled by every label in S_1. Since each label induces the edge (r, v) where r is the root for this label, clearly every label induces a rooted tree containing its root. Moreover, ⊛ holds since every label has the same root.

As an inductive hypothesis suppose that for any $D' < D$ and $\mathcal{S}'$ of size $C \cdot 2^{D'-1}$, we have that every label in $M_{\mathcal{S}'}$ induces a tree containing the label's root and all induced trees in $M_{\mathcal{S}'}$ satisfy ⊛. Thus, our inductive hypothesis tells us that every label in $M_{\mathcal{S}'}$ for $\mathcal{S}' \in I(\mathcal{S})$ induces a tree containing the label's root where all labels satisfy ⊛.

We will first verify that each T_χ is a tree containing r_χ. Clearly, since the edge leaving r_χ is labeled χ, we have that $r_\chi \in T\chi$. Let $\mathcal{T}'_\chi$ be all trees induced by label χ in $M'_{\mathcal{S}}$ for $\mathcal{S}' \in I(\mathcal{S})$. $M_{\mathcal{S}}$ is created by taking the disjoint union of $G_{\mathcal{S}'}$ for $\mathcal{S}' \in I(\mathcal{S})$, identifying several roots of $M_{\mathcal{S}'}$ trees and then adding new roots and edges. Notice that for any χ this identifying of nodes as the same nodes does not cause any cycles in a T_χ since by ⊛ we identify exactly one node from each tree in $\mathcal{T}'_\chi$ with another node. Next, to see that ⊛ still holds notice that if a node is designated a root in $M_{\mathcal{S}}$ then it is incident to a single edge and is a root for every label this edge was assigned. ◀

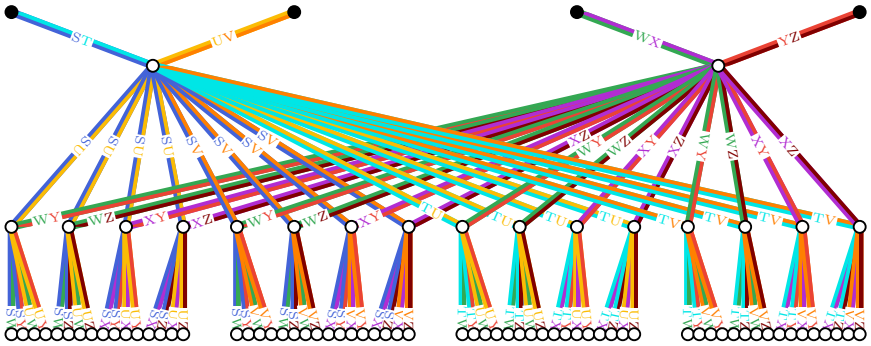


Figure 5 Our construction (i.e. the result of step three) for $\mathcal{S} = (S_1, S_2, S_3, S_4) = (\{S,T\}, \{U,V\}, \{W,X\}, \{Y,Z\})$, $C = 2$ and $D = 3$. Notice that we have modified the graph in Figure 4 by adding one edge for each S_{2i-1} (resp. S_{2i}) colored by the labels in S_{2i-1} (resp. S_{2i}) going from root r_{2i-1} (resp. r_{2i}) to v_i.

4.2 Proof of $\Omega(CD)$ Lower Bound

An induction on D demonstrates that our simultaneous multicast instance has the appropriate congestion and dilation.

▶ **Lemma 4.2.** *$M_{\mathcal{S}}$ has congestion C and dilation D.*

Proof. As each edge receives C labels in our construction, each of which corresponds to a multicast tree, clearly the congestion is C. For the dilation, we prove by induction on D. As a base case notice that if D is 1 then $|\mathcal{S}|$ is 1 and so G consists of a single edge used by all all trees, giving a dilation of 1. Suppose that for $D' < D$ we have that the dilation of $M_{\mathcal{S}'}$ is D' where $|\mathcal{S}'| = C \cdot 2^{D-1}$. The claim follows by simply noticing that each tree in $M_{\mathcal{S}}$ extends the root of every tree in $M_{\mathcal{S}'}$ by 1 edge. ◀

Another simple induction on D and standard approximations allows us to bound the number of nodes in our lower bound graph.

▶ **Lemma 4.3.** $|V(G_{\mathcal{S}})| \leq 2^{C(2^{D+1})}$.

Proof. Clearly, to upper bound the total number of vertices it suffices to upper bound the total number of edges introduced. Thus, we will count the number of edges introduced at each level of our recursion.

Fix C. Define $m_D := |E(G_{\mathcal{S}})|$. We claim by induction on D that $m_D \leq 2^{C(2^D)+D}$. As a base case notice that when $D = 1$ we have $m_1 = 1 \leq 2^{C(2^D)+D}$. For our inductive step consider $G_{\mathcal{S}}$. $G_{\mathcal{S}}$ is constructed by introducing 2^{D-1} edges and unioning together $G_{\mathcal{S}'}$ for $\mathcal{S}' \in I(\mathcal{S})$ of which there are $\binom{C}{C/2}^{2^{D-1}}$, each of which have m_{D-1} edges. Thus, we have

$$
\begin{aligned}
m_D &= 2^{D-1} + \binom{C}{C/2}^{2^{D-1}} m_{D-1} \\
&\leq 2^{D-1} + 2^{C2^{D-1}} 2^{C2^{D-1}+D-1} && \left(\text{By } \binom{C}{C/2} \leq 2^C \text{ and inductive hypothesis}\right) \\
&\leq 2^{D-1} + 2^{C2^D+D-1} \\
&\leq 2^{C2^D+D} && \left(\text{By } 2^{D-1} \leq 2^{C2^D+D-1}\right)
\end{aligned}
$$

Finally, we conclude the (somewhat loose) bound of $2^{C2^{D+1}}$ on the number of vertices in our graph since $D \leq C \cdot 2^D$. ◀

Having established the basic properties of our instance, we now argue that (asymptotically) the best one can hope for on our instance is the trivial $O(CD)$-round schedule. As discussed in Section 3.1, we will prove this by arguing that for any fixed schedule, some sub-graph in G was correct in "guessing" which multicast trees were slowed down. In particular, we will argue that for any fixed schedule there is some smaller instance of simultaneous multicast which this schedule must solve as a sub-problem which takes at least $\frac{C(D-1)}{2}$ rounds but which the schedule does not start making progress towards solving until at least $\frac{C}{2}$ rounds have passed.

▶ **Lemma 4.4.** *The optimal schedule on $M_{\mathcal{S}}$ is of length at least $\frac{CD}{2}$.*

Proof. Fix an arbitrary simultaneous multicast schedule. We will prove by induction on D that $M_{\mathcal{S}}$ requires at least $\frac{CD}{2}$ rounds. The base case of $D = 1$ is trivial, as in this case $M_{\mathcal{S}}$ is a single edge with congestion C and so clearly requires at least $C \geq \frac{CD}{2}$ rounds.

For the inductive step, $D > 1$, suppose that for any partition $\mathcal{S}'$ of $C \cdot 2^{D-1}$ distinct labels into sets of size C, we have that $M_{\mathcal{S}'}$ requires at least $\frac{C(D-1)}{2}$ rounds. By definition of $M_{\mathcal{S}}$, any schedule which solves $M_{\mathcal{S}}$ can be projected in the natural way onto $M_{\mathcal{S}'}$ as a schedule which solves $M_{\mathcal{S}'}$ for any $\mathcal{S}' \in I(\mathcal{S})$. For example, any schedule which solves the instance in Figure 5 induces a schedule which when projected onto Figure 4 solves $M_{\mathcal{S}'}$ for each of the recursively constructed $M_{\mathcal{S}'}$. Even stronger, notice that $M_{\mathcal{S}}$ is created by combining the union of all $M_{\mathcal{S}'}$ for $\mathcal{S}' \in I(\mathcal{S})$ in such a way that any schedule which solves $M_{\mathcal{S}}$ must also send all messages from roots of trees in $M_{\mathcal{S}}$ to corresponding roots of trees in $M_{\mathcal{S}'}$ and solve $M_{\mathcal{S}'}$. That is, let r' be an arbitrary root for $M_{\mathcal{S}'}$ and let $\chi(r')$ be the labels associated with the one edge for root r' in $M_{\mathcal{S}'}$. Then, if we identify r' with v_i when constructing $M_{\mathcal{S}}$ then a schedule for $M_{\mathcal{S}}$ must both send r' the C messages of $\chi(r')$ from r_{2i-1} and r_{2i} and solve $M_{\mathcal{S}'}$. Thus, clearly the time our schedule takes is at least the time it takes to send one message in $\chi(r')$ to r' from r_{2i-1} and r_{2i} for some root in $M_{\mathcal{S}'}$ plus the time it takes to solve $M_{\mathcal{S}'}$. For example, if we let $\mathcal{S}' = (\{S, U\}, \{W, Y\})$ then any schedule which solves Figure 5 solves $M_{\mathcal{S}'}$ but before doing so must clearly send at least one message of $\{S, U\}$ to v_1 or at least one message $\{W, Y\}$ to v_2.

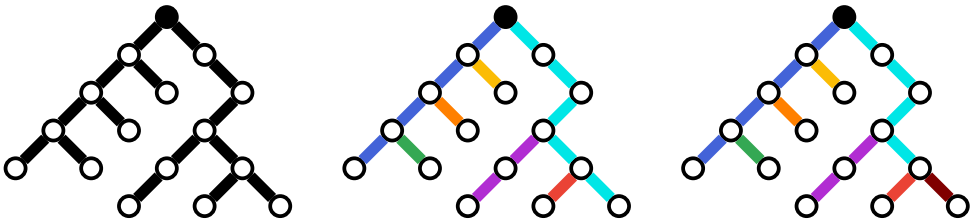
We now define $\bar{\mathcal{S}}'$ where $\bar{\mathcal{S}}' \in I(\mathcal{S})$ so that $M_{\bar{\mathcal{S}}'}$ is an instance of simultaneous multicast embedded in $M_{\mathcal{S}}$ which our fixed schedule must solve in order to solve $M_{\mathcal{S}}$ but which it does not start start solving until at least $C/2$ rounds have passed. In particular, consider what our fixed schedule does in the first $C/2$ rounds. As C messages must cross each edge e_j and only one such message can cross per rounds, there are some $C/2$ multicast trees whose message cannot cross e_j before $C/2$ rounds have passed; let S'_j be these "slow" trees for edge e_j and let

$$\bar{\mathcal{S}}' := (S'_{2i-1} \cup S'_{2i})_{i=1}^{2^{D-2}}$$

be a partition of the labels corresponding to these slow edges into sets of size C.

The tuple $\bar{S}'$ belongs to $I(\mathcal{S})$ and so, as discussed above, the fixed schedule must first send at least one message to a root in $M_{\bar{S}'}$ and then solve $M_{\bar{S}'}$. By definition of $\bar{S}'$, no messages arrive at roots of trees in $M_{\bar{S}'}$ until at least $\frac{C}{2}$ rounds have passed. On the other hand, by the inductive hypothesis, the latter sub-instance takes at least $\frac{C(D-1)}{2}$ additional rounds. Thus, the schedule must use at least $\frac{C(D-1)}{2} + \frac{C}{2} = \frac{CD}{2}$ rounds. ◀

Combining Lemmas 4.2, Lemma 4.3 and 4.4 and noting that we can always add dummy nodes to increase the number of vertices in our graph to a desired n immediately yields Theorem 1.1.



(a) Multicast tree T_i. **(b)** Heavy-light decomposition. **(c)** Cut heavy paths.

Figure 6 Our decomposition for multicast tree T_i. Each heavy path in Figure 6b and short path in Figure 6c drawn in different colors. Notice the far right path is cut into two short paths since its length is $5 > \log_2 n = 4$.

5 Existence of $O(C + D + \log^2 n)$-Length Schedules

Here we demonstrate that length $O(C + D + \log^2 n)$ simultaneous multicast schedules always exist. For this result we rely on heavy path decompositions, introduced by Sleator and Tarjan [42].

▶ **Defintion 5.1** (Heavy path decomposition [42]). *A* heavy path decomposition *of a rooted tree T is obtained as follows. First, each non-leaf node selects one* heavy *edge, which is an edge to a child with the greatest number of descendants (breaking ties arbitrarily). Other edges are termed* light. *We consider inclusion-wise maximal paths consisting of heavy edges, and for each highest node v of such a path p, we add to the path p the edge from v to its parent (if any). The obtained paths form the heavy path decomposition.*

It is easy to see that this is indeed a decomposition of the tree; that is, that each edge belongs to exactly one path in the heavy path decomposition. Moreover, each root-to-leaf path intersects at most $\log_2 n$ heavy paths, as each such path can have at most $\log_2 n$ light edges because the number of nodes in a subtree decreases by at least a factor of two every time one traverses down a light edge. This will allow us to decompose the trees into "short paths" such that each root-to-leaf path intersects few short paths. Specifically, we define a refinement of this decomposition in a top-down fashion, by breaking up each heavy path into short paths of length at most $\log_2 n$; that is, starting from the top of a heavy path of length l, we cut it into $\lceil l/\log n \rceil$ short paths. See Figure 6. Both the decomposition and its refinement exist, and are even computable deterministically in linear time.

As each root-to-leaf path intersects at most $\log_2 n$ heavy paths, this refined decomposition has each root-to-leaf path intersect at most $\frac{D}{\log_2 n} + \log_2 n$ short paths. We will refer to such a decomposition as a $(\log n, \log n)$-short (path) decomposition. We use this particular name as we generalize this notion further in Section 7 and the full version to (l, k)-decompositions for any integers k and l. This refined path decomposition together with some additional random delays will allow us to reduce the task of simultaneous multicast to that of $O\big(\frac{C}{\log n} + \frac{D}{\log n} + \log n\big)$ many simultaneous unicast instances with congestion and dilation $O(\log n)$, from which we obtain the following result. We illustrate the schedules in this result in Figure 7.

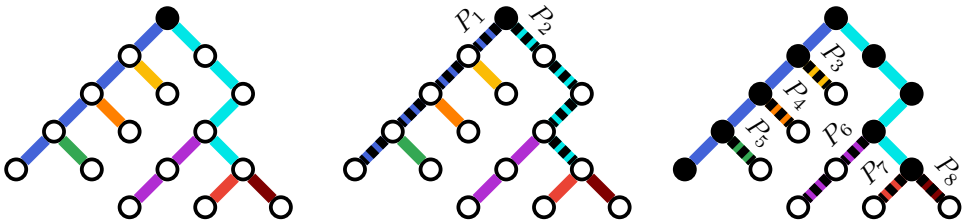
▶ **Theorem 1.2.** *Each simultaneous multicast instance with congestion C and dilation D in an n-node network admits a schedule of length at most $O(C + D + \log^2 n)$.*

Proof. We prove this result by means of the probabilistic method. First, we consider a $(\log n, \log n)$ decomposition of each multicast tree. For each short path p in the $(\log n, \log n)$-decomposition of a tree, we say p is at *level* j if there are exactly $j - 1$ other short paths

between p's root and the tree's root. That is, if we were to schedule a particular tree by forwarding along all paths of level $j = 1, 2, \ldots$ during $R = O(\log n)$ rounds, the path of level j would be scheduled in rounds $((j-1) \cdot R, j \cdot R]$, which we refer to as the j-th *frame*. Our goal will be to schedule the sets of short paths with limited congestion in parallel, using simultaneous unicast schedules guaranteed by [30].

In order to break up simultaneous multicast to multiple simultaneous unicasts, we shift the levels of each tree T_i by a random offset X_{T_i} chosen uniformly in $[C/\log n]$. Now a short path of level j in tree T_i will be scheduled during frame $j + X_{T_i}$. Since each edge e has congestion C, the expected number of paths of different trees that use e during any given frame is at most $O(\log n)$. So, by standard Chernoff concentration inequalities, the congestion of each edge during any frame is at most $O(\log n)$ w.h.p. Therefore, applying a union bound over all edges and time frames, we find that w.h.p., all edges have congestion at most $O(\log n)$ for all (shifted) frames $j = 1, 2, \ldots, \frac{C}{\log n} + \frac{D}{\log n} + \log n$ (recall that each root-to-leaf path intersects at most $\frac{D}{\log n} + \log n$ paths of length at most $\log n$). In particular, there exist random delays such that each time frame consists of a simultaneous unicast instance with congestion $C' = O(\log n)$ and dilation $D' = O(\log n)$. Therefore, by [30, 31], there exists a schedule of length $O(C' + D') = O(\log n)$ for these time frames' simultaneous unicasts. Combining these schedules one time frame after another, we obtain a schedule of length

$$\left(\frac{C+D}{\log n} + \log n\right) \cdot O(\log n) = O(C + D + \log^2 n). \qquad \blacktriangleleft$$



(a) Wait for T_i's random delay. **(b)** Run LMR on P_1, P_2. **(c)** Run LMR on $P_3, \ldots, P_8$.

Figure 7 Our multicast schedule on T_i using the decomposition from Figure 6. Nodes with m_i colored in black. Short unicast paths are dashed in black. "LMR" is the schedule given by [30].

The above proof can be made algorithmic, deterministic, and even allows for efficient distributed algorithms. See Section 7 for details.

6 Additive $\Omega(\log n)$ Necessary

In this section we use our $\Omega(CD)$ lower bound (Theorem 1.1) to show that any simultaneous multicast bound of the form $O(C + D) + f(n)$ must have $f(n) = \Omega(\log n)$ (Theorem 1.3). This result demonstrates the near optimality of the length $O(C + D + \log^2 n)$ schedules we gave in the previous section.

▶ **Theorem 1.3.** *Suppose there is a function f such that for any simultaneous multicast instance with congestion C and dilation D, there is a schedule delivering all packets in $O(C + D) + f(n)$ steps. Then $f(n) = \Omega(\log n)$.*

Proof of Theorem 1.3. Assume for the sake of contradiction that every simultaneous multicast instance admitted a schedule of length $\alpha(C+D)+f(n)$ for constant α and $f(n) = o(\log n)$.

Let $D = 4\alpha$ and let $C = \frac{\log n}{2^{4\alpha+1}}$ for n to be fixed later. Consider the simultaneous multicast instance given by $M_{\mathcal{S}}$ on graph $G_{\mathcal{S}}$ as defined in Section 4 whose properties are given by Theorem 1.1. Notice that $C2^{D+1} = \frac{\log n}{2^{4\alpha+1}} 2^{4\alpha+1} = \log n$ and so indeed we may apply Theorem 1.1.

Furthermore, by Theorem 1.1 we have that the optimal schedule of this simultaneous multicast instance with congestion C, dilation D and n nodes has length at least

$$L := \frac{CD}{2} = \frac{\alpha}{2^{4\alpha}} \log n.$$

But, by our assumption for contradiction we have that this instance admits a schedule of length at most

$$U := \alpha(C + D) + f(n) = \frac{\alpha}{2^{4\alpha+1}} \log n + 4\alpha^2 + f(n).$$

We then have a contradiction because $U < L$. In particular for n sufficiently large,

$$U - L = -\frac{\alpha}{2^{4\alpha+1}} \log n + 4\alpha^2 + o(\log n) < 0. \quad \blacktriangleleft$$

7 Algorithmic Results

In this section we present centralized and distributed algorithms for the computation of simultaneous multicast schedules of length $O(C + D + \log^2 n)$, as guaranteed to exist by Theorem 1.2.

7.1 Centralized Algorithm

It is easy to see that the probabilistic method proof in Theorem 1.2 yields a randomized algorithm which succeeds with high probability. Moreover, by standard limited independence methods [41], one can make this algorithm deterministic.

▶ **Theorem 7.1.** *There exists a deterministic, centralized algorithm which, given a simultaneous multicast instance, outputs a schedule of length $O(C + D + \log^2 n)$ in time polynomial in $|\mathcal{T}|$ and n.*

Proof. Let us begin by explaining why the proof of Theorem 1.2 immediately yields a polynomial-time randomized algorithm which succeeds with high probability. Recall that the schedules in Theorem 1.2 were produced by taking a heavy path decomposition of each T_i, delaying each T_i by $X_{T_i} \sim [C/\log n]$ and then concatenating together unicast schedules given by [30]. As noted in Section 5, a heavy path decomposition can be computed deterministically in polynomial (in fact, linear) time. Clearly, drawing a random delay from $[C/\log n]$ for each T_i is also doable in polynomial time by a randomized algorithm. Lastly, by [31], the schedules of [30] can be computed deterministically in polynomial time. By Theorem 1.2 the resulting schedule is of the appropriate length.

Let us now explain how this algorithm can be made deterministic. Let $n' = n + |\mathcal{T}|$. The only randomization used in the above algorithm is the random delays drawn from $[C/\log n]$. As with most proofs that show concentration by Chernoff bounds, it is easy to see that each X_{T_i} need only be $\frac{1}{\text{poly}n'}$-approximate, $O(\log n')$-wise independent for the above algorithm to succeed with high probability in n'. (For more background on limited independence, see [41].) Recalling that one can generate polynomially-many binary $\frac{1}{\text{poly}n'}$-approximate $O(\log n')$-wise independent random variables with only $O(\log n')$ random bits, our deterministic algorithm can simply brute force over all possible assignments to these $O(\log n')$ bits, and check if each resulting schedule is of the appropriate length. The result is a deterministic algorithm which is polynomial-time in $|\mathcal{T}|$ and n and outputs a schedule of the stated length. $\blacktriangleleft$

7.2 Distributed Algorithm

In this section we outline our distributed simultaneous multicast algorithm in the CONGEST model. Please see the full version for additional details.

In the classic CONGEST model of distributed communication [37], a network is modeled as an undirected simple n-node graph $G = (V, E)$. Communication is conducted over discrete, synchronous rounds. During each round each node can send an $O(\log n)$-bit message along each of its incident edges. Every node has an arbitrary and unique ID of $O(\log n)$ bits, first only known to itself (this is the KT_0 model of Awerbuch et al. [3]).

In the CONGEST model in a simultaneous multicast instance, each node initially knows a unique ID associated with each tree T_i to which it belongs, as well as which of its incident edges occur in which trees. We think of m_i in this setting as being an $O(\log n)$-bit message, which is therefore transmittable along an edge in a single round. As in the centralized version of the problem, initially only r_i knows m_i.

In the full version of this work we show CONGEST algorithms with the following guarantees exists.

▶ **Theorem 7.2.** *For any constant $\epsilon > 0$, there exists a CONGEST algorithm which given access to shared randomness solves simultaneous multicast in time*

$$O\left((C+D)\cdot\left(1+\frac{\log\min\{C,D\}}{\log\log n}\right)+\log^{2+\epsilon} n\right).$$

with high probability. If nodes also know their depth in each tree, then there exists another CONGEST algorithm which solves simultaneous multicast in $O(C + D + \log^{2+\epsilon})$ time.

As noted in the introduction, simultaneous multicast has proven to be a crucial subroutine in many recent algorithms in CONGEST for fundamental problems like MST, shortest path and approximate min cut. Therefore, improving simultaneous multicast in the CONGEST model is an important step towards obtaining better algorithms for many of these fundamental problems. Furthermore, in the above applications of simultaneous multicast the parameters C and D are equal to the diameter of the graph up to polylogarithmic in n terms, provided the input graph has certain structure such as being planar [14, 18, 16, 19]. If C and D are sufficiently large polylogarithmic terms, i.e., $\max\{C, D\} = \Omega(\log^{2+\epsilon} n)$, then, assuming nodes know their heights, our distributed algorithm gives an optimal $O(C + D)$ time distributed algorithm. Thus, we view our distributed algorithm as an important step towards obtaining better algorithms for many distributed problems, including MST, shortest path and approximate minimum cut.

Before proceeding, let us discuss the preprocessing assumptions in Theorem 7.2. Our distributed algorithms assume nodes have access to shared randomness or to their height in each of their incident multicast trees. Both of these assumptions can be dispensed with provided nodes are allowed to do some preprocessing: see [11] for how to share randomness and note that nodes can compute their heights by a single simultaneous multicast computation where we could, for example, use the aforementioned $O(C + D \log n)$ length schedules. If this preprocessing is performed only once and many simultaneous multicasts are performed, the preprocessing step's cost amortizes away. Thus, provided nodes share randomness we have that after a preprocessing step equivalent to the current state of the art distributed simultaneous multicast algorithm, subsequent simultaneous multicasts can be performed in time $O(C + D + \log^{2+\epsilon} n)$, which as discussed earlier, is essentially as close as one can get to a bound of $O(C + D)$.

7.2.1 Intuition and Overview

Here we provide an intuition for and an overview of our distributed algorithms. As mentioned earlier, Ostrovsky and Rabani [34] provided a distributed algorithm for simultaneous unicast using $O(C+D+\log^{1+\epsilon} n)$ rounds. Since our centralized algorithm has shown that simultaneous multicast can be reduced to simultaneous unicast by way of a $(\log n, \log n)$-short decomposition (i.e. a heavy path decomposition), the focus of our distributed algorithm is the efficient distributed computation of a $(\log n, \log n)$-short decomposition.

The challenge of computing such a decomposition in a distributed manner is that it seems as hard as solving simultaneous multicast. In particular, computing a heavy path decomposition requires that every node in a T_i aggregate information from all of its children. It is not hard to see that performing such a "convergecast" at every node can be seen as performing a multicast on every T_i in reverse. Even worse, the message size sent by nodes to their parents in such a convergecast to compute a heavy path decomposition must consist of $\log_2 n$ bits to count the size of their sub-tree; i.e. sending just one such message fully uses the bandwidth of a CONGEST link in one round. Thus, it seems that if we want to solve simultaneous multicast by using a $(\log n, \log n)$-short decomposition, then we must circularly solve a simultaneous convergecast – i.e. simultaneous multicast in reverse – in which large messages must be sent.

However, we show that, in fact, one can compute what is essentially a $(\log n, \log n)$-short decomposition more efficiently than one can solve simultaneous multicast. In particular, we show how to efficiently compute what we call a $(\log^{1+\epsilon} n, \log n)$-short decomposition. We formally define these decompositions in the full version of this work but for now note that they are a simple relaxation of heavy path decompositions. We demonstrate that a $(\log^{1+\epsilon} n, \log n)$-short decomposition for every T_i can be efficiently computed in a distributed fashion by using what we call a rank-decomposition – defined in the full version. Computing a rank-decomposition will require nodes to send exponentially fewer bits to their parents than computing a heavy path decomposition. That is, it will require that each node send only $O(\log \log n)$ bits to its parent rather than the $O(\log n)$ bits needed to encode the size of a subtree by requiring nodes to send information about their "rank" rather than their subtree size. By exploiting this exponential decrease in the total number of bits that must be passed, we are able to efficiently pack rank information into messages and compute a $(\log^{1+\epsilon} n, \log n)$-short decomposition. Specifically, by using random delays each edge transmits a message for $O(\log n/\log \log n)$ trees to which it belongs with high probability. Thus, we are able to to transmit the $O(\log \log n)$ bits of $O(\log n/\log \log n)$ trees for each edge using the $O(\log n)$ bits of one round of CONGEST. This becomes more challenging if an edge does not know which trees' messages are being sent which is what incurs the $\log(\min\{C, D\})/\log \log n)$ factor. After computing the above decomposition in this way we are then able to translate our centralized algorithm to the distributed setting by making use of the distributed simultaneous unicast algorithms of [34].

8 Future Directions

We conclude our paper with future directions for work in the scheduling of simultaneous multicasts. Of course, one can try and tighten the polylogarithmic additive terms in our results. More interestingly, one could extend the simultaneous multicast setting in ways similar to how the simultaneous unicast scheduling work of Leighton et al. [30, 31] has been extended.

We give two notable examples. First, one could study what sort of approximation algorithms are possible if one is permitted to choose the trees over which multicast is performed as was done in the simultaneous unicast setting [43, 5, 28]. Roughly speaking, this corresponds to a depth-bounded version of the multicast congestion problem [45, 7, 26]. We point out that choices of trees with optimal congestion + dilation (or nearly-optimal, up to constant multiplicative and additive polylogarithmic terms) combined with our algorithm to output length $O(C + D + \log^2 n)$-length schedules would imply near-optimal simultaneous multicasts for this setting. Second, we note that our schedules have logarithmic-sized edge queues. That is, messages may have to wait up to $\Theta(\log n)$ rounds before being sent over an edge. This is not due to our use of the schedules of Leighton et al. [30], whose queue sizes are constant, but rather due to $\Theta(\log n)$ messages arriving to a node by the end of simultaneous unicast frames used in our schedules. An interesting open question is whether there exist efficient simultaneous multicast schedules which minimize both time and edges' queue sizes.

References

1 Noga Alon and Joel H Spencer. *The probabilistic method*. John Wiley & Sons, 2016.

2 Friedhelm Meyer auf der Heide and Berthold Vöcking. Shortest-path routing in arbitrary networks. *Journal of Algorithms*, 31(1):105–131, 1999.

3 Baruch Awerbuch, Oded Goldreich, Ronen Vainish, and David Peleg. A trade-off between information and communication in broadcast protocols. *Journal of the ACM (JACM)*, 37(2):238–256, 1990.

4 Amotz Bar-Noy, Sudipto Guha, Joseph Naor, and Baruch Schieber. Message multicasting in heterogeneous networks. *SIAM Journal on Computing (SICOMP)*, 30(2):347–358, 2000.

5 Dimitris Bertsimas and David Gamarnik. Asymptotically optimal algorithms for job shop scheduling and packet routing. *Journal of Algorithms*, 33(2):296–318, 1999.

6 Costas Busch, Malik Magdon-Ismail, Marios Mavronicolas, and Paul Spirakis. Direct routing: Algorithms and complexity. In *Proceedings of the 12th Annual European Symposium on Algorithms (ESA)*, pages 134–145, 2004.

7 Robert Carr and Santosh Vempala. Randomized metarounding. *Random Structures & Algorithms*, 20(3):343–352, 2002.

8 Yang Chu, Sanjay Rao, Srinivasan Seshan, and Hui Zhang. Enabling conferencing applications on the internet using an overlay muilticast architecture. *ACM SIGCOMM computer communication review*, 31(4):55–67, 2001.

9 Michal Dory and Mohsen Ghaffari. Improved distributed approximations for minimum-weight two-edge-connected spanning subgraph. In *Proceedings of the 38th ACM Symposium on Principles of Distributed Computing (PODC)*, pages 521–530, 2019.

10 Michael Elkin and Guy Kortsarz. Sublogarithmic approximation for telephone multicast: path out of jungle. In *Proceedings of the 14th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 76–85, 2003.

11 Mohsen Ghaffari. Distributed broadcast revisited: Towards universal optimality. In *Proceedings of the 42nd International Colloquium on Automata, Languages and Programming (ICALP)*, pages 638–649. Springer, 2015.

12 Mohsen Ghaffari. Near-optimal scheduling of distributed algorithms. In *Proceedings of the 34th ACM Symposium on Principles of Distributed Computing (PODC)*, pages 3–12, 2015.

13 Mohsen Ghaffari and Bernhard Haeupler. Distributed algorithms for planar networks I: Planar embedding. In *Proceedings of the 35th ACM Symposium on Principles of Distributed Computing (PODC)*, pages 29–38, 2016.

14 Mohsen Ghaffari and Bernhard Haeupler. Distributed algorithms for planar networks II: Low-congestion shortcuts, mst, and min-cut. In *Proceedings of the 27th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 202–219, 2016.

15 Mohsen Ghaffari and Bernhard Haeupler. Low-congestion shortcuts for graphs excluding dense minors. *arXiv preprint*, 2020. `arXiv:2008.03091`.

16 Mohsen Ghaffari and Jason Li. New distributed algorithms in almost mixing time via transformations from parallel algorithms. In *Proceedings of the 32nd International Symposium on Distributed Computing (DISC)*, pages 31:1–31:16, 2018.

17 Bernhard Haeupler, D Ellis Hershkowitz, and David Wajc. Round-and message-optimal distributed graph algorithms. In *Proceedings of the 37th ACM Symposium on Principles of Distributed Computing (PODC)*, pages 119–128, 2018.

18 Bernhard Haeupler, Taisuke Izumi, and Goran Zuzic. Low-congestion shortcuts without embedding. In *Proceedings of the 35th ACM Symposium on Principles of Distributed Computing (PODC)*, pages 451–460, 2016.

19 Bernhard Haeupler, Taisuke Izumi, and Goran Zuzic. Near-optimal low-congestion shortcuts on bounded parameter graphs. In *Proceedings of the 30th International Symposium on Distributed Computing (DISC)*, pages 158–172, 2016.

20 Bernhard Haeupler, Jason Li, and Goran Zuzic. Minor excluded network families admit fast distributed algorithms. In *Proceedings of the 37th ACM Symposium on Principles of Distributed Computing (PODC)*, pages 465–474, 2018.

21 Bernhard Haeupler, David Wajc, and Goran Zuzic. Universally-optimal distributed algorithms for known topologies. In *Proceedings of the 53rd Annual ACM Symposium on Theory of Computing (STOC)*, 2021. To appear.

22 Stephan Holzer and Roger Wattenhofer. Optimal distributed all pairs shortest paths and applications. In *Proceedings of the 31st ACM Symposium on Principles of Distributed Computing (PODC)*, pages 355–364, 2012.

23 Jennifer Iglesias, Rajmohan Rajaraman, R Ravi, and Ravi Sundaram. Rumors across radio, wireless, telephone. In *35th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS)*, 2015.

24 Jennifer Iglesias, Rajmohan Rajaraman, R Ravi, and Ravi Sundaram. Plane gossip: Approximating rumor spread in planar graphs. In *Proceedings of the 13th Latin American Theoretical Informatics Symposium (LATIN)*, pages 611–624, 2018.

25 John Jannotti, David K Gifford, Kirk L Johnson, M Frans Kaashoek, et al. Overcast: reliable multicasting with on overlay network. In *Proceedings of the 4th USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, page 14, 2000.

26 K Jansen and H Zhang. An approximation algorithm for the multicast congestion problem via minimum steiner trees. In *Proceedings of the 3rd International Workshop on Approximation Algorithms for Combinatorial Optimization Problems (APPROX)*, pages 77–90, 2002.

27 Naoki Kitamura, Hirotaka Kitagawa, Yota Otachi, and Taisuke Izumi. Low-congestion shortcut and graph parameters. In *Proceedings of the 33rd International Symposium on Distributed Computing (DISC)*, pages 25:1–25:17, 2019.

28 Ronald Koch, Britta Peis, Martin Skutella, and Andreas Wiese. Real-time message routing and scheduling. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques*, pages 217–230. Springer, 2009.

29 Shimon Kogan and Merav Parter. Low-congestion shortcuts in constant diameter graphs. In *Proceedings of the 40th ACM Symposium on Principles of Distributed Computing (PODC)*, 2021. To appear.

30 Thomson Leighton, Bruce M Maggs, and Satish B Rao. Packet routing and job-shop scheduling in O(congestion+ dilation) steps. *Combinatorica*, 14(2):167–186, 1994.

31 Tom Leighton, Bruce Maggs, and Andrea W Richa. Fast algorithms for finding o (congestion+ dilation) packet routing schedules. *Combinatorica*, 19(3):375–401, 1999.

32 Christoph Lenzen and David Peleg. Efficient distributed source detection with limited bandwidth. In *Proceedings of the 32nd ACM Symposium on Principles of Distributed Computing (PODC)*, pages 375–382, 2013.

33 Friedhelm Meyer and Berthold Vöcking. A packet routing protocol for arbitrary networks. In *Proceedings of the 12th International Symposium on Theoretical Aspects of Computer Science (STACS)*, pages 291–302. Springer, 1995.

34 Rafail Ostrovsky and Yuval Rabani. Universal $O(congestion+dilation+\log^{1+\epsilon} n)$ local control packet switching algorithms. In *Proceedings of the 29th Annual ACM Symposium on Theory of Computing (STOC)*, volume 29, pages 644–653, 1997.

35 Britta Peis, Martin Skutella, and Andreas Wiese. Packet routing: Complexity and algorithms. In *Proceedings of the 7th Workshop on Approximation and Online Algorithms (WAOA)*, pages 217–228, 2009.

36 Britta Peis and Andreas Wiese. Universal packet routing with arbitrary bandwidths and transit times. In *Proceedings of the 13th Conference on Integer Programming and Combinatorial Optimization (IPCO)*, pages 362–375, 2011.

37 David Peleg. Distributed computing. *SIAM Monographs on discrete mathematics and applications*, 5:1–1, 2000.

38 Yuval Rabani and Éva Tardos. Distributed packet switching in arbitrary networks. In *Proceedings of the 28th Annual ACM Symposium on Theory of Computing (STOC)*, volume 96, pages 366–375, 1996.

39 Thomas Rothvoß. A simpler proof for $O(Congestion+Dilation)$ packet routing. In *Proceedings of the 16th Conference on Integer Programming and Combinatorial Optimization (IPCO)*, pages 336–348, 2013.

40 Christian Scheideler. *Universal routing strategies for interconnection networks*, volume 1390. Springer, 2006.

41 Jeanette P Schmidt, Alan Siegel, and Aravind Srinivasan. Chernoff–hoeffding bounds for applications with limited independence. *SIAM Journal on Discrete Mathematics*, 8(2):223–250, 1995.

42 Daniel D Sleator and Robert Endre Tarjan. A data structure for dynamic trees. *Journal of Computer and System Sciences*, 26(3):362–391, 1983.

43 Aravind Srinivasan and Chung-Piaw Teo. A constant-factor approximation algorithm for packet routing and balancing local vs. global criteria. *SIAM Journal on Computing (SICOMP)*, 30(6):2051–2068, 2001.

44 Donald M. Topkis. Concurrent broadcast for information dissemination. *IEEE Transactions on Software Engineering*, SE-11(10):1107–1112, 1985.

45 Santosh Vempala and Berthold Vöcking. Approximating multicast congestion. In *Proceedings of the 10th Annual International Symposium on Algorithms and Computation (ISAAC)*, pages 367–372, 1999.

Analysis of Smooth Heaps and Slim Heaps

Maria Hartmann
Institut für Informatik, Freie Universität Berlin, Germany

László Kozma
Institut für Informatik, Freie Universität Berlin, Germany

Corwin Sinnamon
Department of Computer Science, Princeton University, NJ, USA

Robert E. Tarjan
Department of Computer Science, Princeton University, NJ, USA
Intertrust Technologies, Sunnyvale, CA, USA

Abstract

The smooth heap is a recently introduced self-adjusting heap [Kozma, Saranurak, 2018] similar to the pairing heap [Fredman, Sedgewick, Sleator, Tarjan, 1986]. The smooth heap was obtained as a heap-counterpart of Greedy BST, a binary search tree updating strategy conjectured to be *instance-optimal* [Lucas, 1988], [Munro, 2000]. Several adaptive properties of smooth heaps follow from this connection; moreover, the smooth heap itself has been conjectured to be instance-optimal within a certain class of heaps. Nevertheless, no general analysis of smooth heaps has existed until now, the only previous analysis showing that, when used in *sorting mode* (n insertions followed by n delete-min operations), smooth heaps sort n numbers in $O(n \lg n)$ time.

In this paper we describe a simpler variant of the smooth heap we call the *slim heap*. We give a new, self-contained analysis of smooth heaps and slim heaps in unrestricted operation, obtaining amortized bounds that match the best bounds known for self-adjusting heaps. Previous experimental work has found the pairing heap to dominate other data structures in this class in various settings. Our tests show that smooth heaps and slim heaps are competitive with pairing heaps, outperforming them in some cases, while being comparably easy to implement.

2012 ACM Subject Classification Theory of computation → Data structures design and analysis

Keywords and phrases data structure, heap, priority queue, amortized analysis, self-adjusting

Digital Object Identifier 10.4230/LIPIcs.ICALP.2021.79

Category Track A: Algorithms, Complexity and Games

Funding *Maria Hartmann*: Work supported by DFG grant KO 6140/1-1.
László Kozma: Work supported by DFG grant KO 6140/1-1.
Robert E. Tarjan: Research at Princeton University partially supported by an innovation research grant from Princeton and a gift from Microsoft.

1 Introduction

A heap (priority queue) data structure stores a collection of items, each with an associated real-valued key (priority). Operations include inserting an item, deleting an item with smallest key, decreasing the key of an item, or melding (merging) two heaps. Heaps are among the most basic and well-studied structures in computer science, with applications in sorting, event simulation, graph algorithms, and many other settings (see e.g. [2, 17] and the references therein).

Williams' implicit binary heap [27] and its variants remain the simplest, and in many applications, the fastest implementation. Carefully engineered *sequence-based* heaps [22] tend to be even faster in several (although not all) practical scenarios, because of their better use of caching.

To implement efficient *meld* and to obtain theoretically optimal running times for *insert* and *decrease-key*, more sophisticated data structures have been proposed. The Fibonacci heap [10] was the first heap implementation to obtain the asymptotically optimal amortized bounds of $O(\lg n)$ time for *delete-min* and *delete* in a size-n heap, and $O(1)$ time for all other operations. (We denote by lg the base-two logarithm.) A large number of alternative designs that match these bounds have been explored throughout the years, see e.g. [11] and the references therein. Due to their relatively complex implementation, Fibonacci heaps and other theoretically optimal heaps have been found to be slower in practice than simpler heaps with sub-optimal guarantees [17].

Self-adjusting heaps. Is it possible to design data structures that are simple and efficient in practice but also efficient in theory, at least in an amortized sense? Self-adjusting data structures attempt to achieve this goal by allowing a very flexible structure with no (or very little) bookkeeping. They react to user operations with local structural readjustments, attempting to bring the data structure into a more favorable state for future operations. Examples include the *splay tree* [24], a self-adjusting binary search tree, and the *pairing heap* [9], a self-adjusting heap.

A pairing heap is a rooted, ordered tree (the children of each node are ordered) whose nodes are the heap items, arranged in (*min-*)*heap* order: the parent of a node x has key no greater than that of x. Heap order implies that the root has minimum key. A *delete-min* operation deletes the root, thereby making each of its children into a root. These new roots retain the order they had when they were children of the deleted root. The *delete-min* combines the new roots into a single tree by *linking* them two at a time. One link operation combines two adjacent roots, making the one with larger key the new *leftmost* child of the other. In its original variant, the pairing heap combines roots by first linking pairs of adjacent roots (the first and second, the third and fourth, etc.) from left to right, and then linking the remaining roots consecutively from right to left. This variant performs all operations in $O(\lg n)$ amortized time [9]. Although it was originally conjectured that pairing heaps match the theoretical guarantees of Fibonacci-heaps, Fredman [8] and Iacono and Özkan [14] showed that the amortized time of *decrease-key* in pairing heaps is $\Omega(\lg \lg n)$, and that the same lower bound holds for broader classes of self-adjusting heaps.

The exact amortized complexity of the *decrease-key* operation in pairing heaps remains unknown [13, 21, 5][1]. In practice, however, pairing heaps have been found to dominate Fibonacci heaps and other theoretically optimal heaps across a wide range of experimental settings, and for some specific types of workloads (e.g. with a high frequency of *decrease-key* operations) they even compare favorably with implicit heaps [17]. The pairing heap can thus be considered a "robust choice" [23] in a variety of applications.

In addition, pairing heaps and other self-adjusting data structures hold the promise of *adaptivity*, i.e. the ability to take advantage of regularities or biases in the usage pattern, which may give improved performance in specific scenarios. The adaptivity of splay trees has inspired a long line of research and is the subject of the *dynamic optimality conjecture* [24, 18]. Adaptivity in heaps is relatively less studied (see e.g. [16] and references therein).

Smooth heaps and slim heaps. The recently introduced *smooth heap* [16] is a self-adjusting heap with a structure similar to that of the pairing heap. Like pairing heaps, smooth heaps are built of rooted, ordered trees combined by links. The two data structures differ however,

[1] The lack of structure of self-adjusting data structures makes their analysis both difficult and interesting.

both in the order in which they do links, and in the implementation of the linking primitive itself. Smooth heaps use *stable linking*: when linking two adjacent roots, the one with larger key becomes the leftmost – or the *rightmost* – child of the other, depending on the original left-to-right order of siblings.

The smooth heap was obtained from a correspondence between heaps and binary search trees. In sorting mode (n insertions followed by n delete-min operations), the smooth heap has the same asymptotic running time as Greedy BST, a self-adjusting binary search tree strategy[2]. Greedy BST was proposed as a candidate *dynamically optimal* search tree implementation [19, 20, 4]. In its original form, Greedy BST needs information about *future* queries. Remarkably, this requirement can be removed (with a constant factor slowdown) [4]. Nonetheless, Greedy BST should be seen as a theoretical "proof of concept" that is largely impractical, as compared to splay trees, which are simple and widely used in practice. Surprisingly, despite its correspondence to Greedy BST, the smooth heap is simple and easy to implement. In particular, operations do not require knowledge of the future. The pointer structure and low-level complexity of operations in smooth heaps are comparable to those in pairing heaps.

In [16], several bounds were shown for the time of operations on smooth heaps used in sorting mode, by transferring results known for Greedy BST. The correspondence with Greedy BST breaks down when additional operations are supported, or even when *insert* and *delete-min* operations are intermixed. The smooth heap may be a viable general-purpose data structure, but so far it has lacked a complete, self-contained analysis that covers unrestricted operations.

In this paper we provide such an analysis. We begin by describing the implementation of all standard heap operations, some of which were not fully specified in [16]. We also describe and analyze the variant of smooth heaps that uses unstable (classical) linking, which we call the *slim heap*. The bounds we obtain for smooth heaps and slim heaps match the best known bounds for any self-adjusting heap, and differ from those of Fibonacci heaps only in the $O(\lg \lg n)$ bound (versus $O(1)$) for *decrease-key* (see Table 1). Despite the similarities between pairing heaps, smooth heaps, and slim heaps, the analysis of pairing heaps does not directly transfer to smooth heaps or slim heaps, since the behaviours of the data structures differ. Indeed, even minor variants of the standard pairing heap are difficult to analyze, with open questions remaining. See e.g. [5]. On the other hand, our analysis of smooth heaps and slim heaps does use a variant of the potential function used to analyze pairing heaps.

Our $O(\lg \lg n)$ bound for *decrease-key* in smooth heaps and slim heaps matches the lower bounds of Fredman [8] and Iacono and Özkan [14], but our implementation, which is based on Elmasry's [6] implementation for pairing heaps, violates the assumptions of these bounds, so they do not in fact apply to our algorithm. It remains open whether these lower bounds can be extended to encompass an Elmasry-type implementation. A related question is whether there is *any* self-adjusting heap with an $O(\lg \lg n)$ bound for *decrease-key* that satisfies the assumptions of these lower bounds. The *sort heap* [14] satisfies the assumptions of the Iacono-Özkan bound, but its analysis is flawed [15], [12, § 6]. We also give a simple implementation of *decrease-key* in smooth heaps and slim heaps based on the original implementation for pairing heaps. This implementation satisfies the assumptions of both lower bounds.

We complement our theoretical results with a brief experimental study (Section 6). Since the pairing heap was previously found to have good experimental performance and has been extensively compared with other heaps, we limit ourselves to comparing smooth heaps and

[2] The correspondence is subtle: going from the smooth heap to Greedy BST involves *inverting* and *reversing* the permutation that is being sorted.

slim heaps with pairing heaps. Our initial results show that smooth heaps and slim heaps are competitive with pairing heaps, outperforming pairing heaps in some cases, particularly for some structured usage patterns. This aligns with the adaptive properties shown previously in an asymptotic sense for smooth heaps. Our results suggest the smooth heap or slim heap as a drop-in alternative heap in applications where the pairing heap works well.

2 Smooth Heaps and Slim Heaps

In this section, we introduce the smooth heap and its variant, the slim heap, and state our efficiency bounds. Both of these data structures store a collection of nodes, each node having a real-valued key (not necessarily distinct for different nodes). They support the following standard heap operations:

- *make-heap*(h): Create a new, empty heap h.
- *find-min*(h): Return a node of smallest key in heap h, or null if h is empty.
- *insert*(x, h): Insert node x with predefined key into heap h. Node x must be in no other heap.
- *delete-min*(h): Delete from h the node that would be returned by *find-min*(h).
- *meld*(h, h'): Meld node-disjoint heaps h and h'.
- *decrease-key*(x, k, h): Decrease to k the key of node x in heap h, assuming that x is a node in h and k is no larger than the current key of x. (Operation *decrease-key* is given a pointer to node x in heap h, not just its key or some other identifier.)
- *delete*(x, h): Delete node x from heap h, assuming that x is a node in heap h. Again, *delete* is given a pointer to x in h.

Table 1 The best time bounds known for smooth, slim, Fibonacci, and pairing heaps. All bounds are amortized. Smooth heaps are competitive with the best variants of pairing heaps. They fall short of Fibonacci heaps on *decrease-key* (though self-adjusting heaps often perform better in practice [17]). In all cases, *make-heap* and *find-min* take $O(1)$ time, and *delete* takes $O(\lg n)$ time. Strict Fibonacci heaps [3] achieve the same bounds as Fibonacci heaps, but without amortization.

	insert	*delete-min*	*meld*	*decrease-key*
Smooth Heaps (this paper)	$O(1)$	$O(\lg n)$	$O(1)$	$O(\lg \lg n)$
Slim Heaps (this paper)	$O(1)$	$O(\lg n)$	$O(1)$	$O(\lg \lg n)$
Fibonacci Heaps [10]	$O(1)$	$O(\lg n)$	$O(1)$	$O(1)$
Pairing Heaps [9, 25, 13]	$O(1)$	$O(\lg n)$	$O(1)$	$O(\lg n)$
Alternative Analysis (Pettie [21])	$O(4^{\sqrt{\lg \lg n}})$	$O(\lg n)$	$O(4^{\sqrt{\lg \lg n}})$	$O(4^{\sqrt{\lg \lg n}})$
Elmasry Pairing Heaps [6][3]	$O(1)$	$O(\lg n)$	$O(\lg \lg n)$	$O(\lg \lg n)$

2.1 Efficiency

Table 1 states our time bounds for smooth heaps and summarizes known results for some competing heaps. In stating these bounds, we assume that $n \geq 4$, so that $\lg \lg n \geq 1$: if $n < 4$ all operations take $O(1)$ time. We shall prove the following theorem:

[3] Elmasry [6] also claims a *meld* time of $O(1)$ with a *delete-min* time of $O(\lg n + \lg \lg N)$, where N is the number of items in all heaps.

▶ **Theorem 1.** *In a smooth heap or slim heap,* delete-min *and* delete *take* $O(\lg n)$ *amortized time,* decrease-key *takes* $O(\lg \lg n)$ *amortized time, and* make-heap*,* find-min*,* insert*, and* meld *take* $O(1)$ *time, where there are* n *nodes in the heap at the time of operation.*

In some applications, a large number of nodes may be inserted and then never accessed, for example if one inserts many nodes with large keys, and then repeatedly inserts and deletes a few nodes with small keys. For this case, we shall prove the following theorem, which states that nodes that are never deleted do not slow down deletions:

▶ **Theorem 2.** *Operations* delete-min *and* delete *take* $O(\lg t)$ *amortized time, with the other amortized bounds unchanged, where* t *is the number of nodes in the heap at the time of the operation that will be deleted in the future.*

2.2 Data structures and terminology

A smooth heap or slim heap consists of a forest of rooted, ordered trees whose nodes are the nodes of the heap. Each tree is (min-)heap ordered: the key of a non-root node is no less than that of its parent. The roots of the trees are stored in a list, with a root of minimum key, the *min-root*, first. Each node stores a list of its children. For both the root list and the lists of children, we identify the front of the list with "left" and the back of the list with "right", so that nodes early in the list are considered left of nodes later in the list.

The forest is altered by *linking* pairs of *adjacent* nodes in the list of roots or in a list of children. A link makes the node with smaller key (the *winner* of the link) the parent of the node of larger key (the *loser* of the link). If the keys are equal, the node on the right is the winner. The link is a *left link* if the loser is originally left of the winner, a *right link* if the loser is originally right of the winner.

The only difference between smooth and slim heaps is the position of the loser in its list of new siblings after a link. (See Figure 1.) Slim heaps use *one-sided* links: the loser of the link becomes the new leftmost child of the winner. This is the type of linking used in Fibonacci heaps, pairing heaps, and many other similar data structures.

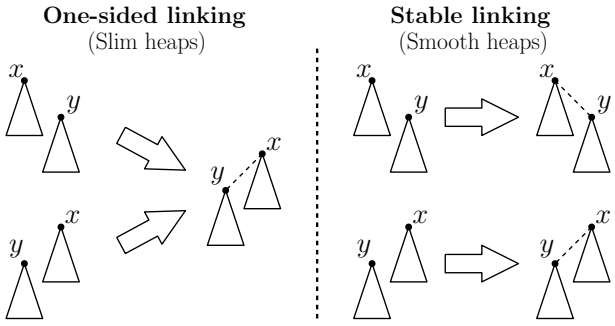


Figure 1 The result of a link of adjacent nodes x and y, where x has the smaller key, using one-sided linking (left) and stable linking (right). A one-sided link always adds the child on the left, whereas a stable link preserves left-to-right order. Slim heaps use one-sided links, smooth heaps use stable links. In both smooth and slim heaps, all links are between adjacent nodes.

Smooth heaps use *stable* links: the loser becomes the new leftmost or rightmost child of the winner, depending on whether the link is a left link or a right link. Stable links maintain the left-to-right order of nodes (although *insert*, *meld*, and *decrease-key* operations perturb it, as we shall see). A child of a node is either a *left* child or a *right* child, depending on whether it was left or right of its parent just before they were linked. In a smooth heap all left children in a list of children precede all right children; in a slim heap, left and right children are in general intermixed.

2.3 Operations

In this subsection we describe the implementation of all the heap operations except *decrease-key* and *delete*; we cover these in Section 4. The description of smooth heaps in [16] did not include a find-min operation, nor was the implementation of *decrease-key* fully specified. In addition to adding these operations, we have changed some details of the original presentation.

As noted in the last subsection, slim heaps differ from smooth heaps only in the linking method. The following descriptions apply to both smooth heaps and slim heaps if links are done using the appropriate linking method, stable for smooth heaps, one-sided for slim heaps.

We store the root list of a heap in a circular singly-linked list, with the min-root first (leftmost). Access is via the min-root. Circular linking supports melding in $O(1)$ time. We store each list of children in a singly-linked list, circular for smooth heaps. In a slim heap, access to a list of children is via the leftmost child. In a smooth heap, access to such a list is via the *rightmost* child; circular linking allows a new child to be added as the leftmost or rightmost in $O(1)$ time. Each node needs two pointers: to its right adjacent root or sibling, and to its leftmost (slim heap) or rightmost (smooth heap) child. To support *delete* and *decrease-key* we shall add a third pointer per node. See Section 4.

- *make-heap*(h): Create and return an empty heap.
- *find-min*(h): Return the min-root of h.
- *insert*(x, h): Make x into a one-node tree and insert x into the root list of h, in first or second position depending on whether its key is less than or not less than that of the min-root of h; in the former case x becomes the new min-root.
- *meld*(h, h'): Catenate the root lists of h and h'; set the min-root of the melded heap to that of h or h', whichever has smaller key.
- *delete-min*(h): Delete the min-root x of h. Replace x in the list of roots of h by the list of children of x, in the order they occur in the list of children. These *new roots* precede all the undeleted (*old*) roots in the root list. Repeatedly link pairs of adjacent roots until there is only one root remaining, using the *leftmost locally maximum linking rule* given below. Make the remaining root the min-root of h.

The main novelty in smooth heaps and slim heaps is the leftmost locally maximum linking rule used in *delete-min*. It is: find the leftmost node v in the root list whose key is no less than those of both adjacent nodes, say u and w, and link v with whichever of u and w has larger key, breaking a tie in favor of the node left of v. As special cases, if the leftmost root has key no less than that of the right adjacent root, link these roots (the right one is the winner); if the keys of all the roots are in strictly increasing order left-to-right, link the rightmost root with the left adjacent root (the left one is the winner). One can eliminate the special cases by adding dummy leftmost and rightmost nodes, both with key minus infinity, and applying the leftmost locally maximum linking rule until there is only one non-dummy root.)

To implement leftmost locally maximum linking, start at the leftmost root and proceed rightward until finding a root v whose key is not less than that of its right adjacent root w, or until reaching the rightmost root. In the latter case, link the two rightmost roots and repeat until there is only one root, completing the linking. In the former case, link v with whichever of the adjacent roots has higher key, choosing the left adjacent node in case of a tie and linking with w (the right adjacent node) if v is the leftmost root. Then repeat, but starting from the winner of the link: all roots preceding this winner are in strictly decreasing order by key, left to right, and hence none is the leftmost local maximum.

Leftmost locally maximum linking *treapifies* the list of roots: it arranges the roots into a binary tree symmetrically ordered by root list order and heap-ordered by key, with ties broken in favor of nodes on the right: if x is a node, its new left child, if any, has no smaller key, and its new right child, if any, has strictly greater key.

In both linking and in the definition of "leftmost local maximum" our tie-breaking rule in key comparisons is that the node on the right is treated as having smaller key. Other tie-breaking rules work equally well: what is required is consistency (the outcome of a key comparison between two nodes must remain the same throughout a *delete-min*) and transitivity (if x, y, and z have equal keys and the tie-breaking rule declares that x has smaller key than y and y has smaller key than z, then it must declare that x has smaller key than z). An alternative tie-breaking rule with these properties is to break ties by node identifier. The tie-breaking rule can depend on the keys, the nodes, and the position of the nodes in the root list. Leftmost locally maximum linking is easy to implement, but a more general linking rule does the same links if the tie-breaking rule is the same: Find *any* locally maximum root v (a root with key greater than those of both adjacent roots); link it with the adjacent node of greater key; repeat until one root remains.

Smooth heaps and slim heaps are efficient for two reasons. The first is that locally maximum linking guarantees that a node acquires at most two new children during a *delete-min* operation (Lemma 3). Pairing heaps do not have this property: in the second, right-to-left linking pass during a pairing heap *delete-min*, a node can acquire an arbitrary number of new children (see Figure 2). The second is that insertions and melds are lazy (they do no links), but *delete-min* operations are eager (they do as many links as possible). Pairing heaps are eager: except in the middle of an operation, a pairing heap consists of a single tree, and insertions and melds are done by single links. We do not know whether it is possible to obtain our bounds for a single-tree smooth heap or pairing heap.

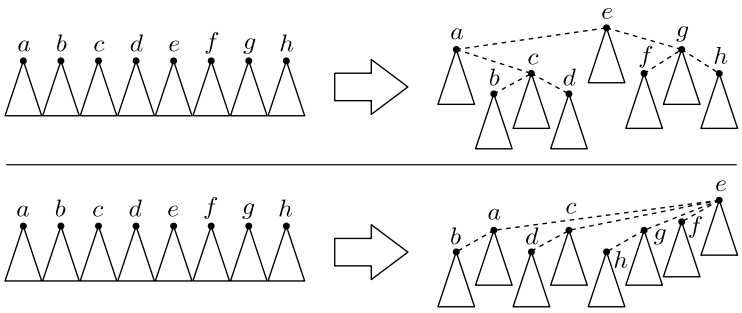


Figure 2 A possible linking of roots during a *delete-min* in a smooth heap (above) and a pairing heap (below), assuming key order e, a, g, c, b, d, f, h from smallest to largest.

▶ **Lemma 3.** *During a* delete-min*, a root wins at most one left link and at most one right link. Hence during a* delete-min *each node acquires at most one new left child and at most one new right child.*

Proof. Consider a link of the leftmost locally maximum node v during a *delete-min*. Node v is linked with either the left adjacent node u or the right adjacent node w. Suppose the link is with u. Then u has key strictly less than that of v; otherwise, u has locally maximum key and is left of v, contradicting the choice of v. The link is a right link won by u. After this link, either u is the rightmost node in the list or the node w adjacent to u on the right has no larger key, and the node adjacent to u on the left, if any, has key strictly smaller than

that of u. This makes u the leftmost local maximum, so u will participate in the next link and lose it: if the link is with w, w will win by the link tie-breaking rule if the keys of u and w are equal. Hence after the link of u and v, u cannot win another link, either left or right.

Suppose the link of v is with w. Since the key of w is no greater than that of v, w wins the link, by the link tie-breaking rule if the keys of v and w are equal. The link is a left link. After the link, the node u adjacent to w on the left, if it exists, has key strictly smaller than that of w by the tie-breaking rule for choosing whether to link v with u or with w. Node u cannot become a leftmost local maximum until w loses a link. Hence w cannot win another left link, since it would be with u. ◀

3 Efficiency of operations

With the two-pointer-per-node representation in Section 2.3, each *make-heap*, *find-min*, *insert*, and *meld* operation takes $O(1)$ time worst-case: catenation of the two root lists during *meld* takes two pointer changes, and each link, whether one-sided or stable, takes O(1) time. The time for a *delete-min* is O(1) plus at most a constant times the number of links. We normalize this time to be one plus the number of links. The number of links is one less than the number of roots, new and old, after deletion of the min-root. We shall prove a bound of $O(\lg n)$ on the amortized time of *delete-min* and of $O(1)$ on the amortized times of the other operations by using the *potential method* [26].

The previous analysis of smooth heaps [16] was restricted to *sorting mode*, and made use of a *geometric view* of binary search trees [4]. Our analysis is free of such restrictions and is self-contained.

3.1 Potential method

In the potential method, we assign to each state of the data structure a real-valued potential. We define the *amortized time* of an operation to be its actual time plus the potential of the structure after the operation minus the potential of the structure before it. That is, the amortized time is the actual time plus the net increase in potential caused by the operation. If we sum the amortized times of the operations in a sequence, the sum of the potential differences telescopes: the sum of the actual times of the operations equals the sum of their amortized times, plus the final potential (after the last operation), minus the initial potential (before the first operation). If the initial potential is 0 (corresponding to an empty data structure) and the final potential is non-negative, then the sum of the amortized times of the operations is an upper bound on the sum of their actual times, allowing us to use the former as a conservative bound on the latter.

3.2 Definition of the potential

The analyses of slim heaps and smooth heaps differ slightly. We develop them both at the same time and point out the differences.

For each node x having children, we define the *link order* of the children to be the order in which these children lost links to x, latest first, earliest last. This is exactly the order of the children in the list of children of x if the heap is a slim heap, but not necessarily if the heap is a smooth heap: in the latter, the link order is a merge of the left children in their order in the list of children of x and of the right children in the reverse of their order in the list of children of x.

We define the size $size(x)$ of a node x in a heap to be the number of its descendants, including itself. We define the mass $mass(x)$ of a child x to be the sum of the sizes of x and of all its siblings after it in link order (those linked to their common parent before x). (We do not need to define the masses of roots.)

We define the potential of a root x to be $2 + 2 \lg size(x)$. In a slim heap, we define the potential of a child x to be 0 if x is the first or second child of its parent, $\lg mass(x)$ otherwise (at least two children were linked to the parent of x after x). In a smooth heap, we define the potential of a child x to be 0 if it is the leftmost left child or rightmost right child in its sibling list, $\lg mass(x)$ otherwise. We define the potential of a collection of heaps to be the sum of the potentials of their nodes.

The slim heap potential is closely related to the one used in the original analysis of pairing heaps [9], which assigns $\lg size(x)$ potential to each root x and $\lg mass(x)$ potential to each child x. Here we give (at most) two children of a node zero potential. This limits the increase in potential caused by a link in a *delete-min* to the logarithm of the size of the winner *before* the *delete-min*. Our analysis relies on this limit.

3.3 Amortized bounds

The amortized time of a *make-heap*, *find-min*, *insert*, or *meld* is $O(1)$, since the worst-case time of each is $O(1)$, and only an *insert* changes the potential, increasing it by 2.

We shall show that the amortized time of a *delete-min* is $O(\lg n)$.

Let x be the min-root. After its deletion, the list of roots consists of new roots, those that were children of x, followed by old roots, those that were roots before the deletion of x. See Figure 3. We separately analyze links won by old roots and links won by new roots.

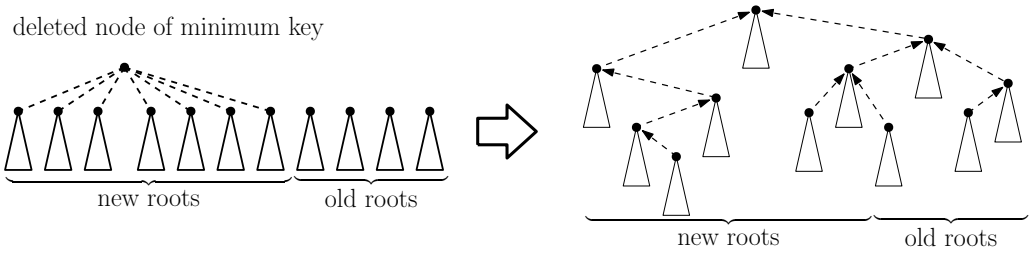


Figure 3 The result of the delete-min operation. The root with smallest key is deleted, its children become the new roots in its place, and then all roots are linked in a binary tree.

Each old root u has potential $2 + 2 \lg size(u)$, where $size(u)$ is the size of u just before the *delete-min*. This potential covers the actual time (1 or 2) of the link or links that u wins during the *delete-min*, if there are any, plus the increase in the potential of children of u when u wins a link or two: if v and w respectively are the at most two children of u that have potential zero before the *delete-min*, then when u wins one link, the potential of one of v and w increases from 0 to $\lg mass(w) < \lg size(u)$, and when u wins a second link the potential of the other of v and w increases from 0 to $\lg mass(v) < \lg size(u)$.

The heart of the analysis, the hard part, is that of links won by new roots. Let the masses of the new roots be their masses before the deletion of x. Before this deletion, x has potential $2 + 2 \lg size(x)$. The deletion of x frees its potential. We transfer $\lg size(x)$ of this potential from x to each of the at most two new roots with zero potential. Then every new root u has a potential of at least $\lg mass(u)$.

Each new root u can acquire one or two new children by winning one or two links during the *delete-min*. By the argument we used for old roots, the increase in potential of the children of u when u wins a link is less than $\lg size(u)$ per link, where $size(u)$ is the size of u

just before the *delete-min*. We shall show that the sum of the base-two logarithms of the masses of the new roots, which is at most the sum of their potentials after x is deleted and its freed potential is transferred, plus $2 + \lg size(x)$ for a slim heap or $2 + 2\lg size(x)$ for a smooth heap, is at least the sum of the increases in potential of the children of these roots, plus at least 2 per new root that wins a link. The extra 2 pays for the actual time of the links won by new roots during the *delete-min*.

The first step is to shift some of the potential of the new roots so that each new root u has at least $\lg mass(u)$ potential for each link it wins during the *delete-min*. This shifting is different for slim heaps and smooth heaps. In slim heaps, for each new root u except the last one in link order, if the root v after u in link order wins a left link, we shift $\lg mass(u) \geq \lg mass(v)$ from u to v. See Figure 4.

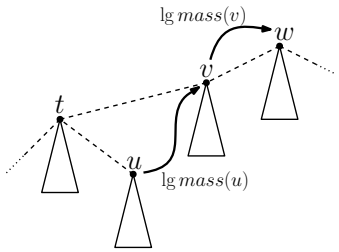


Figure 4 An example of shifting potential among new roots in a slim heap. The dotted lines represent links made during delete-min, with the losers are pictured lower than the winners, and curved lines represent shifted potential. New root u gives potential to its adjacent sibling v because v wins a left link with t and v gives potential to w because w wins a left link with v.

▶ **Lemma 4.** *After the potentials are shifted in a slim heap, each new root u has potential at least* $\lg mass(u)$ *if it wins one link during the* delete-min*, at least* $2\lg mass(u)$ *if it wins two.*

Proof. Consider a new root v that wins one or two links during the *delete-min*. If v wins a left link, it acquires at least $\lg mass(v)$ potential from the root before it in link order. For v to lose potential to the node w after it in link order, w must win a left link. But then v cannot win a right link, since if this link occurs before the left link that w wins, w must be the loser, and hence cannot later win a link, and if this link occurs after the left link that w wins, v must be the loser, and cannot later win a link. We conclude that if v wins both a left link and a right link, it acquires at least $\lg mass(v)$ potential from the node before it in link order and retains its own potential of at least $\lg mass(v)$; if v wins only a right link, it retains its own potential of at least $\lg mass(v)$; and if v wins only a left link, it acquires at least $\lg mass(v)$ potential from the node before it in link order, although it may lose its own potential. Hence the lemma holds. ◀

In the case of smooth heaps, we call a new root a *left* or *right* root depending on whether it was a left or right child of x before x was deleted. We must shift the potential of left and right roots separately, since the link order of the latter is the reverse of their order on the list of children of x before x was deleted. Specifically, for each left root u except the last in link order, if the next left root v in link order wins a left link, we shift $\lg mass(u) \geq \lg mass(v)$ from u to v; for each right root u except the last in link order, if the next right root v in link order wins a right link, we shift $\lg mass(u) \geq \lg mass(v)$ from u to v. In addition, if the first right root u in link order wins a (right) link with an old root, we give u $\lg size(u) \leq \lg size(x)$ of the $2 + 2\lg size(x)$ extra potential allocated to the *delete-min*. This accounts for the extra potential needed in the analysis for smooth heaps: since all the new roots precede all the

old roots, a new root can only win a right link with an old root, not a left link. This is a problem only when shifting potential to cover right links, which does not happen in slim heaps. See Figure 5.

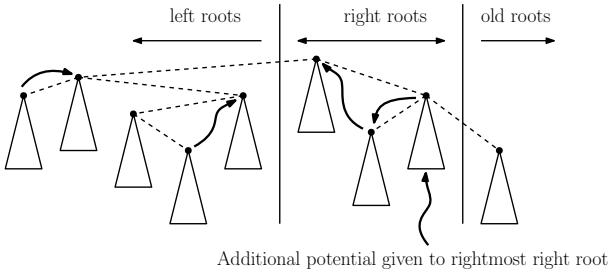


Figure 5 An example of shifting potential in a smooth heap. Every curved line represents a transfer of potential equal to the logarithm of the mass of the new root at the start of the line. Left roots transfer potential in the same way as a slim heap. Right roots transfer potential in the symmetric way, and the rightmost right root receives additional potential to pay for a right link.

▶ **Lemma 5.** *After the potentials are shifted in a smooth heap, each new root u has potential at least* $\lg mass(u)$ *if it wins one link during the* delete-min*, at least* $2 \lg mass(u)$ *if it wins two.*

Proof. The proof is that of Lemma 4 applied separately to left roots and right roots. Consider a left root v that wins one or two links during the *delete-min*. If v wins a left link, it acquires at least $\lg mass(v)$ potential from the left root preceding it in link order: there must be such a root since if v wins a left link it must be with a left root. For v to lose potential, it must be to the left root w after it in link order, and w must win a left link. But then v cannot win a right link, since if this link occurs before the left link that w wins, w must be the loser, and hence cannot later win a link, and if this link occurs after the left link that w wins, v must be the loser, and cannot later win a link. We conclude that if v wins both a left link and a right link, it acquires at least $\lg mass(v)$ potential from the left root preceding it in link order and retains its own potential of at least $\lg mass(v)$; if v wins only a right link, it retains its own potential of at least $\lg mass(v)$; and if v wins only a left link, it acquires at least $\lg mass(v)$ potential from the left root preceding it in link order, although it may lose its own potential. The symmetric argument applies to right roots, except that the first right root in link order may win a right link (with an old root), and not receive any potential for it. The additional $\lg size(x)$ potential allocated to a smooth heap *delete-min* covers this. Hence the lemma holds. ◀

By Lemmas 4 and 5, every new root has enough potential to cover the increase in the potential of its children if it wins one or two links. For each new root u that wins two links during the linking, we allocate $\lg size(u)$ potential from u to the child whose potential increases when the second link occurs. Each new root u that wins at least one link retains at least $\lg mass(u)$ potential. We shall show that if we add $2 + \lg size(x)$ additional potential, there are at least two units of extra potential per new root u that wins at least one link, in addition to the increase in potential of a child of u when u first wins a link. The latter is less than $\lg size(u)$. We consider new roots that win at least one link from first to last in link order. We maintain the invariant that the next root u that wins a link has at least $2 \lg mass(u)$ potential. To establish the invariant for the first such root u, we give it $\lg mass(u) < \lg size(x)$ of the $2 + \lg size(x)$ additional potential.

Let u be the current root under consideration and let v be the next new root after u that wins a link. Node u currently has potential at least $2 \lg \mathit{mass}(u)$. Since $\mathit{mass}(u) \geq \mathit{size}(u) + \mathit{mass}(v)$, the inequality $2 \lg(\mathit{size}(u) + \mathit{mass}(v)) \geq \lg \mathit{size}(u) + \lg \mathit{mass}(v) + 2$ allows us to move $\lg \mathit{size}(u)$ potential from u to its child whose potential increases when u first wins a link, and to move $\lg \mathit{mass}(u)$ potential from u to v, establishing the invariant for v, with at least two units of potential left over to pay for the link or links that u wins during the *delete-min*. This argument covers all but the last new root w in link order that wins at least one link, but w ends up with potential at least $2 \lg \mathit{mass}(w)$, more than enough to cover the increase in potential of one of its children when it first wins a link. We use the two units of remaining added potential to pay for the actual time of the one or two links won by w.

The single root remaining after all the linking has a potential of $2 + 2 \lg n$. We conclude that the amortized time of a *delete-min* is at most $5 + 3 \lg n$ in a slim heap, at most $5 + 4 \lg n$ in a smooth heap: the links are covered by potential decreases, we added $2 + \lg n$ or $2 + 2 \lg n$ extra potential, the remaining root needs potential $2 + 2 \lg n$, and the *delete-min* takes time 1 in addition to the links. This finishes the proof that *delete-min* takes $O(\lg n)$ amortized time.

4 Implementation of *decrease-key* and *delete*

In this section we implement *decrease-key* and *delete*. We give two implementations of *decrease-key*, a simple one for which it is easy to prove an $O(\lg n)$ amortized bound, and a more complicated one based on Elmasry's for pairing heaps, which has an $O(\lg \lg n)$ bound.

Our implementations of *decrease-key* and *delete* require adding a third pointer to each node. If x is a child that is not leftmost on its list of siblings, its third pointer indicates its previous sibling; if it is leftmost, its third pointer indicates its parent. If x is a root, its third pointer indicates the left adjacent root on the root list, or the rightmost root on the root list if x is the min-root. This makes the lists of children and the list of roots doubly linked, and supports deletion of a node (and its subtree) from such a list in $O(1)$ time.

4.1 Key decrease

A simple implementation of *decrease-key*(x, k, h) is the one used in pairing heaps: Set the key of x to k. If x is not a root, cut x (and its subtree) from the list of children containing it, and add x to the root list. Update the min-root. This takes O(1) actual time. Cutting x from its parent and making it a root increases the potential only of x, by at most $2 + 2 \lg \mathit{size}(x)$. Thus the amortized time of *decrease-key* is $O(\lg n)$. A much more complicated argument based on Pettie's proof of the same method for *decrease-key* in pairing heaps gives a stronger amortized bound. For lack of space, we have omitted this proof from this version of our paper.

We obtain a smaller amortized bound for *decrease-key* if we use Elmasry's implementation for pairing heaps [6]. We maintain a *buffer* that contains roots whose keys have been decreased. The buffer of a new heap is initially empty. To do *decrease-key*(x, k, h), we set the key of x to k. If x is not a root, we first cut x (and its subtree) from its sibling list L. Second, if x has a child, we cut its leftmost child y (and its subtree) from x, and replace x by y in L. Third, we add x to the buffer. If the buffer contains at least $\lg n$ roots, we empty it. Whether or not x was a root, we finish the *decrease-key* by updating the min-root.

When doing a *delete-min*, we begin by emptying the buffer. When doing a *meld*, we empty the buffer of the smaller heap; the buffer of the larger heap becomes the buffer of the melded heap.

To empty the buffer, we sort the roots in the buffer in non-increasing order by key, and link them using leftmost locally maximum linking, which makes each root the leftmost child of the one of next-smaller key, whether the heap is a slim heap or a smooth heap. Each link is a left link. We add to the root list the root remaining after the roots in the buffer are linked.

To support *find-min* in $O(1)$-time, we maintain the min-root of the buffer as well as the min-root of the roots not in the buffer. This adds $O(1)$ time to each operation. We also need to store with each heap the number of nodes it contains. This is easy to maintain in $O(1)$ time per operation.

This way of doing *decrease-key* reduces the amortized time to $O(\lg\lg n)$. To prove this we use the potential function of Section 3.2, with three changes:

- **(i)** If x is a root in the buffer that lost a child y when its key was decreased, we give its children the potential they had before y was cut from x. Since y is the leftmost child of x before it is cut, if the heap is a slim heap y has zero potential. If the heap is a smooth heap, y has zero potential unless y and all the children of x are right children, not left children. In the latter case, x has at most one child of zero potential both before and after y is cut. We conclude that after y is cut, x has at most one child of zero potential, its most recently acquired child if the heap is a slim heap; its most recently acquired right child if the heap is a smooth heap. This allows x to win a new left link without increasing the potential of any of its children, including its new one.
- **(ii)** To pay for emptying the buffer when it is full, we give each root u in the buffer a potential of $4 + \lg\lg n$, rather than the $2 + \lg \mathit{size}(u)$ potential it would have if it were in the root list.
- **(iii)** To pay for emptying the buffer of a heap when it is melded with a larger heap, we give each heap of n nodes an extra potential of $4\lg n$.

The actual time to decrease the key of a node x is $O(1)$. After x is cut from its sibling list and its leftmost child is cut from it, its children have exactly the potential they need when x is in the buffer. Replacing x by the first child of x does not increase the potential of any node in the tree previously containing x. Thus the potential increase caused by a *decrease-key* is at most the potential of the new root in the buffer, which is $4 + \lg\lg n$, making the amortized time of the *decrease-key* $O(\lg\lg n)$ unless the buffer becomes full and is emptied.

The actual time to empty the buffer is $O(\lg\lg n)$ per root in the buffer, with the sorting time dominant. The potential of $\lg\lg n$ per root in the buffer covers this time. Because of the sorting, each root in the buffer acquires at most one new left child, which does not increase the potential of any node in any of the trees rooted at these nodes. One root, say u, remains after the roots in the buffer are linked. This node needs potential $2 + 2\mathit{size}(u) \leq 4\lg n$ when it is added to the root list. Since the buffer is full when it is emptied in a *decrease-key*, u acquires the needed potential from the extra 4 units of potential per root in the buffer before these roots are linked. Hence *decrease-key* takes $O(\lg\lg n)$ time whether or not the buffer is emptied.

If the buffer is emptied during a *delete-min*, the root formed by linking the roots in the buffer needs potential at most $2 + 2\lg n$, increasing the amortized time of the *delete-min* by this amount, but the amortized time remains $O(\lg n)$.

It remains to consider *insert* and *meld*. Insertion is just a special case of meld, in which one of the heaps contains only one item and an empty buffer. Consider a meld of two heaps, with the larger heap, say h, containing n nodes. The heap resulting from the meld has size at most $2n$. The heap potential of the smaller heap covers the potential of the root formed

by linking the roots in its buffer when it is emptied. The heap potential of the new heap is at most $4\lg(2n) = 4\lg n + 4$, which is an increase of at most 4 of the heap potential of h. Let $k \leq \lg n$ be the number of roots in the buffer of h. The meld increases the sum of the potentials of these roots by at most

$$k(\lg\lg(2n) - \lg\lg n) \leq (\lg n)\lg\left(\frac{\lg n + 1}{\lg n}\right) = (\lg n)\lg(1 + 1/\lg n) \leq (\lg n)(\lg e)/\lg n = \lg e$$

by the inequality $\lg(1 + 1/\lg n) \leq \lg e/\lg n$ if $n \geq 2$. We conclude that the amortized times of *insert* and *meld* remain $O(1)$.

4.2 Arbitrary deletion

To delete a node x in a heap h, if x is the min-root of h, we merely do *delete-min*(h). If x is not the min-root of h, we offer three ways of doing the deletion. One is to decrease the key of x to minus infinity and then do *delete-min*(h). Using either of our implementations of *decrease-key*, this takes $O(\lg n)$ amortized time, with the *delete-min* time dominating. An alternative that does not use *decrease-key* is to repeatedly link adjacent children of x using leftmost locally maximum linking until x has only one child, and then replace x by its only child in the root list, or in its list of siblings if x is not a root. This implementation of *delete* also takes $O(\lg n)$ time, by an analysis like that of *delete-min* in Section 3.3. A third alternative is to delay linking the children of the deleted node, by merely replacing the deleted node in its list of siblings (or in the root list if it is a root) by its list of children. This also takes $O(\lg n)$ amortized time, by an extension of the analysis in Section 3.3: we must add $O(\lg n)$ additional potential to each group of siblings whose parent is deleted, to help pay for the links they eventually win.

5 Permanent nodes do not count

In this section we prove Theorem 2, which states that nodes that are never deleted do not slow down operations:

▶ **Theorem 2.** *Operations* delete-min *and* delete *take* $O(\lg t)$ *amortized time, with the other amortized bounds unchanged, where* t *is the number of nodes in the heap at the time of the operation that will be deleted in the future.*

Proof. Call a node *temporary* if it will eventually be deleted, and *permanent* otherwise. The total time of a sequence of operations is O(1) per operation plus the number of links. All the links except at most one per *decrease-key* operation are done during *delete-min* operations and when a buffer of nodes whose keys have decreased is emptied. Thus it suffices to count links. We separately count links won by permanent nodes and those won by temporary nodes. A link won by a permanent node either remains permanent, or is cut by a subsequent *decrease-key operation*: such links cannot be cut by deletions. It follows that there is at most one such link per *insert* and at most two per *decrease-key*.

It remains to count links won by temporary nodes. To do this we redefine the size of a node to be the number of temporary nodes in its subtree, and we redefine mass and potential accordingly, except that we give a root with size 0 potential 2 and a child with mass 0 potential 0. It is easy to check that with the new potential function *make-heap*, *find-min*, *insert*, and *meld* still take constant amortized time, and *decrease-key* still takes $O(\lg\lg n)$ amortized time.

We make a few small changes to adapt the analysis of *delete-min* to the new potential function. We apply the analysis of links won by old roots only to those that are temporary. We add and shift potentials only among temporary new roots. The additional potential needed for the analysis to work is $O(\lg t)$ rather than $O(\lg n)$. Thus we obtain an $O(\lg t)$ amortized time bound for *delete-min*. We also obtain an $O(\lg t)$ amortized time bound for either implementation of arbitrary deletion that does not use *decrease-key*; for the implementation that uses *decrease-key*, the bound is $O(\lg t + \lg\lg n)$. ◀

6 Experimental evaluation

We implemented the pairing heap, the smooth heap, and the slim heap in Python 3, with the same generic heap interface. We compared their performance in two scenarios: sorting and Dijkstra's single-source shortest paths algorithm. In both tasks we counted two types of logical operations: *comparisons* between keys and *links* between nodes. Note that in pairing heaps comparisons and links occur together, making the two counts equal. By contrast, in smooth heaps and slim heaps the number of comparisons is larger than the number of links by a factor of at most two (see [16])[4]. In practice, links are typically costlier than comparisons, requiring several pointer changes. Thus the number of links is expected to be indicative of the actual running time. We summarize the experimental setup and findings (Figures 6, 7, 8) and refer to [12] for further experiments.[5]

Sorting. Given a list of roots containing the keys $\pi_1, \ldots, \pi_n$ of an input permutation π, we execute n *delete-min* operations, sorting π. It has been hypothesized that the smooth heap performs particularly well on structured inputs. To test this experimentally, we considered, besides uniformly random inputs, classes of *semi-random* permutations. We tested four families of input permutations:

- **(a)** *Uniformly random*: We generated permutations of sizes $n = 2, 2^2, \ldots, 2^{17}$ using the pseudo-random *random.shuffle* function of Python, with 5 independent runs for each size.
- **(b)** *Separable:* Starting with the sequence $1, \ldots, n$, for $n = 2, 2^2, \ldots, 2^{17}$, we did the following shuffling: reverse the sequence with probability $1/2$, then recursively process the first half and second half of the sequence in the same way, doing 20 independent runs for each size. The permutations obtained are *separable* (see e.g. [1]).
- **(c)** *Localized:* We generated a sequence of length $n = 10000$ where each element is drawn from a Gaussian normal distribution centered at its index, with standard deviation proportional to a parameter ε, doing 10 independent runs for each value $\varepsilon = 0, 0.01, \ldots, 0.3$.
- **(d)** *Sorted blocks:* Starting with a uniform random permutation of size $n = 10000$, we sorted contiguous blocks of elements, where the block sizes are chosen uniformly at random from the range $[1, \ldots B]$, with 20 independent runs for each value $B = 100, 200, \ldots, 2000$.

Shortest paths. For all three heaps we used a similar multi-tree implementation of *insert* and *decrease-key* that simply adds the respective node to the end of the root list. (This is *not* the standard pairing heap implementation, which maintains a single tree, and the

[4] In the worst case $2k - \Theta(\lg k)$ comparisons are necessary to combine k roots during *delete-min*, since the outcome corresponds to one of $\Theta(4^k/k^{3/2})$ possible binary trees.

[5] The scripts used can be found at `https://git.imp.fu-berlin.de/hlm/smooth-heap-pub`.

implementation of *decrease-key* in smooth heaps and pairing heaps is the simple one.) Since these structures differ only in the restructuring during delete-min, we counted the number of links and comparisons only for delete-mins, taking averages over 10 independent runs. We considered two families of undirected input graphs, with edge weights in both cases chosen uniformly at random from $\{1, \ldots, 10000\}$:

(e) *Random graphs:* with $n = 500$ vertices, generated according to the Erdős-Rényi model, with edges present with probability p (for values $p = 0, 0.05, \ldots, 1$), and
(f) *Random* 10*-regular graphs:* with $n = 500, 1000, \ldots, 10000$ vertices.

Findings. In (a) the number of comparisons (and links) done by the pairing heap is a small factor (≈ 1.2) above the information-theoretical lower bound of $\lg(n!)$. On our largest test, the smooth (slim) heap performs about 22% (44%) more comparisons. In turn, the pairing heap performs about 47% (11%) more links than the smooth (slim) heap.

In (b) the number of operations appears to be $O(n)$ for all data structures. This is consistent with the fact that there are $2^{O(n)}$ such permutations of size n. In fact, for the smooth heap, it is known [16] that sorting separable permutations takes time $O(n)$, as implied by more general results for Greedy BST. It can be observed that the pairing heap performs on average both more comparisons and more links than the smooth and slim heaps. The differences in the number of comparisons are small, but in the number of links the overhead of pairing heaps is about 58% (65%) in our largest test. One can further notice that there is less variation in the number of operations in smooth/slim heaps, whereas the pairing heap is sensitive to the exact choice of input permutation.

In (c) the situation is similar to the uniformly random case, with the ordering slim, smooth, pairing in the number of comparisons, and pairing, slim, smooth in the number of links. Although the overall costs are smaller to (a), the smooth/slim heaps do not appear to have a particular advantage in adapting to this type of structure.

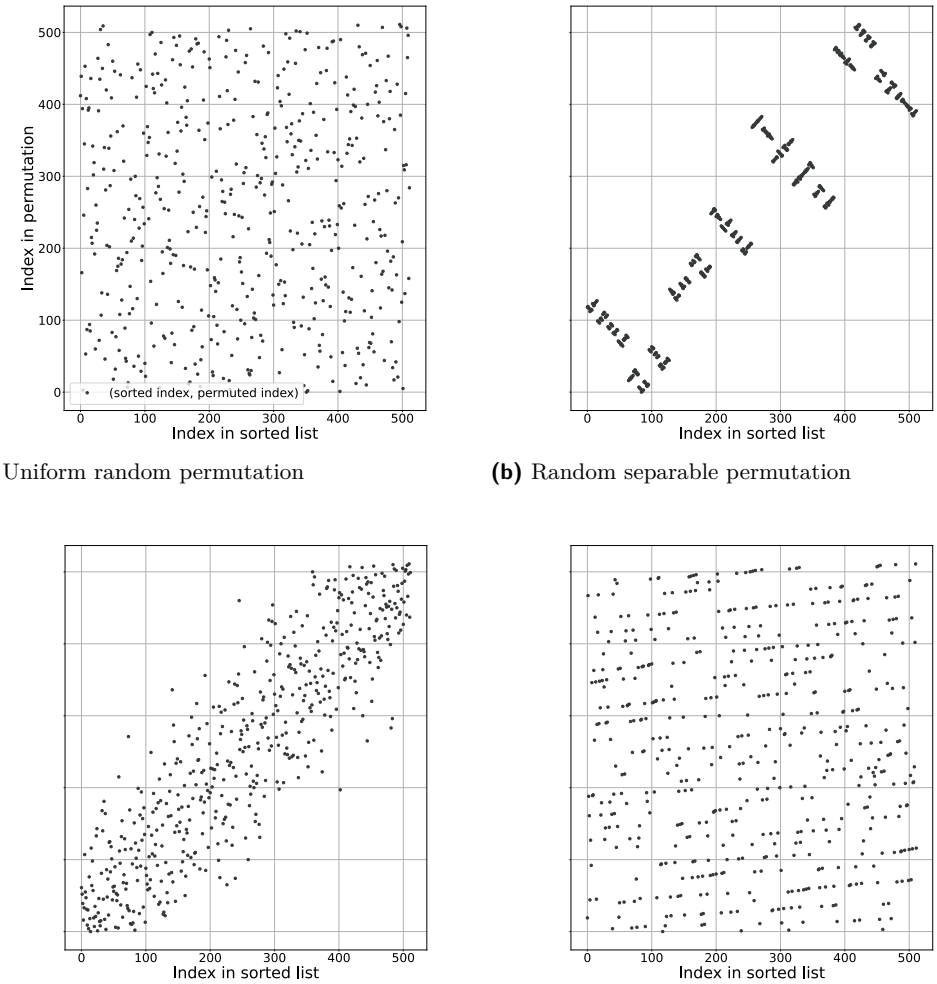
In (d) the smooth/slim heaps have a clearer advantage. When the input permutation consists of just about 10 sorted blocks, the pairing heap performs about 7 comparisons and links per key, whereas both smooth and slim heaps perform about 6 comparisons and 4 links per key. The advantage diminishes as the sizes of the sorted blocks decrease.

In the densest case of (e), the smooth (slim) heaps perform about 40% (45%) more comparisons than the pairing heap, while the pairing heap performs about 22% (14%) more links than the smooth (slim) heaps. In the largest case of (f), the smooth (slim) heaps perform about 36% (48%) more comparisons, while the pairing heap performs about 23% (6%) more links.

7 Remarks

Self-adjusting data structures are simple to describe and implement but hard to analyze. Developing new approaches and tools for analyzing such data structures remains an exciting field, with many questions still open. Our design of slim heaps and our analysis of smooth and slim heaps (Theorems 1 and 2) add to what we know about such structures.

One property that makes smooth heaps and slim heaps efficient is that each *delete-min* links at most two new children to each node. Neither the original version of pairing heaps nor any of the proposed variants has this property, but there is one variant that does: the *pure pairing heap*, in which the heap is a forest instead of a single tree, *insert* and *meld* are done lazily by catenating the root lists, and *delete-min* performs a single pairing pass: after the min-root is deleted, the remaining roots are linked in pairs, first and second, third and fourth, and so on. During a *delete-min*, each node acquires at most *one* new child. We offer



(a) Uniform random permutation

(b) Random separable permutation

(c) Random permutation with locality parameter $\varepsilon = 0.15$

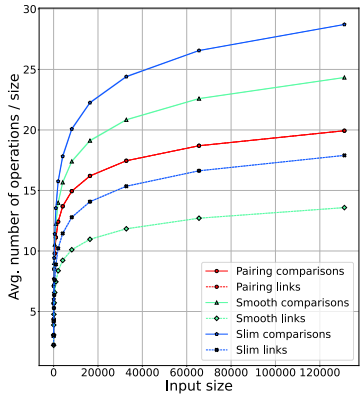
(d) Random permutation with average sorted blocks of length 15

Figure 6 Sample permutations of size 512 generated for the different families of permutations.

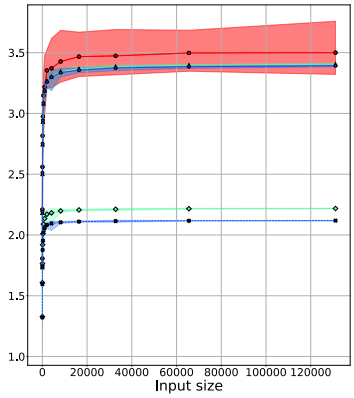
the pure pairing heap as a data structure for further study: our analysis fails, because this structure does not have the second property that makes smooth and slim heaps efficient; a *delete-min* can result in a list of roots rather than a single one.

We have proved that the simple method of doing *decrease-key* in smooth and slim heaps, by merely cutting the node whose key decreases and adding it to the root list, takes $O(\lg n)$ amortized time. For pairing heaps, Pettie has shown that the same method takes $O(4^{\sqrt{\lg \lg n}})$ amortized time. We think that his techniques will give a similar bound for smooth and slim heaps, and we are currently working on this question.

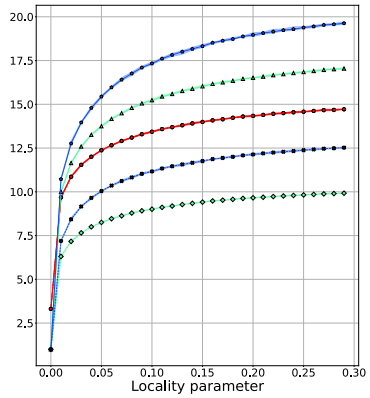
Similarly to other self-adjusting data structures, smooth heaps and slim heaps are expected to be *adaptive*, i.e. to show greater efficiency on some specific inputs. Through the connection with Greedy BST [16], smooth heaps are known to be highly adaptive in sorting mode, but these results do not seem easy to transfer to slim heaps. Adaptivity in general operation sequences is much less understood. Our Theorem 2 can be seen as a first result in this direction. We expect that other adaptive bounds, e.g. similar to those in [7] can also be shown. A possible way for proving such bounds is to generalize the analysis in this paper to



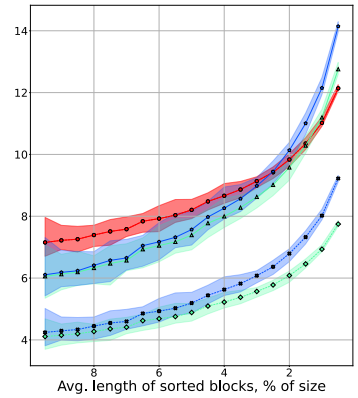
(a) Uniform random permutations.



(b) Random separable permutations.

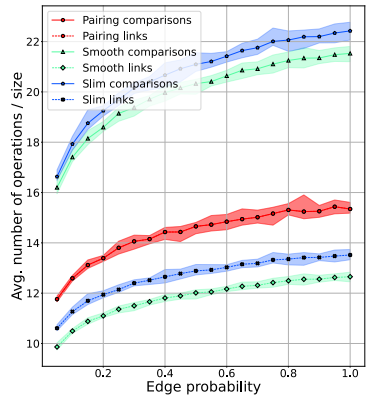


(c) Random permutations of size 10^4 with locality parameter ε.

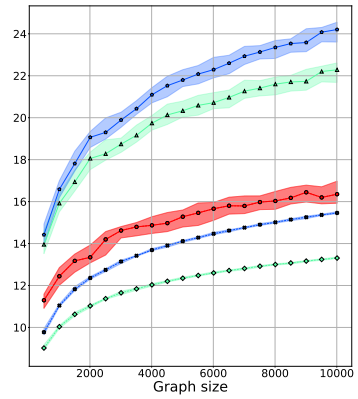


(d) Random permutations of size 10^4 with sorted blocks.

Figure 7 Average number of links (dashed lines) and average number of comparisons (continuous lines) for sorting permutations using pairing heap (red), smooth heap (green), and slim heap (blue); note that for pairing heaps the two counts are equal. Shaded areas indicate ranges between minimum and maximum costs.



(a) Random graphs with varying edge probability.



(b) Random regular graphs of varying size.

Figure 8 Average number of links (dashed lines) and average number of comparisons (continuous lines) for Dijkstra's algorithm using pairing heap (red), smooth heap (green), and slim heap (blue).

a weighted setting, much in the spirit of splay trees [24]. Such an extension of Theorem 1 is indeed possible, but with the current potential function it does not appear to yield nontrivial new bounds. A weighted analysis combined with a *linear* potential function, i.e. $O(n)$ rather than $O(n \lg n)$ may lead to such results.

References

1 Prosenjit Bose, Jonathan F. Buss, and Anna Lubiw. Pattern matching for permutations. *Inf. Process. Lett.*, 65(5):277–283, 1998. doi:10.1016/S0020-0190(97)00209-3.

2 Gerth Stølting Brodal. A survey on priority queues. In Andrej Brodnik, Alejandro López-Ortiz, Venkatesh Raman, and Alfredo Viola, editors, *Space-Efficient Data Structures, Streams, and Algorithms - Papers in Honor of J. Ian Munro on the Occasion of His 66th Birthday*, volume 8066 of *Lecture Notes in Computer Science*, pages 150–163. Springer, 2013. doi:10.1007/978-3-642-40273-9_11.

3 Gerth Stølting Brodal, George Lagogiannis, and Robert E Tarjan. Strict fibonacci heaps. In *Proceedings of the forty-fourth annual ACM symposium on Theory of computing*, pages 1177–1184, 2012.

4 Erik D. Demaine, Dion Harmon, John Iacono, Daniel M. Kane, and Mihai Pătraşcu. The geometry of binary search trees. In *SODA 2009*, pages 496–505, 2009. URL: http://dl.acm.org/citation.cfm?id=1496770.1496825.

5 Dani Dorfman, Haim Kaplan, László Kozma, Seth Pettie, and Uri Zwick. Improved bounds for multipass pairing heaps and path-balanced binary search trees. In Yossi Azar, Hannah Bast, and Grzegorz Herman, editors, *26th Annual European Symposium on Algorithms, ESA 2018, August 20-22, 2018, Helsinki, Finland*, volume 112 of *LIPIcs*, pages 24:1–24:13. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2018. doi:10.4230/LIPIcs.ESA.2018.24.

6 Amr Elmasry. Toward optimal self-adjusting heaps. *ACM Trans. Algorithms*, 13(4):55:1–55:14, 2017. doi:10.1145/3147138.

7 Amr Elmasry, Arash Farzan, and John Iacono. A priority queue with the time-finger property. *Journal of Discrete Algorithms*, 16:206–212, 2012.

8 Michael L. Fredman. On the efficiency of pairing heaps and related data structures. *J. ACM*, 46(4):473–501, 1999. doi:10.1145/320211.320214.

9 Michael L. Fredman, Robert Sedgewick, Daniel Dominic Sleator, and Robert Endre Tarjan. The pairing heap: A new form of self-adjusting heap. *Algorithmica*, 1(1):111–129, 1986. doi:10.1007/BF01840439.

10 Michael L. Fredman and Robert Endre Tarjan. Fibonacci heaps and their uses in improved network optimization algorithms. *J. ACM*, 34(3):596–615, 1987. doi:10.1145/28869.28874.

11 Thomas Dueholm Hansen, Haim Kaplan, Robert E. Tarjan, and Uri Zwick. Hollow heaps. *ACM Trans. Algorithms*, 13(3):42:1–42:27, 2017. doi:10.1145/3093240.

12 Maria Hartmann. Efficiency of self-adjusting heap variants. Master's thesis, Freie Universität Berlin, 2020. URL: http://www.mi.fu-berlin.de/inf/groups/ag-ti/theses/master_finished/hartmann_maria/index.html.

13 John Iacono. Improved upper bounds for pairing heaps. In Magnús M. Halldórsson, editor, *Algorithm Theory - SWAT 2000, 7th Scandinavian Workshop on Algorithm Theory, Bergen, Norway, July 5-7, 2000, Proceedings*, volume 1851 of *Lecture Notes in Computer Science*, pages 32–45. Springer, 2000. doi:10.1007/3-540-44985-X_5.

14 John Iacono and Özgür Özkan. Why some heaps support constant-amortized-time decrease-key operations, and others do not. In *Automata, Languages, and Programming - 41st International Colloquium, ICALP 2014, Copenhagen, Denmark, July 8-11, 2014, Proceedings, Part I*, pages 637–649, 2014. doi:10.1007/978-3-662-43948-7_53.

15 John Iacono and Thatchaphol Saranurak. personal communication.

16 László Kozma and Thatchaphol Saranurak. Smooth heaps and a dual view of self-adjusting data structures. *SIAM J. Comput.*, 49(5), 2020. doi:10.1137/18M1195188.

17 Daniel H. Larkin, Siddhartha Sen, and Robert Endre Tarjan. A back-to-basics empirical study of priority queues. In Catherine C. McGeoch and Ulrich Meyer, editors, *2014 Proceedings of the Sixteenth Workshop on Algorithm Engineering and Experiments, ALENEX 2014, Portland, Oregon, USA, January 5, 2014*, pages 61–72. SIAM, 2014. doi:10.1137/1.9781611973198.7.

18 Caleb C. Levy and Robert E. Tarjan. A new path from splay to dynamic optimality. In Timothy M. Chan, editor, *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2019, San Diego, California, USA, January 6-9, 2019*, pages 1311–1330. SIAM, 2019. doi:10.1137/1.9781611975482.80.

19 Joan M. Lucas. Canonical forms for competitive binary search tree algorithms. *Tech. Rep. DCS-TR-250, Rutgers University*, 1988.

20 J.Ian Munro. On the competitiveness of linear search. In Mike S. Paterson, editor, *Algorithms - ESA 2000*, volume 1879 of *Lecture Notes in Computer Science*, pages 338–345. Springer Berlin Heidelberg, 2000. doi:10.1007/3-540-45253-2_31.

21 Seth Pettie. Towards a final analysis of pairing heaps. In *46th Annual IEEE Symposium on Foundations of Computer Science (FOCS 2005), 23-25 October 2005, Pittsburgh, PA, USA, Proceedings*, pages 174–183. IEEE Computer Society, 2005. doi:10.1109/SFCS.2005.75.

22 Peter Sanders. Fast priority queues for cached memory. *ACM J. Exp. Algorithmics*, 5:7, 2000. doi:10.1145/351827.384249.

23 Peter Sanders, Kurt Mehlhorn, Martin Dietzfelbinger, and Roman Dementiev. *Sequential and Parallel Algorithms and Data Structures - The Basic Toolbox*. Springer, 2019. doi:10.1007/978-3-030-25209-0.

24 Daniel Dominic Sleator and Robert Endre Tarjan. Self-adjusting binary search trees. *Journal of the ACM (JACM)*, 32(3):652–686, 1985.

25 John T. Stasko and Jeffrey Scott Vitter. Pairing heaps: Experiments and analysis. *Commun. ACM*, 30(3):234–249, 1987. doi:10.1145/214748.214759.

26 Robert Endre Tarjan. Amortized computational complexity. *SIAM Journal on Algebraic Discrete Methods*, 6(2):306–318, 1985.

27 John William Joseph Williams. Algorithm 232: heapsort. *Commun. ACM*, 7:347–348, 1964.

Approximating Maximum Integral Multiflows on Bounded Genus Graphs

Chien-Chung Huang
CNRS, ENS, PSL, Paris, France

Mathieu Mari
University of Warsaw, Poland

Claire Mathieu
CNRS, IRIF, Université de Paris, France

Jens Vygen
Research Institute for Discrete Mathematics & Hausdorff Center for Mathematics,
University of Bonn, Germany

Abstract

We devise the first constant-factor approximation algorithm for finding an integral multi-commodity flow of maximum total value for instances where the supply graph together with the demand edges can be embedded on an orientable surface of bounded genus. This extends recent results for planar instances. Our techniques include an uncrossing algorithm, which is significantly more difficult than in the planar case, a partition of the cycles in the support of an LP solution into free homotopy classes, and a new rounding procedure for freely homotopic non-separating cycles.

2012 ACM Subject Classification Theory of computation → Routing and network design problems

Keywords and phrases Multi-commodity flows, approximation algorithms, bounded genus graphs

Digital Object Identifier 10.4230/LIPIcs.ICALP.2021.80

Category Track A: Algorithms, Complexity and Games

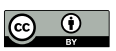
Related Version *Full Version*: `https://arxiv.org/abs/2005.00575`

Acknowledgements The authors would like to thank Arnaud de Mesmay for useful suggestions. This work was partially funded by the grant ANR-19-CE48-0016 from the French National Research Agency (ANR).

1 Introduction

Multi-commodity flows, or *multiflows* for short, are well-studied objects in combinatorial optimization; see, e.g., Part VII of [34]. A multiflow of maximum total value can be found in polynomial time by linear programming. Often, a multiflow must be integral, and then the problem is much harder; the well-known edge-disjoint paths problem is a special case. Recently, constant-factor approximation algorithms have been found for maximum edge-disjoint paths and integral multiflows in *fully planar instances*, i.e., when $G + H$, the supply graph together with the demand edges, can be embedded in the plane [21, 16]. We generalize these results to surfaces of bounded genus and devise the first constant-factor approximation algorithm for that case.

Beyond using some ideas of [16, 21], we need several new ingredients. Like [16], we start by computing an optimal (fractional) multiflow and "uncross" the cycles in its support as much as possible, but uncrossing is significantly more complicated on general surfaces than in the plane. Next, we need to deal with two cases separately: depending on whether most of the fractional multiflow is on separating cycles (that case is similar to the planar case)



or on non-separating cycles. In the latter case we partition the cycles into free homotopy classes and define a cyclic order in each free homotopy class, which is possible due to the uncrossing and allows for a simple greedy algorithm.

1.1 Our results

The (fractional) *maximum multiflow problem* can be described as follows. An instance consists of an undirected graph $(V, D \dot\cup E)$ whose edge set is partitioned into *demand edges*, in D, and *supply edges*, in E. We write $G = (V, E)$, $H = (V, D)$, and $G + H = (V, D \dot\cup E)$. Moreover we have a function $u : D \dot\cup E \to \mathbb{Z}_{>0}$ which defines a *capacity* $u(e)$ for each supply edge $e \in E$ and a *demand* $u(d)$ for each demand edge $d \in D$. The goal is to satisfy as much of the demand as possible by routing flow on supply edges. More precisely, we ask for an s-t-flow f^d of value at most $u(d)$ for every demand edge $d = \{t, s\}$ such that the total flow on each supply edge is at most its capacity and the total value of all those flows is maximum.

It is well known that every s-t-flow can be decomposed into flow on s-t-paths and on cycles, and for integral flows there is an integral decomposition. The cycles in such a decomposition do not contribute to the value of the s-t-flow and can be ignored. An s-t-path in (V, E) together with the demand edge $d = \{t, s\}$ forms a *D-cycle*: a cycle in $G + H$ that contains exactly one demand edge. If we let $\mathcal{C}$ denote the set of all D-cycles in $G + H$, we can write the maximum multiflow problem equivalently as

$$\max \sum_{C \in \mathcal{C}} f_C \quad \text{s.t.} \quad \begin{cases} \sum_{C \in \mathcal{C} : C \ni e} f_C \leqslant u(e) & \text{for all } e \in D \dot\cup E \\ f_C \geqslant 0 & \text{for all } C \in \mathcal{C} \end{cases} \tag{1}$$

In some previous works, the problem has been defined with $u(d) = \infty$ for $d \in D$, and this variant is easily seen to be equivalent. We call the linear program (1) the *maximum multiflow LP*. The *maximum integral multiflow* problem is identical, except that the flow must be integral:

$$\max \sum_{C \in \mathcal{C}} f_C \quad \text{s.t.} \quad \begin{cases} \sum_{C \in \mathcal{C} : C \ni e} f_C \leqslant u(e) & \text{for all } e \in D \dot\cup E \\ f_C \in \mathbb{Z}_{\geqslant 0} & \text{for all } C \in \mathcal{C} \end{cases} \tag{2}$$

The special case where $u(e) = 1$ for every edge $e \in D \dot\cup E$ is known as the *maximum edge-disjoint paths* problem. Even that special case is unlikely to have a constant-factor approximation algorithm for general graphs (see Section 1.2). Our main result is a constant-factor approximation algorithm in the case when $G + H$ can be embedded on an orientable surface of bounded genus.

▶ **Theorem 1.** *There is a polynomial-time algorithm which takes as input an instance (G, H, u) of the maximum integral multiflow problem such that $G + H$ is embedded on an orientable surface of genus g, and which outputs an integral multiflow whose value is at most a factor $O(g^2 \log g)$ smaller than the value of any fractional multiflow.*

See Section 3 for an outline of the algorithm and the proof. In the full version of the paper, we explain how to improve the approximation ratio of the algorithm to $O(g^2)$. It is worth pointing out that almost all known hardness results for the maximum edge-disjoint paths problem hold even when G is planar (see Section 1.2). Theorem 1, along with the two recent papers [16, 21], highlight that for tractability one needs more than the planarity of G alone. The topology of $G + H$ together plays an important role.

The dual LP of (1) is:

$$\min \sum_{e \in D \dot\cup E} u(e) y_e \quad \text{s.t.} \quad \begin{cases} \sum_{e \in C} y_e \geqslant 1 & \text{for all } C \in \mathcal{C} \\ y_e \geqslant 0 & \text{for all } e \in D \dot\cup E \end{cases} \tag{3}$$

and this may be called the *minimum fractional multicut problem*. The *minimum multicut problem* results from replacing the inequality $y_e \geqslant 0$ in (3) by $y_e \in \{0,1\}$ for all edges $e \in D \dot\cup E$. Again, many previous works considered the equivalent special case where $u(d) = \infty$ for $d \in D$, in which case no dual variables for demand edges are needed. By weak duality, the value of any multiflow is at most the capacity of any multicut. Using Theorem 1 and a previous result of [36], we obtain (in Section 9):

▶ **Corollary 2.** *For any instance (G, H, u) of the maximum integral multiflow problem such that $G + H$ is embedded on an orientable surface of genus g, the minimum capacity of a multicut is at most $O(g^{3.5} \log g)$ times the maximum value of an integral multiflow.*

In general the integral multiflow-multicut gap[1], and even the integrality gap of (1), can be as large as $\Theta(|D|)$, even when G is planar and $G + H$ is embedded in the projective plane [18]. In this paper we consider orientable surfaces only. Corollary 2 states that the gap becomes constant when $G + H$ has bounded genus. So far very few such constant integral multiflow-multicut gaps are known, for example when G is a tree [18], or when $G + H$ is planar, as recently shown in [16, 21].

1.2 Related Work

Approximation algorithms and hardness for integral multiflows. Most of the hardness results for the maximum integral multiflow problem follow from the special case of the maximum edge-disjoint paths problem (EDP). The decision version of EDP is one of Karp's original NP-complete problems [22], and remains NP-complete even in many special cases [30], including the case of interest in this paper, namely even when $G + H$ is planar [28]. In terms of approximation, EDP is APX-hard [2]. Assuming that $\mathrm{NP} \not\subseteq \mathrm{DetTIME}(n^{O(\log n)})$, where $n = |V|$, there is no $n^{o(1/\sqrt{\log n})}$ approximation for EDP, even when G is planar and sub-cubic [6]. Assuming that for some positive δ, $\mathrm{NP} \not\subseteq \mathrm{RandTIME}(2^{n^\delta})$, there is no $n^{O(1/(\log\log n)^2)}$ approximation for EDP, even when G is planar and sub-cubic [7]. As far as we know, no stronger hardness result is known for integral mutliflows.

On the positive side, EDP can be solved in polynomial time when the number of demand edges is bounded by a constant [33]. The same holds for integral multiflows when $G + H$ is planar [35]. For exact algorithms in various special cases, see the survey [30]. In general, the best known approximation guarantee for EDP and maximum integral multiflows is $O(\sqrt{n})$ [4]. Approximation algorithms with better approximation ratios for various special cases have been designed. We refer the readers to the survey [10] and to [18, 23, 30] and the references therein.

Recent work on the planar case. Recently, [16] and [21] gave constant-factor approximation algorithms for maximum integer multiflows when $G + H$ is planar. Both papers proceed by first obtaining a half-integral multiflow and then using the four color theorem to round it to an integral solution (similar to Section 6). The main difference of the two works is the way such half-integral multiflows are obtained. In [16], it is constructed by uncrossing a fractional multiflow (see Section 5 for a definition) to construct a certain network matrix, which is known to be totally unimodular; in [21], such a half-integral multiflow is obtained

[1] There is a closely related, but different, notion of integral flow-cut gap introduced in [5]: they study the smallest constant c such that whenever $u(C \cap E) \geqslant u(C \cap D)$ for every cut C (the cut condition), there is an integral multiflow satisfying all demands and violating capacities by at most a factor c.

by rounding a feasible solution of a related problem in the planar dual graph of $G+H$. Both approaches do not extend to higher genus graphs in a straightforward way, because the dual of a cycle is no longer a cut in general and cycles cannot always be uncrossed.

Minimum multicut problem. The minimum multicut problem is NP-hard even when there are only three demand edges [11]. In general, assuming that the Unique Games conjecture holds, there is no $O(1)$-approximation [3], but a $O(\log |D|)$-approximation algorithm [17]. Better approximations also have been shown for special cases; see [18, 36] and the references therein. In particular, when $G+H$ is planar, [25] gave an approximation scheme. When G has genus g, an FPT-approximation scheme with parameters of g and $|D|$ has been proposed [8].

Tools from topology. The design of multiflows on surfaces is closely related to the properties of sets of curves on a surface. In a recent breakthrough, Przytycki [31] proved that the maximum number of essential curves on a closed surface of genus g such that no two of them are freely homotopic or intersect more than once is $O(g^3)$, improving on the previous exponential upper bound by [27]. Very recently, this number was shown to be $O(g^2 \log g)$ by [19], which almost matches the lower bound $\Omega(g^2)$ on the size of such sets [27]. We will use this result in Section 7.

2 Preliminaries

Consider an instance (G, H, u) of the maximum integral multiflow problem, and let $G+H = (V, E \dot\cup D)$ be the graph whose edge set is the disjoint union of the edge sets of the supply graph $G = (V, E)$ and the demand graph $H = (V, D)$. Throughout the paper, we assume that the graph $G+H$ is connected, otherwise we can run the algorithm on each of its connected components.

Graphs on surfaces. Surfaces are either orientable or non-orientable; in this paper we only consider closed orientable surfaces. A closed orientable surface of genus g can be seen as a connected sum of g tori, or equivalently a sphere with g handles attached on it, where g is called the *genus* of the surface. Given an integer $g \geqslant 0$, all closed surfaces with genus g are mutually homeomorphic, and we refer to any one of them as $\mathbb{S}_g$. For instance, $\mathbb{S}_0$ is the sphere and $\mathbb{S}_1$ is the torus.

A (multi)graph has *genus* g or is a *genus-g graph*, if it can be drawn on $\mathbb{S}_g$ without edge crossings, but not on $\mathbb{S}_{g-1}$. A genus-g graph may have several non-equivalent embeddings on $\mathbb{S}_g$, but all of them satisfy the same invariant, called the *Euler characteristic*: #Faces − #Edges + #Vertices $= 2 - 2g$.

A simple application of Euler's formula gives the following upper bound on the coloring number of genus-g graphs, when $g \geqslant 1$.

▶ **Theorem 3** (Map color theorem)**.** *A genus-g graph can be colored in polynomial time with at most $\chi_g \leqslant \lfloor \frac{7+\sqrt{1+48g}}{2} \rfloor$ colors.*

For $g = 0$, this is an algorithmic version of the 4-color theorem [32]. For $g \geqslant 1$, the coloring is obtained in polynomial time by a simple recursive algorithm that removes a vertex of minimum degree and colors the remaining graph [20]. For additional details and results about graphs on surfaces see e.g. [29, 9].

Combinatorial embeddings. Given a graph, let $\delta(v)$ denote the set of edges incident to a vertex v, and $\delta(U)$ the set of edges with exactly one endpoint in vertex set U. Given an embedding of a graph on an orientable surface, and an arbitrary orientation of this surface, for each vertex v, a clockwise cyclic order can be defined on the edges of $\delta(v)$. Note that contracting an edge $e = \{u, v\}$ results in removing e from $\delta(u)$ and from $\delta(v)$ and concatenating the orders to obtain the clockwise cyclic order of the edges around the vertex created by the contraction. Using these orders together with the incidence relation between edges and faces, embeddings become purely combinatorial objects. For additional details see, e.g., [29], Chapter 4.

Graph duality. Given an embedding of a genus-g graph G on $\mathbb{S}_g$, there exists a uniquely defined dual graph, denoted as G^*. This graph can be embedded on the same surface as G. There exists a bijection between the faces of G and the vertices of G^*, a bijection between the vertices of G and the faces of G^*, and a bijection between the edge sets of G and of G^*. Moreover, the embeddings of G and G^* are consistent: with this bijection, every edge only crosses its dual edge, every face only contains its corresponding dual vertex and reciprocally. For notational simplicity, the latter bijection is implicit.

Cycles and cuts. A *path* in a graph G is a sequence $(v_0, e_1, v_1, \dots, e_k, v_k)$ for some $k \geqslant 0$, where $v_0, \dots, v_k$ are distinct vertices and $e_i = \{v_{i-1}, v_i\}$ is an edge for all $i = 1, \dots, k$. A *cycle* in a graph G is a sequence $(v_0, e_1, v_1, \dots, e_k, v_k)$ such that $v_1, \dots, v_k$ are distinct vertices, $\{v_{i-1}, v_i\}$ is an edge for all $i = 1, \dots, k$, and $v_0 = v_k$. Sometimes we view cycles as edge sets or as graphs. A *cut* is an edge set $\delta(U)$ for some proper subset $\emptyset \neq U \subset V$. A cut $\delta(U)$ is *simple* if both U and $V \setminus U$ are connected.

We say that an edge set F in a graph is a (simple) *dual cut* if the corresponding set of edges F^* in the dual is a (simple) cut. A cycle C in G is called *separating* if it is a dual cut, and *non-separating* otherwise. Note that every separating cycle is a simple dual cut.

Homotopy. Given a surface $\mathbb{S}$, a (simple) *topological cycle* is a continuous injective map γ from the unit cycle $S^1 := \{z \in \mathbb{C},\ ||z|| = 1\}$ to $\mathbb{S}$. Two topological cycles γ_1 and γ_2 are *freely homotopic* if there exists a continuous function $\varphi : [0, 1] \times S^1 \to \mathbb{S}$ such that $\varphi(0, \cdot) = \gamma_1$ and $\varphi(1, \cdot) = \gamma_2$. Intuitively, cycle γ_1 is transformed into cycle γ_2 by continuously moving it on the surface. Free homotopy is an equivalence relation.

Given an embedding of the graph $G + H$ on $\mathbb{S}$, we say that a cycle C in $G + H$ *is represented*[2] *by* a topological cycle γ of $\mathbb{S}$ if the image of γ is the embedding of C on $\mathbb{S}$. Two cycles in $G + H$ are freely homotopic if and only if they can be represented by two freely homotopic topological cycles. In the sequel, we use the following well-known fact.

▶ **Fact 4.** *If two cycles C and C' are freely homotopic, then their symmetric difference is a dual cut. If C and C' are additionally disjoint and non-separating, then their union is a simple dual cut.*

Intuitively, the image of the continuous homotopy function from C to C' on the surface forms an annulus [12]. See Figure 1 for an illustration.

[2] Topological cycles are considered up to orientation-preserving reparameterization. Therefore, a cycle in $G + H$ may be represented by a topological cycle from two classes, one for each orientation: the class of γ and the class of γ' where $\gamma'(e^{i\theta}) = \gamma(e^{-i\theta})$.

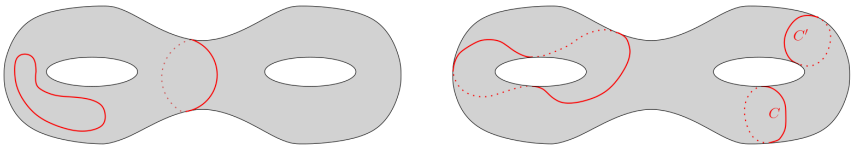


Figure 1 Some cycles on an orientable surface of genus 2. On the left, two *separating* cycles. On the right, three *non-separating* cycles. C and C' are *freely homotopic* and their union disconnects the surface.

3 Overview

In this section, we give an overview of our constant-factor approximation algorithm for the maximum integral multiflow problem when $G+H$ is embedded on an orientable surface $\mathbb{S}_g$ of genus g, where g is bounded by a constant (Theorem 1). Again, without loss of generality, we assume that $G+H$ is connected. Here is the main algorithm. Steps 1,2,3,4 will be described in detail in Sections 4,5,6,7, respectively.

1. Solve the linear program (1) to obtain a (fractional) multiflow f^*.
2. Construct another multiflow $\overline{f}$ such that any two cycles in the support of $\overline{f}$ cross at most once (Lemma 7). See Definition 6 for the definition of "crossing."
3. If at least half of the total value of $\overline{f}$ is contributed by separating cycles, these cycles now form a laminar family. Construct a half-integral multiflow f^{half} (Theorem 10), and from there, using the map color theorem (Theorem 3), compute an integral multiflow f' (Lemma 11), which is the output.
4. Otherwise, partition the non-separating cycles in the support of $\overline{f}$ into free homotopy classes. Pick the class $\mathcal{H}$ with largest total flow value. Remove the flow on all other cycles and greedily construct an integral multiflow (Lemmas 20 and 17), which is the output.

It can be proved that we only lose a constant factor at every step of the algorithm: see Section 8 for the analysis of the above algorithm, proving Theorem 1.

4 Finding a fractional multiflow (Step 1)

A feasible solution f to the maximum multiflow LP (1) will be simply called a *multiflow*. Recall that $\mathcal{C}$ denotes the set of all D-cycles, i.e., all cycles in $G+H$ that contain precisely one demand edge. We denote by $|f| = \sum_{C\in\mathcal{C}} f_C$ the *value* of f, and by $\mathcal{C}(f) := \{C \in \mathcal{C} \mid f_C > 0\}$ the *support* of f. Although formulation (1) has an exponential number of variables, it is well known that it can be reformulated by polynomially many flow variables and constraints (see, e.g., [15, 1]) and thereby solved in polynomial time:

▶ **Proposition 5.** *There is an algorithm that finds an optimal solution f^* to the maximum multiflow LP (1) such that $|\mathcal{C}(f^*)| \leqslant |D||E|$. Its running time is polynomial in the size of the input graph.*

Proof. By introducing flow variables $x_e^d := \sum_{C\in\mathcal{C}:d,e\in C} f_C$ for all $d \in D$ and $e \in D \dot\cup E$ we can maximize the total value $\sum_{d\in D} x_d^d$ subject to nonnegativity and flow conservation constraints (for each $d \in D$ and for each vertex). This is a linear program of polynomial size. By flow decomposition, one can then construct a feasible solution to (1) of the same value and with support at most $|D||E|$. ◀

Later we will restrict a multiflow to subsets of D-cycles. For $\mathcal{C}' \subseteq \mathcal{C}$ we define a multiflow f' by $f'_C := f_C$ for $C \in \mathcal{C}'$ and $f'_C := 0$ for $C \in \mathcal{C} \setminus \mathcal{C}'$, and write $f(\mathcal{C}') := f'$.

5 Making a fractional flow minimally crossing (Step 2)

In this section we show that for a given embedding, we can "uncross" a multiflow in such a way that any two D-cycles in the support cross at most once. While doing this we will lose only an arbitrarily small fraction of the multiflow value.

Uncrossing is a well-known technique in combinatorial optimization, but in most cases it is applied to families of subsets of a ground set U. Such a family is said to be cross-free if, for any two of its sets, A and B, at least one of the four sets $A \setminus B$, $B \setminus A$, $A \cap B$, and $U \setminus (A \cup B)$ is empty. Here we want to uncross D-cycles in the topological sense, and this can be reduced to the above (with some extra care) only if all these cycles are separating (which, for example, is always the case if $G + H$ is planar; cf. [16]).

▶ **Definition 6.** *We say that two D-cycles C_1 and C_2* cross *if there exists a path P (possibly a single vertex), which is a subpath of both C_1 and C_2, and such that in the embedding, after contracting the edges of P, the vertex v thus obtained is incident to two edges of C_1 and to two edges of C_2, all distinct, and in the embedding the restriction of the cyclic order of $\delta(v)$ to those four edges alternates between an edge of C_1 and an edge of C_2.*

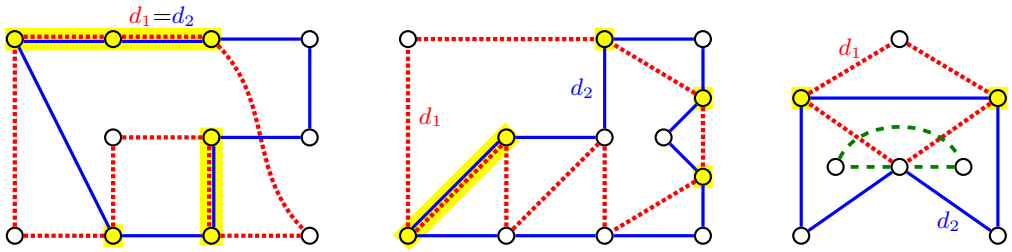


Figure 2 Each of the two figures on the left show two D-cycles, C_1 (red, dotted) and C_2 (blue, solid). The edges belonging to D are marked as d_1 and d_2. Edges are arranged at every vertex in the order of their embedding. Crossings are marked by yellow shade. The two D-cycles on the left cross three times. The two D-cycles in the middle cross four times. The figure on the right shows two D-cycles C_1 and C_2 that cross twice, and a third D-cycle C_3 (green, dashed) that crosses neither C_1 nor C_2. Uncrossing C_1 and C_2 here generates a crossing of C_3 with a new D-cycle (namely with the triangle containing d_2).

Two cycles may cross multiple times. We denote by $cr(C, C')$ the number of times that C and C' cross. See Figure 2 for three examples. In contrast to the planar case, it is possible that two cycles cross exactly once and cannot be uncrossed. The third example in Figure 2 shows another difficulty: when uncrossing two D-cycles it might be necessary to generate new crossings with other cycles.

▶ **Lemma 7.** *Let $\epsilon > 0$ be fixed. Given a multiflow f whose support has size at most $|E||D|$, there is a polynomial-time algorithm to construct another multiflow $\overline{f}$, of value at least $|\overline{f}| \geqslant (1 - \epsilon)|f|$, and such that any two cycles in the support of $\overline{f}$ cross at most once.*

Proof. First we discretize the multiflow, losing an ϵ fraction in value; then we iteratively modify it, without changing its value, to reduce the number of crossings or the total amount of flow on all edges; finally, we analyze the process and argue that the number of iterations is polynomially bounded.

Discretization. The statement is trivial if $|f| = 0$. Otherwise, before uncrossing, we round down the flow on every D-cycle to integer multiples of $\frac{\epsilon|f|}{|E||D|}$. That is, we define $f'_C := \frac{\epsilon|f|}{|E||D|}\left\lfloor \frac{|E||D|f_C}{\epsilon|f|} \right\rfloor$ for all $C \in \mathcal{C}$. Note that f' is a multiflow. We claim that $|f'| \geqslant (1-\epsilon)|f|$. Indeed,

$$|f'| = \sum_{C\in\mathcal{C}} f'_C \geqslant \sum_{C\in\mathcal{C}(f)} \left(f_C - \frac{\epsilon|f|}{|E||D|} \right) = |f| - |\mathcal{C}(f)|\frac{\epsilon|f|}{|E||D|} \geqslant |f| - \epsilon|f|.$$

The discretized multiflow f' can be represented by a multi-set $\mathcal{S}$ of unweighted D-cycles: if $f'_C = k\frac{\epsilon|f|}{|E||D|}$, then k identical copies of cycle C are added to $\mathcal{S}$. The number of cycles in $\mathcal{S}$ (counting multiplicities) is at most $\frac{|E||D|}{\epsilon}$ because $|\mathcal{S}| = \sum_{C\in\mathcal{C}} f'_C \frac{|E||D|}{\epsilon|f|} \leqslant \sum_{C\in\mathcal{C}} f_C \frac{|E||D|}{\epsilon|f|} = \frac{|E||D|}{\epsilon}$.

Uncrossing. To construct $\overline{f}$, we perform a sequence of transformations of the multiflow. We will modify $\mathcal{S}$ while maintaining the following invariants:

(a) The number of elements of $\mathcal{S}$ (counting multiplicities) remains constant.

(b) For every $e \in D \dot\cup E$, the number of elements of $\mathcal{S}$ (counting multiplicities) that contain e never increases.

Thanks to (b), at any stage, $\overline{f}$ is a multiflow, where $\overline{f}$ is defined by $\overline{f}_C = k\frac{\epsilon|f|}{|E||D|}$ for $C \in \mathcal{C}$, where k is the multiplicity of C in $\mathcal{S}$. Initially $\overline{f} = f'$. Thanks to (a), the value of the multiflow is preserved. In the following we work only with $\mathcal{S}$.

While there exist two cycles C_1 and C_2 in $\mathcal{S}$ that cross at least twice, do the following *uncrossing* operation (on one copy of C_1 and one copy of C_2). Let d_1 be the edge in $C_1 \cap D$, and let d_2 be the edge in $C_2 \cap D$. Let P and Q be two of the paths where C_1 and C_2 cross (cf. Definition 6), such that Q contains only edges of E. Orient C_1 so that in that orientation, when traversing the entirety of P and then walking towards Q, edge d_1 is traversed before reaching Q. Let $\vec{C}_1$ denote the resulting directed cycle. Let a be the first vertex on P in the orientation of $\vec{C}_1$, and let b be an arbitrary vertex on Q. Vertices a and b partition $\vec{C}_1$ into a path C_1^+ from a to b that contains d_1 and a path C_1^- from b to a that does not contain d_1.

Case 1: P contains an edge of D. Then this edge is $d_1 = d_2$. We orient C_2 so that the orientation on P agrees with the orientation of $\vec{C}_1$ on P. Let $\vec{C}_2$ denote the resulting directed cycle. Then the vertices a and b also partition $\vec{C}_2$ into a path C_2^+ from a to b that contains d_2 and a path C_2^- from b to a that does not contain d_2.

Case 2: P contains edges of E only. Then we orient C_2 so that in that orientation, when traversing the entirety of P and then walking towards Q, edge d_2 is traversed before reaching Q. Let $\vec{C}_2$ denote the directed cycle. With that orientation, vertices a and b also partition $\vec{C}_2$ into a path C_2^+ from a to b that contains d_2 and a path C_2^- from b to a that does not contain d_2.

To obtain C'_1, we concatenate C_1^+ and C_2^-, remove any cycle that does not contain d_1, and remove the orientation. To obtain C'_2, we concatenate C_2^+ and C_1^-, remove any cycle that does not contain d_2, and remove the orientation. Note that C'_1 and C'_2 are D-cycles because each of C_1^+ and C_2^+ contains exactly one demand edge, and C_1^- and C_2^- contain no demand edge.

See Figure 3 for two examples, one for each case.

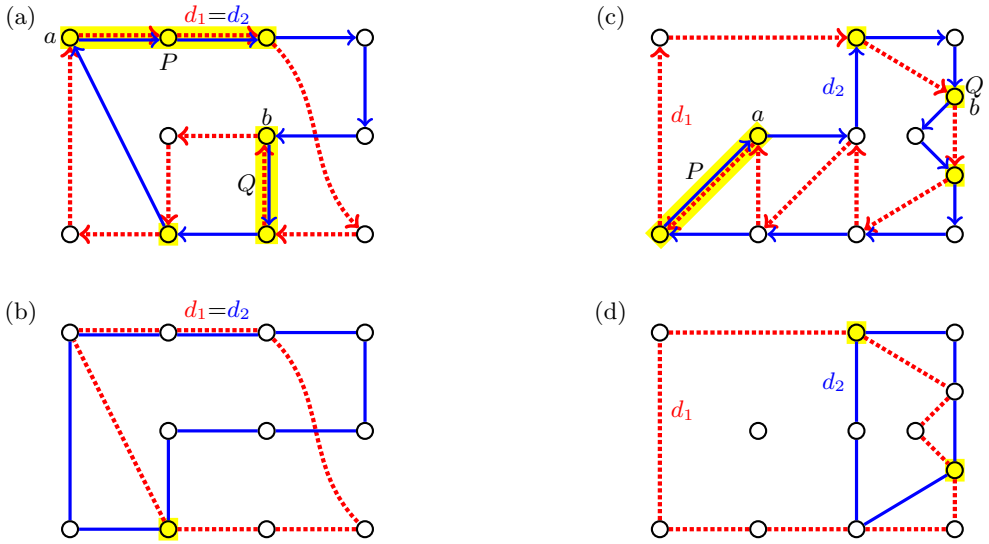


Figure 3 Uncrossing the pairs of D-cycles from Figure 2. (a) and (b) show an example for Case 1, (c) and (d) an example for Case 2. The initial situation (C_1 red, dotted, and C_2 blue, solid) and a possible choice of P, Q, a, b and the resulting orientation is shown in (a) and (c). As the result of the uncrossing operation, shown in (b) and (d), we have the new D-cycles C_1' (red, dotted) and C_2' (blue, solid) with fewer crossings among each other.

Analysis. From the construction it follows that C_1' and C_2' are D-cycles and $C_1' \dot\cup C_2' \subseteq C_1 \dot\cup C_2$. Hence removing one copy of C_1 and C_2 from $\mathcal{S}$ and adding one copy of C_1' and C_2' to $\mathcal{S}$ maintains the invariants (a) and (b).

To show that the after a polynomial number of uncrossing operations any pair of cycles in $\mathcal{S}$ crosses at most once, we consider the total number of edges $\Phi_1 = \sum_{C \in \mathcal{S}} |C|$ (counting multiplicities) and the total number of crossings $\Phi_2 = \sum_{C,C' \in \mathcal{S}} cr(C, C')$ (where we again count multiplicities). Note that $|\mathcal{S}|$ remains constant by invariant (a), and Φ_1 never increases by invariant (b). Moreover $0 \leqslant \Phi_1 \leqslant |V||\mathcal{S}|$ and $0 \leqslant |\Phi_2| \leqslant |V||\mathcal{S}|^2$.

▷ Claim 8. Each uncrossing operation either decreases Φ_1 or leaves Φ_1 unchanged and decreases Φ_2.

To prove Claim 8, consider an uncrossing operation that replaces C_1 and C_2 by C_1' and C_2', and suppose that Φ_1 remains the same, so C_1' consists of C_1^+ plus C_2^-, and C_2' consists of C_2^+ plus C_1^-. We first observe that $cr(C_1', C_2') < cr(C_1, C_2)$. Indeed, the crossings at P and at Q go away, and no new crossing arises.

Finally we need to show that for any cycle $C \in \mathcal{C}$,

$$cr(C, C_1') + cr(C, C_2') \leqslant cr(C, C_1) + cr(C, C_2). \tag{4}$$

To show (4), consider a crossing of C and $C' \in \{C_1', C_2'\}$ at a path R. Let $e_1' = \{v_0, v_1\}, \dots, e_k' = \{v_{k-1}, v_k\}$ be the edges of R ($k \geqslant 0$), and let $e_0, e_{k+1}, e_0', e_{k+1}'$ be edges such that $e_0, e_1', \dots, e_k', e_{k+1}$ are subsequent on C and $e_0', e_1', \dots, e_k', e_{k+1}'$ are subsequent on C'. After contracting R, the incident edges $e_0, e_0', e_{k+1}, e_{k+1}'$ are embedded in this cyclic order. (Note that $e_0 = e_{k+1}$ or $e_0' = e_{k+1}'$ is possible if $k \geqslant 1$, then contracting R yields a loop.) See Figure 4 (a).

Now e_0' belongs to C_1 or C_2, say C_1. If R contains neither a nor b, then $e_0', \dots, e_{k+1}'$ all belong to C_1, and C_1 crosses C at R. If R contains either a or b, say at v_i, then $e_0', \dots, e_i'$ belong to C_1 and $e_{i+1}', \dots, e_{k+1}'$ belong to C_2. Moreover C_1 and C_2 cross at a path containing

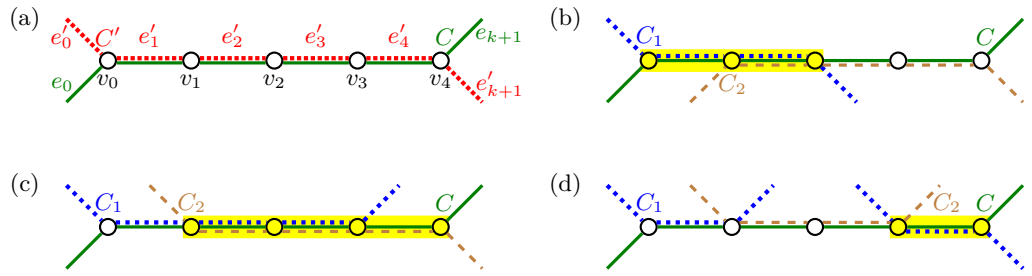


Figure 4 For each crossing of C with a new cycle $C' \in \{C'_1, C'_2\}$ at a path R there is a crossing of C with one of the old cycles C_1 and C_2 at a subpath of R. This crossing is marked with yellow shade in the three examples.

v_i, so either C_1 crosses C at a subpath of R (Figure 4(b)) or C_2 crosses C at a subpath of R (Figure 4(c)). Finally, if R contains a and b, say at v_i and v_j for $0 \leqslant i < j \leqslant k$, then $e'_0, \ldots, e'_i$ and $e'_{j+1}, \ldots, e'_{k+1}$ belong to C_1 and $e'_{i+1}, \ldots, e'_j$ belong to C_2 (Figure 4(d)). Again, C_1 or C_2 crosses C at a subpath of R. This concludes the proof of Claim 8.

We can now conclude the proof of Lemma 7 because Φ_1 decreases at most $|V||\mathcal{S}|$ times, and while Φ_1 is constant, Φ_2 decreases at most $|V||\mathcal{S}|^2$ times, so the total number of uncrossing operations is at most $|V|^2|\mathcal{S}|^3 \leqslant \frac{|V|^2|E|^3|D|^3}{\epsilon^3}$. ◀

6 Separating cycles: routing an integral flow (Step 3)

Let $\overline{f}$ result from Lemma 7, and let $\mathcal{C}_{\text{sep}}$ denote the set of separating cycles in the support of $\overline{f}$. We now consider the case when the separating cycles contribute at least half to the total flow value, i.e., $|\overline{f}(\mathcal{C}_{\text{sep}})| \geqslant \frac{1}{2}|\overline{f}|$.

This branch of our algorithm consists of two steps:

1. Given $\overline{f}(\mathcal{C}_{\text{sep}})$, construct a half-integral multiflow f^{half} of value at least $|\overline{f}|/2$;
2. Given f^{half}, construct an integral multiflow of value at least $|f^{\text{half}}|/\Theta(\sqrt{g})$.

6.1 Obtaining a half-integral multiflow

To obtain a half-integral multiflow, we follow the technique used by [16] for the case where $G + H$ is planar. By the Jordan curve theorem, any cycle in a planar graph is separating. As for the plane, the following property is easy to check for higher genus surfaces.

▶ **Proposition 9.** *If C and C' are two cycles embedded on a surface, and C' is a separating cycle, then C and C' must cross an even number of times.*

Proof. C' is separating the surface into two sides. While walking along C from a vertex v, we go from one side to the other each time we cross C'. When we return at v, we are on the same side where we started so the number of crossing is even. ◀

Since any pair of cycles in the support of $\overline{f}$ crosses at most once, $\mathcal{C}_{\text{sep}}$ must be a non-crossing family by Proposition 9. In particular, we can show that $\mathcal{C}_{\text{sep}}$ have a laminar structure.

We say that a family of subsets of the dual vertex set V^* is laminar if any two members either are disjoint or one contains the other. Let us take any face of $G + H$ that we call ∞. For any cycle $C \in \mathcal{C}_{\text{sep}}$ we define $\text{in}(C)$ and $\text{out}(C)$ to be the two connected components of $(G + H)^* \setminus C^*$, such that $\infty \in \text{out}(C)$. We claim that the family $\mathcal{L} := \{\text{in}(C) : C \in \mathcal{C}_{\text{sep}}\}$ is laminar.

Indeed, take any two cycles C and C' in $\mathcal{C}_{\text{sep}}$. Since they do not cross, either (i) $(C' \setminus C)^* \subseteq \text{in}(C)$ or, (ii) $(C' \setminus C)^* \subseteq \text{out}(C)$. In case (i) we must have $\text{in}(C') \subseteq \text{in}(C)$. In case (ii), we have either (ii.a) $\text{in}(C) \subseteq \text{in}(C')$ or (ii.b) $\text{in}(C) \cap \text{in}(C') = \emptyset$, hence laminarity.

Using the terminology in [16], we say that a multiflow f is *laminar* if $\{C^* : C \in \mathcal{C},\ f_C > 0\} = \{\delta(U) : U \in \mathcal{L}\}$ where $\mathcal{L}$ is a laminar family (of subsets of V^*). Thus, $\overline{f}(\mathcal{C}_{\text{sep}})$ is laminar and we can apply the following result to get f^{half}.

▶ **Theorem 10** ([16]). *If f is a laminar multiflow, then there exists a laminar half-integral multiflow f' such that $\mathcal{C}(f') \subseteq \mathcal{C}(f)$ of value $|f'| \geqslant \frac{1}{2}|f|$. Such a multiflow can be computed in polynomial time.*

6.2 Obtaining an integral multiflow

In this section we show the following result, which is an extension of a result from [21, 16], who proved it for planar graphs.

▶ **Lemma 11.** *Let (G, H, u) be an instance of the maximum multiflow problem such that $G+H$ has genus g, and let f^{half} be a laminar half-integral multiflow whose support $\mathcal{C}(f^{\text{half}})$ contains only separating cycles. Then there exists an integral multiflow f' of value $|f'| \geqslant 2|f^{\text{half}}|/\chi_g$ (such that $\mathcal{C}(f') \subseteq \mathcal{C}(f^{\text{half}})$). Such a multiflow can be found in polynomial time.*

Our proof follows the same outline as the proof of Theorem 1 of Fiorini et al. [14]. Let $\mathcal{C}^{\text{half}} := \mathcal{C}(f^{\text{half}})$ be the set of D-cycles C such that $f^{\text{half}}_C > 0$. We first reduce the problem to the case where all cycles in $\mathcal{C}^{\text{half}}$ have flow value $\frac{1}{2}$ and every edge has capacity 1. To do that, we reduce the flow f^{half}_C by $\lfloor f^{\text{half}}_C \rfloor$ for each cycle $C \in \mathcal{C}^{\text{half}}$, and reduce edge capacities accordingly. Then, since now f^{half} is small, we can further reduce demands and capacities to $u'(e) = \min\{u(e), |\mathcal{C}(f^{\text{half}})|\}$ for each $e \in E \cup D$, so that $\sum_{e \in D \dot\cup E} u(e)$ is polynomially bounded. We can then replace each edge e by $u(e)$ parallel edges of unit capacity. Given a cycle C such that $f^{\text{half}}_C = \frac{1}{2}$, we replace each edge $e \in C$ by one of its parallel edges. This can be done while ensuring that the resulting flow is still feasible and laminar. To facilitate the proof, we still denote this graph by $G + H$ and keep all other notations.

Recall that cycles in $\mathcal{C}^{\text{half}} \subseteq \mathcal{C}_{\text{sep}}$ are separating and do not cross each other, so that the family $\{\text{in}(C), C \in \mathcal{C}^{\text{half}}\}$ is laminar. We partially order $\mathcal{C}^{\text{half}}$ with the following relation: $C \prec C'$ if $\text{in}(C) \subset \text{in}(C')$. We have the following simple property:

▶ **Lemma 12.** *If $C_1, C_2, C' \in \mathcal{C}^{\text{half}}$ are such that $C_1 \prec C'$ and $C_2 \not\prec C'$, then C_1 and C_2 are edge-disjoint.*

For the proof, see the full version of the paper. Our goal is to get a large subset $\mathcal{C}' \subseteq \mathcal{C}^{\text{half}}$ such that any two cycles in $\mathcal{C}'$, are edge-disjoint. This is equivalent to finding a large independent set in a properly defined graph $\text{Int}(\mathcal{C}^{\text{half}})$ with vertex set $\mathcal{C}^{\text{half}}$ and such that two cycles are adjacent if they share at least one edge. Using Lemma 12 we can show:

▶ **Lemma 13.** *Given a graph embedded in $\mathbb{S}_g$, let $\mathcal{C}^{half}$ be defined as above. Let $Int(\mathcal{C}^{half})$ be the graph with vertex set $\mathcal{C}^{half}$ and such that two cycles are adjacent if they share at least one edge. Then $Int(\mathcal{C}^{half})$ is a genus-g graph.*

Proof. We prove the statement by induction on $g + |\mathcal{C}^{\text{half}}|$. When $g + |\mathcal{C}^{\text{half}}| \leqslant 2$, it is trivial. Otherwise let G be a connected genus-g graph, embedded on $\mathbb{S}_g$, and $\mathcal{C}^{\text{half}}$ a family as described above.

Suppose first that $\{\text{in}(C) \mid C \in \mathcal{C}^{\text{half}}\}$ are pairwise disjoint. Then, contract in G^* each set $\text{in}(C)$ into a single node. Two cycles C and C' share an edge if and only if in this contracted graph, the nodes corresponding to $\text{in}(C)$ and $\text{in}(C')$ are adjacent. This means that $\text{Int}(\mathcal{C}^{\text{half}})$ is a minor of G^*, and in particular has genus less than or equal to the genus of G^*.

The case where there is one cycle $\bar{C}$ such that $C \prec \bar{C}$ for all $C \in \mathcal{C}^{\text{half}} \setminus \bar{C}$ and $\{\text{in}(C) \mid C \in \mathcal{C}^{\text{half}} \setminus \bar{C}\}$ are pairwise disjoint works similarly; here we contract $\text{out}(\bar{C})$.

Otherwise there exists a triple $C_1, C_2, C \in \mathcal{C}^{\text{half}}$ such that $C_1 \prec C$ and $C_2 \not\prec C$. The separating cycle C divides $\mathbb{S}_g$ into two sides. Each side can be closed – by identifying the boundary of a disk with the boundary form by C – so that they are homeomorphic to $\mathbb{S}_{g_{\text{in}}}$ and $\mathbb{S}_{g_{\text{out}}}$, respectively. The connected sum of these two surfaces is homeomorphic to $\mathbb{S}_g$, and in particular we have $g_{\text{in}} + g_{\text{out}} = g$. This equality can easily be checked with Euler's formula.

Let G_{in} (*resp.* G_{out}) be the subgraph of G induced by the vertices embedded on the side corresponding to $\mathbb{S}_{g_{\text{in}}}$ (*resp.* $\mathbb{S}_{g_{\text{out}}}$), such that both contain C. The embedding of G in $\mathbb{S}_g$ induces an embedding of G_{in} in $\mathbb{S}_{g_{\text{in}}}$ and an embedding of G_{out} in $\mathbb{S}_{g_{\text{out}}}$. Thus, $\text{genus}(G_{\text{in}}) + \text{genus}(G_{\text{out}}) \leqslant g$.

Now we define $\mathcal{C}^{\text{half}}_{\preceq C} := \{C' \in \mathcal{C}^{\text{half}} | C' \prec C\} \cup \{C\}$ and $\mathcal{C}^{\text{half}}_{\not\prec C} := \{C' \in \mathcal{C}^{\text{half}} | C' \not\prec C\} \cup \{C\}$. The choice of C implies that these two families are proper subsets of $\mathcal{C}^{\text{half}}$. Since the cycles in $\mathcal{C}^{\text{half}}$ do not cross, we have $\{C \in \mathcal{C}^{\text{half}} : C \subseteq G_{\text{in}}\} = \mathcal{C}^{\text{half}}_{\preceq C}$ and $\{C \in \mathcal{C}^{\text{half}} : C \subseteq G_{\text{out}}\} = \mathcal{C}^{\text{half}}_{\not\prec C}$.

By the induction hypothesis, $\text{Int}(\mathcal{C}^{\text{half}}_{\preceq C})$ and $\text{Int}(\mathcal{C}^{\text{half}}_{\not\prec C})$ can be embedded on $\mathbb{S}_{g_{\text{in}}}$ and $\mathbb{S}_{g_{\text{out}}}$, respectively. By Lemma 12, the graph $\text{Int}(\mathcal{C}^{\text{half}})$ arises from $\text{Int}(\mathcal{C}^{\text{half}}_{\preceq C})$ and $\text{Int}(\mathcal{C}^{\text{half}}_{\not\prec C})$ by identifying the two vertices that correspond to C.

Finally we prove that $\text{Int}(\mathcal{C}^{\text{half}})$ can be embedded on a surface genus $g_{\text{in}} + g_{\text{out}} \leqslant g$. To see that, remove small disks D_{in} and D_{out} in $\mathbb{S}_{g_{\text{in}}}$ and $\mathbb{S}_{g_{\text{out}}}$, respectively, around the point that corresponds to vertex C and that intersects only edges incident to C, and glue them together by identifying boundaries of D_{in} and D_{out}. The surface obtained is homeomorphic to $\mathbb{S}_{g_{\text{in}}+g_{\text{out}}}$ It is easy to see that C, and the edges incident to C, can be re-embedded in this surface without intersecting any other edges. This terminates the proof of Lemma 13. ◀

Using Theorem 3, this lemma ensures that one can compute in polynomial time a subset $\mathcal{C}' \subseteq \mathcal{C}^{\text{half}}$ of at least $|\mathcal{C}^{\text{half}}|/\chi_g$ pairwise edge-disjoint D-cycles. From this set, we define an integral multiflow by setting $f'_C = 1$ for $C \in \mathcal{C}'$ and $f'_C = 0$ for $C \in \mathcal{C} \setminus \mathcal{C}'$. It is easy to check that f' is a multiflow that satisfies the properties of Lemma 11.

7 Non-separating cycles: routing an integral multiflow (Step 4)

If the separating cycles contribute less than half to the total value of the multiflow $\overline{f}$ obtained by Lemma 7, we consider the non-separating cycles in the support of $\overline{f}$. We first partition them into *free homotopy classes*. The next theorem gives an upper bound on the number of such classes.

▶ **Theorem 14** ([19]). *Let $\mathbb{S}_g$ be an orientable surface of genus g. Then there are at most $O(g^2 \log g)$ topological cycles such that any two of them are in different free homotopy classes and cross each other at most once.*

▶ **Corollary 15.** *The D-cycles in the support of $\overline{f}$ can be partitioned into $O(g^2 \log g)$ free homotopy classes in polynomial time.*

Proof. Take pairs of cycles in the support of $\overline{f}$ and check whether they are freely homotopic, for example as in [13, 26]. ◀

7.1 Greedy algorithm

Let $\mathcal{H}$ be a free homotopy class of non-separating cycles whose total flow value $|\overline{f}(\mathcal{H})|$ is largest. We will run the following simple greedy algorithm (Algorithm 1) on $\mathcal{H}$ to get an integral multiflow.

Algorithm 1 Greedy algorithm for integral multiflows.

Input: a sequence $C_1, \dots, C_k$ of D-cycles of $\mathcal{C}(\overline{f})$.
Output: an integral multiflow f.
$f \leftarrow$ the all-zero multiflow;
for $i = 1$ *to* k **do**
  Set f_{C_i} to be the greatest integer such that f remains feasible.

The value of the integral multiflow returned by this algorithm depends on the order of the D-cycles in the input. If it is ordered according to the following definition, then we show that we lose only a constant fraction of the flow value.

▶ **Definition 16.** *A family of cycles $\{C_1, C_2, \dots, C_k\}$ is* cyclically ordered*, or has a* cyclic order *if, whenever two cycles C_a and C_b share an edge, where $a < b$, then this edge is:*

1. *shared by all cycles $C_a, C_{a+1}, \dots, C_{b-1}, C_b$,*
2. *or shared by all cycles $C_b, C_{b+1}, \dots, C_k, C_1, \cdots, C_{a-1}, C_a$.*

The following lemma establishes the approximation ratio of Algorithm 1 on cyclically ordered input.

▶ **Lemma 17.** *Let $\overline{f}$ be a multiflow and $\mathcal{H} = \{C_1, C_2, \dots, C_k\}$ a cyclically ordered family of $\mathcal{C}(\overline{f})$. Then Algorithm 1 returns in polynomial time an integral multiflow of value at least $|\overline{f}(\{C_1, \dots, C_k\})|/2$.*

Proof. Let $\overline{f}$ be a multiflow and $\mathcal{H} = \{C_1, C_2, \dots, C_k\}$ a cyclically ordered family of $\mathcal{C}(\overline{f})$. It is clear that Algorithm 1 runs in polynomial time and returns an integral multiflow. Let f be this flow. We show that its value is at least $|\overline{f}(\mathcal{H})|/2$.

Let us define $\mathcal{H}_{a,b} = \{C_a, C_{a+1}, \dots, C_{b-1}\}$ and $\mathcal{H}_{b,a} = \{C_b, C_{b+1}, \dots, C_k, C_1, \dots, C_{a-1}\}$ for all $1 \leqslant a \leqslant b \leqslant k$. Additionally, for all edges $e \in \bigcup_{C \in \mathcal{H}} C$, we define $\mathcal{H}^e := \{C \in \mathcal{H} \mid e \in C\}$. Since we assumed that $\mathcal{H}$ is cyclically ordered, we know that for each $e \in \bigcup_{C \in \mathcal{H}} C$, there are indexes $1 \leqslant a, b \leqslant k$, such that $\mathcal{H}^e = \mathcal{H}_{a,b}$.

We call i_0 the smallest index $1 \leqslant i \leqslant k$ such that there exists an edge $e \in C_i$ such that $f(\mathcal{H}_{1,i+1})(e) = u(e)$ and $\mathcal{H}_{i,1} \subseteq \mathcal{H}^e$. Remark that in particular, for all $i > i_0$, we must have $f_{C_i} = 0$, and thus $|f| = |f(\mathcal{H}_{1,i_0+1})|$.

We first show by induction that for all $1 \leqslant i < i_0$ we have $|f(\mathcal{H}_{1,i+1})| \geqslant |\overline{f}(\mathcal{H}_{1,i+1})|$. For $i = 1$, we have $|f(\mathcal{H}_{1,i+1})| = |f(\mathcal{H}_{1,2})| = f_{C_1} = \min\{u(e) | e \in C_1\} \geqslant \overline{f}_{C_1} = |\overline{f}(\mathcal{H}_{1,2})|$.

Assume now that at some iteration $1 < i < i_0$ of the algorithm we set $f_{C_i} = x$. By the choice of x, we know that there is an edge $e \in C_i$ such that $u(e) = f(\mathcal{H}_{1,i+1})(e)$. In particular, notice that $|f(\mathcal{H}^e)| = |f(\mathcal{H}^e \cap \mathcal{H}_{1,i+1})| = u(e)$. By feasibility of $\overline{f}$, we have

$$|f(\mathcal{H}^e \cap \mathcal{H}_{1,i+1})| = u(e) \geqslant |\overline{f}(\mathcal{H}^e)|. \tag{5}$$

Now, let a, b be the two indexes such that $\mathcal{H}_{a,b} = \mathcal{H}^e$. Since we assumed that $i < i_0$, we must have $i < b \leqslant k$. There are two cases: either $1 \leqslant a \leqslant i < b$ or $1 < i < b < a$.

If $1 \leqslant a \leqslant i < b$, then equation (5) becomes $|f(\mathcal{H}_{a,i+1})| \geqslant |\overline{f}(\mathcal{H}^e)| \geqslant |\overline{f}(\mathcal{H}_{a,i+1})|$. Together with the induction hypothesis we obtain:

$$|f(\mathcal{H}_{1,i+1})| = |f(\mathcal{H}_{1,a})| + |f(\mathcal{H}_{a,i+1})| \geqslant |\overline{f}(\mathcal{H}_{1,a})| + |\overline{f}(\mathcal{H}_{a,i+1})| = |\overline{f}(\mathcal{H}_{1,i+1})|.$$

Otherwise if $1 < i < b < a$, then $\mathcal{H}_{1,i+1} \subseteq \mathcal{H}^e$, and thus the inequality claimed follows directly from equation (5). We have established the induction. In particular, we have proved that $|f| = |f(\mathcal{H}_{1,i_0+1})| \geqslant |f(\mathcal{H}_{1,i_0})| \geqslant |\overline{f}(\mathcal{H}_{1,i_0})|$. To conclude the proof of Lemma 17, it remains to show that $|f| \geqslant |\overline{f}(\mathcal{H}_{i_0,1})|$.

By definition of i_0, we know that there exists an edge $e \in C_{i_0}$ such that $f(e) = u(e)$ and such that $\mathcal{H}_{i_0,1} \subseteq \mathcal{H}^e$. By feasibility of $\overline{f}$, we deduce that $|\overline{f}(\mathcal{H}_{i_0,1})| \leqslant u(e) = f(e) \leqslant |f|$. This concludes the proof. ◀

▶ Remark 18. The analysis of Algorithm 1 for cyclically ordered inputs is tight. To see this, imagine that $\mathcal{H} = \{C_1, \ldots, C_{2k-1}\}$, and there are two edges e_1, e_2, both of capacity k, such that $\{C \in \mathcal{H} \mid e_1 \in C\} = \{C_1, \ldots, C_k\}$ and $\{C \in \mathcal{H} \mid e_2 \in C\} = \{C_{k+1}, \ldots, C_{2k-1}, C_1\}$. Then Algorithm 1 may only set $f_{C_1} = k$ while $\overline{f}$ could be such that $\overline{f}_C = 1$ for all $C \in \mathcal{H}$, for a total value $2k - 1$.

7.2 Computing a cyclic order

Lemma 20, the second main result of the section, states that a family $\mathcal{H}$ of pairwise freely homotopic cycles crossing at most once can be cyclically ordered in polynomial time. One key ingredient in the proof is that cycles in $\mathcal{H}$ are pairwise non-crossing. This fact uses the assumption that the surface is orientable. In a non-orientable surface, two freely homotopic cycles may cross exactly once.

Recall that $\overline{f}$ denotes the minimally-crossing multiflow obtained by Lemma 7.

▶ **Lemma 19.** *Two freely homotopic cycles in $\mathcal{C}(\overline{f})$ do not cross.*

For the proof of this simple topological fact, see the full version.

▶ **Lemma 20.** *A family of non-separating, pairwise non-crossing and freely homotopic cycles of a graph embedded in an orientable surface can be cyclically ordered. Such a cyclic order can be found in polynomial time.*

This result holds more generally for a family of *non-contractible*[3], pairwise non-crossing and freely homotopic cycles. For simplicity, we only consider the special case of non-separating cycles, which is sufficient for our main result.

Proof. Let $\mathcal{H}$ be a set of non-separating, pairwise freely homotopic and non-crossing cycles. We first order the cycles in $\mathcal{H}$ and then prove that this is a cyclic order. We assume that $|\mathcal{H}| \geqslant 3$, otherwise any order on $\mathcal{H}$ is a cyclic order.

In topology it is usually more convenient to work with disjoint cycles. If two (graph) cycles do not cross, but may share common edges, it is possible to continuously deform by free homotopy one of them, into an arbitrarily small open neighborhood so that the two resulting (topological) cycles are now disjoint.

In the context of graph cycles, we now give a reduction from the setting of Lemma 20 to the special case where the cycles are disjoint. Initially, $Q = G + H$.

[3] all cycles that are not freely homotopic to a point on the surface.

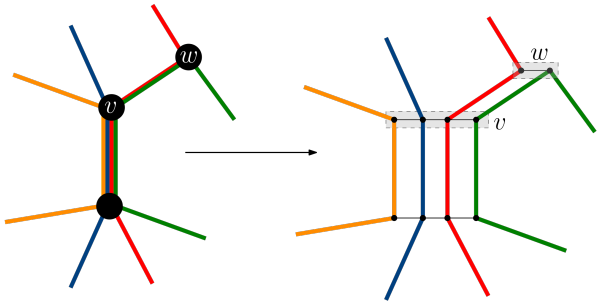


Figure 5 Construction of Q.

Step 1: If an edge is shared by s cycles, replace it s parallel edges. Each of these edges corresponds to a different cycle so that the resulting set of cycles is still pairwise non-crossing. Now the cycles are pairwise edge-disjoint but may still share some vertices.

Step 2: Let v be a vertex shared by two cycles C and C'. Edges incident to v are embedded around v in the cyclic order $e_1, a_1, \dots, a_i, e_2, b_1, \dots, b_j$ where $C \cap \delta(v) = \{e_1, e_2\}$. Since C and C' do not cross, we have $C' \cap \delta(v) \subseteq \{a_1, \dots, a_i\}$ or $C' \cap \delta(v) \subseteq \{b_1, \dots, b_j\}$. Then replace v by two adjacent vertices v', v'' and distribute the incident edges so that $\delta(v') = (e_1, a_1, \dots, a_i, e_2, \{v', v''\})$ and $\delta(v'') = (\{v', v''\}, b_1, \dots, b_j)$. Repeat step 2 until all cycles are vertex-disjoint.

It is easy to see that this graph is connected (since $G + H$ is connected) and can be embedded in the same surface $\mathbb{S}_g$. Figure 5 illustrates the construction of Q. Moreover, a cyclic ordering of the resulting cycles naturally induces a cyclic ordering of H. This completes the reduction. For simplicity, let us also call $\mathcal{H}$ the family of cycles in Q.

In the dual Q^*, let $\mathcal{K}$ denote the set of connected components of $Q^* \setminus \left(\bigcup_{C \in \mathcal{H}} C^*\right)$. They correspond to the connected components of $\mathbb{S}_g \setminus \left(\bigcup_{C \in \mathcal{H}} C\right)$. We say that a cycle $C \in \mathcal{H}$ is *incident* to a connected component $K \in \mathcal{K}$ if there is an edge in C^* with one endpoint in K. Consider the bipartite graph B that has a vertex for each cycle in $\mathcal{H}$ and a vertex for each element of $\mathcal{K}$, and whose edges represent the incidence relation. Next we show that the graph B is a cycle, and we order the D-cycles in $\mathcal{H}$ according to the cyclic order induced by B.

▷ Claim 21. B is a cycle.

The connectivity of B follows by construction from the connectivity of Q. Then it is enough to prove that this graph is 2-regular.

We first prove that each vertex of B that corresponds to a cycle in $\mathcal{H}$ has degree two in B. Since the cycles in $\mathcal{H}$ are disjoint, each cycle C has one component on its left, and one on its right, when we walk along the cycle. Assume, for a contradiction, that they are the same component: C is incident to only one component of $\mathbb{S}_g \setminus \left(\bigcup_{C \in \mathcal{H}} C\right)$. This cycle is also incident to only one component of $\mathbb{S}_g \setminus (C \cup C')$ where C' is any other cycle in $\mathcal{H}$. By Fact 4, we know that $\mathbb{S}_g \setminus (C \cup C')$ has two connected components. But since C is incident to only one connected component of $\mathbb{S}_g \setminus (C \cup C')$, $\mathbb{S}_g \setminus C'$ must also have two connected components, which contradicts the assumption that C' is non-separating. Thus, each cycle in B must have degree two.

Now we prove that each element of $\mathcal{K}$ has degree two. For a contradiction, assume that an element of $\mathcal{K}$ is incident to three cycles C, C', C'' or more. Then one component of $Q^* \setminus (C \cup C' \cup C'')$ is also incident to C, C' and C'', and $Q^* \setminus (C \cup C' \cup C'')$ has two or three components in total. If it has three components, then one of the other two components

would be incident to exactly one cycle, which would mean that this cycle is separating, a contradiction. If $Q^* \setminus (C \cup C' \cup C'')$ has exactly two connected components, then $Q^* \setminus (C \cup C')$ must be connected which contradicts Fact 4. Thus, each component is incident to exactly two cycles. This concludes the proof of the claim.

It remains to show that the order induced by B satisfies the property of Definition 16. If an edge $e = \{u, v\}$ of $G + H$ is shared by some cycles $C'_1, \ldots, C'_\ell$, then the vertex v can be mapped to a path $P = (v_1, \ldots, v_\ell)$ in Q, so that $C'_i \cap P = \{v_i\}, 1 \leqslant i \leqslant \ell$. See Figure 5. It follows that for all $1 \leqslant i \leqslant \ell - 1$, C'_i and C'_{i+1} are both incident the same connected component of $Q^* \setminus \left(\bigcup_{C \in \mathcal{H}} C\right)$ that contains the edge $\{v_i, v_{i+1}\}^*$. In particular, C'_i and C'_{i+1} are consecutive in the order induced by B. ◀

8 Proof of Theorem 1

By construction, the output of the algorithm is a feasible solution. We now analyze the value of the output. Since (1) is a relaxation of the maximum integral multiflow problem, $|f^*| \geqslant \mathrm{OPT}$. By Lemma 7, $|\overline{f}| \geqslant (1 - \epsilon)|f^*|$. For $\epsilon = \frac{1}{2}$ we have $|\overline{f}| \geqslant \frac{1}{2}|f^*|$.

Consider the multiflow restricted to separating cycles, $\overline{f}_{\mathrm{sep}}$. If $|\overline{f}_{\mathrm{sep}}| \geqslant \frac{1}{2}|\overline{f}|$, then by Theorem 10, Lemma 11, and Theorem 3 we obtain an integral flow of value at least $|\overline{f}_{\mathrm{sep}}|/\Theta(\sqrt{g})$.

Otherwise, by Theorem 14 there exists a free homotopy class $\mathcal{H}$ of non-separating cycles such that $|\overline{f}(\mathcal{H})| \geqslant |\overline{f}|/\Theta(g^2 \log g)$. Use Lemmas 17 and 20 to obtain that the output has value at least $|f^*|/\Theta(g^2 \log g)$.

Finally, we analyze the running time. As observed in Section 4, an optimum fractional multiflow f^* can be found in polynomial time. (Discretizing and) uncrossing is done in time polynomial in $|E||D|$ by Lemma 7. Partitioning into free homotopy classes is done by Corollary 15. Finally, the operations of Theorem 10, Theorem 3, Lemma 11, Lemma 17 and Lemma 20 can all be done in polynomial time, hence polynomial running time overall.

9 Proof of Corollary 2

In this section, we observe how Corollary 2 follows from Theorem 1 and the following result by Tardos and Vazirani [36] (based on work by Klein, Plotkin and Rao [24]).

▶ **Theorem 22** ([36]). *Let (G, H, u) be a multiflow instance and $\gamma > 1$ such that the supply graph G does not have a $K_{\gamma,\gamma}$ minor. Then the minimum capacity of a multicut is $O(\gamma^3)$ times the maximum value of a (fractional) multiflow.*

The following is well known.

▷ Claim 23. If a graph G has genus at most g, where $g \geqslant 1$, then it has no $K_{\gamma,\gamma}$ minor for any $\gamma > 2(\sqrt{g} + 1)$.

Proof. Suppose that such a minor $K_{\gamma,\gamma}$ exists in G. As the three operations for obtaining a minor (deleting edges/vertices and contracting edges) do not increase the genus, $K_{\gamma,\gamma}$ has genus at most g. Furthermore, K has 2γ vertices, γ^2 edges, and at most $\frac{\gamma^2}{2}$ faces (since there is no odd cycle in a bipartite graph). By Euler's formula, $2 - 2g \leqslant 2\gamma - \gamma^2 + \frac{\gamma^2}{2}$, which implies $\gamma \leqslant 2(\sqrt{g} + 1)$. ◁

By Claim 23 and Theorem 22, the ratio between the minimum capacity of a multicut and the maximum value of a (fractional) multiflow is $O(g^{1.5})$. This, combined with Theorem 1, proves Corollary 2.

References

1 Ravindra K. Ahuja, Thomas L. Magnanti, and James B. Orlin. *Network Flows*. Prentice-Hall, 1993.

2 Matthew Andrews, Julia Chuzhoy, Sanjeev Khanna, and Lisa Zhang. Hardness of the undirected edge-disjoint paths problem with congestion. In *Proceedings of the 46th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 226–244, 2015.

3 Shuchi Chawla, Robert Krauthgamer, Ravi Kumar, Yuval Rabani, and D. Sivakumar. On the hardness of approximating multicut and sparsest-cut. *Computational Complexity*, 15(2):94–114, 2006.

4 Chandra Chekuri, Sanjeev Khanna, and F. Bruce Shepherd. An $O(\sqrt{n})$ approximation and integrality gap for disjoint paths and unsplittable flow. *Theory of Computing*, 2:137–146, 2006.

5 Chandra Chekuri, F. Bruce Shepherd, and Christophe Weibel. Flow-cut gaps for integer and fractional multiflows. *Journal of Combinatorial Theory, Series B*, 103(2):248–273, 2013.

6 Julia Chuzhoy, David H. K. Kim, and Rachit Nimavat. New hardness results for routing on disjoint paths. In *Proceedings of the 49th Annual ACM Symposium on Theory of Computing Conference (STOC)*, pages 86–99, 2017.

7 Julia Chuzhoy, David H. K. Kim, and Rachit Nimavat. Almost polynomial hardness of node-disjoint paths in grids. In *Proceedings of the 50th Annual ACM Symposium on Theory of Computing Conference (STOC)*, pages 1220–1233, 2018.

8 Vincent Cohen-Addad, Éric Colin de Verdière, and Arnaud de Mesmay. A near-linear approximation scheme for multicuts of embedded graphs with a fixed number of terminals. In *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1439–1458, 2018.

9 Éric Colin de Verdière. Topological algorithms for graphs on surfaces. In J.E. Goodman and J. O'Rourke, editors, *Handbook of Discrete and Computational Geometry*. CRC Press, 2017. Chapter 23.

10 Marie-Christine Costa, Lucas Letocart, and Frederic Roupin. Minimal multicut and maximal integer multiflow: A survey. *European Journal of Operational Research*, 162(1):55–69, 2005.

11 Elias Dahlhaus, David S. Johnson, Christos H. Papadimitriou, Paul D. Seymour, and Mihalis Yannakakis. The complexity of multiterminal cuts. *SIAM Journal on Computing*, 23(4):864–894, 1994.

12 David B. A. Epstein. Curves on 2-manifolds and isotopies. *Acta Mathematica*, 115:83–107, 1966. doi:10.1007/BF02392203.

13 Jeff Erickson and Kim Whittlesey. Transforming curves on surfaces redux. In *Proceedings of the Twenty-Fourth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1646–1655. SIAM, 2013. doi:10.1137/1.9781611973105.118.

14 Samuel Fiorini, Nadia Hardy, Bruce Reed, and Adrian Vetta. Approximate min–max relations for odd cycles in planar graphs. *Mathematical Programming*, 110(1):71–91, 2007.

15 Lester R. Ford and Delbert R. Fulkerson. *Flows in Networks*. Princeton University Press, 1962.

16 Naveen Garg, Nikhil Kumar, and András Sebő. Integer plane multiflow maximisation: Flow-cut gap and one-quarter-approximation. In *Proceedings of IPCO*, pages 144–157, 2020.

17 Naveen Garg, Vijay Vazirani, and Mihalis Yannakakis. Approximate max-flow min-(multi)cut theorems and their applications. *SIAM Journal on Computing*, 25(2):235–251, April 1996.

18 Naveen Garg, Vijay Vazirani, and Mihalis Yannakakis. Primal-dual approximation algorithms for integral flow and multicut in trees. *Algorithmica*, 18(1):3–20, 1997.

19 Joshua Evan Greene. On curves intersecting at most once, 2018. arXiv:1807.05658.

20 Percy J. Heawood. Map colour theorem. *Quarterly Journal of Mathematics*, 24:332–338, 1890.

21 Chien-Chung Huang, Mathieu Mari, Claire Mathieu, Kevin Schewior, and Jens Vygen. An approximation algorithm for fully planar edge-disjoint paths. *SIAM Journal on Discrete Mathematics*, 35:752–769, 2021.

22 Richard M. Karp. On the computational complexity of combinatorial problems. *Networks*, 5(1):45–68, 1975.

23 Ken-ichi Kawarabayashi and Yusuke Kobayashi. An $O(\log n)$-approximation algorithm for the edge-disjoint paths problem in Eulerian planar graphs. *ACM Transactions on Algorithms*, 9(2):16:1–16:13, 2013.

24 Philip Klein, Serge A. Plotkin, and Satish Rao. Excluded minors, network decomposition, and multicommodity flow. In *Proceedings of the Twenty-Fifth Annual ACM Symposium on Theory of Computing (STOC)*, pages 682–690, 1993. doi:10.1145/167088.167261.

25 Philip N. Klein, Claire Mathieu, and Hang Zhou. Correlation clustering and two-edge-connected augmentation for planar graphs. In *Proceedings of the 32nd International Symposium on Theoretical Aspects of Computer Science (STACS)*, pages 554–567, 2014.

26 Francis Lazarus and Julien Rivaud. On the homotopy test on surfaces. In *Proceedings of the 53rd Annual IEEE Symposium on Foundations of Computer Science, (FOCS)*, pages 440–449. IEEE Computer Society, 2012. doi:10.1109/FOCS.2012.12.

27 Justin Malestein, Igor Rivin, and Louis Theran. Topological designs. *Geometriae Dedicata*, 168:221–233, August 2010. doi:10.1007/s10711-012-9827-9.

28 Matthias Middendorf and Frank Pfeiffer. On the complexity of the disjoint paths problem. *Combinatorica*, 13(1):97–107, March 1993. doi:10.1007/BF01202792.

29 Bojan Mohar and Carsten Thomassen. *Graphs on Surfaces*. Johns Hopkins Series in the Mathematical Sciences. Johns Hopkins University Press, 2001. URL: http://jhupbooks.press.jhu.edu/ecom/MasterServlet/GetItemDetailsHandler?iN=9780801866890&qty=1&source=2&viewMode=3&loggedIN=false&JavaScript=y.

30 Guyslain Naves and András Sebő. Multiflow feasibility: an annotated tableau. In W. Cook, L. Lovász, and J. Vygen, editors, *Research Trends in Combinatorial Optimization*, pages 261–283. Springer, 2009.

31 Piotr Przytycki. Arcs intersecting at most once. *Geometric and Functional Analysis*, 25:658–670, February 2015. doi:10.1007/s00039-015-0320-0.

32 Neil Robertson, Daniel P. Sanders, Paul Seymour, and Robin Thomas. Efficiently four-coloring planar graphs. In *Proceedings of the Twenty-Eighth Annual ACM Symposium on Theory of Computing (SODA)*, pages 571–575, 1996.

33 Neil Robertson and Paul D. Seymour. Graph minors. XIII. The disjoint paths problem. *Journal of Combinatorial Theory, Series B*, 63(1):65–110, 1995.

34 Alexander Schrijver. *Combinatorial Optimization: Polyhedra and Efficiency*. Springer, 2003.

35 András Sebö. Integer plane multiflows with a fixed number of demands. *Journal of Combinatorial Theory, Series B*, 59(2):163–171, 1993. doi:10.1006/jctb.1993.1062.

36 Éva Tardos and Vijay V. Vazirani. Improved bounds for the max-flow min-multicut ratio for planar and $K_{r,r}$-free graphs. *Information Processing Letters*, 47:77–80, August 1993.

Minimum-Norm Load Balancing Is (Almost) as Easy as Minimizing Makespan

Sharat Ibrahimpur
Department of Combinatorics and Optimization, University of Waterloo, Canada

Chaitanya Swamy
Department of Combinatorics and Optimization, University of Waterloo, Canada

Abstract

We consider the *minimum-norm load-balancing* (MinNormLB) problem, wherein there are n jobs, each of which needs to be assigned to one of m machines, and we are given the processing times $\{p_{ij}\}$ of the jobs on the machines. We also have a monotone, symmetric norm $f : \mathbb{R}^m \to \mathbb{R}_{\geq 0}$. We seek an assignment σ of jobs to machines that minimizes the f-norm of the induced load vector $\overrightarrow{\mathsf{load}}_\sigma \in \mathbb{R}^m_{\geq 0}$, where $\mathsf{load}_\sigma(i) = \sum_{j:\sigma(j)=i} p_{ij}$. This problem was introduced by [4], and the current-best result for MinNormLB is a $(4+\epsilon)$-approximation [5]. In the stochastic version of MinNormLB, the job processing times are given by nonnegative random variables X_{ij}, and jobs are independent; the goal is to find an assignment σ that minimizes the *expected* f-norm of the induced random load vector.

We obtain results that (essentially) *match* the best-known guarantees for deterministic makespan minimization (MinNormLB with ℓ_∞ norm). For MinNormLB, we obtain a $(2+\epsilon)$-approximation for unrelated machines, and a PTAS for identical machines. For stochastic MinNormLB, we consider the setting where the X_{ij}s are *Poisson* random variables, denoted PoisNormLB. Our main result here is a novel and powerful reduction showing that, for any machine environment (e.g., unrelated/identical machines), *any* α-approximation algorithm for MinNormLB in that machine environment yields a randomized $\alpha(1+\epsilon)$-approximation for PoisNormLB in that machine environment. Combining this with our results for MinNormLB, we immediately obtain a $(2+\epsilon)$-approximation for PoisNormLB on unrelated machines, and a PTAS for PoisNormLB on identical machines. The latter result substantially generalizes a PTAS for makespan minimization with Poisson jobs obtained recently by [6].

2012 ACM Subject Classification Theory of computation → Approximation algorithms analysis

Keywords and phrases Approximation algorithms, Load balancing, Minimum-norm optimization, LP rounding

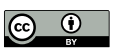
Digital Object Identifier 10.4230/LIPIcs.ICALP.2021.81

Category Track A: Algorithms, Complexity and Games

Funding Supported in part by NSERC grant 327620-09 and an NSERC DAS Award.

1 Introduction

In the *minimum-norm load-balancing* (MinNormLB) problem, we are given a set J of n jobs, a set of m machines, and nonnegative job processing times (or sizes) $\{p_{ij}\}_{i\in[m],j\in J}$. We use $[m]$ to denote $\{1,\ldots,m\}$. We are also given a *monotone, symmetric norm* $f : \mathbb{R}^m \mapsto \mathbb{R}_{\geq 0}$. Recall that f being a norm means that: (i) $f(x) = 0$ iff $x = 0$; (ii) $f(x+y) \leq f(x) + f(y)$ for all $x, y \in \mathbb{R}^m$, and (iii) $f(\theta x) = |\theta| f(x)$ for all $x \in \mathbb{R}^m, \theta \in \mathbb{R}$. A monotone norm f satisfies $f(x) \leq f(y)$ for all $0 \leq x \leq y$, and symmetry is the property that permuting the coordinates of x does not change its norm. An assignment $\sigma : J \to [m]$ of jobs to machines induces the machine-load vector $\overrightarrow{\mathsf{load}}_\sigma = \big(\mathsf{load}_\sigma(i)\big)_{i\in[m]} \in \mathbb{R}^m_{\geq 0}$, where $\mathsf{load}_\sigma(i) := \sum_{j:\sigma(j)=i} p_{ij}$. The goal in MinNormLB is to find an assignment σ that minimizes the f-norm of $\overrightarrow{\mathsf{load}}_\sigma$.



This problem was first considered by [4], as a natural problem in the genre of minimum-norm optimization problems that they introduce. They gave a $(38+\varepsilon)$-approximation algorithm for MinNormLB, and the approximation factor was subsequently improved to $(4+\varepsilon)$ by [5]. Ibrahimpur and Swamy [12] introduced the genre of *stochastic* minimum-norm optimization, and considered StochNormLB, the stochastic generalization of MinNormLB, as a prominent problem in this genre. In StochNormLB, the job processing times are given by nonnegative random variables $\{X_{ij}\}_{i\in[m],j\in J}$ with specified distributions, and the jobs are independent (but X_{ij} and $X_{i'j}$ could be correlated); the goal is to find an assignment that minimizes the *expected* f-norm of the induced random load vector. As discussed in these works, one of the chief motivations and benefits of working with the rather broad class of monotone, symmetric norms is that it captures a variety of appealing objectives, including the frequently-considered (in both deterministic and stochastic settings) min-max (ℓ_∞) and min-sum (ℓ_1) objectives, general ℓ_p-norms, as also another important class of norms called Top_ℓ*norms* (for $x \geq 0$, $\mathsf{Top}_\ell(x)$ is the sum of the ℓ largest coordinates of x). Moreover, by exploiting the closure properties of monotone, symmetric norms, one can also model seemingly more general settings, such as when we have *multiple* monotone-symmetric-norm budget constraints $f_\ell(x) \leq B_\ell$ (a flexibility we leverage in Section 3).

The special case of MinNormLB where $f = \ell_\infty$ yields the classical (deterministic) *makespan-minimization* problem, which has been well studied for various machine environments. For unrelated machines, a 2-approximation is known [17, 23], and improving this factor remains a longstanding open problem, while various PTAS'es are known for identical and related machines [11, 10, 13, 14]. Similar guarantees are known for MinNormLB with (general) ℓ_p norms [2, 16, 20, 1]. Stochastic load balancing is less well-understood than its deterministic counterpart (even for the makespan objective). Constant-factor approximations are known for stochastic makespan minimization on identical [15] and unrelated [8] machines, StochNormLB with ℓ_p norms [22] and Top_ℓ norms [12], and StochNormLB with (an arbitrary f and) Bernoulli job sizes [12]. A common shortcoming of all these works is that the approximation factors obtained are quite large, at least in the 100s (although these works were not aiming to optimize the constant). With an eye towards obtaining small approximation factors, Goel and Indyk [7] considered stochastic makespan minimization on identical machines with structured distributions. They obtained (among other results) a 2-approximation for StochNormLB with Poisson job sizes – i.e., where each X_{ij} is a Poisson random variable – and very recently, this was improved to a PTAS [6].

Our contributions. As suggested by the title of the paper, we obtain results for both deterministic and stochastic load balancing with *approximation factors that (essentially) match the best-known approximation factors for the deterministic makespan-minimization problem.* Our salient results are as follows.

- For MinNormLB, we devise a $(2+\epsilon)$-approximation for unrelated machines (Theorem 4.1), and a PTAS for identical machines (Theorem 5.1).
  For unrelated machines, this improves upon the previous-best $(4+\epsilon)$-approximation [5].
- We consider StochNormLB with Poisson job sizes, denoted PoisNormLB, and give a novel, clean, and fairly general reduction showing that, for any machine environment (e.g., unrelated/identical machines), *any* α-approximation algorithm for MinNormLB in that machine environment can be used to obtain a randomized $\alpha(1+\epsilon)$-approximation for PoisNormLB in that machine environment (Theorem 3.1). Combining this with our results for MinNormLB, we obtain a $(2+\epsilon)$-approximation for PoisNormLB on unrelated machines, and a PTAS for PoisNormLB on identical machines.

Our approximation factors (which are for general monotone, symmetric norms) are *considerably* smaller than the factors known (even) for stochastic makespan minimization with general distributions. Our PTAS for PoisNormLB on identical machines substantially generalizes the PTAS by [6] for stochastic makespan minimization with Poisson jobs. Our techniques are quite different, and our approach, based on the reduction to MinNormLB, while being more general, is in fact also simpler and cleaner than the one in [6].

All our algorithms require only a *value oracle* for the norm f. (In contrast, the results of [4, 5] need more-sophisticated oracle access to f.)

Our techniques. We briefly discuss the key techniques underlying some of our results.

Consider first the reduction from PoisNormLB to MinNormLB. Let $\mathsf{Pois}(\lambda)$ denote a Poisson random variable with parameter λ; recall that this has mean λ. The sum $\mathsf{Pois}(\lambda_1) + \mathsf{Pois}(\lambda_2)$ of independent Poisson random variables is a $\mathsf{Pois}(\lambda_1 + \lambda_2)$ random variable. So given an assignment $\sigma : J \to [m]$, the load on any machine i is a $\mathsf{Pois}(\Lambda_i^\sigma)$ random variable, where Λ_i^σ is the expected load on machine i, and the objective value of σ is $g(\Lambda^\sigma) := \mathbf{E}\left[f(\mathsf{Pois}(\Lambda_1^\sigma), \mathsf{Pois}(\Lambda_2^\sigma), \ldots, \mathsf{Pois}(\Lambda_m^\sigma))\right]$. It is known that if $\mathsf{Top}_i(y) \leq \mathsf{Top}_i(y')$ for all $i \in [m]$ then $g(y) \leq g(y')$ (Theorem 3.3). We argue that g is *subhomogeneous*, i.e., $g(\theta y) \leq \theta \cdot g(y)$ for any $\theta \geq 1$, and so one can generalize the above statement to say that if $\mathsf{Top}_i(y) \leq \theta \cdot \mathsf{Top}_i(y')$ for all $i \in [m]$, then $g(y) \leq \theta \cdot g(y')$. Let Λ^* denote the Λ-vector of an optimal solution. The idea now is to "guess" $\mathsf{Top}_\ell(\Lambda^*)$ within a $(1+\varepsilon)$-factor for all ℓ in a certain sparse set $\mathsf{POS} \subseteq [m]$; let B_ℓ^* denote such an overestimate of $\mathsf{Top}_\ell(\Lambda^*)$. We now seek a solution σ such that $\mathsf{Top}_\ell(\Lambda^\sigma) \leq \alpha B_\ell^*$ for all $\ell \in \mathsf{POS}$, for some $\alpha \geq 1$; this will imply that $\mathsf{Top}_i(\Lambda^\sigma) \leq \alpha\big(1 + O(\varepsilon)\big)\mathsf{Top}_i(\Lambda^*)$ for all $i \in [m]$, and hence $g(\Lambda^\sigma) \leq \alpha(1 + O(\varepsilon))g(\Lambda^*)$. This is where the generality of monotone, symmetric norms comes in handy. *We can cast this multiple-*Top_ℓ*-norm-budgets problem as a* MinNormLB *problem, with* $\{\lambda_{ij}\}$ *job sizes, and monotone, symmetric norm given by* $h(v) := \max_{\ell \in \mathsf{POS}} \mathsf{Top}_\ell(v)/B_\ell^*$; we can now use an α-approximation algorithm for MinNormLB, say $\mathcal{A}^{\mathsf{Det}}$, to find σ.

Note that the versatility afforded by MinNormLB as a model is crucial for the reduction, and we really need $\mathcal{A}^{\mathsf{Det}}$ to work for an arbitrary monotone, symmetric norm. We need to have fine-grained control of the Λ^σ vector (as dictated by the multiple Top_ℓ-norm budget constraints), and we define a suitable (monotone, symmetric) norm h to enforce this. In contrast, De et al. [6], who devise a PTAS for stochastic makespan minimization with Poisson jobs follow a completely different approach, based on proving suitable concentration results.

Our $(2+\epsilon)$-approximation for MinNormLB on unrelated machines (Section 4) is based on rounding the solution to a novel LP-relaxation for the problem. Let σ^* be an optimal solution, $\vec{o}$ be the induced load vector, and $\mathsf{OPT} = f(\vec{o})$. It suffices to obtain an assignment σ such that $\mathsf{Top}_\ell(\overrightarrow{\mathsf{load}}_\sigma) \leq \big(2 + O(\delta)\big)\mathsf{Top}_\ell(\vec{o})$ for all $\ell \in \mathsf{POS}$ (see Theorem 2.4 and Claim 2.6). Roughly speaking, for all $\ell \in \mathsf{POS}$, we guess the ℓ-th largest entry of $\vec{o}$, and use this to linearly encode the constraint that $\mathsf{Top}_\ell(\overrightarrow{\mathsf{load}}_\sigma) \leq \mathsf{Top}_\ell(\vec{o})$. LP-rounding algorithms for the special case of makespan minimization typically return a guarantee where the load on a machine is at most its LP-load + (maximum cost (i.e., size) of a job assigned to it). In order to bound the f-norm of the portion of the load-vector, say P, arising from the most-costly jobs, we also consider the vector $\overrightarrow{\mathsf{JL}^*}$ comprising the costs of the m most-costly jobs under σ^*. In [5], it is shown that $f(\overrightarrow{\mathsf{JL}^*}) \leq \mathsf{OPT}$. We indirectly encode that $f(P) \leq f(\overrightarrow{\mathsf{JL}^*})$ by guessing the ℓ-th largest entry, say ζ_ℓ, of $\overrightarrow{\mathsf{JL}^*}$ and encoding that there are at most $\ell - 1$ jobs of higher cost, for all $\ell \in \mathsf{POS}$. We round a fractional solution to the resulting LP using *iterative rounding* to obtain two types of guarantees *simultaneously* (see Lemma 4.3):

(i) similar to above, the load on machine i is at most its LP-load + $(1+\delta)\times$(maximum cost of a job assigned to i); and (ii) the number of jobs having cost larger then ζ_ℓ is at most $(1+\delta)\ell - 1$. The LP constraints explicitly encode that the Top_ℓ-norms of the LP-load vector are roughly speaking at most $\mathsf{Top}_\ell(\vec{o})$, for all $\ell \in \mathsf{POS}$; guarantee (ii) allows us to argue that $f(P) \leq \big(1 + O(\delta)\big)\mathsf{OPT}$. Together, these imply a $(2+\epsilon)$-approximation.

We remark that the approach taken in the GAP-rounding algorithm of [23] (and also used for Top_ℓ-norm minimization in [4]), wherein the problem is essentially reduced to a bipartite matching problem, does not seem to be helpful in obtaining a $(2+\epsilon)$-approximation for MinNormLB. This approach "hard-codes" the distinction between the most-costly job assigned to a machine and the remaining jobs as a means of bounding the load on a machine by its LP-load + (maximum cost of a job assigned to it). However, bounding $f(P)$ then entails solving (to within a $(1+\epsilon)$-factor) a (bipartite) matching problem with $|\mathsf{POS}| = O(\log m)$ side-constraints, or essentially a min-norm matching problem, but neither of these has (even) an $O(1)$-approximation. Instead, iterative rounding seems crucial for simultaneously obtaining guarantees (i) and (ii) above, and (ii) allows us to obtain a sufficiently-tight bound on the Top_ℓ-norms (and hence the f-norm) of P and thereby obtain our $(2+\epsilon)$-approximation.

2 Preliminaries

For a vector $v \in \mathbb{R}^m_{\geq 0}$, we use $v^\downarrow$ to denote v with its coordinates sorted in non-increasing order; i.e., we have $v^\downarrow_i = v_{\pi(i)}$, where π is a permutation of $[m]$ such that $v_{\pi(1)} \geq v_{\pi(2)} \geq \ldots \geq v_{\pi(m)}$. We say that v is non-increasing if $v_1 \geq \ldots \geq v_m$ (i.e., $v = v^\downarrow$). Whenever we say norm in the sequel, we always mean a monotone, symmetric norm. The following claim from [12] will be useful in obtaining bounds on the optimal value.

▷ Claim 2.1 (Claim 3.2 in [12]). Let $h : \mathbb{R}^m \mapsto \mathbb{R}_{\geq 0}$ be a monotone, symmetric norm. For any $v \in \mathbb{R}^m_{\geq 0}$, we have $\frac{\sum_{i\in[m]} v_i}{m} \leq \max_{i\in[m]} v_i \leq \frac{h(v)}{h(1,0,\ldots,0)} \leq \sum_{i\in[m]} v_i$.

As established in prior work on minimum-norm optimization [4, 12], we can control the norm of a vector by controlling all its Top_ℓ norms, which are defined below. Theorem 2.4 makes this notion precise. For $x \in \mathbb{R}$, we use $(x)^+$ to denote $\max\{0, x\}$.

▶ **Definition 2.2.** Let $\ell \in [m]$. The Top_ℓ *norm* is defined as follows: for $v \in \mathbb{R}^m_{\geq 0}$, $\mathsf{Top}_\ell(v)$ is the sum of the ℓ largest coordinates of v, i.e., $\mathsf{Top}_\ell(v) = \sum_{i=1}^{\ell} v^\downarrow_i$.

The following ways of reformulating the Top_ℓ-norm will be useful. For a vector $v \in \mathbb{R}^m$ and $\theta \in \mathbb{R}$, define $N^{>\theta}(v) := \big|\{i \in [m] : v_i > \theta\}\big|$.

▷ Claim 2.3. Let $v \in \mathbb{R}^m_{\geq 0}$, and $\ell \in [m]$. Then

$$\mathsf{Top}_\ell(v) = \min_{t\geq 0}\big(\ell t + \sum_{i\in[m]} (v_i - t)^+\big) = \ell v^\downarrow_\ell + \sum_{i\in[m]} (v_i - v^\downarrow_\ell)^+ = \int_0^\infty \min\{\ell, N^{>\theta}(v)\} d\theta.$$

▶ **Theorem 2.4** (Follows from structural result in [4], or majorization theory of [9]). *If $x, y \in \mathbb{R}^m_{\geq 0}$ are such that $\mathsf{Top}_\ell(x) \leq \alpha\mathsf{Top}_\ell(y) + \beta$ for all $\ell \in [m]$, where $\alpha, \beta \geq 0$, then $h(x) \leq \alpha \cdot h(y) + \beta \cdot h(1, 0, \ldots, 0)$ for any monotone, symmetric norm $h : \mathbb{R}^m \mapsto \mathbb{R}_{\geq 0}$.*

Theorem 2.4 will be our chief means for reasoning about the norm of a vector. Our algorithms will "guess" (i.e., enumerate) the Top_ℓ norms (or certain associated quantities) of the load vector $\vec{o}$ induced by an optimal solution, and will aim to obtain a solution whose induced load vector $\overrightarrow{\mathsf{load}}$ satisfies $\mathsf{Top}_\ell(\overrightarrow{\mathsf{load}}) = O\big(\mathsf{Top}_\ell(\vec{o})\big)$. However, to make this approach

polynomial time, we will only be able to enumerate the Top_ℓ norms for a certain *sparse* subset of indices $\mathsf{POS} \subseteq [m]$. The next few definitions and results make this precise, and show that the move to this sparse subset only incurs the loss of a small factor.

Let $\delta > 0$ be a parameter. We define $\mathsf{POS}_{m,\delta} \subseteq [m]$ iteratively as follows: include the index 1 in $\mathsf{POS}_{m,\delta}$; as long as the largest index $\ell \in \mathsf{POS}_{m,\delta}$ is such that $\lceil (1+\delta)\ell \rceil \leq m$, include $\lceil (1+\delta)\ell \rceil$ (which is larger than ℓ) in $\mathsf{POS}_{m,\delta}$. (This definition is mathematically slightly more convenient to work with than the one in [4], where $\mathsf{POS}_{m,\delta}$ is defined as $\left\{\min\{\lceil (1+\delta)^s \rceil, m\} : s \in \mathbb{Z}_{\geq 0}\right\}$, but this change is not crucial.)

▷ Claim 2.5. We have $|\mathsf{POS}_{m,\delta}| \leq 1 + \log_{1+\delta} m = O\left(\frac{\log m}{\delta}\right)$.

We frequently abbreviate $\mathsf{POS}_{m,\delta}$ to POS in the remainder of this section, and whenever m, δ are clear from the context. For $i \in [m]$, let $\mathsf{next}(i)$ be the smallest index in POS (strictly) larger than i; if no such index exists, then we define $\mathsf{next}(i) := m+1$ for notational convenience. Similarly, let $\mathsf{prev}(i)$ be the largest index in POS (strictly) smaller than i; set $\mathsf{prev}(1) := 0$. It is immediate from the definition of POS that $\mathsf{next}(\ell) - 1 \leq (1+\delta)\ell$ for all $\ell \in \mathsf{POS}$; it follows also that $\mathsf{next}(i) - 1 \leq (1+\delta)i$ for all $i \in [m]$. Claim 2.6 and Lemma 2.7 show that focusing on only the indices in POS only results in a $(1+\delta)$-factor loss.

▷ Claim 2.6. Let $u, v \in \mathbb{R}^m_{\geq 0}$ be such that $\mathsf{Top}_\ell(u) \leq \mathsf{Top}_\ell(v)$ for all $\ell \in \mathsf{POS}_{m,\delta}$. Then we have $h(u) \leq (1+\delta)h(v)$ for any monotone, symmetric norm $h : \mathbb{R}^m \mapsto \mathbb{R}_{\geq 0}$.

Proof. Let $i \in [m] \setminus \mathsf{POS}$, and $\ell = \mathsf{prev}(i)$. Then $i \leq (1+\delta)\ell$, and therefore $\mathsf{Top}_i(u) \leq (1+\delta)\mathsf{Top}_\ell(u) \leq (1+\delta)\mathsf{Top}_\ell(v) \leq (1+\delta)\mathsf{Top}_i(v)$. The claim now follows from Theorem 2.4. ◁

▶ **Lemma 2.7.** *Let $u, v \in \mathbb{R}^m_{\geq 0}$ be such that $u^\downarrow_\ell \leq v^\downarrow_\ell$ for all $\ell \in \mathsf{POS}_{m,\delta}$. Then, we have $h(u) \leq (1+\delta)h(v)$ for any monotone, symmetric norm $h : \mathbb{R}^m \mapsto \mathbb{R}_{\geq 0}$.*

Proof. By Theorem 2.4, it suffices to show that $\mathsf{Top}_i(u) \leq (1+\delta)\mathsf{Top}_i(v)$ for all $i \in [m]$. So fix $i \in [m]$. We mimic the proof of Lemma 4.2 in [4]. Define vectors $\alpha, \beta \in \mathbb{R}^m_{\geq 0}$ as follows:

$$\alpha_k = \begin{cases} u^\downarrow_k; & \text{if } k \in \{1, \ldots, i\} \\ 0 & \text{otherwise} \end{cases} \qquad \beta_k = \begin{cases} v^\downarrow_k; & \text{if } k \in \{1, \ldots, i\} \\ 0 & \text{otherwise.} \end{cases}$$

Clearly, both α and β are non-increasing vectors. For notational convenience, set $\alpha_k := 0, \beta_k := 0$ for any $k > m$. We have

$$\begin{aligned} \mathsf{Top}_\ell(u) = \sum_{k=1}^{m} \alpha_k &= \sum_{\ell \in \mathsf{POS}} \sum_{k=\ell}^{\mathsf{next}(\ell)-1} \alpha_k \\ &\leq \sum_{\ell \in \mathsf{POS}} \alpha_\ell \big((\mathsf{next}(\ell) - 1) - (\ell - 1) \big) = \sum_{\ell \in \mathsf{POS}} \big(\mathsf{next}(\ell) - 1 \big)(\alpha_\ell - \alpha_{\mathsf{next}(\ell)}) \end{aligned}$$

where the inequality follows because α is a non-increasing vector. Now using the fact that $\mathsf{next}(\ell) - 1 \leq (1+\delta)\ell$ for all $\ell \in \mathsf{POS}$, and $\alpha_\ell \leq \beta_\ell$ for all $\ell \in \mathsf{POS}$, we obtain that

$$\begin{aligned} \mathsf{Top}_\ell(u) &\leq (1+\delta) \sum_{\ell \in \mathsf{POS}} \ell(\alpha_\ell - \alpha_{\mathsf{next}(\ell)}) = (1+\delta) \sum_{\ell \in \mathsf{POS}} \alpha_\ell \big(\ell - \mathsf{prev}(\ell) \big) \leq (1+\delta) \sum_{\ell \in \mathsf{POS}} \beta_\ell \big(\ell - \mathsf{prev}(\ell) \big) \\ &\leq (1+\delta) \sum_{\ell \in \mathsf{POS}} \sum_{k=\mathsf{prev}(\ell)+1}^{\ell} \beta_k \leq (1+\delta) \sum_{k=1}^{m} \beta_i = (1+\delta)\mathsf{Top}_\ell(v). \end{aligned}$$

The third inequality above follows because β is a non-increasing vector. ◀

In our algorithms, we work with estimates of $\{\vec{o}_\ell^{\downarrow}\}_{\ell \in \mathsf{POS}}$, where $\vec{o}$ is the load vector induced by an optimal solution. We show that these estimates then allow us to infer an estimate of $h(\vec{o})$ for any monotone, symmetric norm h. We need the following notation. Given a non-increasing vector $v \in \mathbb{R}^{\mathsf{POS}}_{\geq 0}$, we define its *expansion* to be the vector $v^{\mathsf{exp}} \in \mathbb{R}^m_{\geq 0}$ given by $v_i^{\mathsf{exp}} := v_i$ for $i \in \mathsf{POS}$, and $v_i^{\mathsf{exp}} = v_{\mathsf{prev}(i)}$ for $i \in [m] \setminus \mathsf{POS}$.

▶ **Lemma 2.8.** *Let $u \in \mathbb{R}^m_{\geq 0}$, and $v \in \mathbb{R}^{\mathsf{POS}}_{\geq 0}$ be a non-increasing vector. Let $h : \mathbb{R}^m \mapsto \mathbb{R}_{\geq 0}$ be a monotone, symmetric norm. Let $\varepsilon, \kappa > 0$.*

(a) *If $u_\ell^{\downarrow} \leq v_\ell$ for all $\ell \in \mathsf{POS}$, then $\mathsf{Top}_i(u) \leq \mathsf{Top}_i(v^{\mathsf{exp}})$ for all $i \in [m]$, and hence, $h(u) \leq h(v^{\mathsf{exp}})$.*

(b) *If $v_\ell \leq (1+\varepsilon)u_\ell^{\downarrow} + \kappa$ for all $\ell \in \mathsf{POS}$, then $\mathsf{Top}_i(v^{\mathsf{exp}}) \leq (1+\delta)(1+\varepsilon)\mathsf{Top}_i(u) + i\kappa$ for all $i \in [m]$, and hence, $h(v^{\mathsf{exp}}) \leq (1+\delta)(1+\varepsilon)h(u) + m\kappa \cdot h(1,0,\ldots,0)$.*

(c) *Let v be as in part (b). Let $\delta \leq 1$ (in $\mathsf{POS} = \mathsf{POS}_{m,\delta}$). Let $\alpha \in \mathbb{R}^m$ be such that $\alpha_1^{\downarrow} \leq v_1$ and $N^{>v_\ell}(\alpha) \leq (1+\delta)\ell - 1$ for all $\ell \in \mathsf{POS}$. Then, $\mathsf{Top}_i(\alpha) \leq (1+4\delta)(1+\varepsilon)\mathsf{Top}_i(u) + 5i\kappa$ for all $i \in [m]$. Hence, $h(\alpha) \leq (1+4\delta)(1+\varepsilon)h(u) + 5m\kappa \cdot h(1,0,\ldots,0)$.*

Lemma 2.8 (c) can be seen as a generalization of Lemma 2.8 (b): the vector v^{exp} satisfies $N^{>v_\ell}(v^{\mathsf{exp}}) \leq \ell - 1$ for all $\ell \in \mathsf{POS}$, while the vector α satisfies a relaxed version of this bound.

Proof. The inequalities involving $h(.)$ in parts (a)–(c) follow from the corresponding bounds on the Top_i-norms, using Theorem 2.4. So we focus on the proving the bounds for the Top_i-norms.

Let $\gamma = v^{\mathsf{exp}}$. Part (a) follows immediately from the fact that $u^{\downarrow} \leq \gamma$. For part (b), define $\beta \in \mathbb{R}^m_{\geq 0}$ to be the expansion of $(u_\ell^{\downarrow})_{\ell \in \mathsf{POS}}$. Then, we have $\gamma \leq (1+\varepsilon)\beta + \kappa \cdot \mathbf{1}$, where $\mathbf{1}$ is the vector of all 1s, and hence, $\mathsf{Top}_i(\gamma) \leq (1+\varepsilon)\mathsf{Top}_i(\beta) + i\kappa$, for any $i \in [m]$. Observe that $\beta_\ell \leq u_\ell^{\downarrow}$ for all $\ell \in \mathsf{POS}$. Therefore, by Lemma 2.7, we have $\mathsf{Top}_i(\beta) \leq (1+\delta)\mathsf{Top}_i(u)$ for any $i \in [m]$.

For part (c), consider an index $i \in [m]$. We use the reformulation $\mathsf{Top}_i(\alpha) = \int_0^\infty \min\{i, N^{>\theta}(\alpha)\}d\theta$ stated in Claim 2.3. Since $N^{>v_1}(\alpha) = 0$, we only need to go up to v_1 in the above integral. Let $\overline{\ell} = i$ if $i \in \mathsf{POS}$, and $\overline{\ell} = \mathsf{prev}(i)$ otherwise. Observe that $i \leq (1+\delta)\overline{\ell}$. We have the following chain of inequalities.

$$\begin{aligned}
\mathsf{Top}_i(\alpha) &\leq \int_0^{v_{\overline{\ell}}} i d\theta + \sum_{\ell \in \mathsf{POS}: 1 < \ell \leq \overline{\ell}} \int_{v_\ell}^{v_{\mathsf{prev}(\ell)}} N^{>\theta}(\alpha) d\theta \leq i \cdot v_{\overline{\ell}} + \sum_{\ell \in \mathsf{POS}: 1 < \ell \leq \overline{\ell}} (v_{\mathsf{prev}(\ell)} - v_\ell) N^{>v_\ell}(\alpha) \\
&\leq i \cdot v_{\overline{\ell}} + \sum_{\ell \in \mathsf{POS}: 1 < \ell \leq \overline{\ell}} (v_{\mathsf{prev}(\ell)} - v_\ell)\big((1+\delta)\ell - 1\big) \\
&\leq i \cdot v_{\overline{\ell}} + \sum_{\ell \in \mathsf{POS}: 1 < \ell \leq \overline{\ell}} (v_{\mathsf{prev}(\ell)} - v_\ell)\big((1+\delta)^2 \mathsf{prev}(\ell) + \delta\big). \qquad (1)
\end{aligned}$$

The second inequality is because $N^{>\theta}(\alpha)$ is non-increasing in θ; the third is from condition (ii) in the lemma statement; and the final inequality (1) follows since $\ell - 1 \leq (1+\delta)\mathsf{prev}(\ell)$. Recall that $\mathsf{prev}(1) = 0$. Continuing, since $i \leq (1+\delta)\overline{\ell} \leq (1+\delta)^2\overline{\ell}$, the RHS of (1) is at most

$$\begin{aligned}
&(1+\delta)^2 \sum_{\ell \in \mathsf{POS}: \ell \leq \overline{\ell}} v_\ell\big(\ell - \mathsf{prev}(\ell)\big) + \delta v_1 \\
&\leq (1+\delta)^2 \sum_{\ell \in \mathsf{POS}: \ell \leq \overline{\ell}} \big((1+\varepsilon)u_\ell^{\downarrow} + \kappa\big)\big(\ell - \mathsf{prev}(\ell)\big) + \delta(1+\varepsilon) \cdot \mathsf{Top}_i(u) + \delta\kappa
\end{aligned}$$

$$\ldots \le (1+\delta)^2(1+\varepsilon)\sum_{i'=1}^{\overline{\ell}} u^{\downarrow}_{i'} + (1+\delta)^2\overline{\ell}\kappa + \delta(1+\varepsilon)\cdot \mathsf{Top}_i(u) + \delta\kappa$$
$$\le (1+4\delta)(1+\varepsilon)\mathsf{Top}_i(u) + 5i\kappa. \qquad \text{(since } \delta \le 1\text{)}$$ ◀

Our algorithms will often need to estimate a non-increasing vector α (such as $(\vec{o}^{\downarrow}_\ell)_{\ell\in\mathsf{POS}}$). We show that if we have suitable bounds on the coordinates of α, then one can identify a (polynomially bounded) set containing a vector close to α.

▶ **Lemma 2.9.** *Let $\mathcal{L} \subseteq [m]$ be an index-set. Let $\alpha \in \mathbb{R}^{\mathcal{L}}_{\ge 0}$ be a non-increasing vector, i.e., $\alpha_\ell \ge \alpha_{\ell'}$ for indices $\ell, \ell' \in \mathcal{L}$, $\ell < \ell'$. Let ub be such that $\alpha_\ell \le \mathsf{ub}$ for all $\ell \in \mathcal{L}$. Let $\varepsilon, \kappa > 0$, and $\varepsilon' = \min\{1, \varepsilon\}$.*

(a) *Let $N_1 := (2e)^{|\mathcal{L}|} + \left(\frac{\mathsf{ub}}{\kappa}\right)^{O(\frac{1}{\varepsilon'})}$. We can construct a set $\mathcal{T} \subseteq \mathbb{R}^{\mathcal{L}}_{\ge 0}$ with $|\mathcal{T}| \le N_1$ in $O(N_1)$ time, containing a non-increasing vector $v \in \mathbb{R}^{\mathcal{L}}_{\ge 0}$, such that $\alpha_\ell \le v_\ell \le (1+\varepsilon)\alpha_\ell + \kappa$ for all $\ell \in \mathcal{L}$.*

(b) *Suppose that we also have $\alpha_\ell \ge \mathsf{lb}$ for all $\ell \in \mathcal{L}$, where $\mathsf{lb} > 0$. Let $N_2 := (2e)^{|\mathcal{L}|} + \left(\frac{\mathsf{ub}}{\mathsf{lb}}\right)^{O(\frac{1}{\varepsilon'})}$. We can construct $\mathcal{T} \subseteq \mathbb{R}^{\mathcal{L}}_{\ge 0}$ with $|\mathcal{T}| \le N_2$ in $O(N_2)$ time, containing a non-increasing vector $v \in \mathbb{R}^{\mathcal{L}}_{\ge 0}$, such that $\alpha_\ell \le v_\ell \le (1+\varepsilon)\alpha_\ell$ for all $\ell \in \mathcal{L}$.*

Proof. We utilize the following standard result, lifted from [4], that gives a bound on the number of non-increasing vectors with bounded coordinates.

▷ Claim 2.10. There are at most $(2e)^{\max\{M,k\}}$ non-increasing sequences of k integers chosen from $\{0, \ldots, M\}$.

For part (a), consider the set

$$\mathcal{T} := \Big\{\vec{t} \in \mathbb{R}^{\mathcal{L}}_{\ge 0} : \ \vec{t} \text{ is a non-increasing vector,}$$
$$\forall \ell \in \mathcal{L}, \quad t_\ell = \tfrac{\mathsf{ub}}{(1+\varepsilon)^k}, \text{ where } k \in \mathbb{Z}_{\ge 0}, \quad t_\ell \ge \tfrac{\kappa}{1+\varepsilon}\Big\}.$$

Each $\vec{t} \in \mathcal{T}$ is a non-increasing vector, and there are $I := 1 + \left\lfloor \log_{1+\varepsilon} \frac{(1+\varepsilon)\mathsf{ub}}{\kappa} \right\rfloor = O\big(\frac{\log(\mathsf{ub}/\kappa)}{\varepsilon'}\big)$ choices for $\log_{1+\varepsilon} \frac{\mathsf{ub}}{t_\ell}$ for every $\ell \in \mathsf{POS}$. (Recall that $\varepsilon' = \min\{\varepsilon, 1\}$.) So Claim 2.10 implies that $|\mathcal{T}| \le (2e)^{\max\{|\mathcal{L}|, I\}}$. We have $(2e)^I \le \left(\frac{\mathsf{ub}}{\kappa}\right)^{O(\frac{1}{\varepsilon'})}$, so this yields the bound on $|\mathcal{T}|$ and the time to construct $\mathcal{T}$.

Consider the vector $v \in \mathbb{R}^{\mathcal{L}}_{\ge 0}$, where for every $\ell \in \mathcal{L}$, v_ℓ is the smallest number of the form $\mathsf{ub}/(1+\varepsilon)^k$, $k \in \mathbb{Z}_{\ge 0}$ that is at least $\max\{\kappa/(1+\varepsilon), \alpha_\ell\}$. Then, v is a non-increasing vector, $v \in \mathcal{T}$, and $\alpha_\ell \le v_\ell \le (1+\varepsilon)\alpha_\ell + \kappa$ for all $\ell \in \mathcal{L}$.

Part (b) is proved very similarly. We now take $\mathcal{T}$ to be the set of all non-increasing vectors $\vec{t} \in \mathbb{R}^{\mathcal{L}}_{\ge 0}$ satisfying $t_\ell \ge \mathsf{lb}$, $t_\ell = \frac{\mathsf{ub}}{(1+\varepsilon)^k}$ where $k \in \mathbb{Z}_{\ge 0}$, for all $\ell \in \mathcal{L}$. As before, one can infer that the size of $\mathcal{T}$ and the time taken to construct it are bounded by $(2e)^{|\mathcal{L}|} + \left(\frac{\mathsf{ub}}{\mathsf{lb}}\right)^{O(\frac{1}{\varepsilon'})}$. Now if $v \in \mathbb{R}^{\mathcal{L}}_{\ge 0}$ is such that, for every $\ell \in \mathcal{L}$, v_ℓ is the smallest number of the form $\mathsf{ub}/(1+\varepsilon)^k$, $k \in \mathbb{Z}_{\ge 0}$ that is at least α_ℓ, then, v is a non-increasing vector, $v \in \mathcal{T}$, and $\alpha_\ell \le v_\ell \le (1+\varepsilon)\alpha_\ell$ for all $\ell \in \mathcal{L}$. ◀

Poisson random variables. A discrete random variable Z is said to have a Poisson distribution with parameter $\lambda \ge 0$, denoted $Z \sim \mathsf{Pois}(\lambda)$, if $\mathbf{Pr}\big[Z = k\big] = e^{-\lambda}\lambda^k/k!$ for all $k \in \mathbb{Z}_{\ge 0}$.

▶ **Fact 2.11.** *The following facts about Poisson random variables are well known.*

(a) *The mean and variance of $\mathsf{Pois}(\lambda)$ are both equal to λ.*

(b) *Let $\{Z_j\}_j$ be a collection of independent Poisson variables with parameters $\{\lambda_j\}_j$. Then, $S = \sum_j Z_j$ is distributed as $\mathsf{Pois}(\sum_j \lambda_j)$.*

3 Stochastic Minimum Norm Load Balancing with Poisson Jobs

We now consider StochNormLB with Poisson job sizes, denoted PoisNormLB, wherein the processing time of a job j on machine i is a $\mathsf{Pois}(\lambda_{ij})$ random variable and we seek to minimize the expected f-norm of the load vector. Jobs are independent, but processing times of the same job could be correlated across machines. Our main result is a novel, clean, and versatile *black-box reduction* from PoisNormLB to the deterministic problem, MinNormLB, showing that, for any machine environment (i.e., unrelated/identical machines), guarantees obtained for MinNormLB translate to yield essentially the same guarantees for PoisNormLB.

▶ **Theorem 3.1.** *Let $\mathcal{I}^{\mathsf{Pois}} = (J, [m], \{\lambda_{ij}\}, f)$ be an instance of* PoisNormLB*, and $\mathcal{A}^{\mathsf{Det}}$ be an ρ-approximation algorithm for* MinNormLB*-instances with job-set J, machine-set $[m]$, and $\{\lambda_{ij}\}_{i\in[m],j\in J}$ job sizes. For any $\varepsilon, \eta > 0$, we can utilize $\mathcal{A}^{\mathsf{Det}}$ to obtain an $\rho\big(1+O(\varepsilon)\big)$-approximate solution to* PoisNormLB *with probability at least $1-\eta$, in time $\mathrm{poly}\big(m^{1/\varepsilon}, n, \frac{1}{\eta}\big)$. The run time also bounds the number of calls to $\mathcal{A}^{\mathsf{Det}}$ and the sample size.*

We emphasize that: (a) the above reduction preserves the machine environment: for instance, if we have identical machines ($\lambda_{ij} = \lambda_j$ for all i, j), we only need $\mathcal{A}^{\mathsf{Det}}$ to work for identical machines; and (b) algorithm $\mathcal{A}^{\mathsf{Det}}$ is required to work for an arbitrary monotone, symmetric norm (and not just the norm f): this generality is *crucial* for the above reduction and brings to the fore a prime benefit of working at the level of generality of monotone, symmetric norms. Combining the above reduction with our results for MinNormLB in Sections 4 and 5 *immediately* yields the following results as corollaries. (We do not explicitly indicate the failure probability η below; the sample size, for a fixed ε, is $\mathrm{poly}(m)/\eta$.)

▶ **Theorem 3.2.**

(a) **(Follows from Theorems 3.1 and 4.1)** *For any $\varepsilon > 0$, there is a randomized $\big(2+O(\varepsilon)\big)$-approximation algorithm for* PoisNormLB *on unrelated machines .*

(b) **(Follows from Theorems 3.1 and 5.1)** *There is a randomized PTAS for* PoisNormLB *on identical machines.*

We discuss the chief ideas behind the reduction in Theorem 3.1, deferring some details to the full version of the paper. Since the sum of independent Poisson random variables is another Poisson random variable (Fact 2.11 (b)), the objective value of an assignment $\sigma : J \to [m]$ depends only the aggregate λ-vector $\Lambda^\sigma = (\Lambda^\sigma_i)_{i\in[m]}$, where $\Lambda^\sigma_i := \sum_{j:\sigma(j)=i} \lambda_{ij}$ for all $i \in [m]$. We drop σ in Λ^σ if the assignment σ is clear from the context. Overloading notation, for a vector $y \in \mathbb{R}^m_{\geq 0}$, we use $\mathsf{Pois}(y)$ to denote the random vector $\big(\mathsf{Pois}(y_1), \ldots, \mathsf{Pois}(y_m)\big)$ of *independent* Poisson random variables. Defining $g(y) := \mathbf{E}\big[f(\mathsf{Pois}(y))\big]$ for $y \in \mathbb{R}^m_{\geq 0}$, the goal in PoisNormLB is to find an assignment σ that minimizes $g(\Lambda^\sigma)$. The function g is *not* convex, but it satisfies the following inequality [21] (see Chapter 11, Proposition E.6), which is closely related to a property called *Schur convexity* that is satisfied by all symmetric convex functions. Theorem 3.3 provides a means for controlling $g(y)$, by bounding the Top_ℓ-norms of y, and is key to our approach. We give a self-contained proof of Theorem 3.3 in the full version.

▶ **Theorem 3.3.** *Let $y, y' \in \mathbb{R}^m_{\geq 0}$. If $\mathsf{Top}_i(y) \leq \mathsf{Top}_i(y')$ for all $i \in [m]$, then $g(y) \leq g(y')$.*

To keep notation simple, we assume that f is normalized so that $f(1, 0, \ldots, 0) = 1$; clearly, this is without loss of generality. Let σ^* be an optimal solution to the PoisNormLB-instance $\mathcal{I}^{\mathsf{Pois}}$. Let $\Lambda^* := \Lambda^{\sigma^*}$. The idea underlying our reduction is strikingly simple. Given Theorem 3.3, we aim to (ideally) find an assignment σ such that $\mathsf{Top}_i(\Lambda^\sigma) \leq \mathsf{Top}_i(\Lambda^*)$ for all $i \in [m]$. One of our chief insights is that *this amounts to solving a deterministic*

min-norm load balancing problem with job sizes $\{\lambda_{ij}\}_{i,j}$, and the monotone, symmetric norm $h : \mathbb{R}^m \to \mathbb{R}_{\geq 0}$ given by $h(v) := \max_{i\in[m]} \mathsf{Top}_i(v)/\mathsf{Top}_i(\Lambda^*)$. Now σ^* yields a solution to this MinNormLB-instance of cost 1, and so solving this MinNormLB problem optimally, and utilizing Theorem 3.3, would yield the desired solution.

Further ingredients are needed to make this idea work. We do not know the $\mathsf{Top}_i(\Lambda^*)$ values, and cannot "guess" these values for all $i \in [m]$; moreover, we cannot solve the MinNormLB problem optimally. We utilize the sparsification tools from Section 2, and, with a small loss in approximation, move to the sparse set $\mathsf{POS} = \mathsf{POS}_{m,\delta}$ (for, say, $\delta = \min\{0.5, \varepsilon\}$) and work with estimates B_ℓ of $\mathsf{Top}_\ell(\Lambda^*)$ for all $\ell \in \mathsf{POS}$; so the norm in the MinNormLB instance is now $h(v) := \max_{\ell\in\mathsf{POS}} \mathsf{Top}_\ell(v)/B_\ell$. Now, using the algorithm $\mathcal{A}^{\mathsf{Det}}$ with the correct estimate-vector $B^* \in \mathbb{R}^{\mathsf{POS}}_{\geq 0}$ (where each B^*_ℓ overestimates $\mathsf{Top}_\ell(\Lambda^*)$ within a $(1+\delta)$-factor), we obtain an assignment σ such that $\mathsf{Top}_i(\Lambda^\sigma) \leq \alpha' \mathsf{Top}_i(\Lambda^*)$ for all $i \in [m]$, where $\alpha' = \alpha\big(1 + O(\varepsilon)\big)$. Theorem 3.3 then shows that $g(\Lambda^\sigma) \leq g(\alpha'\Lambda^*)$, but we need a bound in terms of $g(\Lambda^*)$. To this end, we prove the important property that g is *subhomogeneous* (Lemma 3.5): $g(\theta y) \leq \theta \cdot g(y)$ for any $\theta \geq 1$. Finally, we cannot quite identify the correct B^*, but we can isolate it in a polynomial-size set. We show how to estimate $g(y)$ using polynomially many samples (Lemma 3.6), and utilize this estimator to find (loosely speaking) the best solution among those computed for each candidate estimate-vector in this set. Combining these various ingredients yields Theorem 3.1.

▶ **Lemma 3.4.** *Let $y \in \mathbb{R}^m_{\geq 0}$. We have $\max\{f(y), 1 - e^{-\mathsf{Top}_m(y)}\} \leq g(y) \leq \mathsf{Top}_m(y)$.*

Proof. To prove the upper bound on g we use Claim 2.1 (recall that f is normalized) and Fact 2.11 (a): $\mathbf{E}\big[f(\mathsf{Pois}(y))\big] \leq \mathbf{E}\big[\mathsf{Top}_m(\mathsf{Pois}(y))\big] = \mathbf{E}\big[\sum_{i\in[m]} \mathsf{Pois}(y_i)\big] = \mathsf{Top}_m(y)$. The first lower bound follows from convexity of norms: $\mathbf{E}\big[f(\mathsf{Pois}(y))\big] \geq f(\mathbf{E}\big[\mathsf{Pois}(y)\big]) = f(y)$. Lastly, for the second lower bound, we use Claim 2.1:

$$\begin{aligned}\mathbf{E}\big[f(\mathsf{Pois}(y))\big] \geq \mathbf{E}\big[\mathsf{Top}_1(\mathsf{Pois}(y))\big] &\geq \mathbf{Pr}\big[\mathsf{Top}_1(\mathsf{Pois}(y)) > 0\big] \\ &= 1 - \prod_{i\in[m]} \mathbf{Pr}\big[\mathsf{Pois}(y_i) = 0\big] = 1 - e^{-\mathsf{Top}_m(y)}.\end{aligned}$$ ◀

▶ **Lemma 3.5** (Subhomogeneity). *For any $y \in \mathbb{R}^m_{\geq 0}$ and scalar $\theta \geq 1$, we have $g(\theta y) \leq \theta \cdot g(y)$.*

Proof. We prove this for rational θ. The proof for general θ then follows from a continuity argument, which we defer to the full version. Let $\theta = a/b$ for integers $a > b \geq 1$. (If $a = b$, there is nothing to be shown.) Observe that $g(\theta y) = g(az)$ and $g(y) = g(bz)$, where $z = y/b$. So, for the rational case, it suffices to prove that $g(az) \leq \frac{a}{b} g(bz)$ holds for all $z \in \mathbb{R}^m_{\geq 0}$ and integers $a > b \geq 1$. Fix some $z \in \mathbb{R}^m_{\geq 0}$. Let $Z^{(0)}, Z^{(1)}, \ldots, Z^{(a-1)}$ be a independent random vectors that are identically distributed copies of $\mathsf{Pois}(z)$ (so each $Z_i^{(j)}$ is an independent $\mathsf{Pois}(z_i)$ random variable). For any $i \in [m]$, $\mathsf{Pois}(az_i)$ is identically distributed as $\sum_{j=0}^{a-1} Z_i^{(j)}$ (Fact b), so $g(az) = \mathbf{E}\big[f\big(\mathsf{Pois}(az)\big)\big] = \mathbf{E}\big[f\big(\sum_{j=0}^{a-1} Z^{(j)}\big)\big]$. Also, for any subset $S \subseteq \{0, 1, \ldots, a\}$ with $|S| = b$, we have $\mathbf{E}\big[f\big(\sum_{j\in S} Z^{(j)}\big)\big] = g(bz)$. Define size-$b$ index sets $S_k := \{(k+j) \bmod a : j = 0, \ldots, b-1\}$, for $k = 0, 1, \ldots, a-1$. Note that each $j \in \{0, \ldots, a-1\}$ is contained in exactly b of these sets.

$$g(az) = \mathbf{E}\big[f\big(\sum_{j=0}^{a-1} Z^{(j)}\big)\big] = \mathbf{E}\big[f\big(\tfrac{1}{b} \cdot \sum_{k=0}^{a-1} \sum_{j\in S_k} Z^{(j)}\big)\big] \leq \frac{1}{b} \cdot \sum_{k=0}^{a-1} \mathbf{E}\big[f\big(\sum_{j\in S_k} Z^{(j)}\big)\big] = \frac{a}{b} \cdot g(bz).$$ ◀

▶ **Lemma 3.6.** *Let $\varepsilon, \eta > 0$. Let $y \in \mathbb{R}^m_{\geq 0}$ be such that $\mathsf{Top}_m(y) \geq \varepsilon$. Let $N := 2\max\{m^2, 4/\varepsilon\}/(\varepsilon^2\eta)$. Using at most N independent samples from $\mathsf{Pois}(y)$, we can compute an estimate γ satisfying $\mathbf{Pr}\big[\gamma \in [(1-\varepsilon)g(y), (1+\varepsilon)g(y)]\big] \geq 1 - \eta$.*

Proof. Let $x^{(1)}, \dots, x^{(N)}$ be N independent samples from $\mathsf{Pois}(y)$, and let $\gamma := \frac{1}{N}\sum_{j=1}^{N} f(x^{(j)})$ denote the sample f-average. We show that γ is the desired estimator by using Chebyshev's concentration inequality. To this end, we need a bound on the variance of the real-valued random variable $f(\mathsf{Pois}(y))$. We have $\mathrm{Var}\big[f(\mathsf{Pois}(y))\big] \le \mathbf{E}\big[f^2(\mathsf{Pois}(y))\big] \le \mathbf{E}\big[\mathsf{Top}_m^2(\mathsf{Pois}(y))\big]$ by Claim 2.1, and $\mathbf{E}\big[\mathsf{Top}_m^2(\mathsf{Pois}(y))\big] = \mathrm{Var}\big[\mathsf{Pois}(\mathsf{Top}_m(y))\big] + \big(\mathbf{E}\big[\mathsf{Pois}(\mathsf{Top}_m(y))\big]\big)^2$. By Fact b, $\mathsf{Top}_m(\mathsf{Pois}(y))$ is distributed as $\mathsf{Pois}(\mathsf{Top}_m(y))$. As the variance of a Poisson variable with mean λ is λ, we obtain that

$$\mathrm{Var}\big[f(\mathsf{Pois}(y))\big] \le \mathrm{Var}\big[\mathsf{Pois}(\mathsf{Top}_m(y))\big] + \big(\mathbf{E}\big[\mathsf{Pois}(\mathsf{Top}_m(y))\big]\big)^2 \le 2\mathsf{Top}_m(y)\max(1, \mathsf{Top}_m(y)).$$

Since γ is an average of $f(\mathsf{Pois}(y))$ over N independent samples, we have $\mathrm{Var}\big[\gamma\big] = \mathrm{Var}\big[f(\mathsf{Pois}(y))\big]/N$. By Chebyshev's inequality,

$$\mathbf{Pr}\big[|\gamma - g(y)| > \varepsilon g(y)\big] \le \frac{\mathrm{Var}\big[\gamma\big]}{\varepsilon^2 g^2(y)} \le \frac{2}{N\varepsilon^2}\cdot\frac{\mathsf{Top}_m(y)\max(1,\mathsf{Top}_m(y))}{g^2(y)} \le \frac{2\max\{m^2, 4/\varepsilon\}}{N\epsilon^2} \le \eta$$

It remains to justify the penultimate inequality used above. First, observe that $\mathsf{Top}_m(y)/g(y) \le m$ for any $y \in \mathbb{R}^m_{\ge 0}$, so the inequality holds when $\mathsf{Top}_m(y) \ge 1$. Suppose $\mathsf{Top}_m(y) \in [\varepsilon, 1]$. We have $g(y) \ge 1 - e^{-\mathsf{Top}_m(y)}$ (by Lemma 3.4), which is at least $\mathsf{Top}_m(y)(1 - \mathsf{Top}_m(y)/2) \ge \mathsf{Top}_m(y)/2$. This finishes the proof of the lemma. ◀

Proof of Theorem 3.1. Set $\epsilon' = \delta = \min\{0.5, \varepsilon\}$. Let σ^{sum} be the assignment that minimizes the expected sum of machine loads, so $\sigma^{\mathsf{sum}}(j) = \mathrm{argmin}_i \lambda_{ij}$ for all jobs j. Let $\Lambda^{\mathsf{sum}} := \Lambda^{\sigma^{\mathsf{sum}}}$, and $\mathsf{UB} := \mathsf{Top}_m(\Lambda^{\mathsf{sum}})$. If $\mathsf{UB} \le \epsilon'$, then we claim that σ^{sum} is a $(1+\epsilon')$-approximate solution to PoisNormLB. This is because we have $\mathsf{Top}_m(\Lambda^*) \ge \mathsf{UB}$, and so by Lemma 3.4, we have that $g(\Lambda^{\mathsf{sum}}) \le \mathsf{UB}$ and $g(\Lambda^*) \ge 1 - e^{-\mathsf{UB}} \ge \mathsf{UB}(1 - \mathsf{UB}/2) \ge \mathsf{UB}/(1+\epsilon')$, where we use that $\mathsf{UB} \le \epsilon' \le 1$.

So suppose that $\mathsf{UB} \ge \epsilon'$ (and so $\mathsf{Top}_m(\Lambda^\sigma) \ge \epsilon'$ for every assignment σ). We have $\mathsf{Top}_1(\Lambda^*) \ge \mathsf{Top}_m(\Lambda^*)/m \ge \mathsf{UB}/m$. Also, $\mathsf{Top}_m(\Lambda^*) \le m\mathsf{Top}_1(\Lambda^*)$, and we have $\mathsf{Top}_1(\Lambda^*) \le f(\Lambda^*) \le g(\Lambda^*) \le g(\Lambda^{\mathsf{sum}}) \le \mathsf{UB}$; here, we have used Claim 2.1, Lemma 3.4 (twice), and the optimality of σ^*. So we obtain that $\mathsf{Top}_m(\Lambda^*) \le m \cdot \mathsf{UB}$.

Now consider the non-increasing vector u which is $(\Lambda^*_\ell)_{\ell \in \mathsf{POS}}$ with its coordinates listed in *decreasing* order of ℓ. We apply Lemma 2.9 (b) on u, taking $\mathcal{L} = \mathsf{POS}$, and upper and lower bounds $m \cdot \mathsf{UB}$ and UB/m respectively, to obtain a $\mathrm{poly}(m^{1/\delta})$-size set $\mathcal{T} \subseteq \mathbb{R}^{\mathsf{POS}}_{\ge 0}$ containing a vector B^* such that $\mathsf{Top}_\ell(\Lambda^*) \le B^*_\ell \le (1+\delta)\mathsf{Top}_\ell(\Lambda^*)$ for all $\ell \in \mathsf{POS}$.

For each $B \in \mathcal{T}$, we do the following. Let $h_B : \mathbb{R}^m \to \mathbb{R}_{\ge 0}$ be the monotone, symmetric norm defined as $h_B(v) := \max_{\ell \in \mathsf{POS}} \mathsf{Top}_\ell(v)/B_\ell$. We run $\mathcal{A}^{\mathsf{Det}}$ on the MinNormLB instance with $\{\lambda_{ij}\}_{i\in[m], j\in J}$ job sizes and norm h_B, to obtain an assignment σ^B. Let $\Lambda^B = \Lambda^{\sigma^B}$. We use Lemma 3.6 to compute an estimate γ^B such that $\mathbf{Pr}\big[\gamma^B \in [(1-\epsilon')g(\Lambda^B), (1+\epsilon')g(\Lambda^B)]\big] \ge 1 - \frac{\eta}{|\mathcal{T}|}$. We output the assignment $\sigma^{B'}$ with smallest $\gamma^{B'}$ value among all $B' \in \mathcal{T}$.

We argue that this is an $\rho\big(1+O(\varepsilon)\big)$-approximation with probability at least $1-\eta$. By the union bound, with probability at least $1-\eta$, we have that $\gamma^B \in [(1-\epsilon')g(\Lambda^B), (1+\epsilon')g(\Lambda^B)]$ for all $B \in \mathcal{T}$; we assume that this event happens. Consider the correct guess $B^* \in \mathcal{T}$. We have $h_{B^*}(\Lambda^*) \le 1$, and so the solution σ^{B^*} satisfies $\mathsf{Top}_\ell(\Lambda^{B^*}) \le \rho B^*_\ell \le \rho(1+\delta)\mathsf{Top}_\ell(\Lambda^*)$ for all $\ell \in \mathsf{POS}$. Using Claim 2.6, we have $\mathsf{Top}_i(\Lambda^{B^*}) \le \rho(1+\delta)^2\mathsf{Top}_i(\Lambda^*)$ for all $i \in [m]$. By Theorem 3.3 and Lemma 3.5, we then obtain that $g(\Lambda^{B^*}) \le \rho(1+\delta)^2 g(\Lambda^*)$. Accounting for the error due to the γ^B estimates, yields $g(\Lambda^{B'}) \le \frac{1+\epsilon'}{1-\epsilon'} \cdot g(\Lambda^{B^*}) \le \rho(1+15\epsilon')g(\Lambda^*)$. ◀

4 A $(2+\epsilon)$-approximation for MinNormLB on unrelated machines

▶ **Theorem 4.1.** *For any $\varepsilon > 0$, there is a $\big(2+O(\varepsilon)\big)$-approximation algorithm for* MinNormLB *on unrelated machines.*

Our algorithm yielding the above theorem is based on LP-rounding. Our relaxation is different from the LP used in [4] and the convex program used in [5] for MinNormLB. The latter relaxation, which was used to obtain the previous best $(4+\varepsilon)$-approximation has an integrality gap of (roughly) 4 [3], and therefore we need new ideas to obtain our $(2+\varepsilon)$-approximation algorithm. Indeed, our LP and rounding algorithm leverage and build upon the ideas used in the above works in a novel fashion.

Let $n = |J|$ be the number of jobs. Let σ^* denote an optimal solution, $\vec{o} := \big(\overrightarrow{\mathsf{load}}_{\sigma^*}\big)$ be the load vector induced by σ^*, and let $\mathsf{OPT} = f(\vec{o})$ denote the optimal value. Define the cost of a job j under σ^* to be $p_{\sigma^*(j)j}$. For any set $S \subseteq J$ of jobs, define the job-cost vector $\overrightarrow{P^*_S} = \overrightarrow{P^{\sigma^*}_S} := \big(p_{\sigma^*(j)j}\big)_{j\in S}$. Let $\varepsilon > 0$ be a parameter, and let $\delta = \min\{\varepsilon, 1\}$. Recall that we abbreviate $\mathsf{POS}_{m,\delta}$ to POS. We may assume that $n \geq \lceil 2(1+\delta)/\delta^3 \rceil$ as otherwise we can simply exhaustively search for the optimal assignment.

LP relaxation. As is standard, our LP has variables x_{ij} for every machine i and job j denoting if job j is assigned to machine i (or, fractionally, the extent of j assigned to machine i). Constraints (2) enforce that every job is assigned to a machine.

We are guided by Theorem 2.4, which shows that to obtain a solution whose load vector has f-norm equal to $O(\mathsf{OPT})$, it suffices to show that the Top_ℓ-norm of the load vector is $O\big(\mathsf{Top}_\ell(\vec{o})\big)$ for all $\ell \in [m]$; also, by Claim 2.6, one can focus on indices $\ell \in \mathsf{POS}$. We work with "guesses" (i.e., estimates) t_ℓ of $\vec{o}^{\downarrow}_\ell$, for all $\ell \in \mathsf{POS}$. Our LP seeks a fractional solution such that the resulting load vector has Top_ℓ-norm at most $\mathsf{Top}_\ell(\vec{o})$ for all $\ell \in \mathsf{POS}$. Using Claim 2.3 and our estimates, we encode this via the constraint $\ell t_\ell + \sum_i \big(\sum_j p_{ij}x_{ij} - t_\ell\big)^+ \leq \mathsf{Top}_\ell(\overrightarrow{t^{\mathsf{exp}}})$ for all $\ell \in \mathsf{POS}$; constraints (3), (4) linearize these. (Recall that $\overrightarrow{t^{\mathsf{exp}}} \in \mathbb{R}^m_{\geq 0}$ is defined by $t^{\mathsf{exp}}_i = t_i$ for $i \in \mathsf{POS}$, and $t^{\mathsf{exp}}_i = t_{\mathsf{prev}(i)}$ for $i \in [m] \setminus \mathsf{POS}$.)

The last set of constraints of our LP is motivated by an insight in [5]. They show that if $\overrightarrow{\mathsf{JL}^*} \in \mathbb{R}^m_{\geq 0}$ is the vector comprising the costs of the m most costly jobs under σ^*, then we have $f(\overrightarrow{\mathsf{JL}^*}) \leq \mathsf{OPT}$, which also implies that $f(\overrightarrow{P^*_S}) \leq \mathsf{OPT}$ for any set S of m jobs. Chakrabarty and Swamy [5] include this (convex) constraint directly in their convex program, and it plays a crucial role in obtaining their $(4+\varepsilon)$-approximation for MinNormLB. We also utilize this constraint, but incorporate it (loosely speaking) in our LP in a more subtle fashion. We work with guesses ζ_ℓ of $\overrightarrow{\mathsf{JL}^*}^{\downarrow}_\ell$ for all $\ell \in \mathsf{POS}$, and encode (see constraints (5)) that there are at most $\ell - 1$ jobs whose cost is larger than ζ_ℓ, for all $\ell \in \mathsf{POS}$. This can be seen as an indirect way of capturing $f(\overrightarrow{\mathsf{JL}^*}) \leq \mathsf{OPT}$, and this indirect way is crucial for us because we show that we can round a fractional solution x with only a $(1+\delta)$-factor violation of these constraints. For technical reasons, to facilitate this, we use another partial enumeration step. Let ℓ_0 be the smallest index in POS that is at least $\frac{2}{\delta^3}$. Note that $\ell_0 \leq \lceil 2(1+\delta)/\delta^3 \rceil \leq n$. We will guess the ℓ_0 most costly jobs under σ^* and their σ^*-assignments. Let $C = \{(i_1, j_1), (i_2, j_2), \ldots, (i_{\ell_0}, j_{\ell_0})\}$ denote our guess of these jobs and their σ^*-assignments. Note that given C, we know that $\overrightarrow{\mathsf{JL}^*}^{\downarrow}_\ell$ is the ℓ-th largest value in $\{p_{ij} : (i,j) \in C\}$ for all $\ell \in [\ell_0]$, and we therefore set $\zeta_\ell = \overrightarrow{\mathsf{JL}^*}^{\downarrow}_\ell$ for all $\ell \in \mathsf{POS}$ with $\ell \leq \ell_0$. An important consequence of this enumeration step, which will be crucial in the analysis (see the proof of Lemma 4.3), is that this *fixes* the assignment of all jobs whose cost under σ^* is larger than ζ_{ℓ_0}; we encode this in our LP via constraints (6).

This yields the following LP, which is a feasibility LP depending on $\vec{t}$, $\vec{\zeta}$, and C. Throughout, we use i to index $[m]$, and j to index J.

$$
\begin{aligned}
& x \geq 0, \quad \sum_i x_{ij} = 1 && \forall j \in J && (2)\\
& \ell t_\ell + \sum_i W_i^\ell \leq \mathsf{Top}_\ell(\overrightarrow{t^{\mathsf{exp}}}) && \forall \ell \in \mathsf{POS} && (3)\\
(\mathsf{LP}(\vec{t},\vec{\zeta},C)) \qquad & W_i^\ell \geq 0, \quad W_i^\ell \geq \sum_j p_{ij}x_{ij} - t_\ell && \forall i \in [m],\ \ell \in \mathsf{POS} && (4)\\
& \sum_{i,j:p_{ij}>\zeta_\ell} x_{ij} \leq \ell - 1 && \forall \ell \in \mathsf{POS} && (5)\\
& x_{ij} = \begin{cases} 1 & \text{if } (i,j) \in C \\ 0 & \text{otherwise.} \end{cases} && \forall i \in [m], j \in J \text{ s.t. } p_{ij} > \zeta_{\ell_0} && (6)
\end{aligned}
$$

▷ Claim 4.2. $(\mathsf{LP}(\vec{t},\vec{\zeta},C))$ is feasible whenever: (i) $t_\ell \geq \vec{o}^{\downarrow}_\ell$, $\zeta_\ell \geq \overrightarrow{\mathsf{JL}^*}^{\downarrow}_\ell$ for all $\ell \in \mathsf{POS}$; and (ii) C is the correct guess of the ℓ_0 most costly jobs under σ^* and their σ^*-assignments.

Rounding algorithm. Assume that $(\mathsf{LP}(\vec{t},\vec{\zeta},C))$ is feasible and $(\overline{x},\overline{W})$ is a feasible solution. We consider the following auxiliary LP (Aux) for rounding $\overline{x}$.

$$x \geq 0, \quad \sum_i x_{ij} = 1 \ \ \forall j \in J, \quad \sum_j p_{ij}x_{ij} \leq \sum_j p_{ij}\overline{x}_{ij} \ \ \forall i \in [m], \quad \sum_{i,j:p_{ij}>\zeta_\ell} x_{ij} \leq \ell - 1 \quad \forall \ell \in \mathsf{POS}.$$

Clearly, $\overline{x}$ is a feasible solution to (Aux). We show that this can be rounded to an integer solution $\widetilde{x}$ so that $\sum_i \widetilde{x}_{ij} = 1$ for every job j, the load on each machine is at most its $\overline{x}$-load $+ (1+\delta)\times$(cost of the most-costly job assigned to it), and the remaining constraints are violated by a small factor. More precisely, in the analysis, we prove the following result.

▶ **Lemma 4.3.** *We can round $\overline{x}$ efficiently to an integer solution $\widetilde{x}$ satisfying the following.*
(a) *$\widetilde{x}_{ij} = \overline{x}_{ij}$ for all i,j such that $\overline{x}_{ij} \in \{0,1\}$;*
(b) *$\sum_i \widetilde{x}_{ij} = 1$ for each job j;*
(c) *$\sum_j p_{ij}\widetilde{x}_{ij} \leq \sum_i p_{ij}\overline{x}_{ij} + (1+\delta)\max_{j:\widetilde{x}_{ij}=1} p_{ij}$ for every machine i;*
(d) *$\sum_{i,j:p_{ij}>\zeta_\ell} \widetilde{x}_{ij} \leq \ell - 1$ for all $\ell \in \mathsf{POS}$ with $\ell \leq \ell_0$; and*
(e) *$\sum_{i,j:p_{ij}>\zeta_\ell} \widetilde{x}_{ij} \leq (1+\delta)\ell - 1$ for all $\ell \in \mathsf{POS}$ with $\ell > \ell_0$.*

We apply the above lemma, and due to Lemma 4.3 (a), the integer solution $\widetilde{x}$ yields an assignment $\widetilde{\sigma}$ of jobs to machines; we return this assignment.

Analysis. We assume again that f is normalized. Lemma 4.3 is the main technical result that we need to prove. Its proof utilizes an *iterative-rounding* result of independent interest (Theorem 4.6) that is very similar to Corollary 11 in [19], wherein iterative rounding is used to round a point that lies in the base polytope of one matroid $\mathcal{M}_0$ and satisfies various other matroid-independence and knapsack constraints, to a basis of $\mathcal{M}_0$ that is "approximately independent" in the other matroids and violates the knapsack constraints by a certain additive factor. In (Aux), the job assignment constraints correspond to the base-polytope constraints (of a partition matroid), and the remaining constraints correspond to knapsack constraints. First, we establish that, assuming Lemma 4.3, we obtain the stated guarantee.

Suppose that we have the correct set C, and that $\vec{t}$ and $\vec{\zeta}$ are "good" estimates of $(\vec{o}^{\downarrow}_{\ell})_{\ell\in\mathsf{POS}}$ and $(\overrightarrow{\mathsf{JL}^*}^{\downarrow}_{\ell})_{\ell\in\mathsf{POS}}$ respectively (we make this precise later). We analyze the load vector $\overrightarrow{\mathsf{load}}_{\widetilde{\sigma}}$ by considering two vectors $L=(L_i)_{i\in[m]}$ and $P=(P_i)_{i\in[m]}$. For each $i\in[m]$, define $L_i=\sum_j p_{ij}\bar{x}_{ij}$, and $P_i=\max_{j:\widetilde{\sigma}(j)=i} p_{ij}$; note that $P_i=0$ if there is no job assigned by $\widetilde{\sigma}$ to machine i. By Lemma 4.3 (c), we have that $\overrightarrow{\mathsf{load}}_{\widetilde{\sigma}}\le L+(1+\delta)P$, and we bound both $f(L)$ and $f(P)$ by roughly OPT. Constraints (3), (4) of our LP immediately allow us to bound $\mathsf{Top}_\ell(L)$ by $\mathsf{Top}_\ell(\overrightarrow{t^{\mathsf{exp}}})$ for all $\ell\in\mathsf{POS}$, which suffices since $\vec{t}$ well estimates $(\vec{o}^{\downarrow}_{\ell})_{\ell\in\mathsf{POS}}$ (see Lemma 4.4). For the vector P, parts (d) and (e) of Lemma 4.3 yield the desired bound on $f(P)$, due to Lemma 2.8 (c) (see Lemma 4.5). Before proving Lemmas 4.4 and 4.5, we justify, and make precise, the assumption that we have the correct set C (specifying the ℓ_0 most costly jobs under σ^* and their σ^*-assignments), and good estimates $\vec{t}$, $\overrightarrow{\mathsf{JL}^*}$. There are at most $\binom{n}{\ell_0}\cdot m^{\ell_0}$ choices for the set C. As noted earlier, for all $\ell\in\mathsf{POS}$ with $\ell\le\ell_0$, this then fixes $\zeta_\ell=\zeta^*_\ell=\overrightarrow{\mathsf{JL}^*}^{\downarrow}_{\ell}$ to be the ℓ-th largest entry in $\{p_{ij}:(i,j)\in C\}$. Using Claim 2.1, if we consider the assignment σ^{sum} that minimizes the sum of the machine loads – so $\sigma^{\mathsf{sum}}(j)=\mathrm{argmin}_i p_{ij}$ for all jobs j – and set $\mathsf{UB}:=\sum_i \mathsf{load}_{\sigma^{\mathsf{sum}}}(i)$, then we obtain that $\frac{\mathsf{UB}}{m}\le\mathsf{OPT}\le\mathsf{UB}$, and so $\vec{o}_1,\overrightarrow{\mathsf{JL}^*}_1\le\mathsf{UB}$. Set $\kappa=\varepsilon\mathsf{UB}/m^2$. Let $\mathsf{POS}_{>}:=\{\ell\in\mathsf{POS}:\ell>\ell_0\}$. So using Lemma 2.9 (a) (and since $|\mathsf{POS}|=O(\log m/\delta)$), we can identify polynomial-size sets containing vectors $\vec{t}=\vec{t^*}$ and $(\zeta_\ell)_{\ell\in\mathsf{POS}_{>}}=(\zeta^*_\ell)_{\ell\in\mathsf{POS}_{>}}$ such that:

(1) $\vec{o}^{\downarrow}_{\ell}\le t^*_\ell\le(1+\varepsilon)\vec{o}^{\downarrow}_{\ell}+\kappa$ for all $\ell\in\mathsf{POS}$, and $\vec{t^*}$ is a non-increasing vector; and

(2) $\overrightarrow{\mathsf{JL}^*}^{\downarrow}_{\ell}\le\zeta^*_\ell\le(1+\varepsilon)\overrightarrow{\mathsf{JL}^*}^{\downarrow}_{\ell}+\kappa$ for all $\ell\in\mathsf{POS}_{>}$, and $\zeta^*=(\zeta^*_\ell)_{\ell\in\mathsf{POS}}$ is non-increasing.

For (1), we take $\mathcal{L}=\mathsf{POS}$, $\mathsf{ub}=\mathsf{UB}$, and κ as above; for (2), we take $\mathcal{L}=\mathsf{POS}_{>}$, $\mathsf{ub}=\overrightarrow{\mathsf{JL}^*}_{\ell_0}$, and κ as above.) Recall that $\delta=\min\{\varepsilon,1\}$.

▶ **Lemma 4.4.** *We have* $f(L)\le(1+\delta)(1+\varepsilon)\mathsf{OPT}+m\kappa\le(1+\delta+3\varepsilon)\mathsf{OPT}$.

Proof. We show that $\mathsf{Top}_i(L)\le\mathsf{Top}_i(\overrightarrow{t^{*\mathsf{exp}}})$ for *all* $i\in[m]$. For any $\ell\in\mathsf{POS}$, this follows quite directly from constraints (3), (4) of our LP $(\mathsf{LP}(\vec{t},\vec{\zeta},C))$: we have $\mathsf{Top}_\ell(L)\le\ell t^*_\ell+\sum_{i'\in[m]}(L_{i'}-t^*_\ell)^+\le\mathsf{Top}_\ell(\overrightarrow{t^{*\mathsf{exp}}})$. For any $i\in[m]\setminus\mathsf{POS}$, taking $\ell=\mathsf{prev}(i)$, this then implies that

$$\mathsf{Top}_i(L)\le it^*_\ell+\sum_{i'\in[m]}(L_{i'}-t^*_\ell)^+\le\mathsf{Top}_\ell(\overrightarrow{t^{*\mathsf{exp}}})+(i-\ell)t^*_\ell=\mathsf{Top}_i(\overrightarrow{t^{*\mathsf{exp}}}),$$

where the final equality follows because $t^{*\mathsf{exp}}_{i'}=t^*_\ell$ for all $i'\in\{\ell,\ell+1,\dots,\mathsf{next}(\ell)-1\}$. Hence, by Theorem 2.4, we have that $f(L)\le f(\overrightarrow{t^{*\mathsf{exp}}})$.

Since $t^*_\ell\le(1+\varepsilon)\vec{o}^{\downarrow}_{\ell}+\kappa$ for all $\ell\in\mathsf{POS}$, by Lemma 2.8 (b), taking $u=\vec{o}$ and $v=\vec{t^*}$, we obtain that $f(\overrightarrow{t^{*\mathsf{exp}}})\le(1+\delta)(1+\varepsilon)f(\vec{o})+m\kappa$. The final inequality in the lemma statement follows because $m\kappa\le\varepsilon\mathsf{OPT}$ and $(1+\delta)(1+\varepsilon)\le(1+\delta+2\varepsilon)$ as $\delta\le1$. ◀

▶ **Lemma 4.5.** *We have* $f(P)\le(1+4\delta+5\varepsilon)f(\overrightarrow{\mathsf{JL}^*})+5m\kappa\le(1+4\delta+10\varepsilon)\mathsf{OPT}$.

Proof. We apply Lemma 2.8 (c), taking $u=\overrightarrow{\mathsf{JL}^*}$, $\alpha=P$, and $v=(\zeta^*_\ell)_{\ell\in\mathsf{POS}}$. The conditions in Lemma 2.8 (c) hold due to parts (d) and (e) of Lemma 4.3. This yields that $f(P)\le(1+4\delta)(1+\varepsilon)f(\overrightarrow{\mathsf{JL}^*})+5m\kappa$. The lemma follows since $\delta\le1$, $m\kappa\le\varepsilon\mathsf{OPT}$, and $f(\overrightarrow{\mathsf{JL}^*})\le\mathsf{OPT}$. ◀

Proof of Theorem 4.1. Recall that $\widetilde{\sigma}$ is the assignment returned by the algorithm. Lemmas 4.4 and 4.5 together with the fact that $\overrightarrow{\mathsf{load}}_{\widetilde{\sigma}}\le L+(1+\delta)P$ immediately yield that

$$f(\overrightarrow{\mathsf{load}}_{\widetilde{\sigma}})\le(1+\delta+3\varepsilon)\mathsf{OPT}+(1+\delta)(1+4\delta+10\varepsilon)\mathsf{OPT}\le(2+10\delta+23\varepsilon)\mathsf{OPT}.$$ ◀

Proof of Lemma 4.3

We reproduce the system (Aux) below for easy reference.

$$\text{(Aux)}\quad \begin{aligned} x \geq 0, \quad \sum_i x_{ij} &= 1 && \forall j \in J && (7)\\ \sum_j p_{ij} x_{ij} &\leq \sum_j p_{ij} \overline{x}_{ij} && \forall i \in [m] && (8)\\ \sum_{i,j: p_{ij} > \zeta_\ell} x_{ij} &\leq \ell - 1 && \forall \ell \in \mathsf{POS} && (9) \end{aligned}$$

We utilize the following iterative-rounding result that is quite similar to Corollary 11 in [19].

▶ **Theorem 4.6** (Similar to Corollary 11 in [19]). *Let $\mathcal{M}_s = (N_s, \mathcal{I}_s)$, for $s = 0, \ldots, k$ be $(k+1)$ matroids, where each N_s is a subset of $N := N_0$. Let r_s be the rank function of matroid $\mathcal{M}_s$, for $s = 0, \ldots, k$. Let $w \in \mathbb{R}^N$, $A \in \mathbb{R}^{d \times N}_{\geq 0}$, $b \in \mathbb{R}^d_{\geq 0}$. Consider the following LP.*

$$\begin{aligned} \max \quad & w^T x && && (\mathrm{LP}_{\mathrm{matkn}})\\ \textit{s.t.} \quad & x(N) = r_0(N), \quad x(T) \leq r_0(T) && \forall T \subseteq N\\ & x(T) \leq r_s(T) && \forall s = 1, \ldots, k, \forall T \subseteq N_i\\ & Ax \leq b, \quad x \geq 0. \end{aligned}$$

Let $q_1, \ldots, q_k \geq 1$ be integers, and $\mu_1, \ldots, \mu_d \in \mathbb{R}_{\geq 0}$ be such that:

$$\sum_{s \in [k]: e \in N_s} \frac{1}{q_s} + \sum_{s \in [d]: A_{se} > 0} \frac{1}{\mu_s} \leq 1 \qquad \forall e \in N. \tag{10}$$

If $(\mathrm{LP}_{\mathrm{matkn}})$ is feasible, then we can efficiently obtain a basis R of $\mathcal{M}_0$ such that:

- **(i)** *$|T| \leq q_s r_s(T)$ for all $s \in [k]$ and $T \subseteq R \cap N_s$;*
- **(ii)** *$\sum_{e \in R} A_{se} \leq b_s + \mu_s \cdot \max_{e \in R} A_{se}$ for all $s \in [d]$; and*
- **(iii)** *$w(R) \geq \mathsf{OPT}_{LP_{\mathrm{matkn}}}$.*

▶ Remark 4.7. The statement of Theorem 4.6 differs from that of Corollary 11 in [19] in three respects. In [19]: (1) $\mu_1, \ldots, \mu_d$ are assumed to be positive integers (but their proof does not need the integrality requirement); (2) condition (10) is replaced by the *weaker* condition, $\sum_{s \in [k]: e \in N_s} \frac{1}{q_s} + \sum_{s \in [d]} \frac{A_{se}}{(\max_{e \in N} A_{se})\mu_s} \leq 1$ for all $e \in N$; and (3) one obtains the slightly *weaker* additive error of $\mu_s \cdot \max_{e \in N} A_{se}$ in (ii) for the knapsack constraint for index $s \in [d]$. The proof of Corollary 11 in [19] is however easily adapted to yield the above statement; we include the proof of Theorem 4.6 in the full version.

For our purposes, we only need a simpler version of Theorem 4.6, where we have only one matroid $\mathcal{M}_0$ and other knapsack constraints. We include the more general statement above (with its subtly stronger guarantee for the knapsack constraints, which turns out to be crucial for us; see Remark 4.8) because we believe that this is of independent interest.

Before delving into the details of how we apply Theorem 4.6 to prove our lemma, we briefly discuss the main idea, which will also elucidate why we need the partial enumeration step of "guessing" the set C specifying the ℓ_0 most costly jobs under σ^* and their σ^*-assignments.

We can cast (Aux) as an instance of $(\mathrm{LP}_{\mathrm{matkn}})$ as follows. Let $U = [m] \times J$. The weight vector w is irrelevant. The matroid $\mathcal{M}_0$ is the partition matroid over U encoding that each job j is assigned to at most one machine. It is immediate that (7) shows that

$\overline{x}$ lies in the base polytope of $\mathcal{M}_0$. Constraints (8) correspond to knapsack constraints in (LP_{matkn}). Constraints (9) can be incorporated into (LP_{matkn}) in three ways: (i) as matroid-independence constraints in the (single) laminar matroid formed by the nested family of sets $\{(i,j) \in U : p_{ij} > \zeta_\ell\}_{\ell \in \mathsf{POS}}$; (ii) as multiple matroid-independence constraints, where for each $\ell \in \mathsf{POS}$, we have a matroid encoding the cardinality bound from $\{(i,j) \in U : p_{ij} > \zeta_\ell\}$; or (iii) as additional knapsack constraints.

But utilizing Theorem 4.6 directly on the above system does not quite work for us. To elaborate, in order to obtain the guarantee in 4.3 (c), we need to set the μ_is for the knapsack constraints (8) to $(1+\delta)$. But then if we view (9) as matroid independence constraints (in one laminar matroid, or multiple matroids), we need to set the q_s value for the corresponding matroid(s) to be at least 2 in order to satisfy (10) resulting in (at least) a multiplicative factor-2 violation of constraints (9), which is too large a violation. Suppose we incorporate (9) as knapsack constraints and, for some constant ρ, take $\mu_\ell = \rho\ell$ for constraint (9) for index ℓ. But to satisfy (10), we need ρ to be at least $\left(1 - \frac{1}{1+\delta}\right)^{-1} \cdot \sum_{\ell' \in \mathsf{POS}:\ell' > 1} \frac{1}{\ell'} = \Omega\left(\frac{1}{\delta^2}\right)$, which again yields too large a violation of (9). (A more accurate estimate of $\sum_{\ell' \in \mathsf{POS}:\ell' > 1}$ is $\ln(\lceil 1/\delta \rceil) + 1 + \delta$, but this still requires ρ to be large.)

However, a tweak of the latter approach does work, by noting that if x_{ij} first appears in constraint (9) for index $\ell \in \mathsf{POS}$, then to satisfy (10), we really need that $\rho \geq \left(1 - \frac{1}{1+\delta}\right)^{-1} \cdot \sum_{\ell' \in \mathsf{POS}:\ell' \geq \ell} \frac{1}{\ell'}$; the RHS is at most $\frac{(1+\delta)^2}{\delta^2 \ell}$, which is at most δ for *large enough* ℓ. *This is precisely where the enumeration of* C *turns out to be key, since it allows us to eliminate constraints* (9) *for* $\ell \leq \ell_0$. Recall that C specifies the ℓ_0 most-costly jobs under σ^* and their σ^*-assignments, where ℓ_0 is the smallest index in POS that is at least $\frac{2}{\delta^3}$. Due to this enumeration, $\overline{x}$ is integral whenever $p_{ij} > \zeta_\ell$, and so if we drop integral variables and corresponding constraints from (Aux), we are left with constraints (9) for indices $\ell > \ell_0$ in POS. Now we can take $\mu_\ell = \delta\ell$ for all $\ell \in \mathsf{POS}$, $\ell > \ell_0$, and these values satisfy (10), since each $\ell > \ell_0$ is large.

We now describe precisely how we utilize Theorem 4.6 to prove our lemma. We apply Theorem 4.6 to (Aux) after getting rid of all integral variables, and modifying or dropping constraints to take into account the integral variables. Recall that $U = [m] \times J$. Let $E = \{(i,j) \in U : \overline{x}_{ij} \in (0,1)\}$. We retain only the variables in E, and only these will be rounded, so this takes care of part (a) of the lemma. We drop any constraint that contains only integral variables. Thus, the POS-constraints (9) for all $\ell \in \mathsf{POS}$, $\ell \leq \ell_0$ are dropped, and so part (d) holds. Let J' be the set of jobs corresponding to the remaining constraints (7); note that all jobs in J' are completely assigned by (the variables in) E, and the remaining jobs are completely assigned by the integral variables. Let $I \subseteq [m]$ index the machine-constraints (8) that remain, and $I^{\mathsf{POS}} \subseteq \{\ell \in \mathsf{POS} : \ell > \ell_0\}$ index the POS-constraints (9) that remain. This yields the following system of constraints.

$$\text{(P')} \quad \begin{cases} \displaystyle\sum_{i:(i,j)\in E} x_{ij} = 1 & \forall j \in J' \\ \displaystyle\sum_{j:(i,j)\in E} p_{ij} x_{ij} \leq \sum_j p_{ij}\overline{x}_{ij} - \sum_{j:\overline{x}_{ij}=1} p_{ij} & \forall i \in I \\ \displaystyle\sum_{(i,j)\in E:p_{ij}>\zeta_\ell} x_{ij} \leq \ell - 1 - \left|\{(i,j) : p_{ij} > \zeta_\ell, \overline{x}_{ij} = 1\}\right| & \forall \ell \in I^{\mathsf{POS}} \\ x \geq 0. \end{cases}$$

Clearly, $(\overline{x}_{ij})_{(i,j)\in E}$ is a feasible solution to (P'). We apply Theorem 4.6 to (P'), taking $\mathcal{M}_0$ to be the partition matroid now with ground set E encoding that every job in J' is assigned to at most one machine, and treating both the machine constraints indexed by I, and the POS-constraints indexed by I^{POS} as knapsack constraints. The weight vector w is irrelevant. We take $\mu_i = (1+\delta)$ for all $i \in I$, and $\mu_\ell = \delta\ell$ for all $\ell \in I^{\mathsf{POS}}$, and we claim that this satisfies condition (10). To see this, consider any $(i,j) \in E$. Let ℓ be the smallest index in I^{POS} whose POS-constraint contains the variable x_{ij}. Then, x_{ij} appears possibly in the machine constraint for machine i, and the POS-constraints for indices $\ell' \in \mathsf{POS}$, $\ell' \geq \ell$. We have

$$\sum_{\ell' \in I^{\mathsf{POS}}: \ell' \geq \ell} \frac{1}{\mu_{\ell'}} \leq \sum_{\ell' \in \mathsf{POS}: \ell' \geq \ell} \frac{1}{\delta \ell'} \leq \frac{1}{\delta\ell} \cdot \sum_{k \geq 0} (1+\delta)^{-k} = \frac{1}{\delta\ell} \cdot \frac{1+\delta}{\delta} \leq \frac{1}{\delta^2 \ell_0} \leq \frac{\delta}{2}.$$

Since $\frac{1}{1+\delta} + \frac{\delta}{2} \leq 1$, it follows that (10) is satisfied for (i,j).

Finally, we argue that applying Theorem 4.6 with the above parameters yields the lemma. Let $\widetilde{x}$ be the solution obtained by concatenating the integral portion of $\overline{x}$, and the integer solution returned by Theorem 4.6. Part (b) holds because we return a basis of $\mathcal{M}_0$, and the jobs not in J' are completely assigned by the integral portion of $\overline{x}$. Parts (c) and (e) follow directly from our choice of the μ values. ◀

▶ Remark 4.8. We note that the subtly stronger guarantee obtained for the knapsack constraint for index s, in Theorem 4.6, versus Corollary 11 in [19] – i.e., an additive error of $\mu_s \cdot \max_{e \in R} A_{se}$ versus an additive error of $\mu_s \cdot \max_{e \in N} A_{se}$ – is crucial for us. The weaker guarantee would yield, after dropping integral (and hence 0-valued) variables, that the load on each machine i is at most $\sum_j p_{ij}\overline{x}_{ij} + (1+\delta)\max_{j:\overline{x}_{ij}>0} p_{ij}$. But it is hard to obtain a bound on the f-norm of the vector $\{\max_{j:\overline{x}_{ij}>0} p_{ij}\}_{i\in[m]}$, since multiple coordinates here could correspond to the same job, and therefore constraints (9) do not help.

5 A PTAS for MinNormLB on identical machines

We now consider the oft-studied and important special case where all machines are identical – that is, we have $p_{ij} = p_j$ for all jobs j and machines i – and devise a PTAS in this setting.

▶ **Theorem 5.1.** *There is a PTAS for* MinNormLB *on identical machines.*

The above result substantially generalizes the well-known PTAS by Hochbaum and Shmoys for minimizing makespan (i.e., the special case of ℓ_∞ norm) on identical machines [11]. We utilize some of their insights, but need to combine them with various novel ideas to handle the substantial richness of arbitrary monotone, symmetric norms. In the full version, we show that our PTAS extends to a further generalization of MinNormLB, wherein each machine incurs a *cost* given by a "bounded-growth" convex, non-decreasing function μ of its load, and we seek to minimize the f-norm of the *machine-cost vector*. We also observe that Graham's list-scheduling rule – considering jobs in any sequence, schedule the next job on the currently least-loaded machine – yields a simple 2-approximation algorithm.

▶ **Theorem 5.2.** *Graham's list-scheduling rule yields a* 2*-approximation for* MinNormLB *on identical machines.*

We defer the proof of Theorem 5.2 to the full version, and discuss the PTAS for MinNormLB here. It is useful to first recall how the PTAS in [11] for makespan works. Their algorithm considers a "guess", say $t_1 \geq \max_j p_j$, of the optimal makespan, *opt*. We classify jobs as large or small, based on whether $p_j \geq \varepsilon t_1$ (large jobs) or $p_j < \varepsilon t_1$ (small jobs). A key insight is

that since the large jobs have $p_j \in [\varepsilon t_1, t_1]$, rounding their p_js to the nearest power of $(1+\varepsilon)$ yields $O\big(\log_{1+\varepsilon} \frac{1}{\varepsilon}\big)$ distinct rounded job sizes, and a constant (depending on ε) number of distinct possibilities, called *configurations*, for assigning large jobs to a machine so that its rounded load is at most $(1+\varepsilon)t_1$. One can write an *integer program*, with a (ε-dependent) constant number of variables, to determine which configurations are used on the machines, which can be solved in time $\text{poly}(m,n)$ for any fixed $\varepsilon > 0$ using the algorithm of Lenstra [18]. If $opt \le t_1$, then this yields an assignment of large jobs having makespan at most $(1+\varepsilon)t_1$. Finally, one can argue that either the small jobs can be packed (arbitrarily) on the machines while maintaining a makespan of at most $(1+\varepsilon)t_1$, or we have $opt > t_1$.

Now consider MinNormLB. We may assume that $n > m$ as otherwise it is easy to see that an optimal solution is to assign each job to a separate machine. Clearly, we may also assume that $p_j > 0$ for all jobs j. Recall that $\vec{o}$ is an optimal load vector, and $\mathsf{OPT} = f(\vec{o})$. As before, we assume that f is normalized. Since all machines are identical, it will be convenient to re-index machines so that $\vec{o} = \vec{o}^{\downarrow}$. Since we now need to control all Top_ℓ norms, following a common theme, it is natural that we now work with guesses t_ℓ of o_ℓ (which is $\vec{o}^{\downarrow}_\ell$) for all $\ell \in \mathsf{POS}$. The chief issue that arises is: *how do we now define large and small jobs given the multiple t_ℓs*? Suppose we choose a threshold of εt_ℓ, for some $\ell \in \mathsf{POS}$, for this purpose. For indices $\ell' < \ell$, the problem then is that we need to consider all jobs with $p_j \in [\varepsilon t_\ell, t_{\ell'}]$; this would yield a *non-constant number* of job types, and hence a non-constant number of configurations for machines $i < \ell$. Machines $i > \ell$ also present a problem: the packing of small jobs may cause us to exceed the target load of o_i (or rather t_i^{exp}) by εt_ℓ, which can create too large an error compared to the target load.

We circumvent these issues as follows. We select a *specific* optimal solution. Given distinct vectors $u, v \in \mathbb{R}^m$, we say that u is *lexicographically smaller* than v, denoted $u \prec_{\mathsf{lex}} v$ if there exists some index $i \in [m]$ such that $u_{i'} = v_{i'}$ for all $i' = 1, \ldots, i-1$, and $u_i < v_i$. Let $\vec{o}^{\downarrow}$ be the *lexicographically smallest* sorted load-vector among all optimal solutions, i.e., $\vec{o}^{\downarrow} \prec_{\mathsf{lex}} (\overrightarrow{\mathsf{load}}_\sigma)^{\downarrow}$ for every optimal solution σ. As before, we may assume that $\vec{o} = \vec{o}^{\downarrow}$. Let σ^* be an optimal solution yielding the load-vector $\vec{o}$. Let i^* be the smallest index such that machine i^* has at least two jobs assigned to it under σ^*; this is well defined since $n > m$.

▷ Claim 5.3. Let $\vec{o}$, σ^* and i^* be as defined above. Then we have $o_i \ge o_{i^*}/2$ for all $i \ge i^*$.

In addition to our guess $\vec{t}$ of $(o_\ell)_{\ell \in \mathsf{POS}}$, we now also guess i^* and the load o_{i^*}, and use εo_{i^*} as the threshold demarcating large and small jobs. The small jobs are now indeed small compared to the target load of o_i for $i \ge i^*$. Also, since machines $i < i^*$ only have one job assigned to them, we can infer that these machines are assigned the $i^* - 1$ largest jobs.

We describe our algorithm below, and conclude with its analysis.

▶ **Algorithm 1 (PTAS for MinNormLB on identical machines).**
Input: a non-increasing vector $\vec{t} \in \mathbb{R}^{\mathsf{POS}}_{\ge 0}$, an index $i^* \in [m]$, $\theta \in \mathbb{R}_{\ge 0}$ intended to be a good overestimate of o_{i^*}, and a parameter $0 < \varepsilon \le 1$. We set $\delta = \varepsilon$ in the definition $\mathsf{POS} = \mathsf{POS}_{m,\delta}$.
Output: an assignment $\widetilde{\sigma}$ such that $(\overrightarrow{\mathsf{load}}_{\widetilde{\sigma}})^{\downarrow}_\ell \le (1+\varepsilon)t_\ell + \varepsilon\theta$ for all $\ell \in \mathsf{POS}$, or that $(\vec{t}, i^*, \theta)$ is an invalid guess.

P1. Perform the following checks, and if any of these fail, then declare that $(\vec{t}, i^*, \theta)$ is invalid and return. Define $t_{m+1} := 0$ for notational convenience.
- **(a)** If $i^* \in \mathsf{POS}$, then check that $\theta = t_{i^*}$, otherwise check that $t_{\mathsf{next}(i^*)} \le \theta \le t_{\mathsf{prev}(i^*)}$.
- **(b)** For all $\ell \in \mathsf{POS}$, $\ell \le i^*$, check that there are at most $\ell - 1$ jobs with $p_j > t_\ell$.
- **(c)** Check that there are at most $i^* - 1$ jobs with $p_j > \theta$.

P2. Assign the $i^* - 1$ largest jobs to machines $1, \ldots, i^* - 1$, assigning the largest job to machine 1, the second-largest job to machine 2, and so on. Let J' be the remaining set of jobs. Note that check (c) in step P1 implies that $p_j \le \theta$ for all $j \in J'$.

P3. Let $J'_L := \{j \in J' : p_j \geq \varepsilon\theta\}$, and $J'_S := \{j \in J' : p_j < \varepsilon\theta\}$. We refer to the jobs in J'_L and J'_S as large and small jobs respectively.

P4. Round the processing time of each large job to the nearest power of $(1+\varepsilon)$, i.e., for each $j \in J'_L$, round its processing time to $p'_j := (1+\varepsilon)^{\lceil \log_{1+\varepsilon} p_j \rceil}$. Let $R = \{p'_j : j \in J'_L\}$ be the distinct rounded processing times, and $N = |R|$; note that $N \leq 1 + \log_{1+\varepsilon}\left(\frac{1}{\varepsilon}\right)$. For a budget $B \geq 0$, define $\mathcal{C}(B) := \left\{\nu \in \mathbb{Z}^R_{\geq 0} \setminus \{\vec{0}\} : \sum_{p\in R} \nu_p \cdot p \leq B\right\}$ as the set of non-trivial *configurations* that yield a total rounded load of at most B. (Observe that $\mathcal{C}(0) = \emptyset$.)

Recall that $\overrightarrow{t^{\mathsf{exp}}} \in \mathbb{R}^m_{\geq 0}$ is the expansion of $\vec{t}$, defined by $t^{\mathsf{exp}}_i = t_i$ for $i \in \mathsf{POS}$, and $t^{\mathsf{exp}}_i = t_{\mathsf{prev}(i)}$ for $i \in [m] \setminus \mathsf{POS}$. For notational convenience, define $t^{\mathsf{exp}}_{m+1} := 0$. For $i \in \{i^*, \ldots, m+1\}$, define its load budget to be $B_i = (1+\varepsilon)\min\{\theta, t^{\mathsf{exp}}_i\}$. (Note that $B_{m+1} = 0$.) For any $0 \leq B \leq B_{i^*}$, any $\nu \in \mathcal{C}(B)$ and $p \in R$, we have $\nu_p \leq B/p \leq \frac{1+\varepsilon}{\varepsilon}$; so $|\mathcal{C}(B)| \leq \left(\frac{1}{\varepsilon}+2\right)^N$, and we can enumerate the configurations in $\mathcal{C}(B)$ in polynomial time. Assign the large jobs by using the algorithm of Lenstra [18] to solve the integer program (Config-IP), wherein the integer variable x_ν specifies the number of machines in $\{i^*, \ldots, m\}$ for which configuration $\nu \in \mathcal{C}(B_{i^*})$ is used.

$$\sum_{\nu \in \mathcal{C}(B_{i^*})} \nu_p \cdot x_\nu = \left|\{j \in J'_L : p'_j = p\}\right| \qquad \forall p \in R \tag{11}$$

(Config-IP)
$$\sum_{\nu \in \mathcal{C}(B_{i^*}) \setminus \mathcal{C}(B_\ell)} x_\nu \leq \ell - i^* \qquad \forall \ell \in \mathsf{POS} \cup \{m+1\} : \ell > i^* \tag{12}$$

$$x_\nu \in \mathbb{Z}_{\geq 0} \qquad \forall \nu \in \mathcal{C}(B_{i^*}) \tag{13}$$

If (Config-IP) is infeasible, declare that $(\vec{t}, i^*, \theta)$ is invalid and return. Otherwise, suppose that $\widetilde{x}$ is a feasible solution to (Config-IP). Obtain an assignment of the large jobs to machines $i^*, i^*+1, \ldots, m$ from $\widetilde{x}$ as follows. Let $\widetilde{\mathcal{C}}$ be the *multiset* of configurations where we have $\widetilde{x}_\nu$ copies of each configuration $\nu \in \mathrm{supp}(\widetilde{x})$. We map each $\nu \in \widetilde{\mathcal{C}}$ to a disjoint set A_ν of large jobs, comprising ν_p large jobs with $p'_j = p$ for every $p \in R$, so that $(A_\nu)_{\nu \in \widetilde{\mathcal{C}}}$ forms a partition of J'_L. Constraint (11) shows that this is always possible.

We go over the configurations $\nu \in \widetilde{\mathcal{C}}$ in non-increasing order of their load, $\sum_{p\in S} \nu_p \cdot p$, and assign one configuration each to machines $i^*, i^*+1, \ldots, i^* + |\widetilde{\mathcal{C}}| - 1$ in this order, where by assigning a configuration ν to a machine i, we mean that we assign the jobs in A_ν to machine i. Note that $|\widetilde{\mathcal{C}}| = \sum_{\nu \in \mathcal{C}(B_{i^*})} \widetilde{x}_\nu \leq m + 1 - i^*$ due to constraint (12) for index $m+1$.

P5. We assign the small jobs to the machines in $\{i^*, \ldots, m\}$ arbitrarily while ensuring that the total actual load (i.e., under the p_j processing times) on each machine i is at most $B_i + \varepsilon\theta$. If we are unable to assign all small jobs this way, then we declare that $(\vec{t}, i^*, \theta)$ is invalid, and return.

P6. Return the assignment $\widetilde{\sigma}$ computed in steps P2, P4, and P5.

Analysis. Recall that σ^* is the optimal solution yielding the sorted load vector $\vec{o}^{\downarrow} = \vec{o}$. We first assume that $(\vec{t}, i^*, \theta)$ is such that $\vec{t} \geq (o_\ell)_{\ell \in \mathsf{POS}}$, we have the right i^*, $\theta \geq o_{i^*}$, and $(\vec{t}, i^*, \theta)$ passes the check in step P1(a). Under these assumptions, we show that the algorithm returns an assignment $\widetilde{\sigma}$, and $\overrightarrow{\mathsf{load}}^{\downarrow}_{\widetilde{\sigma}}$ is "close" to $\overrightarrow{t^{\mathsf{exp}}}$. This will then imply Theorem 5.1, since we can find in polytime $(\vec{t^*}, i^*, \theta^*)$ satisfying the above properties, and such that $t^*_\ell \leq (1+\varepsilon)o_\ell$ for all $\ell \in [m]$, and $\theta^* \leq (1+\varepsilon)o_{i^*}$.

We begin by observing in Claim 5.4 that (under the above assumptions), $(\vec{t}, i^*, \theta)$ passes checks (b) and (c) in step P1, and that step P2 is valid. Then, in Lemmas 5.5 and 5.6, we argue that steps P4 and P5 are successful.

▷ Claim 5.4. (i) $(\vec{t}, i^*, \theta)$ passes checks (b) and (c) in step P1. (ii) The optimal solution σ^* assigns the $i^* - 1$ largest jobs to machines $1, \ldots, i^* - 1$, assigning one job per machine.

▶ **Lemma 5.5.** *Step P4 successfully returns an assignment of large jobs such that every machine $i \in \{i^*, \dots, m\}$ has (rounded and actual) load at most B_i.*

Proof. Consider the following solution to (Config-IP). Each machine $i \in \{i^*, \dots, m\}$ that has jobs assigned to it under σ^* naturally maps to a corresponding configuration ν, where ν_p is the number of large jobs assigned to i with $p'_j = p$. We have

$$\sum_{p \in R} \nu_p \cdot p = \sum_{j \in J'_L : \sigma^*(j) = i} p'_j \leq (1+\varepsilon) \sum_{j : \sigma^*(j) = i} p_j = (1+\varepsilon) o_i \leq (1+\varepsilon) \min\{\theta, t_i^{\mathsf{exp}}\} = B_i$$

where the first inequality is because $p'_j \leq (1+\varepsilon) p_j$ for all $j \in J'_L$. Thus, each machine $i \in \{i^*, \dots, m\}$ maps to a configuration in $\mathcal{C}(B_i) \subseteq \mathcal{C}(B_{i^*})$. Let $\hat{x} \in \mathbb{Z}_{\geq 0}^{\mathcal{C}(B_{i^*})}$ be the integral vector, where $\hat{x}_\nu$ is the number of machines that map to configuration ν, for each $\nu \in \mathcal{C}(B_{i^*})$. By construction, it is easy to see that constraints (11) are satisfied. It is also clear that constraint (12) holds for index $\ell = m+1$. So consider constraint (12) for some index $\ell \in \mathsf{POS}$, $\ell > i^*$. A configuration ν with $\hat{x}_\nu > 0$ whose load is larger than B_ℓ corresponds to a machine in $\{i^*, \dots, m\}$ whose actual load is larger than $B_\ell/(1+\varepsilon) \geq o_\ell$. There are at most $\ell - i^*$ such machines, so (12) holds for index ℓ.

Since (Config-IP) is feasible, it follows that we return an assignment of large jobs to machines $i^*, i^*+1, \dots, m$ where every machine has rounded (and hence, actual) load at most B_{i^*}. Consider a machine $i \in \{i^*, \dots, m\}$ that is assigned a configuration. Let $\ell = i$ if $i \in \mathsf{POS} \cup \{i^*\}$, and $\ell = \max\{i^*, \mathsf{prev}(i)\}$ otherwise; note that $B_i = B_\ell$. If $\ell = i^*$, then $B_i = B_{i^*}$, and there is nothing left to prove. So suppose $\ell \neq i^*$. Then $\ell \in \mathsf{POS}$, $\ell > i^*$. By constraint (12) there are at most $\ell - i^* \leq i - i^*$ configurations in $\widetilde{\mathcal{C}}$ with load larger than B_ℓ. By the way in which we assign configurations to machines, it follows that machine i is *not* assigned one of these configurations. Therefore, the total rounded (and hence, actual) load on machine i is at most $B_\ell = B_i$. ◀

▶ **Lemma 5.6.** *Step P5 successfully assigns all small jobs. At the end of this step every machine $i \in \{i^*, \dots, m\}$ has total actual load at most $B_i + \varepsilon\theta$.*

Proof of Theorem 5.1. We can find $(\vec{t^*}, i^*, \theta^*)$ in polytime such that $o_\ell \leq t^*_\ell \leq (1+\varepsilon) o_\ell$ for all $\ell \in \mathsf{POS}$, we have the right i^*, $o_{i^*} \leq \theta^* \leq (1+\varepsilon) o_{i^*}$, and $(\vec{t^*}, i^*, \theta^*)$ clears check P1 (a). We prove this momentarily, but first show that this yields the desired performance guarantee.

Recall that $\varepsilon \leq 1$ and $\delta = \varepsilon$. By Lemma 2.8 (b), taking $u = \vec{o}$ and $v = \vec{t^*}$, we obtain that $f(\overrightarrow{t^{*\mathsf{exp}}}) \leq (1+\delta)(1+\varepsilon) f(\vec{o})$. Our analysis shows that the algorithm run with input $(\vec{t^*}, i^*, \theta^*)$ returns an assignment $\widetilde{\sigma}$. Since we pass checks (b) and (c) in step P1, each machine $i \in \{1, \dots, i^*-1\}$ has load (due to the one job assigned to it) $o_i \leq {t^*}_i^{\mathsf{exp}}$. Lemmas 5.5 and 5.6 show that the total load on each machine $i \in \{i^*, \dots, m\}$ is at most

$$B_i + \varepsilon\theta^* \leq (1+\varepsilon) {t^*}_i^{\mathsf{exp}} + \varepsilon(1+\varepsilon) o_{i^*} \leq (1+\varepsilon) {t^*}_i^{\mathsf{exp}} + 4\varepsilon o_i \leq (1+5\varepsilon) {t^*}_i^{\mathsf{exp}}.$$

The second inequality follows from Claim 5.3 and since $\varepsilon \leq 1$, and the last one since $\vec{t^*} \geq (o_\ell)_{\ell \in \mathsf{POS}}$ (and therefore $\overrightarrow{t^{*\mathsf{exp}}} \geq o$). It follows that $\overrightarrow{\mathsf{load}}_{\widetilde{\sigma}} \leq (1+5\varepsilon) t^{*\mathsf{exp}}$, and so $f(\overrightarrow{\mathsf{load}}_{\widetilde{\sigma}}) \leq (1+5\varepsilon) f(\overrightarrow{t^{*\mathsf{exp}}})$. Combining this with the bound on $f(\overrightarrow{t^{*\mathsf{exp}}})$, and simplifying, we obtain that $f(\overrightarrow{\mathsf{load}}_{\widetilde{\sigma}}) \leq (1+23\varepsilon) f(\vec{o})$.

We complete the proof by showing how to find $(\vec{t^*}, i^*, \theta^*)$ in polynomial time satisfying the properties stated at the beginning of the proof. There are only m choices for i^*. Given the correct i^*, we know that the load on each machine $i < i^*$ is the processing time of the i-th largest job. So we know $o_1, \dots, o_{i^*-1}$ exactly, and can set $t^*_\ell = o_\ell$ for all $\ell \in \mathsf{POS}$, $\ell < i^*$. Let $\mathsf{POS}' = \{i^*\} \cup \{\ell \in \mathsf{POS} : \ell > i^*\}$. Let J' be the set of jobs excluding the

$i^* - 1$ largest jobs. Let $\mathsf{UB}' := \sum_{j \in J'} p_j$. The jobs in J' are assigned by σ^* to machines in $\{i^*, \ldots, m\}$, and we have $o_{i^*} \geq o_i \geq o_{i^*}/2$ for all $i \geq i^*$ (Claim 5.3). It follows that $\frac{\mathsf{UB}'}{m-i^*+1} \leq o_{i^*} \leq T := \min\{\frac{2\mathsf{UB}'}{m-i^*+1}, o_{i^*-1}\}$. So by Lemma 2.9 (b), taking $\mathcal{L} = \mathsf{POS}'$, $\mathsf{ub} = T$, and $\mathsf{lb} = \frac{\mathsf{UB}'}{2(m-i^*+1)}$, we can identify a set of size $O\big(m^{O(\frac{1}{\varepsilon})}\big)$ containing a non-increasing vector $v \in \mathbb{R}^{\mathsf{POS}'}_{\geq 0}$ such that $o_\ell \leq v_\ell \leq \min\{(1+\varepsilon)o_\ell, T\}$ for all $\ell \in \mathsf{POS}'$. So the vector $\vec{t^*} = \big((o_\ell)_{\ell \in \mathsf{POS}:\ell < i^*}, (v_\ell)_{\ell \in \mathsf{POS}:\ell \geq i^*}\big)$ and $\theta^* = v_{i^*}$ have the desired properties. ◀

References

1 Noga Alon, Yossi Azar, Gerhard J. Woeginger, and Tal Yadid. Approximation schemes for scheduling on parallel machines. *Journal of Scheduling*, 1(1):55–66, 1998.
2 Yossi Azar and Amir Epstein. Convex programming for scheduling unrelated parallel machines. In *Proceedings of the 37th STOC*, pages 331–337, 2005.
3 Deeparnab Chakrabarty. Personal Communication, 2021.
4 Deeparnab Chakrabarty and Chaitanya Swamy. Approximation algorithms for minimum norm and ordered optimization problems. In *Proceedings of the 51st STOC*, pages 126–137, 2019.
5 Deeparnab Chakrabarty and Chaitanya Swamy. Simpler and better algorithms for minimum-norm load balancing. In *Proceedings of the 27th ESA*, pages 27:1–27:12, 2019.
6 Anindya De, Sanjeev Khanna, Huan Li, and Hesam Nikpey. An efficient PTAS for stochastic load balancing with poisson jobs. In *Proceedings of the 47th ICALP*, pages 37:1–37:18, 2020.
7 Ashish Goel and Piotr Indyk. Stochastic load balancing and related problems. In *Proceedings of the 40th FOCS*, pages 579–586, 1999.
8 Anupam Gupta, Amit Kumar, Viswanath Nagarajan, and Xiangkun Shen. Stochastic load balancing on unrelated machines. In *Proceedings of the 29th SODA*, pages 1274–1285, 2018.
9 G. H. Hardy, J. E. Littlewood, and G. Pólya. *Inequalities*. Cambridge Univ Press, 1934.
10 D. S. Hochbaum and D. B. Shmoys. A polynomial approximation scheme for scheduling on uniform processors: Using the dual approximation approach. *SICOMP*, 17(3):539–551, 1988.
11 Dorit Hochbaum and David B. Shmoys. Using dual approximation algorithms for scheduling problems: practical and theoretical results. *J. ACM*, 34(1):144–162, 1987.
12 Sharat Ibrahimpur and Chaitanya Swamy. Approximation algorithms for stochastic minimum-norm combinatorial optimization. In *Proceedings of the 61st FOCS*, pages 966–977, 2020.
13 K. Jansen. An EPTAS for scheduling jobs on uniform processors: Using an MILP relaxation with a constant number of integral variables. In *Proc. 36th ICALP*, pages 562–573, 2009.
14 Klaus Jansen, Kim-Manuel Klein, and José Verschae. Closing the Gap for Makespan Scheduling via Sparsification Techniques. In *Proceedings of the 43rd ICALP*, pages 72:1–72:13, 2016.
15 Jon M. Kleinberg, Yuval Rabani, and Éva Tardos. Allocating bandwidth for bursty connections. *SIAM J. Comput.*, 30(1):191–217, 2000.
16 V. S. Kumar, Madhav V Marathe, S. Parthasarathy, and A. Srinivasan. A unified approach to scheduling on unrelated parallel machines. *Journal of the ACM*, 56(5):28, 2009.
17 Jan Karel Lenstra, David B. Shmoys, and Éva Tardos. Approximation algorithms for scheduling unrelated parallel machines. *Math. Program.*, 46:259–271, 1990.
18 Hendrik W. Lenstra Jr. Integer programming with a fixed number of variables. *Mathematics of Operations Research*, 8(4):538–548, 1983.
19 André Linhares, Neil Olver, Chaitanya Swamy, and Rico Zenklusen. Approximate multi-matroid intersection via iterative refinement. *Math. Program.*, 183(1):397–418, 2020.
20 Konstantin Makarychev and Maxim Sviridenko. Solving optimization problems with diseconomies of scale via decoupling. *J. ACM*, 65(6):42:1–42:27, 2018.
21 Albert W Marshall, Ingram Olkin, and Barry Arnold. *Inequalities: Theory of Majorization and Its Applications*. Springer series in statistics. Springer, 2011.
22 Marco Molinaro. Stochastic ℓ_p load balancing and moment problems via the L-function method. In *Proceedings of the 30th SODA*, pages 343–354, 2019.
23 David B. Shmoys and Éva Tardos. An approximation algorithm for the generalized assignment problem. *Math. Program.*, 62:461–474, 1993.

Quasi-Polynomial Time Algorithms for Free Quantum Games in Bounded Dimension

Hyejung H. Jee
Department of Computing, Imperial College London, UK

Carlo Sparaciari
Department of Computing, Imperial College London, UK
Department of Physics and Astronomy, University College London, UK

Omar Fawzi
Univ Lyon, ENS Lyon, UCBL, CNRS, Inria, LIP, F-69342, Lyon Cedex 07, France

Mario Berta
Department of Computing, Imperial College London, UK
IQIM, California Institute of Technology, Pasadena, CA, USA
AWS Center for Quantum Computing, Pasadena, CA, USA[1]

Abstract

In a recent landmark result [Ji *et al.*, arXiv:2001.04383 (2020)], it was shown that approximating the value of a two-player game is undecidable when the players are allowed to share quantum states of unbounded dimension. In this paper, we study the computational complexity of two-player games when the dimension of the quantum systems is bounded by T. More specifically, we give a semidefinite program of size $\exp\big(\mathcal{O}\big(T^{12}(\log^2(AT)+\log(Q)\log(AT))/\epsilon^2\big)\big)$ to compute additive ϵ-approximations on the value of two-player free games with $T\times T$-dimensional quantum entanglement, where A and Q denote the number of answers and questions of the game, respectively. For fixed dimension T, this scales polynomially in Q and quasi-polynomially in A, thereby improving on previously known approximation algorithms for which worst-case run-time guarantees are at best exponential in Q and A. For the proof, we make a connection to the quantum separability problem and employ improved multipartite quantum de Finetti theorems with linear constraints that we derive via quantum entropy inequalities.

2012 ACM Subject Classification Theory of computation

Keywords and phrases non-local game, semidefinite programming, quantum correlation, approximation algorithm, Lasserre hierarchy, de Finetti theorem

Digital Object Identifier 10.4230/LIPIcs.ICALP.2021.82

Category Track A: Algorithms, Complexity and Games

Related Version *Full Version*: `https://arxiv.org/abs/2005.08883`

1 Introduction

Thanks to the celebrated discovery by John Bell [4], it is well-known that quantum correlations can be used to overcome locality constraints, which was one of the earliest examples of advantages provided by quantum correlations over classical correlations. This led to the development of numerous quantum information processing tasks which make use of quantum correlations as a resource to outperform their classical analogues. In general, understanding the differences in the performance of distinct correlation sets for a given task is important both fundamentally and practically. A common way to measure the quantitative advantages

[1] This work was completed prior to MB joining the AWS Center for Quantum Computing.

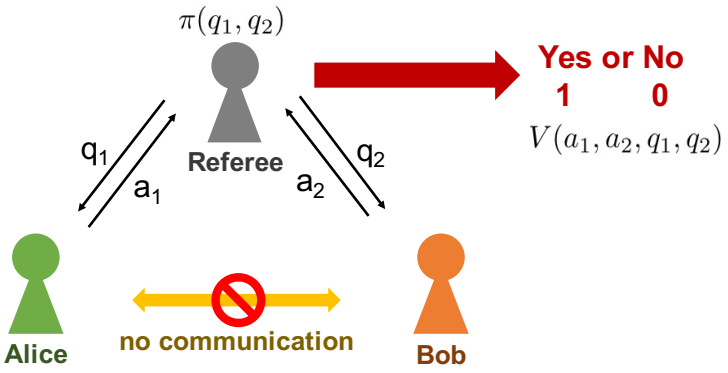


Figure 1 Two-player games. The referee gives Alice and Bob questions $q_1 \in Q_1$ and $q_2 \in Q_2$ according to the question probability distribution $\pi(q_1, q_2)$, and then Alice and Bob give answers $a_1 \in A_1$ and $a_2 \in A_2$ back to the referee depending on the questions they received. The referee decides whether Alice and Bob win or lose according to the rule function $V : A_1 \times A_2 \times Q_1 \times Q_2 \to \{0, 1\}$, where 0 denotes losing the game, and 1 denotes winning the game. Alice and Bob cannot communicate with each other during the game, but they can agree on a strategy beforehand. We are interested in determining the values of the game, i.e., the maximum achievable winning probabilities, for different classes of strategies. For simplicity, we assume that $|Q_1| = |Q_2| = Q$ and $|A_1| = |A_2| = A$.

of different sets of correlations is via a two-player game G (illustrated in Figure 1). In a two-player game, the performance of a given correlation set is quantified by the maximum achievable winning probability. For example, the classical value $\omega_C(G)$ is the maximum winning probability that can be achieved using shared randomness between the two players, while the quantum value $\omega_Q(G)$ is the maximum winning probability that can be achieved by sharing arbitrary quantum states between the players.

In general, it is hard to compute $\omega_C(G)$ and $\omega_Q(G)$ for the given description of a two-player game G. Approximating $\omega_C(G)$ within some constant multiplicative factor is NP-hard [2, 3], while approximating $\omega_Q(G)$ has recently been shown not to be possible for an algorithm running in finite time [20]. Despite these general hardness results, there are some special classes of two-player games for which $\omega_C(G)$ and $\omega_Q(G)$ can be approximated in polynomial time [10, 21, 1, 9]. In particular, for *free games*, i.e., games where the questions for the two players are chosen independently, there exists a quasi-polynomial time algorithm that can approximate $\omega_C(G)$ within any constant additive error [1, 9]. Also, in practice, the Navascués-Pironio-Acín (NPA) hierarchy [27, 29] provides semidefinite programming (SDP) upper bounds on $\omega_Q(G)$ which give approximately tight bounds for many games of interest.

1.1 Contributions

In this paper, we study the dimension-bounded quantum value $\omega_{Q(T)}(G)$ – the maximum winning probability that can be achieved by sharing quantum states of fixed dimension $T \times T$. It is easy to see that $\omega_{Q(1)}(G) = \omega_C(G)$ and $\omega_Q(G) = \sup_{T \geq 1} \omega_{Q(T)}(G)$. Computing $\omega_{Q(T)}(G)$ is of particular interest since it can be used as a *dimension witness* for an underlying system in semi-device-independent quantum information processing protocols, see for example [15]. SDP upper bounds have been derived for $\omega_{Q(T)}(G)$ in [25, 28, 26]. In [25], the authors exploit a connection to the quantum separability problem, and in [28, 26], the authors employ a moment matrix technique similar to the NPA hierarchy to derive SDP relaxations with better performance than the ones in [25]. However, the worst case runtime guarantees for these works is either not analytically quantified or is at best exponential in the number of questions Q and the number of answers A of the game G.

In our work, we provide approximation algorithms for $\omega_{Q(T)}(G)$ whose runtime has an improved dependence on both A and Q. More specifically, we construct a new hierarchy of SDP relaxations, providing a sequence of upper bounds for $\omega_{Q(T)}(G)$ for a given game G, and then derive analytical bounds on the convergence speed. This gives an upper bound on the computational complexity of calculating $\omega_{Q(T)}(G)$ in terms of the size of the game G. For the case of free games, a semidefinite program of size

$$\exp\left(\mathcal{O}\left(\frac{T^{12}}{\epsilon^2}\log(AT)\left(\log(Q)+\log(AT)\right)\right)\right) \tag{1}$$

is sufficient for computing additive ϵ-approximations of $\omega_{Q(T)}(G)$, where A and Q denote the number of answers and questions, respectively. The dependence is quasi-polynomial in A and polynomial in Q thus improving on the best previously known approximation algorithms [25, 28, 26], for which only exponential bounds in A and Q are known. In the classical limit ($T = 1$), our result recovers the quasi-polynomial time approximation scheme for computing $\omega_C(G)$ for two-player free games – which has a matching hardness result assuming the Exponential Time Hypothesis [1, 9]. Besides analysing free games, we give an algorithm for general games as well, leading to approximation algorithms that are still quasi-polynomial in A but exponential in Q.

We construct our SDP relaxations by drawing a connection to a variant of the quantum separability problem where the optimisation variables are additionally subject to some linear constraints. Similar variants of the quantum separability problem have been studied in [35, 34, 6]. The main tool we use to obtain the analytical convergence speed is improved multipartite quantum de Finetti theorems with linear constraints, which we derive in our work. One of the contributions towards this result, which we believe is of independent interest, is an improved version of the optimal loss in distinguishability relative to quantum side information.

1.2 Preliminaries on two-player games

A non-local game is a mathematical formulation for the correlations between distant parties. In this paper, we will consider *two-player games* where only two distant parties are involved. In this formulation, the correlation between two parties is considered to be a resource to win the games.

In a two-player game G, two spatially separated agents, Alice and Bob, need to provide correct answers $a_1 \in A_1$ and $a_2 \in A_2$ to the referee depending on the questions $q_1 \in Q_1$ and $q_2 \in Q_2$ they received (see Figure 1). The correct answers are determined by a given rule function of G

$$V : A_1 \times A_2 \times Q_1 \times Q_2 \to \{0, 1\}, \tag{2}$$

where 0 means the answer is incorrect, and 1 means the answer is correct. The questions q_1 and q_2 are chosen by the referee according to a given probability distribution $\pi(q_1, q_2)$ of G. A specific two-player game G can be represented by the pair of rule function $V(a_1, a_2, q_1, q_2)$ and question probability distribution $\pi(q_1, q_2)$, and hereafter we will denote a game G as (V, π). Alice and Bob cannot communicate with each other during the game, but they can agree on a strategy beforehand as well as make use of systems whose correlations lie within a given class. When only classical shared randomness is allowed, the correlations take the form

$$p(a_1, a_2|q_1, q_2) = e(a_1|q_1)d(a_2|q_2), \tag{3}$$

where $e(a_1|q_1)$ and $d(a_2|q_2)$ are conditional probability distributions for Alice and Bob respectively. That is, $\sum_{a_1} e(a_1|q_1) = 1\ \forall q_1 \in Q_1$, and $\sum_{a_2} d(a_2|q_2) = 1\ \forall q_2 \in Q_2$. When quantum resources are allowed, the correlations have a more general form

$$p(a_1, a_2|q_1, q_2) = \operatorname{tr}\left[\rho_{T\hat{T}}\left(E_T(a_1|q_1) \otimes D_{\hat{T}}(a_2|q_2)\right)\right], \tag{4}$$

where $\rho_{T\hat{T}}$ is a possibly entangled quantum state shared by Alice and Bob, and $\{E_T(a_1|q_1)\}_{a_1}$ and $\{D_{\hat{T}}(a_2|q_2)\}_{a_2}$ are positive-operator valued measurements (POVMs) performed by Alice and Bob respectively for given q_1 and q_2, i.e., $\sum_{a_1} E_T(a_1|q_1) = \mathbb{I}_T\ \forall q_1 \in Q_1$ and $\sum_{a_2} D_{\hat{T}}(a_2|q_2) = \mathbb{I}_{\hat{T}}\ \forall q_2 \in Q_2$.

The quantitative advantage of each set of correlations can be captured by the maximum winning probabilities achievable using the given correlation set. For a given two-player game $G = (V, \pi)$, the classical value is defined as

$$\omega_C(V, \pi) := \max_{(e,d)} \sum_{a1,q1,a2,q2} \pi(q_1, q_2) V(a_1, a_2, q_1, q_2) e(a_1|q_1) d(a_2|q_2), \tag{5}$$

and the quantum value is given by

$$\omega_Q(V, \pi) := \sup_{\substack{(E\otimes D,\rho)\\ \text{on } \mathcal{H}_{T\hat{T}}}} \sum_{\substack{a1,q1\\ a2,q2}} \pi(q_1, q_2) V(a_1, a_2, q_1, q_2) \operatorname{tr}\left[\rho_{T\hat{T}}\left(E_T(a_1|q_1) \otimes D_{\hat{T}}(a_2|q_2)\right)\right]. \tag{6}$$

Here, the optimisation is taken over not only states and measurements but also the Hilbert space $\mathcal{H}_{T\hat{T}}$. We can define the dimension-bounded quantum value as

$$\omega_{Q(T)}(V, \pi) := \max_{\substack{(E\otimes D,\ \rho)\\ \text{on}\\ \mathbb{C}^T\otimes\mathbb{C}^T}} \sum_{\substack{a1,q1\\ a2,q2}} \pi(q_1, q_2) V(a_1, a_2, q_1, q_2) \operatorname{tr}\left[\rho_{T\hat{T}}\left(E_T(a_1|q_1) \otimes D_{\hat{T}}(a_2|q_2)\right)\right], \tag{7}$$

which is the central object of investigation in this paper.

If not stated otherwise, we assume that the choice of questions for Alice and Bob are independent, i.e., $\pi(q_1, q_2) = \pi_1(q_1)\pi_2(q_2)$, which corresponds to *free games*. We denote $\mathcal{H}_A^{\otimes n}$ as A^n, and $\dim(\mathcal{H}_A)$ as $|A|$. For simplicity, we assume that $|Q_1| = |Q_2| = Q$ and $|A_1| = |A_2| = A$.

2 Derivation of semidefinite programming relaxations

2.1 Connection with quantum separability

Quantum separability problems are a special type of optimisation problems, where the optimisation is taken over the set of separable quantum states. We show that computing $\omega_{Q(T)}(V, \pi)$ for a given two-player game (V, π) can be rephrased as an instance of the tripartite quantum separability problem subject to additional linear constraints.

▶ **Lemma 1.** *For a two-player free game with $V(a_1,a_2,q_1,q_2)$, $\pi(q_1,q_2)=\pi_1(q_1)\pi_2(q_2)$, and $|T|^2$-dimensional quantum correlation, we have*

$$\begin{aligned}
\omega_{Q(T)}(V,\pi) = |T|^2 \cdot \max_{(E,D,\rho)} \ & \mathrm{tr}\left[\left(V_{A_1A_2Q_1Q_2}\otimes\Phi_{T\hat{T}|S\hat{S}}\right)\left(E_{A_1Q_1T}\otimes D_{A_2Q_2\hat{T}}\otimes\rho_{S\hat{S}}\right)\right] \\
s.t. \quad & \rho_{S\hat{S}}\geq 0\,, \qquad \mathrm{tr}\left[\rho_{S\hat{S}}\right]=1 \\
& E_{A_1Q_1T}=\sum_{a_1,q_1}\pi_1(q_1)\,|a_1q_1\rangle\langle a_1q_1|_{A_1Q_1}\otimes\frac{E_T(a_1|q_1)}{|T|}\geq 0 \\
& D_{A_2Q_2\hat{T}}=\sum_{a_2,q_2}\pi_2(q_2)\,|a_2q_2\rangle\langle a_2q_2|_{A_2Q_2}\otimes\frac{D_{\hat{T}}(a_2|q_2)}{|T|}\geq 0 \\
& \mathrm{tr}_{A_1}\left[E_{A_1Q_1T}\right]=\sum_{q_1}\pi_1(q_1)\,|q_1\rangle\langle q_1|_{Q_1}\otimes\frac{\mathbb{I}_T}{|T|} \\
& \mathrm{tr}_{A_2}\left[D_{A_2Q_2\hat{T}}\right]=\sum_{q_2}\pi_2(q_2)\,|q_2\rangle\langle q_2|_{Q_2}\otimes\frac{\mathbb{I}_{\hat{T}}}{|T|},
\end{aligned} \tag{8}$$

where $\Phi_{T\hat{T}|S\hat{S}}=|\Phi\rangle\langle\Phi|_{T\hat{T}|S\hat{S}}$ is the (non-normalised) maximally-entangled state, $|\Phi\rangle_{T\hat{T}|S\hat{S}}=\sum_i|i\rangle_{T\hat{T}}|i\rangle_{S\hat{S}}$, and $V_{A_1A_2Q_1Q_2}$ is a diagonal matrix whose entries are given by the rule function $V(a_1,a_2,q_1,q_2)$.

To prove Lemma 1, we need a slightly modified version of the swap trick.

▶ **Lemma 2.** *Let M_{AB} be a linear operator on $\mathcal{H}_A\otimes\mathcal{H}_B$, and N_A be a linear operator on $\mathcal{H}_A$. Then, it holds that*

$$\mathrm{tr}\left[(N_A\otimes\mathbb{I}_B)M_{AB}\right]=\mathrm{tr}\left[\left(F_{\hat{A}|A}\otimes\mathbb{I}_B\right)\left(N_{\hat{A}}\otimes M_{AB}\right)\right], \tag{9}$$

where $F_{\hat{A}|A}$ denotes the swap operator between $\hat{A}$ and A.

Proof. By inspection, we have that

$$\begin{aligned}
& \mathrm{tr}\left[\left(F_{\hat{A}|A}\otimes\mathbb{I}_B\right)\left(N_{\hat{A}}\otimes M_{AB}\right)\right] \\
=\,& \mathrm{tr}\left[\left(F_{\hat{A}|A}\otimes\mathbb{I}_B\right)\left(\sum_{i,j}n_{ij}\,|i\rangle\langle j|_{\hat{A}}\otimes\sum_{k,\ell,s,t}m_{(k\ell)(st)}\,|k\rangle\langle\ell|_A\otimes|s\rangle\langle t|_B\right)\right] \\
=\,& \mathrm{tr}\left[\sum_{i,j,k,\ell,s,t}n_{ij}\,m_{(k\ell)(st)}\,|k\rangle\langle j|_{\hat{A}}\otimes|i\rangle\langle\ell|_A\otimes|s\rangle\langle t|_B\right] \\
=\,& \sum_{i,j,s,t}n_{ij}\,m_{(ji)(st)}=\mathrm{tr}\left[(N_A\otimes\mathbb{I}_B)M_{AB}\right],
\end{aligned} \tag{10}$$

where we used $N_{\hat{A}}=\sum_{i,j}n_{ij}\,|i\rangle\langle j|_{\hat{A}}$ and $M_{AB}=\sum_{k,\ell,s,t}m_{(k\ell)(st)}\,|k\rangle\langle\ell|_A\otimes|s\rangle\langle t|_B$. ◀

Proof of Lemma 1. Let us start from the expression for $\omega_{Q(T)}$ in Eq. (7). For free games, i.e. $\pi(q_1, q_2) = \pi_1(q_1)\pi_2(q_2)$, we can write

$$\omega_{Q(T)}(V,\pi) = |T|^2 \max_{E,D,\rho} \operatorname{tr}\left[\left(V_{A_1A_2Q_1Q_2} \otimes \rho_{T\hat{T}}\right)\left(E_{A_1Q_1T} \otimes D_{A_2Q_2\hat{T}}\right)\right] \tag{11}$$

$$\begin{aligned}
\text{s.t.}\quad & \rho_{T\hat{T}} \geq 0\,, \qquad \operatorname{tr}\left[\rho_{T\hat{T}}\right] = 1 \\
& E_{A_1Q_1T} = \sum_{a_1,q_1} \pi_1(q_1)\, |a_1q_1\rangle\langle a_1q_1|_{A_1Q_1} \otimes \frac{E_T(a_1|q_1)}{|T|} \geq 0 \\
& D_{A_2Q_2\hat{T}} = \sum_{a_2,q_2} \pi_2(q_2)\, |a_2q_2\rangle\langle a_2q_2|_{A_2Q_2} \otimes \frac{D_{\hat{T}}(a_2|q_2)}{|T|} \geq 0 \\
& \operatorname{tr}_{A_1}\left[E_{A_1Q_1T}\right] = \sum_{q_1} \pi_1(q_1)\, |q_1\rangle\langle q_1|_{Q_1} \otimes \frac{\mathbb{I}_T}{|T|} \\
& \operatorname{tr}_{A_2}\left[D_{A_2Q_2\hat{T}}\right] = \sum_{q_2} \pi_2(q_2)\, |q_2\rangle\langle q_2|_{Q_2} \otimes \frac{\mathbb{I}_{\hat{T}}}{|T|},
\end{aligned}$$

where we define $V_{A_1A_2Q_1Q_2} := \sum_{a_1,a_2,q_1,q_2} V(a_1,a_2,q_1,q_2)\, |a_1,a_2,q_1,q_2\rangle\langle a_1,a_2,q_1,q_2|$. Then, using Lemma 2 we can rewrite the objective function in Eq. (11) as

$$\begin{aligned}
& \operatorname{tr}\left[\left(V_{A_1A_2Q_1Q_2} \otimes \rho_{T\hat{T}}\right)\left(E_{A_1Q_1T} \otimes D_{A_2Q_2\hat{T}}\right)\right] \\
&= \operatorname{tr}\left[\left(\mathbb{I}_{A_1A_2Q_1Q_2} \otimes \rho_{T\hat{T}}\right)\left(\left(V_{A_1A_2Q_1Q_2} \otimes \mathbb{I}_{T\hat{T}}\right)\left(E_{A_1Q_1T} \otimes D_{A_2Q_2\hat{T}}\right)\right)\right] \\
&= \operatorname{tr}\left[\left(\mathbb{I}_{A_1A_2Q_1Q_2} \otimes F_{T\hat{T}|S\hat{S}}\right)\left(\left(\left(V_{A_1A_2Q_1Q_2} \otimes \mathbb{I}_{T\hat{T}}\right)\left(E_{A_1Q_1T} \otimes D_{A_2Q_2\hat{T}}\right)\right) \otimes \rho_{S\hat{S}}\right)\right] \\
& \qquad\qquad \text{(by Lemma 2)} \\
&= \operatorname{tr}\left[\left(\left(V_{A_1A_2Q_1Q_2} \otimes F_{T\hat{T}|S\hat{S}}\right)\left(E_{A_1Q_1T} \otimes D_{A_2Q_2\hat{T}} \otimes \rho_{S\hat{S}}\right)\right)\right],
\end{aligned} \tag{12}$$

which has a similar form to the objective function in Lemma 1 with the exception that $F_{T\hat{T}|S\hat{S}}$ replaces $\Phi_{T\hat{T}|S\hat{S}}$. To complete the proof, we write the swap operator $F_{A|\hat{A}}$ in terms of the (non-normalised) maximally-entangled state $\Phi_{A|\hat{A}} = |\Phi\rangle\langle\Phi|_{A|\hat{A}}$, where $|\Phi\rangle_{A|\hat{A}} = \sum_{i=1}^{d_A} |i\rangle_A\, |i\rangle_{\hat{A}}$. Namely, we have $F_{A|\hat{A}} = \Phi_{A|\hat{A}}^{T_A}$, where T_A denotes the transposition over the A subsystem. Redefining the variable ρ as ρ^T, we then immediately obtain Eq. (8) as this last step leaves the constraints invariant. ◀

In Lemma 1, the optimisation is now taken over all product states with respect to the tripartition $A_1Q_1T|A_2Q_2\hat{T}|S\hat{S}$ satisfying the stated linear constraints. Since product states are extreme points in the set of separable states, we can equivalently think of the above as an optimisation over the convex hull of the feasible states, where the feasible states are all product states satisfying the linear constraints. This gives the claimed connection to the quantum separability problem.

2.2 Hierarchy of semidefinite programming relaxations

In the previous section, we showed that $\omega_{Q(T)}(V,\pi)$ can be rephrased as a variant of the quantum separability problem which is subject to additional linear constraints. However, solving quantum separability problems is known to be NP-hard [16, 17], and our mapping does not necessarily make the problem more approachable. Fortunately, there are well-known

relaxations for the quantum separability condition; the Doherty-Parrilo-Spedalieri (DPS) hierarchy [12] based on extendibility, which is strongly related to the notion of monogamy of entanglement [33].

▶ **Definition 3** (Extendibility). *A bipartite quantum state ρ_{AB} is n-extendible if there exists a multipartite quantum state ρ_{AB^n} such that*

$$\mathrm{tr}_{B^{n-1}}\left[\rho_{AB^n}\right] = \rho_{AB}, \quad \left(\mathcal{I}_A \otimes \mathcal{U}^{\pi}_{B^n}\right)\left(\rho_{AB^n}\right) = \rho_{AB^n} \quad \forall \pi \in \mathcal{S}(B^n), \tag{13}$$

where $\mathcal{S}(B^n)$ is the symmetric group over B^n, $\mathcal{U}^{\pi}_{B^n}(\cdot) = U^{\pi}_{B^n}(\cdot)(U^{\pi}_{B^n})^{\dagger}$ is the adjoint representation of the group, and $U^{\pi}_{B^n}$ is a unitary permutation operator acting on B^n.

Extendible states have two main advantages. Firstly, deciding if a state is n-extendible can be done efficiently via SDPs [11, 12]; for fixed n, the computation resources scale polynomially in the system dimension. Secondly, it is shown that a quantum state is n-extendible for all $n \geq 2$ if and only if the state is separable [14, 30]. Thus, the set of n-extendible states is a good outer approximation for the separable set and converges to the separable set when $n \to \infty$. The same idea can be generalised to the tripartite case as well; (n_1, n_2)-extendible states ρ_{ABC} with the two-fold extension $\rho_{AB^{n_1}C^{n_2}}$. As in the bipartite case, the set of (n_1, n_2)-extendible states converges to the set of tripartite separable states when $n_1 \to \infty$ and $n_2 \to \infty$ [13].

To derive SDP relaxations for $\omega_{Q(T)}(V, \pi)$ in Eq. (8), we can simply replace the optimisation variables with (n, n)-extendible states with respect to the appropriate tripartition.

$$\mathrm{sdp}_n(V, \pi, T) := |T|^2 \max_{\rho} \mathrm{tr}\left[\left(V_{A_1A_2Q_1Q_2} \otimes \Phi_{T\hat{T}|S\hat{S}}\right)\rho_{(A_1Q_1T)(A_2Q_2\hat{T})(S\hat{S})}\right] \tag{14}$$

$$\text{s.t.} \quad \rho_{(A_1Q_1T)(A_2Q_2\hat{T})^n(S\hat{S})^n} \geq 0\,, \quad \mathrm{tr}\left[\rho_{(A_1Q_1T)(A_2Q_2\hat{T})^n(S\hat{S})^n}\right] = 1 \tag{15}$$

$$\rho_{(A_1Q_1T)(A_2Q_2\hat{T})^n(S\hat{S})^n} \text{ perm. inv. on } (A_2Q_2\hat{T})^n \text{ wrt } (A_1Q_1T)(S\hat{S})^n \tag{16}$$

$$\rho_{(A_1Q_1T)(A_2Q_2\hat{T})^n(S\hat{S})^n} \text{ perm. inv. on } (S\hat{S})^n \text{ wrt } (A_1Q_1T)(A_2Q_2\hat{T})^n \tag{17}$$

$$\mathrm{tr}_{A_1}[\rho_{(A_1Q_1T)(A_2Q_2\hat{T})^n(S\hat{S})^n}] = \left(\sigma_{Q_1} \otimes \frac{\mathbb{I}_T}{|T|}\right) \otimes \rho_{(A_2Q_2\hat{T})^n(S\hat{S})^n} \tag{18}$$

$$\mathrm{tr}_{A_2}[\rho_{(A_1Q_1T)(A_2Q_2\hat{T})^n(S\hat{S})^n}] = \left(\sigma_{Q_2} \otimes \frac{\mathbb{I}_{\hat{T}}}{|T|}\right) \otimes \rho_{(A_1Q_1T)(A_2Q_2\hat{T})^{(n-1)}(S\hat{S})^n} \tag{19}$$

$$\rho^{T_{A_1Q_1T}}_{(A_1Q_1T)(A_2Q_2\hat{T})^n(S\hat{S})^n} \geq 0\,, \quad \rho^{T_{(A_2Q_2\hat{T})^n}}_{(A_1Q_1T)(A_2Q_2\hat{T})^n(S\hat{S})^n} \geq 0, \ \dots\,, \tag{20}$$

where $\sigma_{Q_i} = \sum_{q_i} \pi_i(q_i)\,|q_i\rangle\langle q_i|_{Q_i}$ for $i = 1, 2$, and the last line Eq. (20) contains all positive partial transpose (PPT) conditions with respect to all the cuts

$$A_1Q_1T : A_2^1Q_2^1\hat{T}^1 : \cdots : A_2^nQ_2^n\hat{T}^n : S^1\hat{S}^1 : \cdots : S^n\hat{S}^n. \tag{21}$$

Note that in addition to the n-extendibility conditions Eq. (16)–(17) enforced by the DPS hierarchy, we arrive at the additional linear constraints, Eq. (18)–(19), originating from the constraints in Eq. (8). These additional constraints are crucial in order to obtain the improved complexity bounds. Furthermore, we are able to combine our SDPs with the NPA constraints [27], so that our new hierarchy is guaranteed to produce at least as good outputs as the ones produced by the NPA hierarchy (see the full version [19, Section 5]),

$$\mathrm{sdp}^{\mathrm{NPA}}_n(V, \pi, T) := \mathrm{sdp}_n(V, \pi, T) \text{ with } \Gamma_n\left(\rho_{(A_1Q_1T)(A_2Q_2\hat{T})^n(S\hat{S})^n}\right) \geq 0, \tag{22}$$

where $\Gamma_n(\rho)$ denotes the n-th level NPA matrix.

It is worth noting that $\mathrm{sdp}_n(V, \pi, T)$ in Eq. (14) is naturally upper bounded by 1.

▶ **Proposition 4.** *Let $sdp_n(V,\pi,T)$ be the n-th level SDP relaxation for the two-player free game with rule matrix V, probability distribution $\pi(q_1,q_2)=\pi_1(q_1)\pi_2(q_2)$, and $|T|^2$-dimensional quantum correlation. Then, we have that*

$$0 \leq sdp_n(V,\pi,T) \leq 1. \tag{23}$$

The proof can be found in the full version [19, Proposition 5].

3 Convergence of the hierarchy

3.1 Tripartite quantum de Finetti theorem with additional linear constraints

Quantum de Finetti theorems provide a quantitative bound on how close n-extendible states are to the set of separable states in trace distance as a function of both n and the system's dimensions. This information can be converted to the upper bound on the accuracy of our SDP relaxations. However, since the quantum separability problem for $\omega_{Q(T)}(V,\pi)$ in Eq. (8) is subject to the additional linear constraints, we cannot directly exploit the standard quantum de Finetti theorem and need an adapted version (we refer to [6, Example 3.7] for a discussion of counterexamples). What we need is an upper bound on how close n-extendible states satisfying the linear constraints are to the separable states satisfying the same linear constraints.

In this paper, we derive improved multipartite quantum de Finetti theorems with additional linear constraints employing the information-theoretic proof technique based on quantum entropy inequalities [8, 9]. Using this adapted quantum de Finetti theorems is crucial to obtain the improved complexity bounds on approximating $\omega_{Q(T)}(V,\pi)$ in the next section. Here, we state the tripartite version of the theorem.

▶ **Theorem 5.** *Let $\rho_{AB^{n_1}C^{n_2}}$ be a quantum state which is invariant under permutations on B^{n_1} with respect to AC^{n_2} and on C^{n_2} with respect to AB^{n_1}, satisfying for linear maps $\mathcal{E}_{A\to\tilde{A}}$, $\Lambda_{B\to\tilde{B}}$, and $\Gamma_{C\to\tilde{C}}$ and operators $\mathbf{X}_{\tilde{A}}$, $\mathbf{Y}_{\tilde{B}}$, and $\mathbf{Z}_{\tilde{C}}$ that*

$$\left(\mathcal{E}_{A\to\tilde{A}}\otimes\mathcal{I}_{B^{n_1}C^{n_2}}\right)\left(\rho_{AB^{n_1}C^{n_2}}\right)=\mathbf{X}_{\tilde{A}}\otimes\rho_{B^{n_1}C^{n_2}} \qquad \text{linear constraint on } A \tag{24}$$

$$\left(\Lambda_{B\to\tilde{B}}\otimes\mathcal{I}_{B^{n_1-1}C^{n_2}}\right)\left(\rho_{B^{n_1}C^{n_2}}\right)=\mathbf{Y}_{\tilde{B}}\otimes\rho_{B^{n_1-1}C^{n_2}} \qquad \text{linear constraint on } B \tag{25}$$

$$\left(\mathcal{I}_{B^{n_1}C^{n_2-1}}\otimes\Gamma_{C\to\tilde{C}}\right)\left(\rho_{B^{n_1}C^{n_2}}\right)=\mathbf{Z}_{\tilde{C}}\otimes\rho_{B^{n_1}C^{n_2-1}} \qquad \text{linear constraint on } C. \tag{26}$$

Then, there exist a probability distribution $\{p_i\}_{i\in I}$ and sets of quantum states $\{\sigma_A^i\}_{i\in I}$, $\{\omega_B^i\}_{i\in I}$ and $\{\tau_C^i\}_{i\in I}$ such that we have that

$$\begin{aligned}&\left\|\rho_{ABC}-\sum_{i\in I}p_i\sigma_A^i\otimes\omega_B^i\otimes\tau_C^i\right\|_1\\&\qquad\leq\min\left\{18^{3/2}\sqrt{|ABC|},4|BC|\right\}\times\sqrt{2\ln 2}\left(\sqrt{\frac{\log|A|+8\log|B|}{n_2}+\frac{\log|A|}{n_1}}\right)\end{aligned} \tag{27}$$

$$\mathcal{E}_{A\to\tilde{A}}\left(\sigma_A^i\right)=\mathbf{X}_{\tilde{A}},\quad \Lambda_{B\to\tilde{B}}\left(\omega_B^i\right)=\mathbf{Y}_{\tilde{B}},\quad \Gamma_{C\to\tilde{C}}\left(\tau_C^i\right)=\mathbf{Z}_{\tilde{C}}\qquad\forall i\in I. \tag{28}$$

Like any other de Finetti theorem, Theorem 5 can be understood as a statement on the monogamy of entanglement; a multipartite system, described by an extendible state, cannot possess much entanglement between any tripartition. Instead of directly working with the

trace distance, we prove the above theorem via quantum entropy inequalities and chain rules. This approach allows us to carefully quantify how correlations are divided between different partitions of the extendible states.

For k given quantum systems $A_1, ..., A_k$ and a classical system R described by the global state $\rho_{A_1A_2\cdots A_kR}$, the conditional multipartite quantum mutual information is defined as

$$I(A_1 : A_2 : \ldots : A_k|R) := \sum_{i=1}^{k} S(A_iR) - S(A_1A_2 \ldots A_kR) - S(R), \tag{29}$$

where $S(A_i) = -\operatorname{tr}[\rho_{A_i} \log \rho_{A_i}]$ is the von Neumann entropy [23] of the marginal state ρ_{A_i}. This quantity has a few useful mathematical properties. One is its relation to the bipartite ones [9, Lemma 3]

$$I(A_1 : \ldots : A_k|R) = I(A_1 : A_2|R) + I(A_1A_2 : A_3|R) + \ldots + I(A_1 \ldots A_{k-1} : A_k|R), \tag{30}$$

and another one is the chain rule

$$I(AB : C|D) = I(B : C|D) + I(A : C|BD). \tag{31}$$

The conditional multipartite quantum mutual information is mathematically equivalent to the relative entropy distance between the state and the tensor product of its conditional marginals

$$I(A_1 : A_2 : \ldots : A_k|R) = D(\rho_{A_1\cdots A_k|R} \| \rho_{A_1|R} \otimes \ldots \otimes \rho_{A_k|R}), \tag{32}$$

where $\rho_{A_i|R}$ is the marginal state of the conditional $\rho_{A_1\cdots A_k|R} = \rho_R^{-1/2}\rho_{A_1\cdots A_k}\rho_R^{-1/2}$, and $D(\rho||\sigma) = \operatorname{tr}[\rho(\log\rho - \log\sigma)]$ is the relative entropy between ρ and σ whenever $\operatorname{supp}(\rho) \subset \operatorname{supp}(\sigma)$. The relative entropy can be further related to the trace distance via Pinsker's inequality. As the tensor product of marginal states is a separable state, if we can find an upper bound on the conditional multipartite quantum mutual information of an extendible state $\rho_{AB^{n_1}C^{n_2}}$, we can show Eq. (27) in Theorem 5.

For the first ingredient, we derive a general upper bound on the conditional multipartite quantum mutual information of a state with classical subsystems.

▶ **Lemma 6.** *Consider a quantum state $\rho_{AZ^{n_1}W^{n_2}}$ classical on the Z- and W-systems. Then, there exist $0 \le \bar{m} < n_1$ and $0 \le \bar{l} < n_2$ such that*

$$I(A : Z_{\bar{m}+1} : W_{\bar{l}+1}|Z^{\bar{m}}W^{\bar{l}}) \le \frac{\log|A|}{n_1} + \frac{\log|A| + \log|Z|}{n_2}. \tag{33}$$

Moreover, by Pinsker's inequality, this implies that

$$\mathbb{E}_{z^{\bar{m}}w^{\bar{l}}} \left\{ \left\| \rho_{AZ_{\bar{m}+1}W_{\bar{l}+1}|z^{\bar{m}}w^{\bar{l}}} - \rho_{A|z^{\bar{m}}w^{\bar{l}}} \otimes \rho_{Z_{\bar{m}+1}|z^{\bar{m}}w^{\bar{l}}} \otimes \rho_{W_{\bar{l}+1}|z^{\bar{m}}w^{\bar{l}}} \right\|_1^2 \right\} \tag{34}$$
$$\le 2\ln 2 \left(\frac{\log|A|}{n_1} + \frac{\log|A| + \log|Z|}{n_2} \right).$$

Here, we use the notation $\rho_{A|z}$ for the conditional state after measurement on classical system Z when the measurement outcome is z, i.e.,

$$\rho_{A|z} := \frac{\operatorname{tr}_Z[\rho_{AZ}(\mathbb{I}_A \otimes |z\rangle\langle z|_Z)]}{\operatorname{tr}[\rho_{AZ}(\mathbb{I}_A \otimes |z\rangle\langle z|_Z)]}. \tag{35}$$

The proof of Lemma 6 is as follows.

Proof of Lemma 6. The multipartite quantum mutual information $I(A : Z_{\bar{m}+1} : W_{\bar{l}+1}|Z^{\bar{m}}W^{\bar{l}})$ can be expressed in terms of bipartite ones using Eq. (30):

$$I(A : Z_{\bar{m}+1} : W_{\bar{l}+1}|Z^{\bar{m}}W^{\bar{l}}) = I(A : Z_{\bar{m}+1}|Z^{\bar{m}}W^{\bar{l}}) + I(AZ_{\bar{m}+1} : W_{\bar{l}+1}|Z^{\bar{m}}W^{\bar{l}}). \tag{36}$$

The two terms in the right hand side (RHS) are the bipartite mutual information between quantum and classical systems, and this allows us to find an upper bound for each term using the chain rule in Eq. (31). Additionally, we also make use of a general upper bound

$$I(A : Z|X) \leq \log|A| \tag{37}$$

for a classical-quantum state ρ_{AZX} with classical Z and X systems [19, Lemma 13].

First term: For any l, it holds that

$$I(A : Z^{n_1}|W^l) = \sum_{m=0}^{n_1-1} I(A : Z_{m+1}|Z^m W^l) \leq \log|A|, \tag{38}$$

where the first equality is the chain rule in Eq. (31) and the second inequality is found by applying Eq. (37) to $I(A : Z^{n_1}|W^l)$. Then, summing over all l gives us

$$\sum_{m=0}^{n_1-1}\sum_{l=0}^{n_2-1} I(A : Z_{m+1}|Z^m W^l) \leq n_2 \log|A|. \tag{39}$$

Second term: Using the same argument, for any m, it holds that

$$I(AZ_{m+1} : W^{n_2}|Z^m) = \sum_{l=0}^{n_2-1} I(AZ_{m+1} : W_{l+1}|Z^m W^l) \leq \log|AZ_{m+1}|, \tag{40}$$

and summing over m gives us

$$\sum_{m=0}^{n_1-1}\sum_{l=0}^{n_2-1} I(AZ_{m+1} : W_{l+1}|Z^m W^l) \leq n_1 \left(\log|A| + \log|Z|\right). \tag{41}$$

Combining Eq. (39) and Eq. (41) gives

$$\begin{aligned} &n_2 \log|A| + n_1 \left(\log|A| + \log|Z|\right) \\ &\geq \sum_{m=0}^{n_1-1}\sum_{l=0}^{n_2-1} \left[I(A : Z_{m+1}|Z^m W^l) + I(AZ_{m+1} : W_{l+1}|Z^m W^l)\right] \\ &\geq n_1 n_2 \left[I(A : Z_{\bar{m}+1}|Z^{\bar{m}}W^{\bar{l}}) + I(AZ_{\bar{m}+1} : W_{\bar{l}+1}|Z^{\bar{m}}W^{\bar{l}})\right], \end{aligned} \tag{42}$$

where $\bar{m}$ and $\bar{l}$ are the indices of the smallest element in the sum. Dividing both sides by $n_1 n_2$ gives us the desired relation,

$$\begin{aligned} I(A : Z_{\bar{m}+1} : W_{\bar{l}+1}|Z^{\bar{m}}W^{\bar{l}}) &= I(A : Z_{\bar{m}+1}|Z^{\bar{m}}W^{\bar{l}}) + I(AZ_{\bar{m}+1} : W_{\bar{l}+1}|Z^{\bar{m}}W^{\bar{l}}) \\ &\leq \frac{\log|A|}{n_1} + \frac{\log|A| + \log|Z|}{n_2}. \end{aligned} \tag{43}$$

This ends the proof of Eq. (33). Then, using Eq. (32) and Pinsker's inequality we can obtain Eq. (34). ◀

As another ingredient, we derive two different types of informationally complete measurements that achieve the optimal loss in distinguishability.

▶ **Lemma 7.**

1. ([8, Lemma 14]) *There exist fixed measurements $\mathcal{M}_A$, $\mathcal{M}_B$, and $\mathcal{M}_C$ with at most $|A|^8$, $|B|^8$, and $|C|^8$ outcomes, respectively, such that for every traceless Hermitian operator γ_{ABC} on $\mathcal{H}_{ABC}$ we have*

$$\|\gamma_{ABC}\|_1 \leq 18^{3/2}\sqrt{|ABC|}\cdot\|(\mathcal{M}_A\otimes\mathcal{M}_B\otimes\mathcal{M}_C)(\gamma_{ABC})\|_1. \tag{44}$$

2. *There exists a fixed measurement $\mathcal{M}_B$ with at most $|B|^6$ outcomes such that for every traceless Hermitian operator γ_{AB} on $\mathcal{H}_{AB}$ we have*

$$\|\gamma_{AB}\|_1 \leq 2|B|\cdot\|(\mathcal{I}_A\otimes\mathcal{M}_B)(\gamma_{AB})\|_1. \tag{45}$$

The first part is straightforward from [8, Lemma 14]. We remark that when a traceless Hermitian operator already has a classical subsystem, i.e., γ_{ABCZ} with classical Z-system, the dimension factor only includes the dimension of the quantum systems

$$\|\gamma_{ABCZ}\|_1 \leq 18^{3/2}\sqrt{|ABC|}\cdot\|(\mathcal{M}_A\otimes\mathcal{M}_B\otimes\mathcal{M}_C\otimes\mathcal{I}_Z)(\gamma_{ABCZ})\|_1. \tag{46}$$

This follows easily as $\|\sum_z \rho_A^z\otimes|z\rangle\langle z|\|_1 = \sum_z\|\rho_A^z\|_1$ for classical-quantum states ρ_{AZ}.

The proof of the second part is given in Section 5. The main idea is to identify the one-way quantum teleportation protocol as a candidate for the optimal measurement and is largely inspired by [22, Theorem 16]. Our result improves on the factor $\sqrt{18}B^{3/2}$ given in [9, Eq.(68)]. Moreover, as there exist quantum states ρ_{AB} and σ_{AB} such that [24]

$$\|\rho_{AB}-\sigma_{AB}\|_1 = 2 \quad\text{and}\quad \sup_{\mathcal{M}_B}\|(\mathcal{I}_A\otimes\mathcal{M}_B)(\rho_{AB}-\sigma_{AB})\|_1 = \frac{2}{|B|+1}, \tag{47}$$

our result establishes that the dimension dependence for the optimal loss in distinguishability relative to quantum side information is $\Theta(|B|)$. This answers a question left open in [6].

Then, for the extendible state $\rho_{AB^{n_1}C^{n_2}}$ in Theorem 5, applying the optimal measurement $\mathcal{M}$ as specified in Lemma 7 to the state (to make it partially classical), and applying Lemma 6 to the resulting classical-quantum state allows us to derive Theorem 5.

Proof of Theorem 5. Let $\mathcal{M}_{B\to Y}$ be a quantum-to-classical measurement from B to the classical system Y, and $\mathcal{M}_{C\to Z}$ be a quantum-to-classical measurement from C to the classical system Z. We apply these measurements to the quantum state $\rho_{AB^{n_1}C^{n_2}}$ and will denote the outcome classical-quantum state as $\rho_{AY^{n_1}Z^{n_2}}$. Then, according to Lemma 6, there exist $m\in\{0,\cdots,n_1-1\}$ and $\ell\in\{0,\cdots,n_2-1\}$ such that

$$\mathbb{E}_{y^m z^\ell}\left\{\left\|\rho_{AY_{m+1}Z_{\ell+1}|y^m z^\ell}-\rho_{A|y^m z^\ell}\otimes\rho_{Y_{m+1}|y^m z^\ell}\otimes\rho_{Z_{\ell+1}|y^m z^\ell}\right\|_1^2\right\} \leq 2\ln 2\left(\frac{\log|A|}{n_1}+\frac{\log|A|+\log|Y|}{n_2}\right). \tag{48}$$

As $\rho_{AB^{n_1}C^{n_2}}$ is invariant under permutations of the systems B^{n_1} and C^{n_2}, we can always find m and l satisfying Eq. (48).

Now, let us define

$$\gamma_{ABC}\equiv\rho_{AB_{m+1}C_{\ell+1}|y^m z^\ell}-\rho_{A|y^m z^\ell}\otimes\rho_{B_{m+1}|y^m z^\ell}\otimes\rho_{C_{\ell+1}|y^m z^\ell}. \tag{49}$$

Note that

$$\mathcal{I}_A\otimes\mathcal{M}_{B\to Y}\otimes\mathcal{M}_{C\to Z}(\gamma_{ABC}) = \rho_{AY_{m+1}Z_{\ell+1}|y^m z^\ell}-\rho_{A|y^m z^\ell}\otimes\rho_{Y_{m+1}|y^m z^\ell}\otimes\rho_{Z_{\ell+1}|y^m z^\ell}. \tag{50}$$

Using the second part of Lemma 7 iteratively, we can obtain

$$\begin{aligned}\|\gamma_{ABC}\|_1 &\leq 2|C|\|(\mathcal{I}_{AB}\otimes\mathcal{M}_{C\to Z})(\gamma_{ABC})\|_1\\ &\leq 2|B|\times 2|C|\|(\mathcal{I}_{AC}\otimes\mathcal{M}_{B\to Y})(\mathcal{I}_{AB}\otimes\mathcal{M}_{C\to Z})(\gamma_{ABC})\|_1\\ &= 4|BC|\|(\mathcal{I}_A\otimes\mathcal{M}_{B\to Y}\otimes\mathcal{M}_{C\to Z})(\gamma_{ABC})\|_1,\end{aligned} \tag{51}$$

with $|Y|\leq|B|^6$. We can also exploit the first part of Lemma 7 to obtain

$$\begin{aligned}\|\gamma_{ABC}\|_1 &\leq \sqrt{18^3|ABC|}\,\|(\mathcal{M}_A\otimes\mathcal{M}_{B\to Y}\otimes\mathcal{M}_{C\to Z})(\gamma_{ABC})\|_1\\ &\leq \sqrt{18^3|ABC|}\,\|(\mathcal{I}_A\otimes\mathcal{M}_{B\to Y}\otimes\mathcal{M}_{C\to Z})(\gamma_{ABC})\|_1\end{aligned} \tag{52}$$

with $|Y|\leq|B|^8$, where the second inequality follows from the monotonicity of the trace norm under completely positive and trace preserving (CPTP) maps. Depending on the dimensions, we can freely choose the tighter bound between the two cases. Combining Eq. (48) with the above two results we obtain

$$\begin{aligned}&\mathbb{E}_{y^m z^\ell}\left\{\left\|\rho_{AB_{m+1}C_{\ell+1}|y^m z^\ell}-\rho_{A|y^m z^\ell}\otimes\rho_{B_{m+1}|y^m z^\ell}\otimes\rho_{C_{\ell+1}|y^m z^\ell}\right\|_1^2\right\}\\ &\quad\leq\min\left\{\sqrt{18^3|ABC|},4|BC|\right\}^2\times 2\ln 2\left(\frac{\log|A|}{n_1}+\frac{\log|A|+8\log|B|}{n_2}\right).\end{aligned} \tag{53}$$

Then, we have

$$\begin{aligned}&\left\|\rho_{AB_{m+1}C_{\ell+1}}-\mathbb{E}_{y^m z^\ell}\left\{\rho_{A|y^m z^\ell}\otimes\rho_{B_{m+1}|y^m z^\ell}\otimes\rho_{C_{\ell+1}|y^m z^\ell}\right\}\right\|_1\\ &\leq\ \mathbb{E}_{y^m z^\ell}\left\{\left\|\rho_{AB_{m+1}C_{\ell+1}|y^m z^\ell}-\rho_{A|y^m z^\ell}\otimes\rho_{B_{m+1}|y^m z^\ell}\otimes\rho_{C_{\ell+1}|y^m z^\ell}\right\|_1\right\}\\ &\leq\ \sqrt{\mathbb{E}_{y^m z^\ell}\left\{\left\|\rho_{AB_{m+1}C_{\ell+1}|y^m z^\ell}-\rho_{A|y^m z^\ell}\otimes\rho_{B_{m+1}|y^m z^\ell}\otimes\rho_{C_{\ell+1}|y^m z^\ell}\right\|_1^2\right\}}\\ &\leq\ \min\left\{\sqrt{18^3|ABC|},4|BC|\right\}\times\sqrt{2\ln 2}\left(\sqrt{\frac{\log|A|}{n_1}+\frac{\log|A|+8\log|B|}{n_2}}\right),\end{aligned} \tag{54}$$

where we used the triangular inequality for Schatten p-norms in the second line and the concavity of the square function in the third line. As $\mathbb{E}_{y^m z^\ell}\left\{\rho_{A|y^m z^\ell}\otimes\rho_{B_{m+1}|y^m z^\ell}\otimes\rho_{C_{\ell+1}|y^m z^\ell}\right\}$ is a separable state with respect to the tripartition $A|B|C$, this proves the first half of the theorem.

The remaining part is to check whether $\rho_{A|y^m z^\ell}$, $\rho_{B_{m+1}|y^m z^\ell}$ and $\rho_{C_{\ell+1}|y^m z^\ell}$ satisfy the desired linear constraints. Let us denote $M_{B_i}^{y_i}$ and $M_{C_i}^{z_i}$ as the POVM elements of the measurements $\mathcal{M}_{B_i\to Y_i}$ and $\mathcal{M}_{C_i\to Z_i}$ corresponding to the measurement outcomes y_i and z_i, respectively. Then, we find

$$\begin{aligned}&\mathcal{E}_{A\to\tilde{A}}\left(\sigma_A^i\right)=\mathcal{E}_{A\to\tilde{A}}\left(\rho_{A|y^m z^\ell}\right)\\ &=\frac{\mathrm{Tr}_{B^m C^\ell}\left[(\mathbb{I}_A\otimes M_{B_1}^{y_1}\otimes\cdots\otimes M_{B_m}^{y_m}\otimes M_{C_1}^{z_1}\otimes\cdots\otimes M_{C_\ell}^{z_\ell})\mathcal{E}_{A\to\tilde{A}}\left(\rho_{AB^m C^\ell}\right)\right]}{\mathrm{Tr}\left[(\mathbb{I}_A\otimes M_{B_1}^{y_1}\otimes\cdots\otimes M_{B_m}^{y_m}\otimes M_{C_1}^{z_1}\otimes\cdots\otimes M_{C_\ell}^{z_\ell})\rho_{AB^m C^\ell}\right]}\\ &=\frac{\mathrm{Tr}_{B^m C^\ell}\left[(\mathbb{I}_A\otimes M_{B_1}^{y_1}\otimes\cdots\otimes M_{B_m}^{y_m}\otimes M_{C_1}^{z_1}\otimes\cdots\otimes M_{C_\ell}^{z_\ell})\left(\mathcal{X}_{\tilde{A}}\otimes\rho_{B^m C^\ell}\right)\right]}{\mathrm{Tr}\left[(\mathbb{I}_A\otimes M_{B_1}^{y_1}\otimes\cdots\otimes M_{B_m}^{y_m}\otimes M_{C_1}^{z_1}\otimes\cdots\otimes M_{C_\ell}^{z_\ell})\rho_{AB^m C^\ell}\right]}\\ &=\mathcal{X}_{\tilde{A}}.\end{aligned} \tag{55}$$

$$\begin{aligned}
&\Lambda_{B\to\tilde{B}}\left(\omega_B^i\right) = \Lambda_{B\to\tilde{B}}\left(\rho_{B_{m+1}|y^m z^\ell}\right) \qquad (56)\\
&= \frac{\mathrm{Tr}_{B^m C^\ell}\left[(\mathbb{I}_{\tilde{B}}\otimes M_{B_1}^{y_1}\otimes\cdots\otimes M_{B_m}^{y_m}\otimes M_{C_1}^{z_1}\otimes\cdots\otimes M_{C_\ell}^{z_\ell})\Lambda_{B\to\tilde{B}}\left(\rho_{B^{m+1}C^\ell}\right)\right]}{\mathrm{Tr}\left[(M_{B_1}^{y_1}\otimes\cdots\otimes M_{B_m}^{y_m}\otimes\mathbb{I}_{B_{m+1}}\otimes M_{C_1}^{z_1}\otimes\cdots\otimes M_{C_\ell}^{z_\ell})\rho_{B^{m+1}C^\ell}\right]}\\
&= \frac{\mathrm{Tr}_{B^m C^\ell}\left[(\mathbb{I}_{\tilde{B}}\otimes M_{B_1}^{y_1}\otimes\cdots\otimes M_{B_m}^{y_m}\otimes M_{C_1}^{z_1}\otimes\cdots\otimes M_{C_\ell}^{z_\ell})\left(\mathcal{Y}_{\tilde{B}}\otimes\rho_{B^m C^\ell}\right)\right]}{\mathrm{Tr}\left[(M_{B_1}^{y_1}\otimes\cdots\otimes M_{B_m}^{y_m}\otimes\mathbb{I}_{B_{m+1}}\otimes M_{C_1}^{z_1}\otimes\cdots\otimes M_{C_\ell}^{z_\ell})\rho_{B^{m+1}C^\ell}\right]}\\
&= \mathcal{Y}_{\tilde{B}}.
\end{aligned}$$

$$\begin{aligned}
&\Gamma_{C\to\tilde{C}}\left(\tau_C^i\right) = \Gamma_{C\to\tilde{C}}\left(\rho_{C_{\ell+1}|y^m z^\ell}\right) \qquad (57)\\
&= \frac{\mathrm{Tr}_{B^m C^\ell}\left[(\mathbb{I}_{\tilde{C}}\otimes M_{B_1}^{y_1}\otimes\cdots\otimes M_{B_m}^{y_m}\otimes M_{C_1}^{z_1}\otimes\cdots\otimes M_{C_\ell}^{z_\ell})\Gamma_{C\to\tilde{C}}\left(\rho_{B^m C^{\ell+1}}\right)\right]}{\mathrm{Tr}\left[(M_{B_1}^{y_1}\otimes\cdots\otimes M_{B_m}^{y_m}\otimes M_{C_1}^{z_1}\otimes\cdots\otimes M_{C_\ell}^{z_\ell}\otimes\mathbb{I}_{C_{\ell+1}})\left(\rho_{B^m C^{\ell+1}}\right)\right]}\\
&= \frac{\mathrm{Tr}_{B^m C^\ell}\left[(\mathbb{I}_{\tilde{C}}\otimes M_{B_1}^{y_1}\otimes\cdots\otimes M_{B_m}^{y_m}\otimes M_{C_1}^{z_1}\otimes\cdots\otimes M_{C_\ell}^{z_\ell})\left(\mathcal{Z}_{\tilde{C}}\otimes\rho_{B^m C^\ell}\right)\right]}{\mathrm{Tr}\left[(M_{B_1}^{y_1}\otimes\cdots\otimes M_{B_m}^{y_m}\otimes M_{C_1}^{z_1}\otimes\cdots\otimes M_{C_\ell}^{z_\ell}\otimes\mathbb{I}_{C_{\ell+1}})\left(\rho_{B^m C^{\ell+1}}\right)\right]}\\
&= \mathcal{Z}_{\tilde{C}}. \qquad \blacktriangleleft
\end{aligned}$$

Theorem 5 describes a general setting; both the extendible state and the linear constraints do not have any refined structures. However, in our case, we have more information about the state and the constraints. The extendible state $\rho_{(A_1Q_1T)(A_2Q_2\hat{T})^n(S\hat{S})^n}$ in $\mathrm{sdp}_n(V,\pi,T)$ to which we apply the de Finetti theorem already has some classical subsystems, and the linear constraints are partial trace constraints. We can exploit this information to obtain a better bound in the quantum de Finetti theorem. We state this special case as a lemma.

▶ **Lemma 8.** *Let $\rho_{(AX\tilde{X})B^{n_1}(CZ\tilde{Z})^{n_2}}$ be a quantum state with classical $X\tilde{X}$- and $Z\tilde{Z}$-systems invariant under permutation on B^{n_1} and $(CZ\tilde{Z})^{n_2}$ with respect to the other systems, satisfying*

$$\mathrm{tr}_X\left[\rho_{(AX\tilde{X})B^{n_1}(CZ\tilde{Z})^{n_2}}\right] = \mathcal{X}_{A\tilde{X}}\otimes\rho_{B^{n_1}(CZ\tilde{Z})^{n_2}} \qquad (58)$$

$$\mathrm{tr}_Z\left[\rho_{(AX\tilde{X})B^{n_1}(CZ\tilde{Z})^{n_2}}\right] = \mathcal{Z}_{C\tilde{Z}}\otimes\rho_{(AX\tilde{X})B^{n_1}(CZ\tilde{Z})^{n_2-1}} \qquad (59)$$

for some operators $\mathcal{X}_{A\tilde{X}}$, and $\mathcal{Z}_{C\tilde{Z}}$. Then, there exist a probability distribution $\{p_i\}_{i\in I}$ and sets of quantum states $\{\sigma^i_{AX\tilde{X}}\}_{i\in I}$, $\{\omega^i_B\}_{i\in I}$ and $\{\tau^i_{CZ\tilde{Z}}\}_{i\in I}$ such that

$$\begin{aligned}
&\left\|\rho_{(AX\tilde{X})B(CZ\tilde{Z})} - \sum_{i\in I} p_i\,\sigma^i_{AX\tilde{X}}\otimes\omega^i_B\otimes\tau^i_{CZ\tilde{Z}}\right\|_1\\
&\leq \min\left\{18^{3/2}\sqrt{|ABC|}, 4|BC|\right\}\times\sqrt{4\ln 2}\left(\sqrt{\frac{\log|X|+8\log|B|}{n_2}}+\frac{\log|X|}{n_1}\right) \qquad (60)
\end{aligned}$$

with $\mathrm{tr}_X\left[\sigma^i_{AX\tilde{X}}\right] = \mathcal{X}_{A\tilde{X}}$ and $\mathrm{tr}_Z\left[\tau^i_{CZ\tilde{Z}}\right] = \mathcal{Z}_{C\tilde{Z}}$ for all $i\in I$.

The proof of Lemma 8 is similar to the one of Theorem 5 apart from the following two ingredients – leading to the tighter bound in Eq. (60) in comparison to Eq. (27):

- The partial trace constraints allow us to use a stronger bound on the conditional quantum mutual information in the proof of Lemma 6 (instead of Eq. (37)). Namely, for a quantum state ρ_{ABCD} satisfying $\mathrm{tr}_A[\rho_{ABCD}] = \rho_B\otimes\rho_{CD}$, we have that

$$I(AB:C|D)_\rho = I(B:C|D) + I(A:C|DB) \leq 2\log|A|. \qquad (61)$$

  Using this results in a better bound with $|X|$ instead of $|AX\tilde{X}|$ in the square root part of Eq. (60). Please see Section 4.2, especially Lemma 6 and Lemma 7, in the full version [19] for a more detailed discussion.

- As we remarked in Eq. (46) after Lemma 7, the dimension factor only comes from the measurements on the quantum systems. This is why there is no $|X\tilde{X}Z\tilde{Z}|$ contribution in the first part of Eq. (60).

3.2 Convergence of the hierarchy

Lemma 8 allows us to find an upper bound on the accuracy of the SDP relaxations in Eq. (14). We derive analytical bounds on the convergence speed of our SDP hierarchy in terms of the dimension $|T|$ and the size of the game.

▶ **Theorem 9.** *Let $sdp_n(V,\pi,T)$ be the n-th level SDP relaxation for the two-player free game with rule matrix V, probability distribution $\pi(q_1,q_2)=\pi_1(q_1)\pi_2(q_2)$, and quantum correlation of dimension $|T|^2$. Then, we have*

$$0 \le sdp_n(V,\pi,T) - \omega_{Q(T)}(V,\pi) \le O\left(|T|^6\sqrt{\frac{\log|T||A|}{n}}\right). \tag{62}$$

Hence, we have $\omega_{Q(T)}(V,\pi) = \lim_{n\to\infty} sdp_n(V,\pi,T)$.

Proof. Let $\rho_{A_1Q_1TA_2Q_2\hat{T}S\hat{S}}$ be the optimal state of the n-th level relaxation $\mathrm{sdp}_n(V,\pi,T)$. The state should be (n,n)-extendible since all feasible states must be (n,n)-extendible states satisfying the linear constraints. Then, we have

$$\begin{aligned}
&\mathrm{sdp}_n(V,\pi,T) = |T|^2\,\mathrm{tr}\left[\left(V_{A_1A_2Q_1Q_2}\otimes\Phi_{T\hat{T}|S\hat{S}}\right)\rho_{A_1Q_1TA_2Q_2\hat{T}S\hat{S}}\right]\\
&= |T|^2\,\mathrm{tr}\left[\left(V_{A_1A_2Q_1Q_2}\otimes\Phi_{T\hat{T}|S\hat{S}}\right)\left(\sum_i p_i\,\sigma^i_{A_1Q_1T}\otimes\omega^i_{A_2Q_2\hat{T}}\otimes\tau^i_{S\hat{S}}\right)\right]\\
&\quad + |T|^2\,\mathrm{tr}\left[\left(V_{A_1A_2Q_1Q_2}\otimes\Phi_{T\hat{T}|S\hat{S}}\right)\left(\rho_{A_1Q_1TA_2Q_2\hat{T}S\hat{S}} - \sum_i p_i\,\sigma^i_{A_1Q_1T}\otimes\omega^i_{A_2Q_2\hat{T}}\otimes\tau^i_{S\hat{S}}\right)\right]\\
&\le \omega_{Q(T)}(V,\pi)\\
&\quad + |T|^2\,\mathrm{tr}\left[\left(V_{A_1A_2Q_1Q_2}\otimes\Phi_{T\hat{T}|S\hat{S}}\right)\left(\rho_{A_1Q_1TA_2Q_2\hat{T}S\hat{S}} - \sum_i p_i\,\sigma^i_{A_1Q_1T}\otimes\omega^i_{A_2Q_2\hat{T}}\otimes\tau^i_{S\hat{S}}\right)\right],
\end{aligned} \tag{63}$$

where $\sum_i p_i\,\sigma^i_{A_1Q_1T}\otimes\omega^i_{A_2Q_2\hat{T}}\otimes\tau^i_{S\hat{S}}$ is one of the close separable states to $\rho_{A_1Q_1TA_2Q_2\hat{T}S\hat{S}}$ specified by Lemma 8. As $\mathrm{sdp}_n(V,\pi,T)$ is an upper bound for $\omega_{Q(T)}(V,\pi)$ we obtain

$$\begin{aligned}
&\left|\mathrm{sdp}_n(V,\pi,T) - \omega_{Q(T)}(V,\pi)\right|\\
&\quad\le |T|^2\left|\mathrm{tr}\left[\left(V_{A_1A_2Q_1Q_2}\otimes\Phi_{T\hat{T}|S\hat{S}}\right)\left(\rho_{A_1Q_1TA_2Q_2\hat{T}S\hat{S}} - \sum_i p_i\,\sigma^i_{A_1Q_1T}\otimes\omega^i_{A_2Q_2\hat{T}}\otimes\tau^i_{S\hat{S}}\right)\right]\right|\\
&\quad\le |T|^2\left\|V_{A_1A_2Q_1Q_2}\otimes\Phi_{T\hat{T}|S\hat{S}}\right\|_\infty\left\|\rho_{A_1Q_1TA_2Q_2\hat{T}S\hat{S}} - \sum_i p_i\,\sigma^i_{A_1Q_1T}\otimes\omega^i_{A_2Q_2\hat{T}}\otimes\tau^i_{S\hat{S}}\right\|_1\\
&\qquad\qquad\text{(by Hölder's inequality)}\\
&\quad= |T|^2\|V_{A_1A_2Q_1Q_2}\|_\infty\left\|\Phi_{T\hat{T}|S\hat{S}}\right\|_\infty\left\|\rho_{A_1Q_1TA_2Q_2\hat{T}S\hat{S}} - \sum_i p_i\,\sigma^i_{A_1Q_1T}\otimes\omega^i_{A_2Q_2\hat{T}}\otimes\tau^i_{S\hat{S}}\right\|_1
\end{aligned}$$

$$= |T|^4 \left\| \rho_{A_1Q_1TA_2Q_2\hat{T}S\hat{S}} - \sum_i p_i \, \sigma^i_{A_1Q_1T} \otimes \omega^i_{A_2Q_2\hat{T}} \otimes \tau^i_{S\hat{S}} \right\|_1$$

$$\left(\text{by } \|V_{A_1A_2Q_1Q_2}\|_\infty = 1, \ \left\|\Phi_{T\hat{T}|S\hat{S}}\right\|_\infty = |T|^2 \right)$$

$$\leq |T|^4 \left[18^{3/2} |T|^2 \left(\sqrt{2 \ln 2}\right) \left(\sqrt{\frac{\log |A_1| + 8 \log |S\hat{S}|}{n}} + \frac{\log |A_1|}{n} \right) \right] \qquad \text{(by Lemma 8)}$$

$$= 18^{3/2} |T|^6 \left(\sqrt{2 \ln 2}\right) \left(\sqrt{\frac{\log |A| + 16 \log |T|}{n}} + \frac{\log |A|}{n} \right). \tag{64}$$

Here, we set $A = T$, $X = A_1$, $\tilde{X} = Q_1$ $B = S\hat{S}$, $C = \hat{T}$, $Z = A_2$, and $\tilde{Z} = Q_2$ when we applied Lemma 8. ◀

It is worth noting that neither the PPT nor NPA constraints are used to derive this convergence speed.

Theorem 9 allows us to provide an upper bound on the computational complexity of calculating $\omega_{Q(T)}(V, \pi)$ for two-player free games. To achieve a constant error ϵ, it is sufficient to go up to the following level of the hierarchy:

$$\mathcal{O}\left(|T|^6 \sqrt{\frac{\log |TA|}{n}} \right) \leq \epsilon \qquad \Longleftrightarrow \qquad n \geq \mathcal{O}\left(|T|^{12} \frac{\log |TA|}{\epsilon^2} \right). \tag{65}$$

The resulting size of the program is stated in Eq. (1), where the dependence is quasi-polynomial in A and polynomial in Q. Our result is the quantum extension of the quasi-polynomial time approximation scheme for computing classical values $\omega_C(V, \pi)$ of two-player free games developed in [1, 9].

3.2.1 General games

We hitherto assume that the choice of questions for Alice and Bob is independent, i.e., $\pi(q_1, q_2) = \pi_1(q_1)\pi_2(q_2)$, which corresponds to free games. We can use the same protocol that we used for free games to derive upper bounds on the computational complexity of calculating $\omega_{Q(T)}(V, \pi)$ of general games, when $\pi(q_1, q_2) \neq \pi_1(q_1)\pi_2(q_2)$. The key difference is that for general games we absorb $\pi(q_1, q_2)$ into the rule matrix $V(a_1, a_2, q_1, q_2)$ instead of $E_{A_1Q_1T}$ and $D_{A_2Q_2\hat{T}}$ when we connect $\omega_{Q(T)}(V, \pi)$ to the quantum separability problem in Lemma 1. This leaves some additional factor $|Q_1||Q_2|$ in the objective function, which leads to a worse upper bound on the computational complexity. For a general two-player game with $|T|^2$-dimensional quantum correlation, we can compute additive ϵ-approximations of $\omega_{Q(T)}(V, \pi)$ with a semidefinite program of size

$$\exp\left(\mathcal{O}\left(\frac{|T|^{12} |Q|^4 \left(\log^2 |A||T| + \log |A||T| \log |Q| \right)}{\epsilon^2} \right) \right), \tag{66}$$

where $|A|$ and $|Q|$ are the number of possible answers and questions, respectively. The dependence is still quasi-polynomial in $|A|$, but exponential in $|Q|$ in contrast to the case of free games in Eq. (1). The detailed derivation can be found in Appendix C of the full version [19, Appendix C].

4 Conclusions

In this paper, we study the characterisation of quantum correlations of fixed dimension and, more specifically, provide a converging hierarchy of SDP relaxations with improved analytical convergence speed for the set of fixed-dimensional quantum correlations. This is done by employing a variant of the quantum separability problem and multipartite quantum de Finetti theorems with additional linear constraints. Our result leads to an upper bound on the computational complexity of additive ϵ-approximation for $\omega_{Q(T)}(V,\pi)$ of two-player free games with $T \times T$-dimensional quantum correlation.

We conclude with a few remarks on possible future studies. Firstly, for a given level n, $\mathrm{sdp}_n(V,\pi,T)$ has a relatively large-sized optimisation variable. One possible way to improve this aspect is to exploit the symmetry embedded in the program to reduce the size of the optimisation variable. We could employ some existing symmetry-finding programs such as [31] to achieve this. Secondly, it is still not certain whether the T-dependence in Eq. (1) is optimal. In the classical limit ($T = 1$), our result matches the best-known classical result for free games in terms of A and Q – which also has a matching hardness result [1]. This implies that the dependence on A and Q in Eq. (1) is optimal, but there could be more efficient approximation algorithm in terms of T-dependence. For example, one could explore ϵ-net based methods as in [7, 32].

5 Proof of Lemma 7

In this section, we prove the second part of Lemma 7 which states that for a traceless Hermitian operator γ_{AB} on $\mathcal{H}_{AB}$, there exists a measurement $\mathcal{M}_B$ on $\mathcal{H}_B$ with at most $|B|^6$ outcomes such that $\|(\mathcal{I}_A \otimes \mathcal{M}_B)(\gamma_{AB})\|_1 \geq \frac{1}{2|B|}\|\gamma_{AB}\|_1$. The proof is inspired by [22, Theorem 16].

Proof of the second part of Lemma 7. Let us start with the maximally entangled state

$$\Phi_{A'|B'} = |\Phi\rangle\langle\Phi|_{A'|B'} \text{ where } |\Phi\rangle_{A'|B'} = \frac{1}{|A'||B'|}\sum_i |i\rangle_{A'}|i\rangle_{B'}, \text{ and } |A'| = |B'|. \tag{67}$$

We can create a separable state $\omega_{A'B'}$ by mixing $\Phi_{A'|B'}$ with another separable state $\sigma_{A'B'} = \frac{\mathbb{I}_{A'B'} - \Phi_{A'|B'}}{|B'|^2-1}$ as

$$\omega_{A'B'} = \frac{1}{|B'|}\Phi_{A'B'} + \frac{|B'|-1}{|B'|}\sigma_{A'B'} \in \mathrm{SEP(A':B')}, \tag{68}$$

where SEP(A':B') denotes the set of separable states with respect to the bipartition $A'|B'$. Hence, we can write $\omega_{A'B'} = \sum_i p_i \omega^i_{A'} \otimes \omega^i_{B'}$ for some probability distribution $\{p_i\}_i$ and states $\{\omega^i_{A'}\}_i$ and $\{\omega^i_{B'}\}_i$ with at most $|A'B'|^2$ elements [18]. Next, we define a measurement $\mathcal{M}_B$ with operators $\{\tilde{M}_B(i,k)\}_{i,k}$, as well as a set of measurements $\{\mathcal{M}^{i,k}_A\}_{i,k}$ with operators $\{\tilde{M}^{i,k}_A(j)\}_j$ as

$$\tilde{M}_B(i,k) = \mathrm{tr}_{B'}\left[p_i U^\dagger_B(k)\sqrt{\omega^i_{B'}}\Phi_{BB'}\sqrt{\omega^i_{B'}}U_B(k)\right] \quad \text{and} \tag{69}$$

$$\tilde{M}^{i,k}_A(j) = \mathrm{tr}_{A'}\left[\sqrt{\omega^i_{A'}}U^\dagger_{A'}(k)N_{AA'}(j)U_{A'}(k)\sqrt{\omega^i_{A'}}\right], \tag{70}$$

where $U(k)$ denote generalised Pauli operators, $\omega^i_{A'}$ and $\omega^i_{B'}$ are the elements of the decomposition of $\omega_{A'B'}$, and $\{N_{AA'}(j)\}_j$ are measurement operators defined later. We can check that both definitions indeed correspond to valid measurements:

$$\sum_{i,k} \tilde{M}_B(i,k) = \mathbb{I}_B,\ \sum_j \tilde{M}_A^{i,k}(j) = \mathbb{I}_A, \text{ and } \tilde{M}_B(i,k), \tilde{M}_A^{i,k}(j) \geq 0 \ \ \forall i,k,j. \tag{71}$$

The goal is to show that $\mathcal{M}_B$ defined in Eq. (69) gives rise to Eq. (45). Before showing that, however, it is helpful to understand where these measurements came from. They are related to the quantum teleportation protocol [5]. Without loss of generality, let us assume that $|A| \geq |B| = |A'| = |B'|$. Then, the quantum teleportation protocol from B to A is a quantum channel defined as [5]

$$\tau_{ABA'B'\to AA'}(\cdot) = \sum_{k=1}^{|B|^2} U_{A'}(k) \operatorname{tr}_{BB'} \left[(\cdot) \left(\mathbb{I}_{AA'} \otimes U_B(k) \Phi_{BB'} U_B^\dagger(k) \right) \right] U_{A'}^\dagger(k). \tag{72}$$

For a traceless Hermitian operator γ_{AB}, we then consider

$$\left\| \tau_{ABA'B'\to AA'} \left(\gamma_{AB} \otimes \omega_{A'B'} \right) \right\|_1 = \sum_j \left| \operatorname{tr} \left[N_{AA'}(j) \left(\tau_{ABA'B'\to AA'} \left(\gamma_{AB} \otimes \omega_{A'B'} \right) \right) \right] \right|, \tag{73}$$

where we used the expression $\|X_A\|_1 = \max_{\{M_A(i)\}_i} \sum_i |\operatorname{tr}[M_A(i) X_A]|$ for the trace norm with corresponding arg max $\{N_{AA'}(j)\}_j$ to be used in Eq. (70). Then, we have

$$\begin{aligned}
&\left\| \tau_{ABA'B'\to AA'} \left(\gamma_{AB} \otimes \omega_{A'B'} \right) \right\|_1 \\
&= \sum_j \left| \sum_k \operatorname{tr} \left[N_{AA'}(j) \left(U_{A'}(k) \operatorname{tr}_{BB'} \left[\left(\gamma_{AB} \otimes \omega_{A'B'} \right) \left(\mathbb{I}_{AA'} \otimes U_B(k) \Phi_{BB'} U_B^\dagger(k) \right) \right] U_{A'}^\dagger(k) \right) \right] \right| \\
&= \sum_j \left| \sum_k \operatorname{tr} \left[\left(U_{A'}^\dagger(k) N_{AA'}(j) U_{A'}(k) \otimes \mathbb{I}_{BB'} \right) \left(\left(\gamma_{AB} \otimes \omega_{A'B'} \right) \left(\mathbb{I}_{AA'} \otimes U_B(k) \Phi_{BB'} U_B^\dagger(k) \right) \right) \right] \right| \\
&= \sum_j \left| \sum_k \operatorname{tr} \left[\left(U_{A'}^\dagger(k) N_{AA'}(j) U_{A'}(k) \otimes U_B^\dagger(k) \Phi_{BB'} U_B(k) \right) \left(\gamma_{AB} \otimes \left(\sum_i p_i \omega_{A'}^i \otimes \omega_{B'}^i \right) \right) \right] \right| \\
&= \sum_j \left| \sum_{i,k} \operatorname{tr} \left[\left(\left(\sqrt{\omega_{A'}^i} U_{A'}^\dagger(k) N_{AA'}(j) U_{A'}(k) \sqrt{\omega_{A'}^i} \right) \right. \right. \right. \\
&\qquad \left. \left. \left. \otimes \left(p_i U_B^\dagger(k) \sqrt{\omega_{B'}^i} \Phi_{BB'} \sqrt{\omega_{B'}^i} U_B(k) \right) \right) \left(\gamma_{AB} \otimes \mathbb{I}_{A'B'} \right) \right] \right| \\
&= \sum_j \left| \sum_{i,k} \operatorname{tr} \left[\gamma_{AB} \left(\tilde{M}_A^{i,k}(j) \otimes \tilde{M}_B(i,k) \right) \right] \right|.
\end{aligned} \tag{74}$$

The measurement $\mathcal{M}_B$ defined in Eq. (69) now gives rise to

$$\left\| \left(\mathcal{I}_A \otimes \mathcal{M}_B \right) \left(\gamma_{AB} \right) \right\|_1 \tag{75}$$

$$= \sum_{i,k} \left\| \operatorname{tr}_B \left[\left(\mathbb{I}_A \otimes \tilde{M}_B(i,k) \right) \gamma_{AB} \right] \right\|_1 \tag{76}$$

$$= \sum_{i,k} \max_{\{M_A^{i,k}(j)\}_j} \sum_j \left| \operatorname{tr} \left[\left(M_A^{i,k}(j) \otimes \tilde{M}_B(i,k) \right) \gamma_{AB} \right] \right| \tag{77}$$

$$\geq \sum_{i,k} \sum_j \left| \operatorname{tr} \left[\left(\tilde{M}_A^{i,k}(j) \otimes \tilde{M}_B(i,k) \right) \gamma_{AB} \right] \right| \tag{78}$$

$$\geq \sum_j \left| \sum_{i,k} \operatorname{tr} \left[\left(\tilde{M}_A^{i,k}(j) \otimes \tilde{M}_B(i,k) \right) \gamma_{AB} \right] \right| \tag{79}$$

$$= \left\|\tau_{ABA'B'\to AA'}\left(\gamma_{AB}\otimes\omega_{A'B'}\right)\right\|_1 \qquad \text{(by Eq. (74))} \tag{80}$$

$$= \left\|\tau_{ABA'B'\to AA'}\left(\gamma_{AB}\otimes\left(\frac{1}{|B|}\Phi_{A'B'}+\frac{|B|-1}{|B|}\sigma_{A'B'}\right)\right)\right\|_1 \tag{81}$$

$$= \left\|\frac{1}{|B|}\tau_{ABA'B'\to AA'}\left(\gamma_{AB}\otimes\Phi_{A'B'}\right)+\frac{|B|-1}{|B|}\tau_{ABA'B'\to AA'}\left(\gamma_{AB}\otimes\sigma_{A'B'}\right)\right\|_1 \tag{82}$$

$$\geq \frac{1}{|B|}\left\|\tau_{ABA'B'\to AA'}\left(\gamma_{AB}\otimes\Phi_{A'B'}\right)\right\|_1 - \left\|\tau_{ABA'B'\to AA'}\left(\gamma_{AB}\otimes\left(\frac{|B|-1}{|B|}\sigma_{A'B'}\right)\right)\right\|_1, \tag{83}$$

where in the third line we substituted the measurement operators $\{\tilde{M}_A^{i,k}(j)\}_j$ instead of the maximisation, and in the last line we used the reverse triangular inequality. Note that the first term in the last line is equivalent to $\|\gamma_{AB}\|_1$ since $\Phi_{A'B'}$ is the maximally entangled state. Let us investigate the second term more closely. We have a chain of elementary implications

$$\frac{|B|-1}{|B|}\sigma_{A'B'} \leq \frac{|B|-1}{|B|}\sigma_{A'B'}+\frac{1}{|B|}\Phi_{A'B'} = \omega_{A'B'}$$

$$\gamma_{AB}\otimes\frac{|B|-1}{|B|}\sigma_{A'B'} \leq \gamma_{AB}\otimes\omega_{A'B'}$$

$$\left\|\tau_{ABA'B'\to AA'}\left(\gamma_{AB}\otimes\left(\frac{|B|-1}{|B|}\sigma_{A'B'}\right)\right)\right\|_1 \leq \left\|\tau_{ABA'B'\to AA'}\left(\gamma_{AB}\otimes\omega_{A'B'}\right)\right\|_1$$

$$\left\|\tau_{ABA'B'\to AA'}\left(\gamma_{AB}\otimes\left(\frac{|B|-1}{|B|}\sigma_{A'B'}\right)\right)\right\|_1 \leq \sum_j\left|\sum_{i,k}\operatorname{tr}\left[\gamma_{AB}\left(\tilde{M}_A^{i,k}(j)\otimes\tilde{M}_B(i,k)\right)\right]\right| \qquad \text{(by Eq. (74))}$$

$$\left\|\tau_{ABA'B'\to AA'}\left(\gamma_{AB}\otimes\left(\frac{|B|-1}{|B|}\sigma_{A'B'}\right)\right)\right\|_1 \leq \left\|\left(\mathcal{I}_A\otimes\mathcal{M}_B\right)\left(\gamma_{AB}\right)\right\|_1 \qquad \text{(by Eq. (79))} \tag{84}$$

and substituting this into Eq. (83) yields the claim

$$\left\|\left(\mathcal{I}_A\otimes\mathcal{M}_B\right)\left(\gamma_{AB}\right)\right\|_1 \geq \frac{1}{|B|}\left\|\gamma_{AB}\right\|_1 - \left\|\left(\mathcal{I}_A\otimes\mathcal{M}_B\right)\left(\gamma_{AB}\right)\right\|_1. \tag{85}$$

It remains to quantify the number of measurement outcomes of $\mathcal{M}_B$ with measurement operators $\{\tilde{M}_B(i,k)\}_{i,k}$ defined in Eq. (70). The index i came from the number of elements in the separable state $\omega_{A'B'}$, which is at most $|A'B'|^2 = |B|^4$, and the index k came from the number of generalised Pauli operators, which is $|B|^2$. Therefore, the number of outcomes is at most $|B|^6$. ◀

References

1 Scott Aaronson, Russell Impagliazzo, and Dana Moshkovitz. AM with multiple Merlins. In *IEEE 29th Conference on Computational Complexity*, pages 44–55, 2014.

2 Sanjeev Arora, Carsten Lund, Rajeev Motwani, Madhu Sudan, and Mario Szegedy. Proof verification and the hardness of approximation problems. *Journal of the ACM*, 45(3):501–555, 1998.

3 Sanjeev Arora and Shmuel Safra. Probabilistic checking of proofs: A new characterization of NP. *Journal of the ACM*, 45(1):70–122, 1998.

4 John S Bell. On the Einstein Podolsky Rosen paradox. *Physics Physique Fizika*, 1(3):195, 1964.

5 Charles H Bennett, Gilles Brassard, Claude Crépeau, Richard Jozsa, Asher Peres, and William K Wootters. Teleporting an unknown quantum state via dual classical and Einstein-Podolsky-Rosen channels. *Physical Review Letters*, 70(13):1895, 1993.

6 Mario Berta, Francesco Borderi, Omar Fawzi, and Volkher B Scholz. Semidefinite programming hierarchies for constrained bilinear optimization. *Mathematical Programming*, pages 1–49, 2021.

7 Fernando GSL Brandão and Aram W Harrow. Estimating operator norms using covering nets. *arXiv:1509.05065*, 2015. `arXiv:1509.05065`.

8 Fernando GSL Brandão and Aram W Harrow. Product-state approximations to quantum states. *Communications in Mathematical Physics*, 342(1):47–80, 2016.

9 Fernando GSL Brandão and Aram W Harrow. Quantum de Finetti theorems under local measurements with applications. *Communications in Mathematical Physics*, 353(2):469–506, 2017.

10 Richard Cleve, Peter Høyer, Benjamin Toner, and John Watrous. Consequences and limits of nonlocal strategies. In *Proceedings 19th IEEE Annual Conference on Computational Complexity*, pages 236–249, 2004.

11 Andrew C Doherty, Pablo A Parrilo, and Federico M Spedalieri. Distinguishing separable and entangled states. *Physical Review Letters*, 88(18):187904, 2002.

12 Andrew C Doherty, Pablo A Parrilo, and Federico M Spedalieri. Complete family of separability criteria. *Physical Review A*, 69(2):022308, 2004.

13 Andrew C Doherty, Pablo A Parrilo, and Federico M Spedalieri. Detecting multipartite entanglement. *Physical Review A*, 71(3):032333, 2005.

14 Mark Fannes, John T Lewis, and André Verbeure. Symmetric states of composite systems. *Letters in Mathematical Physics*, 15(3):255–260, 1988.

15 Rodrigo Gallego, Nicolas Brunner, Christopher Hadley, and Antonio Acín. Device-independent tests of classical and quantum dimensions. *Physical Review Letters*, 105(23):230501, 2010.

16 Sevag Gharibian. Strong NP-hardness of the Quantum Separability Problem. *Quantum Information and Computation*, 10(3&4):343–360, 2010.

17 Leonid Gurvits. Classical deterministic complexity of Edmonds' problem and quantum entanglement. In *Proceedings of the thirty-fifth annual ACM symposium on theory of computing*, pages 10–19, 2003.

18 Pawel Horodecki. Separability criterion and inseparable mixed states with positive partial transposition. *Physics Letters A*, 232:333, 1997.

19 Hyejung H Jee, Carlo Sparaciari, Omar Fawzi, and Mario Berta. Characterising quantum correlations of fixed dimension. *arXiv preprint*, 2020. `arXiv:2005.08883`.

20 Zhengfeng Ji, Anand Natarajan, Thomas Vidick, John Wright, and Henry Yuen. MIP*= RE. *arXiv:2001.04383*, 2020. `arXiv:2001.04383`.

21 Julia Kempe, Oded Regev, and Ben Toner. Unique games with entangled provers are easy. *SIAM Journal on Computing*, 39(7):3207–3229, 2010.

22 Ludovico Lami, Carlos Palazuelos, and Andreas Winter. Ultimate data hiding in quantum mechanics and beyond. *Communications in Mathematical Physics*, 361(2):661–708, 2018.

23 Göran Lindblad. Entropy, information and quantum measurements. *Communications in Mathematical Physics*, 33(4):305–322, 1973.

24 William Matthews, Stephanie Wehner, and Andreas Winter. Distinguishability of quantum states under restricted families of measurements with an application to quantum data hiding. *Communications in Mathematical Physics*, 291(3):813–843, 2009.

25 Miguel Navascués, Gonzalo de la Torre, and Tamás Vértesi. Characterization of quantum correlations with local dimension constraints and its device-independent applications. *Physical Review X*, 4(1):011011, 2014.

26 Miguel Navascués, Adrien Feix, Mateus Araújo, and Tamás Vértesi. Characterizing finite-dimensional quantum behavior. *Physical Review A*, 92(4):042117, 2015.

27 Miguel Navascués, Stefano Pironio, and Antonio Acín. A convergent hierarchy of semidefinite programs characterizing the set of quantum correlations. *New Journal of Physics*, 10(7):073013, 2008.

28 Miguel Navascués and Tamás Vértesi. Bounding the set of finite dimensional quantum correlations. *Physical Review Letters*, 115(2):020501, 2015.

29 Stefano Pironio, Miguel Navascués, and Antonio Acín. Convergent relaxations of polynomial optimization problems with noncommuting variables. *SIAM Journal on Optimization*, 20(5):2157–2180, 2010.

30 GA Raggio and Reinhard F Werner. Quantum statistical mechanics of general mean field systems. *Helvetica Acta Physica*, 62:980, 1989.

31 Denis Rosset. SymDPoly: symmetry-adapted moment relaxations for noncommutative polynomial optimization. *arXiv:1808.09598*, 2018. `arXiv:1808.09598`.

32 Yaoyun Shi and Xiaodi Wu. Epsilon-net method for optimizations over separable states. *Theoretical Computer Science*, 598:51–63, 2015.

33 Barbara M Terhal. Is entanglement monogamous? *IBM Journal of Research and Development*, 48(1):71–78, 2004.

34 Xiao-Dong Yu, Timo Simnacher, H Chau Nguyen, and Otfried Gühne. Quantum-inspired hierarchy for rank-constrained optimization. *arXiv preprint*, 2020. `arXiv:2012.00554`.

35 Xiao-Dong Yu, Timo Simnacher, Nikolai Wyderka, H. Chau Nguyen, and Otfried Gühne. A complete hierarchy for the pure state marginal problem in quantum mechanics. *Nature Communications*, 12(1):1012, 2021.

Fully Dynamic Algorithms for Minimum Weight Cycle and Related Problems

Adam Karczmarz
Institute of Informatics, University of Warsaw, Poland

Abstract

We consider the directed minimum weight cycle problem in the fully dynamic setting. To the best of our knowledge, so far no fully dynamic algorithms have been designed specifically for the minimum weight cycle problem in general digraphs. One can achieve $\widetilde{O}(n^2)$ amortized update time by simply invoking the fully dynamic APSP algorithm of Demetrescu and Italiano [J. ACM '04]. This bound, however, yields no improvement over the trivial recompute-from-scratch algorithm for sparse graphs.

Our first contribution is a very simple deterministic $(1+\epsilon)$-approximate algorithm supporting vertex updates (i.e., changing all edges incident to a specified vertex) in conditionally near-optimal $\widetilde{O}(m \log(W)/\epsilon)$ amortized time for digraphs with real edge weights in $[1, W]$. Using known techniques, the algorithm can be implemented on planar graphs and also gives some new sublinear fully dynamic algorithms maintaining approximate cuts and flows in planar digraphs.

Additionally, we show a Monte Carlo randomized exact fully dynamic minimum weight cycle algorithm with $\widetilde{O}(mn^{2/3})$ worst-case update that works for real edge weights. To this end, we generalize the exact fully dynamic APSP data structure of Abraham et al. [SODA'17] to solve the *multiple-pairs shortest paths* problem, where one is interested in computing distances for some k (instead of all n^2) fixed source-target pairs after each update. We show that in such a scenario, $\widetilde{O}((m+k)n^{2/3})$ worst-case update time is possible.

2012 ACM Subject Classification Theory of computation → Dynamic graph algorithms; Theory of computation → Shortest paths

Keywords and phrases Dynamic graph algorithms, minimum weight cycle, dynamic shortest paths

Digital Object Identifier 10.4230/LIPIcs.ICALP.2021.83

Category Track A: Algorithms, Complexity and Games

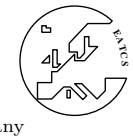
Related Version *Full Version*: `https://arxiv.org/abs/2106.11744`

Funding *Adam Karczmarz*: supported by ERC Consolidator Grant 772346 TUgbOAT and by the Foundation for Polish Science (FNP) via the START programme.

1 Introduction

The all-pairs shortest paths problem (APSP) is one of the most fundamental graph problems. Given a real-weighted *directed* graph G with n vertices, the goal is to compute the distance matrix between all pairs of vertices u, v in G. APSP can be computed in $\widetilde{O}(nm)$ time [29, 41], which is clearly near-optimal for sparse graphs (since the output consists of n^2 numbers), but is also conjectured to be optimal for the entire range of possible graph sparsities. Some of the other core directed graph problems such as computing the diameter, the radius, or the minimum weight cycle[1] can be trivially reduced to APSP in $O(n^2)$ time by simply inspecting the entries of the distance matrix. In fact, as shown by Vassilevska Williams and Williams [47], for dense graphs APSP is known to be subcubically equivalent to many problems which look easier at first sight, especially because their output is just a single

[1] Also called the *girth*, or the *weighted girth* of a digraph. For simplicity, in this paper we very often use *minimum weight cycle* to refer to the *length* of such a cycle rather than to the actual cycle. Moreover, throughout this paper, our focus is on computing/maintaining that length instead of the actual cycle. The obtained algorithms, however, can be easily extended to return a sought cycle with no additional asymptotic overhead.



number (as opposed to n^2 numbers in APSP). These include e.g., the radius, the minimum weight cycle, and the second shortest simple s, t path problems. For all these problems, just like for APSP, the best known algorithms run in $\widetilde{O}(nm)$ time. Lincoln et al. [36] gave some compelling reasons why improving upon this bound may also be impossible.

In this paper, our focus is on *fully dynamic* graph algorithms. Fully dynamic graph algorithms allow updating the graph under both edge insertions and deletions, as opposed to *partially dynamic* algorithms that allow either only insertions (*incremental* setting) or only deletions (*decremental* setting). Fully dynamic APSP has been widely studied in the past. Demetrescu and Italiano [18] showed that the distance matrix can be *explicitly* maintained in $\widetilde{O}(n^2)$ *amortized* time under *vertex updates* which are allowed to change all edges incident to a single vertex at once. Thorup [45] simplified and slightly improved their algorithm. Clearly, if the algorithm is required to maintain all distances explicitly, one cannot break through the $O(n^2)$ time barrier since even a *single* edge update may change *all* the n^2 pairwise distances. Much of the work in this topic [2, 24, 46] has been devoted to obtaining good *worst-case* bounds on the time needed to recompute the distance matrix and it is known that $\widetilde{O}(n^{2+2/3})$ worst-case update time is possible [2, 24]. Interestingly, none of the known fully dynamic algorithms for *real-weighted* dynamic APSP has $o(n^2)$ update time and a non-trivial query procedure running in $o(m)$ time. Such an algorithm, with $\widetilde{O}(m\sqrt{n})$ amortized update time and $\widetilde{O}(n^{3/4})$ query time, has so far been only described for sparse enough unweighted graphs by Roditty and Zwick [43].

The algorithm of Demetrescu and Italiano [18] immediately implies $\widetilde{O}(n^2)$ amortized update bound for fully dynamic variants of all the most fundamental problems "trivially reducible" to APSP – the aforementioned diameter, radius, or minimum weight cycle. Surprisingly, as shown in [3], such an update bound is likely to be the best possible for maintaining both the diameter and the radius (conditionally on so-called Strong Exponential Time- and Hitting Set hypotheses [1, 26]), even if the graph remains sparse at all times and $(3/2 - \epsilon)$-approximation is allowed.

It is thus natural to ask whether there exist fully dynamic algorithms for the *minimum weight cycle problem* that improve upon the reduction to fully dynamic APSP for sparse graphs, possibly allowing some small multiplicative approximation. The fundamental difference between the minimum weight cycle and diameter/radius problems is that the trivial reduction of minimum weight cycle requires reading only m entries of the distance matrix, as opposed to all n^2 in the case of radius and diameter. As a result, by using the aforementioned fully dynamic algorithm of Roditty and Zwick [43], one immediately gets $\widetilde{O}(mn^{3/4})$ amortized update bound but merely for *unweighted* graphs. Note that this bound is always better than recompute-from-scratch, and is truly subquadratic for sparse graphs. It is however not clear whether such a bound can be obtained for real-weighted graphs, nor whether a much better bound is attainable if we allow approximation.

Motivated by the above, in this paper we initiate the study of the directed minimum weight cycle problem in the fully dynamic setting. To the best of our knowledge, this problem has not been explicitly studied in the literature before. It is worth noting, however, that a non-trivial fully dynamic algorithm has been shown for *undirected planar* graphs [37].

1.1 Our results

A fully dynamic approximate minimum weight cycle algorithm. Our first contribution is a simple deterministic fully dynamic algorithm maintaining a $(1 + \epsilon)$-approximation of the minimum weight $\phi(G)$ of a cycle in a real-weighted directed graph G. If G has a negative

cycle, then we define $\phi(G) = -\infty$, thus allowing the sought cycle to be non-simple. Note that if we wanted the minimum weight cycle to be simple and simultaneously allowed negative edge weights, the problem would become NP-hard via a reduction from Hamiltonian cycle.

▶ **Theorem 1.** *Let G be an initially empty fully dynamic real-weighted digraph such that the weight of each positive weight cycle in G always belongs to the interval $[c, C]$, $c, C \in \mathbb{R}$.*

There exists an algorithm maintaining an estimate ϕ' satisfying $\phi(G) \leq \phi' \leq (1+\epsilon)\phi(G)$ under vertex updates to G with amortized update time $O((m + n \log n) \cdot \log(C/c)/\epsilon)$.

By Theorem 1, a simpler amortized update time bound of $O((m + n \log n) \cdot \log(nW)/\epsilon)$ for the fully dynamic $(1+\epsilon)$-approximate minimum weight cycle problem can be obtained in two special cases:

- if G has real-weights in $\{0\} \cup [1, W]$,
- if G has integer weights in $(-\infty, W]$.

Via known conditional lower bounds on the static approximate minimum weight cycle problem, the update time bound in Theorem 1 – as a function of m *alone* – is near-optimal for both vertex and edge updates if we allow approximation factor less than 2 and $O(m^{2-\delta})$ preprocessing time (for some $\delta > 0$). Indeed, Dalirrooyfard and Vassilevska Williams [17] proved that under so-called k-Cycle hypothesis [3], one cannot approximate the minimum weight cycle within factor less than 2 in $O(m^{2-\delta})$ time, for any $\delta > 0$. Clearly, if there was, say, a dynamic 3/2-approximate minimum weight cycle algorithm with $O(m^{2-\delta})$ preprocessing time, $O(m^{1-\delta})$ update time, and the same interface as our algorithm, m edge/vertex updates would be sufficient to obtain a static 3/2-approximate minimum weight cycle algorithm running in $O(m^{2-\delta})$ time. This would refute the k-Cycle hypothesis.

Observe that the $\Omega(m^{2-o(1)})$ conditional lower bound [17] (which implies that the $\Omega(mn^{1-o(1)})$ bound holds for *some* sparsity m) on the complexity of static approximate minimum weight cycle problem does not rule out dynamic vertex update bounds of the form $\widetilde{O}(n^{\alpha} \cdot m^{1-\alpha})$ for some $\alpha \in (0, 1]$ or $\widetilde{O}(m^{1+\beta}/n^{2\beta})$ for some $\beta \in (0, 1/2]$, e.g., $\widetilde{O}(n)$, $\widetilde{O}(\sqrt{nm})$, or $\widetilde{O}(m^{3/2}/n)$. However, if we limit ourselves to *"combinatorial"* algorithms that do not rely on fast matrix multiplication, such $O(m^{1-\epsilon})$ bounds are ruled out for infinitely many sparsities of the form $m = \Theta(n^{1+2/(k-1)})$, where $k \geq 3$ is an odd integer [17, 36].

We stress that the aforementioned static conditional lower bounds do not rule out $\widetilde{O}(n)$ or even $\widetilde{O}(\sqrt{nm})$ amortized update time in the *edge update* model. In this case, for similar reasons, only combinatorial approximate algorithms with amortized update time that is sublinear in n for many sparsities, e.g., $\widetilde{O}(m^{\beta} \cdot n^{1-2\beta})$ for $\beta \in (0, 1/2]$, are unlikely to exist.

Fully dynamic cycles, flows, and cuts in planar graph. Interestingly, if we limit our attention to the case of single edge updates (as opposed to vertex updates) and real weights in $\{0\} \cup [1, W]$, the amortized update cost of the data structure of Theorem 1 can always be charged to the cost of performing a single edge update plus a single distance query on $O(\log(nW)/\epsilon)$ fully dynamic *exact* distance oracles, each maintaining some subgraph of G. For general digraphs, this amounts to running Dijkstra's algorithm in each of these subgraphs since no non-trivial fully dynamic distance oracles with both update and query time $o(m)$ are known. However, such dynamic distance oracles are well-known to exist for planar digraphs [21, 31, 34] which immediately leads to the following result.

▶ **Theorem 2.** *Let G be a planar digraph G with real weights in $\{0\} \cup [1, W]$. There exists an algorithm maintaining an $(1+\epsilon)$-approximate estimate of $\phi(G)$ under planarity preserving edge insertions and deletions with amortized update time $\widetilde{O}(n^{2/3} \log(W)/\epsilon)$.*

Previously, no sublinear fully dynamic algorithm for minimum weight cycle in planar directed graphs has been described. An *exact* algorithm for planar *undirected* graphs with $\widetilde{O}(n^{5/6})$ update time was given by Łącki and Sankowski [37].

There is a well-known correspondence between simple cuts in an undirected plane graph G, and simple cycles in its dual G^*. The correspondence, in a way, extends to *directed* planar graphs (see e.g. [35, 38]). Nevertheless, currently the best known min s,t-cut algorithms in planar digraphs [9, 19] are less efficient and use entirely different techniques than their counterparts for planar undirected graphs [28]. Generally speaking, for cut/flow applications, undirected planar graphs proved much more friendly to work with (see e.g., the discussion in [19] or [38]). As an example of this phenomenon, an *exact* fully dynamic max s,t-flow *oracle* (accepting s,t as query parameters) with $\widetilde{O}(n^{2/3})$ update and query time exists for undirected plane graphs [28], whereas no such dynamic algorithm has been described for *directed* plane graphs, even allowing approximation and just a single fixed source-sink pair.

It is known that in a plane digraph G, an s,t-flow of value f can be routed iff the dual $G^*_{s,t,f}$ of a certain augmentation of $G_{s,t,f}$ depending on s,t and f contains no negative cycles [19, 30, 39]. Roughly speaking, since the algorithm of Theorem 2 supports negative weights, by running it on $G^*_{s,t,f}$ for $O(\log(nW)/\epsilon)$ distinct values of f, we obtain the following.

▶ **Theorem 3.** *Let G be a plane embedded digraph with real edge capacities in $\{0\} \cup [1, W]$ and a* fixed *source/sink pair s,t. There exists an algorithm maintaining a $(1-\epsilon)$-approximate estimate of the value of maximum s,t-flow in G under* embedding preserving *edge insertions and deletions with $\widetilde{O}(n^{2/3}\log(W)/\epsilon)$ amortized update time.*

To the best of our knowledge, the above constitutes the first known fully dynamic maximum s,t-flow algorithm for plane directed graphs with a sublinear update time bound.

Exact fully dynamic minimum weight cycle and MPSP. Finally, we consider maintaining the minimum weight cycle *exactly* in a fully dynamic *real-weighted* digraph. We show:

▶ **Theorem 4.** *Let G be a real-weighted digraph. There exists a Monte Carlo randomized fully dynamic algorithm maintaining $\phi(G)$ under vertex updates with $O((m+n\log n)n^{2/3}\log^{4/3} n)$ worst-case update time. The answers produced are correct with high probability.*[2]

Note that for sparse graphs, Theorem 4 allows recomputing the minimum weight cycle in $\widetilde{O}(n^{5/3})$ time, i.e., polynomially faster than recompute-from-scratch and the dynamic algorithm of Demetrescu and Italiano [18]. However, observe that [18] yields a better amortized update bound for $m = \omega(n^{4/3})$.

In order to obtain Theorem 4, we generalize the fully dynamic APSP algorithm of Abraham et al. [2] in a non-trivial way to solve what we call the *multiple pairs shortest paths* problem (*MPSP*). In the MPSP problem, which may be of independent interest, one requires to maintain only k fixed entries of the distance matrix, i.e., after each update we are interested in distances between some source-target pairs (s_i, t_i) for $i = 1, \ldots, k$. Recall that the minimum weight cycle of a directed graph can be computed by inspecting distances for m source-target pairs. We obtain the following bound for the fully dynamic MPSP problem.

▶ **Theorem 5.** *Let G be a real-weighted digraph. There exists a Monte Carlo randomized fully dynamic MPSP data structure supporting vertex updates with $O((m+n\log n+k)n^{2/3}\log^{4/3} n)$ worst-case update time. The answers produced are correct with high probability.*

[2] That is, with probability at least $1 - 1/n^c$ for any chosen constant $c \geq 1$.

Note that the aforementioned data structure of Roditty and Zwick [43] trivially implies an MPSP data structure for *unweighted* digraphs with $\widetilde{O}(m\sqrt{n} + kn^{3/4})$ amortized update bound. Our result shows that a better (even worst-case) bound for (even real-weighted) sparse graphs can be achieved if the set of source-target pairs is fixed throughout.

Actually, just as the worst-case update time of the data structure of Abraham et al. [2] can be very easily improved to $\widetilde{O}(n^{2.5})$ for *unweighted* graphs [2, Section 4.2], an unweighted variant of our MPSP data structure has $\widetilde{O}((m+k)\sqrt{n})$ worst-case update time.

Interestingly, it seems that the other known approaches to fully dynamic APSP in real-weighted graphs [18, 23, 45], if adjusted, cannot easily yield subquadratic (in n) update times for "sparse" instances of MPSP where $m, k = O(n)$. This is because they all reconstruct shortest paths in a hierarchical manner, by inductively stitching [23] or extending [18, 45] paths recomputed earlier in the process. Even though the number of input source-target pairs of interest may be small, these may require answers for $\Theta(n^2)$ distinct source-target pairs at lower levels of the hierarchy. The data structure of Abraham et al. [2], on the contrary, does not use a hierarchical approach and can be thought as using a single "stitching layer".

Since the algorithm behind Theorem 4 (Theorem 5) is exact, the maintained information, i.e., the minimum weight of a cycle (the entries of the distance matrix of interest, resp.) is unique. Therefore, if we are interested in maintaining the corresponding weight (distances, resp.) only, the bounds in Theorems 4 and 5 hold against an adaptive adversary. However, if we are required to output some actual minimum weight cycle (edges on some of the desired shortest paths, resp.) we have to assume an oblivious adversary.[3]

1.2 Related work

Computing minimum weight cycles statically. The best known algorithm for computing the minimum weight cycle in sparse graphs exactly runs in $O(nm)$ time [40]. One can improve upon this for graphs with small integer weights using matrix multiplication [16, 27, 42]. A subcubic-time $(1+\epsilon)$-approximation can also be achieved this way [10, 48]. Much of the recent work regarded approximating the minimum weight cycle within factor at least 2 [13, 14, 17].

Dynamic APSP. Apart from the fully dynamic setting, APSP has also been widely studied in partially dynamic settings. There exist efficient exact algorithms for *unweighted* digraphs with $\widetilde{O}(n^3)$ total update time in both incremental [4] and decremental [5, 20] settings. The fully dynamic APSP algorithm [18, 45] is known to have total update time $\widetilde{O}(n^3)$ in the decremental setting for real-weighted digraphs, but only when each update removes *all* edges incident to a vertex (and thus there are at most $\leq n$ updates). For weighted digraphs, a nearly optimal $\widetilde{O}(nm\log(W)/\epsilon)$ total update time partially dynamic algorithm is known in the $(1+\epsilon)$-approximate setting [7]. This algorithm assumes an oblivious adversary though. Less efficient algorithms that are either deterministic or assume an adaptive adversary are known [20, 33, 32]. Note that many of the above algorithms maintain the distance matrix explicitly so they can be obviously used to maintain the minimum weight cycle (possibly approximately) in the respective partially dynamic scenarios.

Dynamic APSP has also been studied in undirected graphs [6, 8, 12, 15, 23, 25, 44].

[3] Abraham et al. [2] show how to extend their data structure so that it is capable of tracking lexicographically smallest shortest paths and thus works against an adaptive adversary, even when returning actual paths is required. Out of the box, this additional feature costs $\Omega(n^2)$ extra time per update, though. Adapting this idea to minimum weight cycle and MPSP is an interesting possible further step.

1.3 Organization of the paper

The rest of this paper is organized as follows. In Section 2 we fix the notation. In Section 3 we show a fully dynamic *threshold cycle detection* data structure that constitutes the heart of the fully dynamic $(1+\epsilon)$-approximate minimum weight cycle algorithm of Theorem 1 proved in Section 4. The applications of Theorem 1 to planar graph algorithms, in particular the proofs of Theorems 2 and 3, are covered in detail in Section 5. In Section 6 we describe the exact fully dynamic minimum weight cycle and fully dynamic MPSP algorithms. Due to space constraints, Section 6 contains merely an overview of the adjustments we make to the fully dynamic APSP algorithm of [2], and the details can be found in the full version.

2 Preliminaries

In this paper we deal with *real-weighted directed* graphs. We write $V(G)$ and $E(G)$ to denote the sets of vertices and edges of G, respectively. We denote by n and m numbers of vertices and edges (resp.) in the input graph. A graph H is a *subgraph* of G, which we denote by $H \subseteq G$, if and only if $V(H) \subseteq V(G)$ and $E(H) \subseteq E(G)$. We write $uv \in E(G)$ when referring to edges of G and use $w_G(uv)$ to denote the weight of uv.

For an edge set F, we sometimes write $G + F$ to denote the graph $(V(G), E(G) \cup F)$. If F contains an edge uv of weight x and $uv \in E(G)$, then we assume that $w_{G+F}(uv) = \min(w_G(uv), x)$. For an edge e we sometimes use $G + e$ to denote $G + \{e\}$. For a subset $D \subseteq V$, we define $G \setminus D$ to be the graph G with all edges incident to vertices in D removed.

A sequence of edges $P = e_1 \dots e_k$, where $k \geq 1$ and $e_i = u_i v_i \in E(G)$, is called an $s \to t$ path in G if $s = u_1$, $v_k = t$ and $v_{i-1} = u_i$ for each $i = 2, \dots, k$. For brevity we sometimes also express P as a sequence of $k+1$ vertices $u_1 u_2 \dots u_k v_k$ or as a subgraph of G with vertices $\{u_1, \dots, u_k, v_k\}$ and edges $\{e_1, \dots, e_k\}$. A path P is *simple* if $u_i \neq u_j$ for $i \neq j$. A *cycle* is a path such that $u_1 = v_k$. A *simple cycle* is a cycle that is a simple path.

The *hop-length* of P is the number of edges in P. We also say that P is a *k-hop path*. The *length* of the path $\ell(P)$ is defined as $\ell(P) = \sum_{i=1}^{k} w_G(e_i)$. For convenience, we sometimes consider a single edge uv as a path of hop-length 1. If P_1 is a $u \to v$ path and P_2 is a $v \to w$ path, we denote by $P_1 \cdot P_2$ (or simply $P_1 P_2$) a path obtained by concatenating P_1 with P_2.

The *distance* $\delta_G(u, v)$ between the vertices $u, v \in V(G)$ is the length of the shortest $u \to v$ path in G, or ∞, if no $u \to v$ path exists in G.

Note that the distance is well-defined only if G contains no negative cycles. It is well known that G has no negative cycles if and only if there exists a *feasible price function* $p : V \to \mathbb{R}$ satisfying $w_G(e) + p(u) - p(v) \geq 0$ for all $uv = e \in E(G)$. It is well-known that, given a feasible price function of G, one can compute single-source shortest paths in G using Dijkstra's algorithm even if G has edges with negative weights.

Define $\phi(G)$ to be the infimum of $\ell(C)$ through all cycles $C \subseteq G$. Note that here C is not necessarily a simple cycle: in general finding minimum weight simple cycles with arbitrary negative weights is NP-hard. In particular, if G contains no cycles at all, then we define $\phi(G) := \infty$. If G contains a negative cycle, then $\phi(G) = -\infty$. On the other hand, if $\phi(G) \geq 0$, then G contains a simple cycle C' with $\ell(C') = \phi(G)$. We call any such cycle C' a *minimum weight cycle*. Observe that if $\phi(G) \geq 0$, then $\phi(G) = \min_{uv \in E(G)} \{\delta_G(v, u) + w_G(uv)\}$.

▶ **Observation 6.** *Let H be a non-negatively weighted digraph and let v be its vertex. The minimum weight of a cycle in H that goes through v can be computed in $O(m + n \log n)$ time.*

Proof. First compute single-source shortest paths from v using Dijkstra's algorithm. Note that the minimum weight cycle through v has length $\min_{uv \in E(H)} \{\delta_H(v, u) + w_H(uv)\}$. ◀

When characterizing dynamic graph algorithms, we use the term *edge update* to refer to a graph update that changes (i.e., inserts, removes, or alters the weight) a single edge of G. On the other hand, a *vertex update* can change all edges incident (incoming or outgoing) to a single chosen vertex $v \in V(G)$. In this case, we say that such a vertex update is *centered* at v.

3 Fully dynamic threshold cycle detection

Consider the following decision variant of the fully dynamic minimum weight cycle problem. Suppose we would like to maintain the information whether the minimum weight $\phi(G)$ of a cycle in a real-weighted digraph G is below some threshold $\mu \geq 0$. In this section we show:

▶ **Theorem 7.** *Let G be an initially empty real-weighted digraph and let $\mu \geq 0$. There exist a fully dynamic algorithm maintaining the information whether $\phi(G) < \mu$ and supporting vertex updates in $O(m + n \log n)$ amortized time.*

The idea is to keep the edge set E partitioned into two subsets E_0 and E_1 such that the following two invariants are satisfied:
(1) For $G_0 = (V, E_0)$ we have $\phi(G_0) \geq \mu$.
(2) If $E_1 \neq \emptyset$, then $\phi(G) < \mu$.
Observe that by the above invariants, $\phi(G) < \mu$ *if and only if* $E_1 \neq \emptyset$.

Let us first consider the case when G has non-negative edges only. Then we can assume $\mu > 0$ since the answer for $\mu = 0$ is trivially "no".

We store E_1 partitioned into subsets $E_1(v)$ for $v \in V$, so that each edge $uv \in E_1$ is stored in either $E_1(u)$ or $E_1(v)$ (this choice is arbitrary). Since the data structure is initialized with an empty graph, initially $E_0 = \emptyset$ and $E_1(v) = \emptyset$ for all $v \in V$.

We also store the vertices v with $E_1(v) \neq \emptyset$ of G in a list Q sorted by the time when the last insertion around v happened, i.e., at the end of Q we have a vertex that has been most recently subject to insertion of edges around v.

Let us now describe an auxiliary procedure `update`(v) that will be used to fix the invariants. `update`(v) does the following. We assume that $E_1(v) \neq \emptyset$. We compute the minimum weight x of a cycle going through v in $G_0 + E_1(v) = (V, E_0 \cup E_1(v))$ as described in Observation 6. If $x \geq \mu$, the edges $E_1(v)$ are moved to E_0, and the set $E_1(v)$ is emptied. This change is reflected in Q by removing v from Q.

To handle and insertion of a set F_v of edges centered at some vertex v, we simply add the edges F_v to $E_1(v)$, move v to the end of Q, and, if $Q = \{v\}$, run `update`(v).

To handle a deletion of an arbitrary set of edges $F \subseteq E$, we first remove each edge $f \in F$ from E_0 or some set $E_1(w)$, wherever f resides. If some $E_1(w)$ is emptied this way, w is removed from Q accordingly. Next, while $Q \neq \emptyset$, we repeatedly run `update`(v) for the first element $v \in Q$ and stop if `update`(v) fails to empty the respective set $E_1(v)$.

We now prove the correctness of the algorithm, whose pseudocode is given in Algorithm 1.

▶ **Observation 8.** *Suppose $\phi(G_0) \geq \mu$ and let $v \in V$. Then $\phi(G_0 + E_1(v)) < \mu$ if and only if the shortest cycle going through v in $G_0 + E_1(v)$ has weight less than μ.*

Proof. By $\phi(G_0) \geq \mu$, a cycle of weight less than μ in $G_0 + E_1(v)$ has to go through an edge of $E_1(v)$. All of these edges are incident to the vertex v. ◀

Clearly, E_0 and E_1 form a partition of E after each insertion or deletion: the procedure `update` only moves edges from E_1 to E_0.

Algorithm 1 Detecting a cycle of weight less than μ.

procedure `update`(v)

```
1: x := the minimum weight of a cycle going through v in G_0 + E_1(v)
2: if x ≥ μ then
3:     E_0 := E_0 ∪ E_1(v)
4:     E_1(v) := ∅
5:     Q := Q \ {v}
```

procedure `insert`$(F_v \neq \emptyset)$

```
1: E_1(v) := E_1(v) ∪ F_v
2: move-to-back(Q, v)
3: if Q = {v} then
4:     update(v)
```

procedure `delete`$(F \subseteq E(G))$

```
1: E_0 := E_0 \ F
2: for uv = e ∈ F do
3:     for w ∈ {u, v} do
4:         E_1(w) := E_1(w) \ {e}
5:         if E_1(w) = ∅ then
6:             Q := Q \ {w}
7: while Q ≠ ∅ do
8:     v := front(Q)
9:     update(v)
10:    if E_1(v) ≠ ∅ then
11:        break
```

function `cycle-below-threshold`$()$

```
1: return Q ≠ ∅
```

▶ **Lemma 9.** *Invariant (1) is maintained throughout the updates.*

Proof. Note that no edge is added to E_0 outside the `update` procedure. As a result, since invariant (1) cannot be broken by removing edges from G_0, to establish that invariant (1) is maintained, it is enough to see that `update` only adds edges to E_0 if $\phi(G_0) \geq \mu$ afterwards. ◀

▶ **Lemma 10.** *Invariant (2) is maintained throughout the updates.*

Proof. Let G', G'_0, E'_1 denote G, G_0, E_1 respectively *before* the graph update. Suppose that after processing the update, invariant (2) is broken. Equivalently, $E_1 \neq \emptyset$ and $\phi(G) \geq \mu$.

Suppose the update was insertion of edges F_v centered at v. Since adding edges can only decrease the minimum weight of a cycle, $\phi(G') \geq \mu$. As invariant (2) was satisfied before, $E'_1 = \emptyset$. So after F_v is moved to $E_1(v)$, we indeed have $Q = \{v\}$. Since $\phi(G_0 + E_1(v)) \geq \phi(G) \geq \mu$, $E_1(v)$ should have been moved to E_0 by `update`(v). But $E_1 = E_1 \setminus E'_1 \subseteq E_1(v) = \emptyset$, so $E_1 = \emptyset$, a contradiction.

Now assume that the update deleted an arbitrary subset of edges. If after some `update`(v) call we have $E_1(v) \neq \emptyset$, then $\phi(G_0 + E_1(v)) < \mu$, which implies $\phi(G) < \mu$, a contradiction. If no such v exists, then Q is emptied, i.e., $E_1(v) = \emptyset$ for all $v \in V$ after the deletion is processed. It follows that $E_1 = \emptyset$, which again leads to a contradiction. ◀

Now let us analyze the running time of our algorithm.

▶ **Lemma 11.** *Each insertion is processed in $O(m + n \log n)$ worst-case time.*

Proof. An insertion adds $O(n)$ edges to a single set $E_1(v)$ and causes at most a single `update` call. The running time of `update` is dominated by the time needed to find the minimum weight of a cycle going through some vertex v in some subgraph of the current graph G. By Observation 6, this time is no more than $O(m + n \log n)$. ◀

▶ **Lemma 12.** *The total time needed to process arbitrary k updates is $O\left(\sum_{i=1}^{k}(m_i + n \log n)\right)$, where m_i is the number of edges in G when the i-th update happened. In other words, the amortized update time is $O(m + n \log n)$.*

Proof. By Lemma 11, we only need to prove that the deletions take $O\left(\sum_{i=1}^{k}(m_i + n \log n)\right)$ time in total. The cost of removing the edges from the sets E_0 and $E_1(w)$, $w \in V$, can be charged to the insertions which added those edges to the graph.

After updating the edge set, a deletion is handled using a number of `update`(v) runs, in the order in which vertices v appear in Q. At most one of these runs leaves $E_1(v)$ non-empty afterwards. We charge the cost of this run to the considered deletion. For all other `update`(v) runs during that deletion, they empty the set $E_1(v)$ that previously was non-empty. As a result, we can charge the cost of that run to the last insertion of edges centered at v that happened before the considered deletion.

We need to prove two things. First of all, to see that no insertion is charged twice, note that after an insertion is charged for the first time, $E_1(v)$ is emptied. So, before `update`(v) is called next time when handling a deletion, new edges have to be added to $E_1(v)$, which can only happen during another later insertion centered at v.

We also have to prove that just before $E_1(v)$ is emptied in `update`(v), the number of edges in $G_0 + E_1(v)$ is $O(|E'|)$, where E' is the edge set of G immediately after the last insertion I centered at v happened. To this end, we prove $E(G_0) \cup E_1(v) \subseteq E'$.

We clearly had $E_1(v) \subseteq E'$ immediately after I. Afterwards no more elements were added to $E_1(v)$ (albeit some might have been removed), so we still have $E_1(v) \subseteq E'$.

Now suppose there is an edge $e \in E(G_0)$ with $e \notin E'$. Then, since $G_0 \subseteq G$, e was inserted into G after the insertion I, as a result of a later insertion I' centered at some $w \neq v$. The edge e could have been added to G_0 only if $E_1(w)$ was emptied inside `update`(w) immediately afterwards, but before `update`(v) was called. Since I was the last insertion centered at v before `update`(v) was called, both v and w were in Q when `update`(w) was called. This is a contradiction: `update` is always called on the earliest element of Q, whereas the fact that I happened before I' implies that v lied earlier than w in Q when `update`(w) was called. ◀

▶ Remark 13. When handling a deletion, we could in principle call `update`(v) for vertices v with $E_1(v) \neq \emptyset$ in arbitrary order, as opposed to in the order of least recent centered insertions. However, then one could only show a weaker total update time bound of $O(k(m_{\max} + n \log n))$, where $m_{\max}$ is the maximum number of edges in G during the first k updates.

3.1 Negative weights

In this section we extend the obtained basic algorithm to also work with negative edges. Recall that we still assume $\mu \geq 0$. Note that the case $\mu = 0$ is equivalent to dynamically maintaining whether G has a negative cycle. Recall that if G has a negative cycle, $\phi(G) = -\infty$.

Unfortunately, in presence of negative weights or cycles we cannot simply use the algorithm behind Observation 6 to find the minimum weight cycle through a vertex v in $G_0 + E_1(v)$ as we did in `update`(v). Instead, we use the following lemma.

▶ **Lemma 14.** *Let H be a digraph with no negative cycles. Let $p : V \to \mathbb{R}$ be a feasible price function of H. Let F be a set of edges centered at some vertex v.*

Then in $O(m + n \log n)$ time one can find the minimum weight of a cycle going through v in $H + F$. Moreover, if $H + F$ contains no negative cycles, within the same time bound one can produce a feasible price function on $H + F$.

Proof. Clearly, since H has no negative cycles, a negative cycle in $H+F$ has to go through v. Let E_v^+ be the set of edges in $H + F$ incoming to v. Note that $H' = H + F - E_v^+$ has no negative cycles. Moreover, since H' differs from H by edges incident to v, the edge costs reduced by p are non-negative for all edges of H' possibly except the outgoing edges of v. However, since v has no incoming edges in H', a price function p' obtained from p by sufficiently increasing $p(v)$ (e.g., to $\max\{p(u) - w(vu) : vu \in E(H')\}$) is a feasible price function of H'. With price function p' in hand, we can compute distances from v in H' using Dijkstra's algorithm in $O(m + n \log n)$ time.

Now let $x = \min_{uv \in E_v^+}\{\delta_{H'}(v, u) + w_{H'}(uv)\}$. Observe that x is indeed the minimum weight of a simple cycle in $H + F$. Moreover, $x \geq 0$ implies that $p^*(y) := \delta_{H'}(v, y) = \delta_{H+F}(v, y)$ is a feasible price function on the induced subgraph of $(H + F)[R]$ reachable from v. To extend that price function p^* on $R \subseteq V$ to entire V, it is enough to set $p^*(z) = p(z) + M$ for all $z \in V \setminus R$, where M is a sufficiently large number. To see that, note that p^* is clearly a feasible price function on $(H + F)[R]$, $(H + F)[V \setminus R]$, and there are no edges from R to $V \setminus R$ in $H + F$. For edges $zy \in E(H + F) \cap ((V \setminus R) \times R)$ we have $w_{H+F}(zy) + p^*(z) - p^*(y) = w_{H+F}(zy) + p(z) + M - p^*(y)$. For

$$M = \max\{p^*(y) - p(z) - w_{H+F}(zy) : zy \in E(H + F) \cap ((V \setminus R) \times R)\},$$

all the required reduced costs are non-negative. ◀

Now, given Lemma 14, we modify the basic algorithm as follows. In addition to the partition of E into E_0 and E_1, we always maintain a feasible price function p_0 on G_0. Then, in `update`(v), we use Lemma 14 to find the minimum weight x of a cycle in $G_0 + E_1(v)$. If the edges $E_1(v)$ are moved to E_0 (and thus $x \geq 0$ since $\phi(G_0 + E_1(v)) \geq \mu \geq 0$), we update the price function p_0 to that produced by Lemma 14. Since the worst-case cost of running the algorithm from Lemma 14 matches that of Observation 6, the time analysis remains unchanged. Lemmas 9, 10, 12 and 14 together imply Theorem 7.

▶ Remark 15. For the problem of fully dynamically maintaining the information whether G contains a *negative* cycle (i.e., the special case $\mu = 0$) there exists a better algorithm with $O(m + n \log n)$ *worst-case* (as opposed to only amortized) update time bound (see Theorem 23). In fact, we make use of that algorithm when obtaining exact algorithms with good worst-case bounds in Section 6. The main idea is to generalize the problem to maintaining a minimum cost circulation in the graph G with imposed unit vertex/edge capacities (the details can be found in the full version). This resembles Gabow's reduction of single-source shortest paths with negative weights to the minimum cost perfect matching problem [22]. However, the min-cost circulation based algorithm is not as robust when it comes to obtaining fully dynamic algorithms for planar graphs (described in Section 5).

4 A fully dynamic $(1 + \epsilon)$-approximate algorithm

In this section we show how Lemma 12 can be used to obtain an $(1+\epsilon)$-approximate minimum weight cycle algorithm, for any $\epsilon \in (0, 1]$. Suppose $c \in \mathbb{R}$ ($C \in \mathbb{R}$) is a lower bound (an upper bound, respectively) on the weight of a *positive* cycle in G.

Suppose first that G has positively weighted edges. In order to convert the decision version from Section 3, all we have to do is to run it simultaneously with $\mu = (1+\epsilon)^k$ for all integers $k = \lceil \log_{1+\epsilon}(c) \rceil, \ldots, \lceil \log_{1+\epsilon}(C) \rceil$. . To maintain an approximate minimum weight of a cycle G, one only needs to keep track of the minimum k such that the fully dynamic decision algorithm for $(1+\epsilon)^k$ returns yes. If no such k exists, G is acyclic since $\phi(G) < \infty$ implies $\phi(G) \leq C$. Otherwise, we have $(1+\epsilon)^{k-1} \leq \phi(G) < (1+\epsilon)^k$, so indeed $(1+\epsilon)^k$ approximates $\phi(G)$ with multiplicative error no more than $(1+\epsilon)$. Since each of the $O(\log_{1+\epsilon}(C) - \log_{1+\epsilon}(c)) = O(\log(C/c)/\epsilon)$ decision algorithms has $O(m + n\log n)$ amortized update time, the amortized time of the approximate algorithm is $O((m+n\log n)\log(C/c)/\epsilon)$.

The same bound can be achieved even if G has non-positive edges (without, however, changing the definition of c and C) by extending each threshold data structure as described in Section 3.1. Apart from the data structures for thresholds $\mu = (1+\epsilon)^k$, we also need two more threshold cycle detection data structures: one for $\mu = 0$ to detect a negative cycle, and one for $\mu = c$ to detect whether $\phi(G) = 0$. We have thus proved Theorem 1.

▶ **Theorem 1.** *Let G be an initially empty fully dynamic real-weighted digraph such that the weight of each positive weight cycle in G always belongs to the interval $[c, C]$, $c, C \in \mathbb{R}$.*

There exists an algorithm maintaining an estimate ϕ' satisfying $\phi(G) \leq \phi' \leq (1+\epsilon)\phi(G)$ under vertex updates to G with amortized update time $O((m + n\log n) \cdot \log(C/c)/\epsilon)$.

5 Dynamic algorithms for cycles, cuts and flows in planar graphs

In this section we argue that the fully dynamic threshold cycle detection algorithm can be implemented on planar directed graphs using the known dynamic distance oracles on planar graphs. Since the reduction in Section 4 uses the threshold data structure in a black-box way, this will imply an $(1+\epsilon)$-approximate minimum weight cycle algorithm.

Using known reductions based on plane duality, this will yield fully dynamic $(1+\epsilon)$-approximate algorithms for maintaining (1) the capacity of a global min-cut in a plane digraph, (2) the value of maximum s,t-flow in a plane digraph.

The algorithms in this section handle *edge updates*, as opposed to more general vertex updates as was the case in the previous sections. Observe that achieving sublinear update time for vertex updates is not possible in general since a vertex update may need up to $\Theta(n)$ space to be described. More concretely, we will allow a single update to either insert or remove a single edge uv, provided that this update preserves planarity of G. In the cut/flow applications we will additionally need to assume that the edge insertions are *embedding preserving*, i.e., u and v lie on a single face of the current embedding of G.

Kaplan et al. [31], based on earlier work [21, 34], showed a dynamic distance oracle for real-weighted plane graphs undergoing edge weight updates. As argued in [11], their bound also holds if arbitrary, not necessarily embedding-preserving, edge updates are allowed.

▶ **Theorem 16** ([11, 21, 31, 34])**.** *Let G be a real-weighted planar digraph. There exists a fully dynamic algorithm supporting edge insertions and deletions in $\widetilde{O}(n^{2/3})$ worst-case time, such that for any query vertices s, t, the shortest $s \to t$ path in G can be computed in $\widetilde{O}(n^{2/3})$ time. If an edge insertion creates a negative cycle in G, the update algorithm reports it and refuses to perform that insertion. Edge insertions are not required to be embedding preserving.*

Fully dynamic threshold- and minimum weight cycles. Consider using the fully dynamic threshold cycle detection algorithm of Section 3 in the edge update scenario. Suppose that that algorithm attempts to moves edges from E_1 to E_0 single edge at a time. This does not

influence correctness; the efficiency of processing a *node update* could deteriorate though (which we do not mind). Then, the *amortized* update time to process the update involving an edge uv can be actually bounded by the sum of times needed to:

1. update the set $E_1(u)$ to reflect the graph update,
2. if uv is deleted, remove uv from G_0,
3. for some $xy \in E$, find the minimum weight of a cycle going through xy in $G_0 + xy$,
4. if $\phi(G_0 + xy) \geq \mu$, insert the edge xy into G_0.

Clearly, item 1 takes constant time. If we store the (planar) graph G_0 in the data structure of Theorem 16, items 2-4 above all require $\widetilde{O}(n^{2/3})$ time. Indeed, items 2 and 4 translated to a single edge update to that data structure, whereas item 3 amounts to computing $\delta_{G_0}(y, x) + w_G(xy)$ using a single query. We thus obtain the following analogue of Theorem 7.

▶ **Theorem 17.** *Let G be a real-weighted planar digraph and let $\mu \geq 0$. There exist a fully dynamic algorithm maintaining whether $\phi(G) < \mu$ and supporting planarity-preserving edge insertions and deletions in $\widetilde{O}(n^{2/3})$ amortized time.*

Since Theorem 1 uses the threshold data structure in a black-box way, we obtain:

▶ **Theorem 18.** *Let G be a fully dynamic real-weighted planar digraph G such that the weight of any positive cycle in G always lies in the interval $[c, C]$.*

There exists an algorithm maintaining the minimum weight cycle in G under planarity preserving edge insertions and deletions with amortized update time $\widetilde{O}(n^{2/3} \log(C/c)/\epsilon)$.

Note that Theorem 18 immediately implies Theorem 2.

Fully dynamic directed cuts and flows. Let G be a *plane embedded* digraph with real edge capacities in $\{0\} \cup [1, W]$. Wlog. we assume that every edge e in G has its reverse e^R of capacity 0 embedded into the same curve. We can then think of any edge as traversable in both directions, but the cost of such a traversal is 0 if the edge is traversed in the reverse direction. This assumption clearly does not influence values of max-flows or min-cuts in G, but makes the dual graph G^* possess certain useful properties. We call a cycle in G^* *non-trivial* if it is not of the form ee^R for some edge $e \in E(G^*)$ and its reverse e^R.

We now state well-known properties relating flows/cuts in G to cycles in the dual G^*.

▶ **Lemma 19** (see e.g. [35])**.** *The global minimum cut in a plane graph G corresponds to the minimum weight non-trivial cycle in G^*.*

▶ **Lemma 20** ([19, 30, 39])**.** *Let G be a plane digraph with some fixed source s and sink t. For $f \geq 0$, let $G_{P,f}$ be a plane graph obtained from G adding an embedded $s \to t$ path P such that for each edge e of P, the capacity of e is f, whereas the capacity of e^R is $-f$.*

There exists an s,t-flow of value f in G if and only if the dual $G^*_{P,f}$ of $G_{P,f}$ does not contain negative cycles.

By Lemma 19, maintaining the (approximate) global min-cut dynamically under edge *embedding preserving* insertions/deletions can be reduced to maintaining the (approximate) minimum weight non-trivial cycle in the dual under vertex splits and edge contractions.

Let us now explain how such operations can be simulated using $O(1)$ updates to the data structure of Theorem 18 maintained on a certain augmented version G^*_1 of G^*, so that the minimum weights of a non-trivial cycle in G^* and G^*_1 are equal. A similar reduction has been previously described in [28, 35]. Each vertex v of the dual G^* corresponds in G^*_1 to a path P_v of $\deg_{G^*}(v)$ vertices connected using 0-weight edges traversable in both directions. For an edge $vu \in E(G^*)$ that is the i-th in (some) clockwise edge ring of v, and j-th in (some)

clockwise edge ring of u, the i-th vertex of P_v is connected by an edge of weight $w_{G^*}(vu)$ with the j-th vertex of P_u. This way, (1) each vertex of G_1^* has constant degree, (2) each non-trivial cycle in G^* has a corresponding non-trivial cycle of the same weight in G_1^*, (3) no additional (with respect to G^*) non-trivial cycles are introduced in G_1^*.

It is not hard to verify that each edge contraction or vertex split in G^* can be reflected using $O(1)$ edge insertions or deletions issued to G_1^*.

Observe that the additional constraint that the minimum weight cycle is non-trivial does not introduce any difficulties: in the data structure of Theorem 17 we compute the minimum weight cycle through some edge e by issuing a distance query to a graph that *does not contain* that edge. However, since a minimum weight non-trivial cycle through e in $G_0 + e$ can traverse e in any of the two directions, we need to issue two distance queries instead of one.

We thus obtain the following theorem.

▶ **Theorem 21.** *Let G be a plane digraph with real capacities in $\{0\} \cup [1, W]$. There is an algorithm maintaining a $(1+\epsilon)$-approximate estimate of the capacity of the global min-cut of G under* embedding preserving *edge updates with $\widetilde{O}(n^{2/3} \log W/\epsilon)$ amortized update time.*

To obtain a dynamic max s,t-flow algorithm, we use Lemma 20. We keep track of whether there exists a negative cycle (i.e., we set $\mu = 0$) in the dual of a graph a $G_{P,f}$, where $f = (1+\epsilon)^k$, for each $k = 0, \ldots, \lceil \log_{1+\epsilon}(nW) \rceil$. Similarly as was the case for global min-cut, one can simulate the effect that an embedding preserving edge update in G has on the negative cycles of the dual of $G_{P,f}$ using $O(1)$ updates to the data structure of Theorem 17 maintained on an analogous augmentation $(G_{P,f})_1^*$ of $G_{P,f}^*$.

There is one subtle detail about how $G_{P,f}$ is updated when G is subject to embedding preserving edge insertions and deletions. Note that Lemma 20 requires us to embed any additional simple $s \to t$ path P into G. Embedding P into G subdivides some of the original faces of G. As a result, an edge uv to be inserted inside some face F of G may cross some edges of the currently used path P in $G_{P,f}$. We deal with this problem as follows. We maintain an additional invariant that (the embedding of) the simple path P crosses each face of G at most once.

Now, when a new edge uv is inserted inside F, and P has an edge $e = xy$ inside F that would cross uv, we first remove e from $G_{P,f}$ to allow the insertion of uv. This insertion splits F into two faces F_1, F_2 such that x lies on F_1 and y lies on F_2. We now reconnect the path P by embedding two edges xu, uy with appropriate capacities as required by Lemma 20.

On the other hand, when an edge uv is removed, two faces F_1 of F_2 of G are merged into a single face F. If at most one of them F_1, F_2 contained an edge of P, we do not have to do anything. Otherwise, suppose wlog. that F_1 contains an edge $xy = e_1 \in P$, and F_2 contains an edge $ab = e_2 \in P$, such that e_1 appears before e_2 on P. Then, we remove e_1, e_2, and all edges between e_1 and e_2 on P from $G_{P,f}$, and replace them with a single edge xb embedded in F. Afterwards, the invariant is satisfied and P remains a simple path.

Finally, observe that each update to G adds $O(1)$ new edges to P in the *worst case*. An edge deletion may remove a superconstant number of edges from P, but these removals can be charged to the corresponding additions of new edges to P. To conclude, an edge update to G translates to $O(1)$ amortized edge updates to $G_{P,f}$, and as a result, to $O(1)$ amortized operations on the data structure of Theorem 17 run on the augmented dual $(G_{P,f})_1^*$. We have thus proved:

▶ **Theorem 3.** *Let G be a plane embedded digraph with real edge capacities in $\{0\} \cup [1, W]$ and a* fixed *source/sink pair s, t. There exists an algorithm maintaining a $(1-\epsilon)$-approximate estimate of the value of maximum s,t-flow in G under* embedding preserving *edge insertions and deletions with $\widetilde{O}(n^{2/3} \log(W)/\epsilon)$ amortized update time.*

6 Exact fully dynamic algorithm for minimum weight cycle

In this section we argue that using a variant of the fully dynamic APSP algorithm of Abraham et al. [2] one can achieve subquadratic update bounds for dynamic minimum weight cycle.

We will in fact first solve a slightly more general problem that we call the *fully dynamic multiple-pairs shortest paths* (fully dynamic MPSP for short). Our goal is to have a data structure that maintains distances $\delta_G(s_i, t_i)$ for some *fixed* (throughout the course of the algorithm) k source-target pairs $(s_1, t_1), \ldots, (s_k, t_k)$ subject to fully dynamic vertex updates. Obviously, the classical fully dynamic APSP corresponds to the case $k = n^2$.

In the following we sketch the approach of [2] to fully dynamic APSP. The presentation is however directed towards our goal of obtaining an MPSP data structure. Some details and proofs can be found only in [2]; we focus on the details of our adjustments.

Reduction to batch-deletion MPSP data structure. The first step is to reduce the fully dynamic problem to a certain decremental problem, called the *batch-deletion MPSP*. In this problem, we want to preprocess the input digraph G, so that one can efficiently compute MPSP in $G \setminus D$ for a subset $D \subseteq V$ that constitutes the query parameter. We assume that if $G \setminus D$ has a negative cycle, the data structure has to report its existence instead.

▶ **Lemma 22.** *Suppose we have a batch-deletion MPSP data structure with preprocessing time $T_{\mathrm{pre}}(n, m, k)$ and worst-case query time $T_{\mathrm{q}}(n, m, k, d)$, where $d = |D|$ is the size of the removed vertex set. Then, for any integer $\Delta > 0$, there exists a fully dynamic MPSP algorithm with worst-case update time $O(T_{\mathrm{pre}}(n, m, k)/\Delta + T_{\mathrm{q}}(n, m, k, \Delta) + \Delta(m + k + n \log n))$.*

Proof sketch. To obtain an amortized (as opposed to worst-case) bound from the statement, we split the timeline into phases of Δ updates. When a new phase starts, we rebuild the batch-deletion data structure from scratch on the graph G_0 at the start of the phase; this clearly incurs $O(T_{\mathrm{pre}}(n, m_0, k)/\Delta)$ amortized time cost per update, where $m_0 = |E(G_0)|$. At some point of a phase, let $D \subseteq V$, $|D| \leq \Delta$, be the vertices touched by updates in this phase. To compute MPSP at that point, we first compute MPSP in $G_0 \setminus D = G \setminus D$ in $O(T_{\mathrm{q}}(n, m_0, k, |D|)) = O(T_{\mathrm{q}}(n, m_0, k, \Delta))$ time. To obtain MPSP in G, we need to check if paths going through D in G improve upon those in $G \setminus D$, i.e., we compute MPSP in G according to the equation $\delta_G(s_i, t_i) = \min\left(\delta_{G_0 \setminus D}(s_i, t_i), \min_{v \in D}\{\delta_G(s_i, v) + \delta_G(v, t_i)\}\right)$.

Observe that all distances of the form $\delta_G(\cdot, v)$ or $\delta_G(v, \cdot)$ for $v \in D$ can be obtained by running Dijkstra's algorithm to/from each such v, in $O(\Delta(m + n \log n))$ total time, as long as a feasible price function of G is given. A feasible price function can be maintained in $O(m + n \log n)$ worst-case time after a vertex update using the following theorem, whose proof is deferred to the full version.

▶ **Theorem 23.** *Let G be an initially empty real-weighted digraph. There exists an algorithm maintaining the information whether G has a negative cycle and supporting vertex updates in $O(m + n \log n)$* worst-case *time. Additionally, whenever $\phi(G) \geq 0$, the algorithm maintains a feasible price function p of G.*

Theorem 23 is also used to keep track of whether the current G has a negative cycle. Once these distances are available, the distances $\delta_G(s_i, t_i)$ can be computed in $O(\Delta \cdot k)$ time.

Unfortunately, the above argument is fully valid only if either the number of edges m is of the same order throughout, i.e., $m_0 = O(m)$, or it cannot drop by more than a constant factor during a single phase, e.g., $m = \Omega(n\Delta)$. If, say, $T_{\mathrm{pre}}(n, m, k) = \Theta(nm)$, $\Delta = n^{1/3}$ and $m = n^{5/4} = m_0$ at the beginning of the phase, and during $n^{1/4}$ first updates in that phase m

gets decreased to $O(n)$, then the total update cost coming from the preprocessing in this phase is $\Theta(nm_0) = \Theta(n^{9/4})$. If the amortized update time coming from the preprocessing was indeed $O(T_{\text{pre}}(n, m, k)/\Delta)$, the the total update cost coming from these terms in that phase would be $O(n^{1/4} \cdot nm_0/\Delta + \Delta \cdot n^2/\Delta) = O(n^{13/6})$, i.e., polynomially less.

We circumvent this problem[4] as follows. We build the batch-deletion MPSP data structure on the graph $G'_0 = G_0 \setminus D^*$ instead of G_0, where D^* is the set of Δ vertices of G_0 with highest degree. Then, Dijkstra's algorithm is used to separately compute shortest paths through $D \cup D^*$ in G, as opposed to only through D. Clearly, the cost of such computation remains $O(\Delta(m + k + n \log n))$. However, the update cost coming from the batch-deletion MPSP data structure is decreased to $O(T_{\text{pre}}(n, m'_0, k)/\Delta + T_{\text{q}}(n, m'_0, k, \Delta))$, where m'_0 is the number of edges in G'_0. It is hence enough to observe that $m'_0 \leq m$ throughout this phase. Indeed, the updates centered at vertices D cannot remove more than $\sum_{v \in D} \deg_{G_0}(v)$ edges out of those originally contained in G_0. As a result, $m \geq m_0 - \sum_{v \in D} \deg_{G_0}(v)$. On the other hand, by removing D^* from G_0 we remove at least $\frac{1}{2} \sum_{v \in D^*} \deg_{G_0}(v)$ edges from G_0, i.e, $m'_0 \leq m_0 - \frac{1}{2} \sum_{v \in D^*} \deg_{G_0}(v)$. We obtain $m'_0 \leq m$ as follows:

$$m'_0 \leq m_0 - \frac{1}{2} \sum_{v \in D^*} \deg_{G_0}(v) \leq m_0 - \frac{1}{2} \sum_{v \in D} \deg_{G_0}(v) \leq m_0 + \frac{1}{2}(m - m_0) = \frac{1}{2} m'_0 + \frac{1}{2} m.$$

Since the amortization comes only from a (costly) rebuilding step after every Δ updates, turning the amortized bound into a worst-case one is standard, see e.g., [2, Section 2]. ◀

The batch-deletion data structure. Abraham et al [2] showed a batch-deletion APSP data structure with $\widetilde{O}(n^3)$ preprocessing time and $\widetilde{O}(n^2\sqrt{nd})$ query time which, by Lemma 22, implies $\widetilde{O}(n^{2+2/3})$ worst-case update time for fully dynamic APSP. Their batch-deletion data structure is Monte Carlo randomized and produces answers correct with high probability. We generalize this data structure to MPSP and non-dense graphs.

▶ **Theorem 24.** *There exists a Monte Carlo randomized batch-deletion MPSP data structure with $O((m + k)n \log^2 n)$ preprocessing and $O((m + n \log n + k)\sqrt{nd} \log n)$ query time. The answers produced are correct with high probability.*

Before we prove Theorem 24, let us show how it can be used to obtain fully dynamic MPSP and minimum weight cycle algorithms.

By choosing $\Delta = n^{1/3} \log^{2/3} n$, and applying Lemma 22, we obtain:

▶ **Theorem 5.** *Let G be a real-weighted digraph. There exists a Monte Carlo randomized fully dynamic MPSP data structure supporting vertex updates with $O((m+n \log n+k)n^{2/3} \log^{4/3} n)$ worst-case update time. The answers produced are correct with high probability.*

Now consider the fully dynamic minimum weight cycle problem. The minimum weight of a cycle in G is given by $\phi(G) = \min_{uv \in E(G)}\{\delta_G(v, u) + w_G(uv)\}$. As a result, after each update it is enough to recompute distances $\delta_G(s_l, t_l)$ in G for $k = m$ pairs (s_l, t_l) such that $t_l s_l \in E(G)$. If the edge set of G was fixed (and, for example, the updates were only allowed to change edge weights), so would be the set of source-target pairs of our interest. Hence, we could simply use the fully dynamic MPSP data structure of Theorem 5 in a black-box way. However, in general, $E(G)$ is not fixed and we need to be more careful.

[4] This problem does not arise in [2], since there m is assumed to be $\Theta(n^2)$ throughout.

We proceed as follows. In the reduction of Lemma 22, we will always build a batch-deletion MPSP data structure with the set of source-target pairs equal to the edge set used to build that data structure reversed. This means that at any point of the phase, we can compute the minimum weight cycle in $G \setminus (D^* \cup D) = G_0 \setminus (D^* \cup D)$ in $O(T_{\mathrm{pre}}(n,m,m)/\Delta + T_{\mathrm{q}}(n,m,m,\Delta)) = O((m+n\log n)n^{2/3}\log^{4/3} n)$ worst-case time. Since $G_0 \setminus (D^* \cup D)$ contains only a subset of edges of G_0, reading the subset of entries of the distance matrix of $G_0 \setminus (D^* \cup D)$ corresponding to reversed edges of $E(G_0)$ is enough to this end. In order to find the minimum weight cycle going through some vertex of $D^* \cup D$ in G, we just run the algorithm of Observation 6 (or, more generally, in presence of negative edges – the algorithm of Lemma 14 with a feasible price function maintained by the algorithm of Theorem 23) $|D^* \cup D| = O(\Delta)$ times. This costs $O(\Delta(m + n\log n)) = \widetilde{O}(mn^{1/3})$ time.

▶ **Theorem 4.** *Let G be a real-weighted digraph. There exists a Monte Carlo randomized fully dynamic algorithm maintaining $\phi(G)$ under vertex updates with $O((m+n\log n)n^{2/3}\log^{4/3} n)$ worst-case update time. The answers produced are correct with high probability.*

6.1 Overview of the batch-deletion MPSP data structure

Let us now sketch the idea behind our generalization of the batch-deletion data structure of [2]. Due to space constraints, the detailed description can be found in the full version.

We first need to refer to some details of the construction of Abraham et al. [2]. The batch-deletion data structure separately handles recomputing shortest paths of hop-length at least $\sqrt{n/d}$ ("long" paths), and separately "short" shortest paths – with hop-lengths in the intervals of the form $[h/2, h)$ for $O(\log n)$ values $h = 2^1, 2^2, \ldots, \sqrt{n/d}$.

The main difficulty lies in handling short paths, whereas handling long paths is an easier task. The key idea (which dates back to Thorup [46]) is to compute an ordered subset $\{v_1, \ldots, v_\ell\} \subseteq V$ with the following properties. Let $G_i = G \setminus \{v_1, \ldots, v_{i-1}\}$. Let $\mathcal{P}_i$ be the set of shortest $\leq h$-hop paths from/to v_i in G_i. Then:

(1) For any $s, t \in V$, an $s \to t$ path not longer than the shortest $\leq h$-hop $s \to t$ path in G can be obtained by stitching, for some $i \in \{1, \ldots, \ell\}$, the $s \to v_i$ and $v_i \to t$ paths of $\mathcal{P}_i$.
(2) For any $x \in V$, x lies on at most $\widetilde{O}(hn)$ paths from $\bigcup_{i=1}^{\ell} \mathcal{P}_i$.

Such an ordering, along with the paths $\mathcal{P}_i$, can be computed in $\widetilde{O}(nmh)$ time deterministically (then we have $\ell = n$), or in $\widetilde{O}(nm)$ time using randomization (then $\ell = \widetilde{O}(n/h)$). Each subsequent vertex v_i in the ordering is picked to be, roughly speaking, the "most congested" one out of $V \setminus \{v_1, \ldots, v_{i-1}\}$, i.e., the one that has not been picked yet and appears most often on the previously constructed paths $\bigcup_{j=1}^{i-1} \mathcal{P}_j$.

Given the above, Abraham et al. [2] show that after removing any $D \subseteq V$ from G, the "short" paths in G can be recomputed by:

(1) constructing a number of *sketch graphs* $H_1, \ldots, H_\ell$, where $H_i \subseteq G_i \setminus D$,
(2) rebuilding destroyed (by the removal of D) paths from $\mathcal{P}_i$ by running Dijkstra's algorithm from/to v_i on H_i,
(3) stitching the reconstructed paths back to obtain paths at leas as good as the actual shortest $\leq h$-hop paths in G.

Abraham et al. [2] prove that if we denote by U_i the set of vertices u such that either of the paths $u \to v_i$ or $v_i \to u$ from $\mathcal{P}_i$ has been destroyed by removing D, $d = |D|$, then we have $\sum_{i=1}^{\ell} |U_i| = \widetilde{O}(hnd)$, and the total number of edges M in the sketch graphs is

$$M = O\left(\sum_{i=1}^{\ell}\left(n + \sum_{u \in U_i} \deg_G(u)\right)\right).$$

It is easy to see that $M = \Omega(n\ell)$, and $M = \tilde{O}(hn^2 d)$. Moreover, for each rebuilt path $u \to v_i$ or $v_i \to u$, stitching takes additional $\Theta(n)$ time – as one needs to traverse through $\Theta(n)$ source-target pairs that might benefit from this – for a total of $\widetilde{O}(hn^2 d)$ time. Since h ranges from $O(1)$ to $\Theta\left(\sqrt{n/d}\right)$, rebuilding short paths takes $\Omega(n^2)$ and $\widetilde{O}(n^2\sqrt{nd})$ time as claimed.

Now, to obtain our improved $\widetilde{O}((m+k)\sqrt{nd})$ bound on batch deletion for sparse graphs and small number k of source-target paths (s_i, t_i) of interest, we make two main adjustments.

First of all, we show that even smaller sketch graphs H_i – with $O\left(\sum_i^\ell \sum_{u\in U_i} \deg_G(u)\right)$ edges in total – can be used, thus eliminating the $\Omega(n\ell)$ term, which for small h is $\Omega(n^2)$.

More importantly, we use a different *weighted* scheme for picking the ordered subset $\{v_1, \ldots, v_\ell\}$. Let us denote by K the undirected graph on V whose edges correspond to the source-target pairs (s_i, t_i) of interest. In our scheme, the congestion that a previously computed $\le h$-hop path $P = v_i \to u$ (or $P = u \to v_i$) incurs upon some vertex x with $x \in V(P)$ is $\deg_G(u) + \log n + \deg_K(u)$, as opposed to 1 in [2]. This makes the total congestion of each vertex x in the process possibly increase to $\tilde{\Theta}(h(m + n\log n + k))$, as opposed to $\widetilde{O}(hn)$ in [2]. However, we show that the total cost of running Dijkstra's algorithm on our (more compact) sketch graphs $H_1, \ldots, H_\ell$ can be charged to the part of the total congestion of removed vertices D coming from the $[\deg_G(u) + \log n]$ terms, which is $\widetilde{O}(dh(m + n\log n))$. A similar argument applies to the cost of restitching, which we prove to be $\widetilde{O}(dhk)$.

References

1. Amir Abboud, Virginia Vassilevska Williams, and Joshua R. Wang. Approximation and fixed parameter subquadratic algorithms for radius and diameter in sparse graphs. In Robert Krauthgamer, editor, *Proceedings of the Twenty-Seventh Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2016, Arlington, VA, USA, January 10-12, 2016*, pages 377–391. SIAM, 2016. doi:10.1137/1.9781611974331.ch28.
2. Ittai Abraham, Shiri Chechik, and Sebastian Krinninger. Fully dynamic all-pairs shortest paths with worst-case update-time revisited. In *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2017*, pages 440–452, 2017. doi:10.1137/1.9781611974782.28.
3. Bertie Ancona, Monika Henzinger, Liam Roditty, Virginia Vassilevska Williams, and Nicole Wein. Algorithms and hardness for diameter in dynamic graphs. In Christel Baier, Ioannis Chatzigiannakis, Paola Flocchini, and Stefano Leonardi, editors, *46th International Colloquium on Automata, Languages, and Programming, ICALP 2019, July 9-12, 2019, Patras, Greece*, volume 132 of *LIPIcs*, pages 13:1–13:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019. doi:10.4230/LIPIcs.ICALP.2019.13.
4. Giorgio Ausiello, Giuseppe F. Italiano, Alberto Marchetti-Spaccamela, and Umberto Nanni. Incremental algorithms for minimal length paths. *J. Algorithms*, 12(4):615–638, 1991. doi:10.1016/0196-6774(91)90036-X.
5. Surender Baswana, Ramesh Hariharan, and Sandeep Sen. Improved decremental algorithms for maintaining transitive closure and all-pairs shortest paths. *J. Algorithms*, 62(2):74–92, 2007. doi:10.1016/j.jalgor.2004.08.004.
6. Aaron Bernstein. Fully dynamic (2 + epsilon) approximate all-pairs shortest paths with fast query and close to linear update time. In *50th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2009, October 25-27, 2009, Atlanta, Georgia, USA*, pages 693–702. IEEE Computer Society, 2009. doi:10.1109/FOCS.2009.16.
7. Aaron Bernstein. Maintaining shortest paths under deletions in weighted directed graphs. *SIAM J. Comput.*, 45(2):548–574, 2016. doi:10.1137/130938670.

8 Aaron Bernstein and Liam Roditty. Improved dynamic algorithms for maintaining approximate shortest paths under deletions. In Dana Randall, editor, *Proceedings of the Twenty-Second Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2011, San Francisco, California, USA, January 23-25, 2011*, pages 1355–1365. SIAM, 2011. doi:10.1137/1.9781611973082.104.

9 Glencora Borradaile and Philip N. Klein. An $O(n \log n)$ algorithm for maximum st-flow in a directed planar graph. *J. ACM*, 56(2):9:1–9:30, 2009. doi:10.1145/1502793.1502798.

10 Karl Bringmann, Marvin Künnemann, and Karol Wegrzycki. Approximating APSP without scaling: equivalence of approximate min-plus and exact min-max. In Moses Charikar and Edith Cohen, editors, *Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing, STOC 2019, Phoenix, AZ, USA, June 23-26, 2019*, pages 943–954. ACM, 2019. doi:10.1145/3313276.3316373.

11 Panagiotis Charalampopoulos and Adam Karczmarz. Single-source shortest paths and strong connectivity in dynamic planar graphs. In Fabrizio Grandoni, Grzegorz Herman, and Peter Sanders, editors, *28th Annual European Symposium on Algorithms, ESA 2020, September 7-9, 2020, Pisa, Italy (Virtual Conference)*, volume 173 of *LIPIcs*, pages 31:1–31:23. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPIcs.ESA.2020.31.

12 Shiri Chechik. Near-optimal approximate decremental all pairs shortest paths. In *59th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2018*, pages 170–181. IEEE Computer Society, 2018. doi:10.1109/FOCS.2018.00025.

13 Shiri Chechik and Gur Lifshitz. Optimal girth approximation for dense directed graphs. In Dániel Marx, editor, *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms, SODA 2021, Virtual Conference, January 10 - 13, 2021*, pages 290–300. SIAM, 2021. doi:10.1137/1.9781611976465.19.

14 Shiri Chechik, Yang P. Liu, Omer Rotem, and Aaron Sidford. Constant girth approximation for directed graphs in subquadratic time. In Konstantin Makarychev, Yury Makarychev, Madhur Tulsiani, Gautam Kamath, and Julia Chuzhoy, editors, *Proccedings of the 52nd Annual ACM SIGACT Symposium on Theory of Computing, STOC 2020, Chicago, IL, USA, June 22-26, 2020*, pages 1010–1023. ACM, 2020. doi:10.1145/3357713.3384330.

15 Julia Chuzhoy and Thatchaphol Saranurak. Deterministic algorithms for decremental shortest paths via layered core decomposition. In Dániel Marx, editor, *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms, SODA 2021, Virtual Conference, January 10 - 13, 2021*, pages 2478–2496. SIAM, 2021. doi:10.1137/1.9781611976465.147.

16 Marek Cygan, Harold N. Gabow, and Piotr Sankowski. Algorithmic applications of baur-strassen's theorem: Shortest cycles, diameter, and matchings. *J. ACM*, 62(4):28:1–28:30, 2015. doi:10.1145/2736283.

17 Mina Dalirrooyfard and Virginia Vassilevska Williams. Conditionally optimal approximation algorithms for the girth of a directed graph. In Artur Czumaj, Anuj Dawar, and Emanuela Merelli, editors, *47th International Colloquium on Automata, Languages, and Programming, ICALP 2020, July 8-11, 2020, Saarbrücken, Germany (Virtual Conference)*, volume 168 of *LIPIcs*, pages 35:1–35:20. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPIcs.ICALP.2020.35.

18 Camil Demetrescu and Giuseppe F. Italiano. A new approach to dynamic all pairs shortest paths. *J. ACM*, 51(6):968–992, 2004. doi:10.1145/1039488.1039492.

19 Jeff Erickson. Maximum flows and parametric shortest paths in planar graphs. In Moses Charikar, editor, *Proceedings of the Twenty-First Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2010, Austin, Texas, USA, January 17-19, 2010*, pages 794–804. SIAM, 2010. doi:10.1137/1.9781611973075.65.

20 Jacob Evald, Viktor Fredslund-Hansen, Maximilian Probst Gutenberg, and Christian Wulff-Nilsen. Decremental APSP in directed graphs versus an adaptive adversary. *CoRR*, abs/2010.00937, 2020. arXiv:2010.00937.

21 Jittat Fakcharoenphol and Satish Rao. Planar graphs, negative weight edges, shortest paths, and near linear time. *J. Comput. Syst. Sci.*, 72(5):868–889, 2006. doi:10.1016/j.jcss.2005.05.007.

22 Harold N. Gabow. Scaling algorithms for network problems. *J. Comput. Syst. Sci.*, 31(2):148–168, 1985. doi:10.1016/0022-0000(85)90039-X.

23 Maximilian Probst Gutenberg and Christian Wulff-Nilsen. Deterministic algorithms for decremental approximate shortest paths: Faster and simpler. In *Proceedings of the 2020 ACM-SIAM Symposium on Discrete Algorithms, SODA 2020*, pages 2522–2541. SIAM, 2020. doi:10.1137/1.9781611975994.154.

24 Maximilian Probst Gutenberg and Christian Wulff-Nilsen. Fully-dynamic all-pairs shortest paths: Improved worst-case time and space bounds. In *Proceedings of the 2020 ACM-SIAM Symposium on Discrete Algorithms, SODA 2020*, pages 2562–2574. SIAM, 2020. doi:10.1137/1.9781611975994.156.

25 Monika Henzinger, Sebastian Krinninger, and Danupon Nanongkai. Dynamic approximate all-pairs shortest paths: Breaking the $O(mn)$ barrier and derandomization. *SIAM J. Comput.*, 45(3):947–1006, 2016. doi:10.1137/140957299.

26 Russell Impagliazzo, Ramamohan Paturi, and Francis Zane. Which problems have strongly exponential complexity? *J. Comput. Syst. Sci.*, 63(4):512–530, 2001. doi:10.1006/jcss.2001.1774.

27 Alon Itai and Michael Rodeh. Finding a minimum circuit in a graph. *SIAM J. Comput.*, 7(4):413–423, 1978. doi:10.1137/0207033.

28 Giuseppe F. Italiano, Yahav Nussbaum, Piotr Sankowski, and Christian Wulff-Nilsen. Improved algorithms for min cut and max flow in undirected planar graphs. In *Proceedings of the 43rd ACM Symposium on Theory of Computing, STOC 2011*, pages 313–322, 2011. doi:10.1145/1993636.1993679.

29 Donald B. Johnson. Efficient algorithms for shortest paths in sparse networks. *J. ACM*, 24(1):1–13, 1977. doi:10.1145/321992.321993.

30 Donald B. Johnson. Parallel algorithms for minimum cuts and maximum flows in planar networks. *J. ACM*, 34(4):950–967, 1987. doi:10.1145/31846.31849.

31 Haim Kaplan, Shay Mozes, Yahav Nussbaum, and Micha Sharir. Submatrix maximum queries in monge matrices and partial monge matrices, and their applications. *ACM Trans. Algorithms*, 13(2):26:1–26:42, 2017. doi:10.1145/3039873.

32 Adam Karczmarz and Jakub Lacki. Simple label-correcting algorithms for partially dynamic approximate shortest paths in directed graphs. In *3rd Symposium on Simplicity in Algorithms, SOSA@SODA 2020*, pages 106–120. SIAM, 2020. doi:10.1137/1.9781611976014.15.

33 Adam Karczmarz and Jakub Łącki. Reliable hubs for partially-dynamic all-pairs shortest paths in directed graphs. In *27th Annual European Symposium on Algorithms, ESA 2019*, pages 65:1–65:15, 2019. doi:10.4230/LIPIcs.ESA.2019.65.

34 Philip N. Klein. Multiple-source shortest paths in planar graphs. In *Proceedings of the Sixteenth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2005*, pages 146–155, 2005. URL: http://dl.acm.org/citation.cfm?id=1070432.1070454.

35 Hung-Chun Liang and Hsueh-I Lu. Minimum cuts and shortest cycles in directed planar graphs via noncrossing shortest paths. *SIAM J. Discret. Math.*, 31(1):454–476, 2017. doi:10.1137/16M1057152.

36 Andrea Lincoln, Virginia Vassilevska Williams, and R. Ryan Williams. Tight hardness for shortest cycles and paths in sparse graphs. In Artur Czumaj, editor, *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2018, New Orleans, LA, USA, January 7-10, 2018*, pages 1236–1252. SIAM, 2018. doi:10.1137/1.9781611975031.80.

37 Jakub Łącki and Piotr Sankowski. Min-cuts and shortest cycles in planar graphs in o(n loglogn) time. In Camil Demetrescu and Magnús M. Halldórsson, editors, *Algorithms - ESA 2011 - 19th Annual European Symposium, Saarbrücken, Germany, September 5-9, 2011. Proceedings*, volume 6942 of *Lecture Notes in Computer Science*, pages 155–166. Springer, 2011. doi:10.1007/978-3-642-23719-5_14.

38 Shay Mozes, Kirill Nikolaev, Yahav Nussbaum, and Oren Weimann. Minimum cut of directed planar graphs in $O(n \log \log n)$ time. In Artur Czumaj, editor, *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2018, New Orleans, LA, USA, January 7-10, 2018*, pages 477–494. SIAM, 2018. doi:10.1137/1.9781611975031.32.

39 Yahav Nussbaum. *Network flow problems in planar graphs*. PhD thesis, Tel Aviv University, 2014.

40 James B. Orlin and Antonio Sedeño-Noda. An $O(nm)$ time algorithm for finding the min length directed cycle in a graph. In Philip N. Klein, editor, *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2017, Barcelona, Spain, Hotel Porta Fira, January 16-19*, pages 1866–1879. SIAM, 2017. doi:10.1137/1.9781611974782.122.

41 Seth Pettie. A new approach to all-pairs shortest paths on real-weighted graphs. *Theor. Comput. Sci.*, 312(1):47–74, 2004. doi:10.1016/S0304-3975(03)00402-X.

42 Liam Roditty and Virginia Vassilevska Williams. Minimum weight cycles and triangles: Equivalences and algorithms. In Rafail Ostrovsky, editor, *IEEE 52nd Annual Symposium on Foundations of Computer Science, FOCS 2011, Palm Springs, CA, USA, October 22-25, 2011*, pages 180–189. IEEE Computer Society, 2011. doi:10.1109/FOCS.2011.27.

43 Liam Roditty and Uri Zwick. On dynamic shortest paths problems. *Algorithmica*, 61(2):389–401, 2011. doi:10.1007/s00453-010-9401-5.

44 Liam Roditty and Uri Zwick. Dynamic approximate all-pairs shortest paths in undirected graphs. *SIAM J. Comput.*, 41(3):670–683, 2012. doi:10.1137/090776573.

45 Mikkel Thorup. Fully-dynamic all-pairs shortest paths: Faster and allowing negative cycles. In *Algorithm Theory - SWAT 2004, 9th Scandinavian Workshop on Algorithm Theory, Proceedings*, pages 384–396, 2004. doi:10.1007/978-3-540-27810-8_33.

46 Mikkel Thorup. Worst-case update times for fully-dynamic all-pairs shortest paths. In *Proceedings of the 37th Annual ACM Symposium on Theory of Computing, STOC 2005*, pages 112–119, 2005. doi:10.1145/1060590.1060607.

47 Virginia Vassilevska Williams and R. Ryan Williams. Subcubic equivalences between path, matrix, and triangle problems. *J. ACM*, 65(5):27:1–27:38, 2018. doi:10.1145/3186893.

48 Uri Zwick. All pairs shortest paths using bridging sets and rectangular matrix multiplication. *J. ACM*, 49(3):289–317, 2002. doi:10.1145/567112.567114.

Coboundary and Cosystolic Expansion from Strong Symmetry

Tali Kaufman
Department of Computer Science, Bar-Ilan University, Ramat-Gan, Israel

Izhar Oppenheim
Department of Mathematics, Ben-Gurion University of the Negev, Be'er Sheva, Israel

Abstract

Coboundary and cosystolic expansion are notions of expansion that generalize the Cheeger constant or edge expansion of a graph to higher dimensions. The classical Cheeger inequality implies that for graphs edge expansion is equivalent to spectral expansion. In higher dimensions this is not the case: a simplicial complex can be spectrally expanding but not have high dimensional edge-expansion. The phenomenon of high dimensional edge expansion in higher dimensions is much more involved than spectral expansion, and is far from being understood. In particular, prior to this work, the only known bounded degree cosystolic expanders were derived from the theory of buildings that is far from being elementary.

In this work we study high dimensional complexes which are *strongly symmetric*. Namely, there is a group that acts transitively on top dimensional cells of the simplicial complex [e.g., for graphs it corresponds to a group that acts transitively on the edges]. Using the strong symmetry, we develop a new machinery to prove coboundary and cosystolic expansion.

It was an open question whether the recent elementary construction of bounded degree spectral high dimensional expanders based on coset complexes give rise to bounded degree cosystolic expanders. In this work we answer this question affirmatively. We show that these complexes give rise to bounded degree cosystolic expanders in dimension two, and that their links are (two-dimensional) coboundary expanders. We do so by exploiting the strong symmetry properties of the links of these complexes using a new machinery developed in this work.

Previous works have shown a way to bound the co-boundary expansion using strong symmetry in the special situation of "building like" complexes. Our new machinery shows how to get coboundary expansion for *general* strongly symmetric coset complexes, which are not necessarily "building like", via studying the (Dehn function of the) presentation of the symmetry group of these complexes.

2012 ACM Subject Classification Theory of computation → Expander graphs and randomness extractors

Keywords and phrases High dimensional expanders, Cosystolic expansion, Coboundary expansion

Digital Object Identifier 10.4230/LIPIcs.ICALP.2021.84

Category Track A: Algorithms, Complexity and Games

Related Version *Full Version*: `https://arxiv.org/abs/1907.01259` [13]

Funding *Tali Kaufman*: Research supported by ERC and BSF.
Izhar Oppenheim: Research supported by ISF grant no. 293/18.

Extended abstract

High dimensional expansion is a vibrant emerging field that has found applications to PCPs [5] and property testing [10], to counting problems and matroids [1], to list decoding [4], and recently to a breakthrough construction of decodable quantum error correcting codes that outperform the state-of-the art previously known codes [7]. We refer the reader to [16] for a recent (but already outdated) survey.

The term high dimensional expander means a simplicial complex that has expansion properties that are analogous to expansion in a graph. Nevertheless, the question of what is a high dimensional expander is still unclear. There is a spectral definition of high dimensional expanders that generalizes the spectral definition of expander graphs and a geometrical/topological definition that generalizes the notion of edge expansion (or Cheeger constant) of a graph. For a graph the spectral and the geometric definitions of expansion are known to be equivalent (via the celebrated Cheeger inequality) while in high dimensions the spectral and geometric definitions are known to be NOT equivalent (see [9, Theorem 4] and [19]).

The aim of this paper is to present elementary constructions of new families of 2-dimensional simplicial complexes with high dimensional edge expansion, and in particular, of new elementary *bounded degree* families of cosystolic expanders (see exact definition below). The question of giving an elementary construction of a family of bounded degree spectrally high dimensional expanders got recently a satisfactory answer. Namely, it was understood that such a family needs to obey a specific local spectral criterion and in [11] we used this understanding in order to construct elementary families of high dimensional *spectrally* expanding families (prior non-elementary constructions were known). Here, we further study the examples of [11], and show that they also give rise to bounded degree cosystolic expanders:

▶ **Theorem 1** (New cosystolic expanders, Informal, see also Theorem 22)**.** *For every large enough odd prime power q, the family of 2-skeletons of the 3-dimensional local spectral expanders constructed in [11] using elementary matrices over $\mathbb{F}_q[t]$ is a family of bounded degree cosystolic expanders.*

Prior to this work, the known examples of bounded degree cosystolic expanders arose from the theory of Bruhat-Tits buildings and were far from being elementary.

Relying on the work of the first named author and Evra (see Theorem 11 below), the proof of this Theorem boils down to proving that the links of our construction are coboundary expanders and that their coboundary expansion can be bounded independently of q (i.e., that the coboundary does not deteriorate as q increases). Thus, the real problem is bounding the coboundary expansion of the links. This goal is achieved utilizing the fact that the links are strongly symmetric coset complexes.

Coboundary expansion for strongly symmetric (coset) complexes

We call a simplicial complex strongly symmetric if it has a symmetry group acting transitively on top dimensional simplices. As noted above, our problem is to show that the links in our examples are coboundary expanders. In the graph setting, there is a classical Theorem (see Theorem 12 below) stating that for a strongly symmetric graph the Cheeger constant can be bounded from below by $\frac{1}{2D}$, where D denotes the diameter of the graph.

We generalize this idea: we define a high dimensional notion of radius and show that for strongly symmetric complexes, this radius can be used to bound the coboundary expansion. We then show that this radius can be bounded using filling constants of the complex. These ideas of bounding the coboundary expansion for symmetric complexes using filling constants already appeared implicitly in Gromov's work [8] and in the work of Lubotzky, Meshulam and Mozes [17]. However, these previous works considered the setting of spherical buildings and "building-like complexes" and thus bounding the filling constants in these examples were relatively simple due to the existence of apartments in the building (or "apartment-like" sub-complexes in "building-like" complexes). In our setting, we consider a more general situation (not assuming "apartment-like" sub-complexes) and thus bounding the filling constants becomes a much harder task.

What helps to solve this harder problem of bounding the filling constants is working with strongly symmetric *coset complexes* (see Definition 16). We note that this is not a very restrictive assumption - under some mild assumptions, every strongly symmetric complex is a coset complex (see proof in [13]). For a coset complex one can fully reconstruct the complex via its symmetry group and its subgroup structure. Thus every geometrical/topological property of a coset complex (including coboundary expansion) is encoded in some way in the presentation of its symmetry group. Using this philosophy, we are able to prove a bound for filling constants for two dimensional coset complex in terms of the presentation of its symmetry group (namely, in terms on its Dehn function - see definition below). Thus, for two dimensional coset complexes, we get a bound on the coboundary expansion in terms of presentation-theoretic properties of the symmetry group.

Coboundary expansion of the links in our construction

If follows from our work described above that in order to show that the links in our construction are coboundary expanders, we should verify a presentation-theoretic property for their symmetry group (namely, to bound its Dehn function). Luckily for us, the symmetry group of the links in our construction is a generalization of the group of unipotent groups over finite fields. For the finite field case, the presentation of these unipotent groups was studied by Biss and Dasgupta [2]. Using their ideas, we are able to show that the symmetry groups of links in our construction fulfil the presentation-theoretic condition that allows us to bound their coboundary expansion. Namely, we prove the following:

▶ **Theorem 2** (New coboundary expanders, Informal, see also Theorem 21)**.** *For every odd prime power q, the links of the 3-dimensional local spectral expanders constructed in [11] using elementary matrices over $\mathbb{F}_q[t]$ are coboundary expanders and their coboundary expansion can be bounded from below independent of q.*

Simplicial complexes

An n-dimensional simplicial complex X is a hypergraph whose maximal hyperedges are of size $n+1$, and which is closed under containment. Namely, for every hyperedge τ (called a face) in X, and every $\eta \subset \tau$, it must be that η is also in X. In particular, $\emptyset \in X$. For example, a graph is a 1-dimensional simplicial complex. Let X be a simplicial complex, we fix the following terminology/notation:

1. X is called *pure n-dimensional* if every face in X is contained in some face of size $n+1$.
2. The set of all k-faces (or k-simplices) of X is denoted $X(k)$, and we will be using the convention in which $X(-1) = \{\emptyset\}$.
3. For $0 \leq k \leq n$, the k-skeleton of X is the k-dimensional simplicial complex $X(0) \cup X(1) \cup ... \cup X(k)$. In particular, the 1-skeleton of X is the graph whose vertex set is $X(0)$ and whose edge set is $X(1)$.
4. For a simplex $\tau \in X$, the link of τ, denoted X_τ is the complex

   $$\{\eta \in X : \tau \cup \eta \in X, \tau \cap \eta = \emptyset\}.$$

   We note that if $\tau \in X(k)$ and X is pure n-dimensional, then X_τ is pure $(n-k-1)$-dimensional.
5. A family of pure n-dimensional simplicial complexes $\{X^{(s)}\}_{s\in\mathbb{N}}$ is said to have bounded degree if there is a constant $L > 0$ such that for every $s \in \mathbb{N}$ and every vertex v in $X^{(s)}$, v is contained in at most L n-dimensional simplices of $X^{(s)}$.

The coboundary/cosystolic expansion and high order Cheeger constants

Let us recall the geometric notion of expansion in graphs known as the edge expansion or Cheeger constant of a graph:

▶ **Definition 3** (Cheeger constant of a graph). *For a graph $X = (V, E)$:*

$$h(X) := min_{A \neq \emptyset, V} \frac{|E(A, \bar{A})|}{min\{w(A), w(\bar{A})\}},$$

where for a set of vertices $U \subsetneq V$, $w(U)$ denotes is the sum of the degrees of the vertices in U.

The generalization of the Cheeger constant to higher dimensions originated in the works of Linial, Meshulam and Wallach ([15], [18]) and independently in the work of Gromov ([8]) and is now known as *coboundary expansion*. Later, a weaker variant of high dimensional edge expansion known as *cosystolic expansion* arose in order to answer questions regarding topological overlapping.

In order to define coboundary and cosystolic expansion, we also need some terminology. Let X be an n-dimensional simplicial complex. Fix the following notations/definitions:

1. The space of k-cochains denoted $C^k(X) = C^k(X, \mathbb{F}_2)$ is the $\mathbb{F}_2$-vector space of functions from $X(k)$ to $\mathbb{F}_2$.
2. The *coboundary map* $d_k : C^k(X, \mathbb{F}_2) \to C^{k+1}(X, \mathbb{F}_2)$ is defined as:

$$d_k(\phi)(\sigma) = \sum_{\tau \subset \sigma, |\tau| = |\sigma| - 1} \phi(\tau),$$

3. The spaces of k-coboundaries and k-cocycles are subspaces of $C^k(X)$ defined as:
   $B^k(X) = B^k(X, \mathbb{F}_2) = \text{Image}(d_{k-1}) =$ the space of k-coboundaries.
   $Z^k(X) = Z^k(X, \mathbb{F}_2) = \text{Ker}(d_k) =$ the space of k-cocycles.
4. The function $w : \bigcup_{k=-1}^{n} X(k) \to \mathbb{R}_+$ is defined as

$$\forall \tau \in X(k), w(\tau) = \frac{|\{\sigma \in X(n) : \tau \subseteq \sigma\}|}{\binom{n+1}{k+1}|X(n)|}.$$

   We note that $\sum_{\tau \in X(k)} w(\tau) = 1$.
5. For every $\phi \in C^k(X)$, $w(\phi)$ is defined as

$$w(\phi) = \sum_{\tau \in \text{supp}(\phi)} w(\tau).$$

6. For every $0 \leq k \leq n-1$, define the following k-expansion constants:

$$\text{Exp}_b^k(X) = \min \left\{ \frac{w(d_k \phi)}{\min_{\psi \in B^k(X)} w(\phi + \psi)} : \phi \in C^k(X) \setminus B^k(X) \right\}.$$

$$\text{Sys}^k(X) = \min \left\{ w(\psi) : \psi \in Z^k(X) \setminus B^k(X) \right\},$$

and

$$\text{Exp}_z^k(X) = \min \left\{ \frac{w(d_k \phi)}{\min_{\psi \in Z^k(X)} w(\phi + \psi)} : \phi \in C^k(X) \setminus Z^k(X) \right\}.$$

After these notations, we can define coboundary/cosystolic expansion:

▶ **Definition 4** (Coboundary expansion). *Let $\varepsilon > 0$ be a constant. We say that X is an ε-coboundary expander if for every $0 \leq k \leq n-1$, $\mathrm{Exp}_b^k(X) \geq \varepsilon$.*

▶ Remark 5. We leave it for the reader to verify that in the case where X is a graph, i.e., the case where $n = 1$, $\mathrm{Exp}_b^0(X)$ is exactly the Cheeger constant of X. Thus, we think of $\mathrm{Exp}_b^k(X)$ as the k-dimensional Cheeger constant of X.

▶ **Definition 6** (Cosystolic expansion). *Let $\varepsilon > 0, \mu > 0$ be constants and X an n-dimensional simplicial complex X. We say that X is a (ε, μ)-cosystolic expander if for every $0 \leq k \leq n-1$, $\mathrm{Exp}_z^k(X) \geq \varepsilon$ and $\mathrm{Sys}^k(X) \geq \mu$.*

▶ Remark 7. We note that if $\mathrm{Exp}_b^k(X) > 0$, then it can be shown that $B^k(X) = Z^k(X)$ and thus $\mathrm{Exp}_b^k(X) = \mathrm{Exp}_z^k(X)$. However, there are examples of simplicial complexes with $\mathrm{Exp}_b^k(X) = 0$ and $\mathrm{Exp}_z^k(X) > 0, \mathrm{Sys}^k(X) > 0$.

As in expander graphs, we are mainly interested in a family of bounded degree cosystolic expanders (and not a single complex that is a cosystolic expander):

▶ **Definition 8** (A family of bounded degree cosystolic expanders). *A family of n-dimensional simplicial complexes $\{Y^{(s)}\}_{s\in\mathbb{N}}$ is a family of bounded degree cosystolic expanders if:*

- *The number of vertices of $Y^{(s)}$ tends to infinity with s.*
- *$\{Y^{(s)}\}_{s\in\mathbb{N}}$ has bounded degree.*
- *There are universal constants $\varepsilon > 0, \mu > 0$ such that for every s, $Y^{(s)}$ is a (ε, μ)-cosystolic expander.*

▶ Remark 9. The motivation behind the definition of a family of cosystolic expanders is to proved a family of bounded degree complexes that have the topological overlapping property (see [13] for the exact definition).

The Evra-Kaufman criterion for cosystolic expansion

In [6], Evra and the first named author gave a criterion for cosystolic expansion. In order to state this criterion, we will need the following definition:

▶ **Definition 10** (Local spectral expansion). *For $\lambda \geq 0$, a pure n-dimensional simplicial complex X is called a (one-sided) λ-local spectral expander if for $-1 \leq k \leq n-2$ and every $\tau \in X(k)$, the one-skeleton of X_τ is a connected graph and the second largest eigenvalue of the random walk on the one-skeleton of X_τ is less or equal to λ.*

The idea behind the Evra-Kaufman criterion for cosystolic expansion is the following: We can deduce cosystolic expansion from local spectral expansion and local coboundary expansion (i.e., coboundary expansion in the links) given that the local spectral expansion is "strong enough" so it "beats" the local coboundary expansion. More formally:

▶ **Theorem 11** ([6, Theorem 1] Evra-Kaufman criterion for cosystolic expansion). *For every $\varepsilon' > 0$ and $n \geq 3$ there are $\mu(n, \varepsilon') > 0, \varepsilon(n, \varepsilon') > 0$ and $\lambda(n, \varepsilon') > 0$ such that for every pure n-dimensional simplicial complex if*

- *X is a λ-local spectral expander.*
- *For every $0 \leq k \leq n-2$ and every $\tau \in X(k)$, X_τ is a ε'-coboundary expander.*

Then the $(n-1)$-skeleton of X is a (ε, μ)-cosystolic expander.

Thus, in order to prove cosystolic expansion in examples, we should verify two things: local spectral expansion and coboundary expansion in the links. In our examples from [11] described below, local spectral expansion is already known and we are left with proving coboundary expansion for the links. In order to do so, we will develop machinery to prove coboundary expansion for symmetric complexes of a special type called coset complexes.

Coboundary expansion for strongly symmetric simplicial complexes

As noted above, unlike the case of graphs, in simplicial complexes a high dimensional version of Cheeger inequality does not hold. Thus, there is a need to develop machinery in order to prove coboundary expansion that does not rely on spectral arguments. For graphs such machinery is available, under the assumptions that the graph has a large symmetry group. A discussion regarding the Cheeger constant of symmetric graphs appear in [3, Section 7.2] and in particular, the following Theorem is proven there:

▶ **Theorem 12** ([3, Theorem 7.1]). *Let X be a finite connected graph such that there is a group G acting transitively on the edges of X. Denote $h(X)$ to be the Cheeger constant of X and D to be the diameter of X. Then $h(X) \geq \frac{1}{2D}$.*

▶ Remark 13. Note that the inequality stated in the Theorem does not hold without the assumption of symmetry. For instance, let X_N be the graph that is the ball of radius N in the 3-regular infinite tree. Then the diameter of X is $2N+1$ and $h(X_N)$ is of order $O(\frac{1}{2^N})$.

In this paper, using the ideas of [8] and [17], we prove a generalization of Theorem 12 to the setting of (strongly) symmetric simplicial complexes. We first define the notion of strongly symmetric simplicial complexes.

▶ **Definition 14** (Strongly symmetric complex). *A simplicial complex X is called* strongly symmetric *if there is a group that acts simply transitively on its top dimensional faces. E.g., For graphs (one dimensional complexes) we require a group that acts simply transitively on the edges.*

We then define a high dimensional notion of radius which we call a cone radius, but this definition is a little technical and thus appears in Section 2 (see Definition 28). We then prove the following:

▶ **Theorem 15** (Informal, see Theorem 30 for the formal statement). *Let X be a strongly symmetric simplicial complex. If the k-dimensional (cone) radius of X is bounded by D, then $\mathrm{Exp}_b^k(X) \geq \frac{1}{\binom{n+1}{k+1}D}$, i.e., the k-coboundary expansion is bounded from below as a function of the k-th radius.*

Bounding the high dimensional radius for coset complexes

By Theorem 15, in order to prove coboundary expansion for strongly symmetric complexes, it is enough to bound their high dimensional radius. Following the ideas of Gromov [8], we bound the radius by bounding certain filling constants, that we will not define here. In order to bound these filling constants and thus the high dimensional radius, we will assume that our strongly symmetric complex is of a special type, namely that it is a coset complex:

▶ **Definition 16** (Coset complex). *Given a group G with subgroups $K_{\{i\}}, i \in \mathrm{I}$, where I is a finite set. The coset complex $X = X(G, (K_{\{i\}})_{i\in \mathrm{I}})$ is a simplicial complex defined as follows:*

1. *The vertex set of X is composed of disjoint sets $S_i = \{gK_{\{i\}} : g \in G\}$.*
2. *For two vertices $gK_{\{i\}}, g'K_{\{j\}}$ where $i, j \in \mathrm{I}, g, g' \in G$, $\{gK_{\{i\}}, g'K_{\{j\}}\} \in X(1)$ if $i \neq j$ and $gK_{\{i\}} \cap g'K_{\{j\}} \neq \emptyset$.*
3. *The simplicial complex X is the clique complex spanned by the 1-skeleton defined above, i.e., $\{g_0K_{\{i_0\}}, ..., g_kK_{\{i_k\}}\} \in X(k)$ if for every $0 \leq j, j' \leq k$, $g_jK_{\{i_j\}} \cap g_{j'}K_{\{i_{j'}\}} \neq \emptyset$.*

Although this Definition may seem daunting at first, we note that it is very natural in examples. Namely, in [13] we shows that under some mild assumptions, strongly symmetric simplicial complexes are actually coset complexes.

As noted above, for coset complexes, every property of the complex should be reflected in some way in its symmetry group and its subgroup structure. Following this philosophy, we prove that for coset complexes, the 0-th and 1-th dimensional coboundary expansion can be bounded using the presentation of the group from which the complex arose.

In order to describe our result, we recall some definitions from group theory. Given a group G, a generating set $S \subseteq G$ is a set of elements of G such that every element in G can be written as a finite product (or sum if G is commutative) of elements of S. One can always take $S = G$, but usually one can make due with a smaller set. For example, for the group G of addition of integers modulo n, $G = (\mathbb{Z}/n\mathbb{Z}, +)$, one can take $S = \{\pm 1\}$. Given a group G with a generating set S, a word with letters in S is called trivial if it equal to the identity. For example, in $G = (\mathbb{Z}/n\mathbb{Z}, +)$ with $S = \{\pm 1\}$, the words $1 + 1 + (-1) + (-1)$ and $1 + ... + 1$ (n summands) $= n \cdot 1$ are trivial.

We say that a group G has a presentation $G = \langle S|R\rangle$, if S is a generating set of G and R is a set of trivial words called relations such that every trivial word in G can be written using the words in $R \cup \{ss^{-1}, s^{-1}s : s \in S\}$ (allowing products, conjugations and inverses). Again, one can always take $S = G \setminus \{e\}$ and R to be the entire multiplication table of G, i.e., all the words of the form $g_1 g_2 g_3^{-1} = e$, where $g_1 g_2 = g_3$. However, in concrete examples, one can usually make due with fewer generators and relations. For example, for the group $G = (\mathbb{Z}/n\mathbb{Z}, +)$ it is sufficient to take $S = \{\pm 1\}$ and the single relation $n \cdot 1$. We note that it is not always easy to determine if a set of relations gives a presentation of G.

Given a presentation $G = \langle S|R\rangle$, the Dehn function for this presentation is a function $\mathrm{Dehn} : \mathbb{N} \to \mathbb{N}$ such that $\mathrm{Dehn}(m)$ describes how many elements of $R \cup \{ss^{-1}, s^{-1}s : s \in S\}$ does one need to write a trivial word in G of length $\leq m$. With this terminology, we prove the following:

▶ **Theorem 17.** *Let G be a finite group with subgroups $K_{\{i\}}, i \in \{0,1,2\}$. Denote $X = X(G, (K_{\{i\}})_{i\in\{0,1,2\}})$. Assume that G acts strongly transitively on X.*

For every $i \in \{0,1,2\}$, denote R_i to be all the non-trivial relations in the multiplication table of $K_{\{i\}}$, i.e., all the relations of the form $g_1 g_2 g_3 = e$, where $g_1, g_2, g_3 \in K_{\{i\}} \setminus \{e\}$. Assume that $G = \langle \bigcup_i K_{\{i\}} | \bigcup_i R_i \rangle$ and let Dehn *denote the Dehn function of this presentation.*

Then:

1. *For*
$$N_0' = 1 + \max_{g\in G} \min \left\{ l : g = g_1 ... g_l \text{ and } g_1, ..., g_l \in \bigcup_i K_{\{i\}} \right\},$$
it holds that $\mathrm{Exp}_b^0(X) \geq \frac{1}{3N_0'}$.
2. *There is a universal polynomial $p(x, y)$ independent of X such that*
$$\mathrm{Exp}_b^1(X) \geq \frac{1}{3p(2N_0' + 1, \mathrm{Dehn}(2N_0' + 1))}.$$

Our construction

So far, we described general tools that we developed in order to prove coboundary and cosystolic expansion. Now we will describe our construction from [11] on which we aim to apply these tools.

In [11], we used coset complexes to construct n-dimensional spectral expanders. Below, we only describe the construction for $n = 3$: Fix $s \in \mathbb{N}, s > 4$ and q be a prime power. Denote $G_q^{(s)}$ to be the group of 4×4 matrices with entries in $\mathbb{F}_q[t]/\langle t^s \rangle$ generated by the set

$$\{e_{1,2}(a + bt), e_{2,3}(a + bt), e_{3,4}(a + bt), e_{4,1}(a + bt) : a, b \in \mathbb{F}_q\}.$$

For $0 \leq i \leq 2$, define $H_{\{i\}}$ to be the subgroup of $G_q^{(s)}$ generated by

$$\{e_{j,j+1}(a+bt), e_{4,1}(a+bt) : a,b \in \mathbb{F}_q, 1 \leq j \leq 3, j \neq i+1\}$$

and define $H_{\{3\}}$ to be the subgroup of $G^{(s)}$ generated by

$$\{e_{1,2}(a+bt), e_{2,3}(a+bt), e_{3,4}(a+bt) : a,b \in \mathbb{F}_q\}.$$

Denote $X_q^{(s)} = X(G_q^{(s)}, (H_{\{i\}})_{i\in\{0,\ldots,3\}})$ to be the coset complex as defined above.

The main result of [11] applied to $\{X_q^{(s)}\}_{s>4}$ above can be summarized as follows:

1. The family $\{X_q^{(s)}\}_{s>4}$ has bounded degree (that depends on q).
2. The number of vertices of $X_q^{(s)}$ tends to infinity with s.
3. For every s, $X_q^{(s)}$ is $\frac{1}{\sqrt{q}-3}$-local spectral expander.

In light of Theorem 17, we will also need some facts regrading the links of $X_q^{(s)}$. We give the following explicit description of the links in our construction: We note that for every fixed q it holds that there is a complex X such that for every $s > 4$ and every vertex $v \in X_q^{(s)}$, there is a coset complex denoted $X_{link,q}$ such that the link of v is isomorphic to $X_{link,q}$ (all the links are isomorphic).

The complex $X_{link,q}$ can be described explicitly as follows: Denote the group $G_{link,q}$ to be a subgroup of 4×4 invertible matrices with entries in $\mathbb{F}_q[t]$ in generated by the set $\{e_{i,i+1}(a+bt) : a,b \in \mathbb{F}_q, 1 \leq i \leq 4\}$. More explicitly, an 4×4 matrix A is in $G_{link,q}$ if and only if

$$A(i,j) = \begin{cases} 1 & i = j \\ 0 & i > j \\ a_0 + a_1 t + \ldots + a_{j-i} t^{j-i} & i < j, a_0, \ldots, a_{j-i} \in \mathbb{F}_q \end{cases},$$

(observe that all the matrices in G are upper triangular).

For $0 \leq i \leq 3$, define a subgroup $K_{\{i\}} < G$ as

$$K_{\{i\}} = \langle e_{j,j+1}(a+bt) : j \in \{1,\ldots,4\} \setminus \{i+1\}, a,b \in \mathbb{F}_q \rangle.$$

Define $X_{link,q}$ to be the coset complex $X_{link,q} = X(G_{link,q}, (K_{\{i\}})_{i\in\{0,1,2\}})$. As noted above, for every $s > 4$, all the 2-dimensional links of $X_q^{(s)}$ are isomorphic to $X_{link,q}$. Also,

▶ **Theorem 18** ([12, Theorems 2.4, 3.5]). *The complex $X_{link,q}$ above is strongly symmetric, namely the group $G_{link,q}$ of unipotent matrices described above acts transitively on the triangles of $X_{link,q}$.*

New coboundary and cosystolic expanders

Finally, we describe how the general machinery we developed can be applied in our construction.

First, by applying Theorem 11 on the family $\{X_q^{(s)}\}_{s\in\mathbb{N}}$ yields the following Corollary:

▶ **Corollary 19.** *Let $\{X_q^{(s)}\}_{s\in\mathbb{N}}$ be the family of n-dimensional simplicial complexes from [11]. Assume there is a constant $\varepsilon' > 0$ such that for every odd q, every s, every $0 \leq k \leq n-2$ and every $\tau \in X(k)$, X_τ is a ε'-coboundary expander. Denote $Y_q^{(s)}$ to be the $(n-1)$-skeleton of $X_q^{(s)}$. Then for any sufficiently large odd prime power q, the family $\{Y_q^{(s)}\}_{s\in\mathbb{N}}$ is a family of bounded degree cosystolic expanders.*

Thus, by this Corollary, in order to prove Theorem 1 it is enough to show that for every odd q, there is a constant $\varepsilon' > 0$ such that for every odd q and every $s \in \mathbb{N}$, the 2-skeleton of the link of every vertex v in $X_q^{(s)}$ is a ε' coboundary expander.

As we noted, the links are strongly transitive coset complexes which we denoted $X_{q,link}$ and described explicitly above. By Theorem 17, in order to bound the coboundary expansion of the links, we need to consider the presentation of their symmetry group $G_{link,q}$ defined above. Generalizing on the work of Biss and Dasgupta [2] we prove the following:

▶ **Theorem 20.** *For any prime power q denote $G_{q,link}, K_{\{0\}}, K_{\{1\}}, K_{\{2\}}$ as above and for every $i \in \{0,1,2\}$, denote R_i to be all the non-trivial relations in the multiplication table of $K_{\{i\}}$. Then*

- $$\sup_{q \text{ odd prime power}} \left(\max_{g \in G_{q,link}} \min \left\{ l : g = g_1 ... g_l \text{ and } g_1, ..., g_l \in \bigcup_i K_{\{i\}} \right\} \right) < \infty.$$
- *For every odd q it holds that*

$$G_{q,link} = \langle \bigcup_i K_{\{i\}} | \bigcup_i R_i \rangle$$

and the Dehn function of this presentation is bounded independently of q.

This Theorem combined with Theorem 17 gives:

▶ **Theorem 21** (First Main Theorem - new explicit two dimensional coboundary expanders)**.** *For every odd prime power q, $X_{link,q}$ is a coboundary expander and $\mathrm{Exp}_0(X), \mathrm{Exp}_1(X)$ are bounded from below by a constant that is independent of q.*

Applying Corollary 19 it follows that:

▶ **Theorem 22** (Second Main Theorem - elementary two dimensional bounded degree cosytolic expanders)**.** *Let $s \in \mathbb{N}, s > 4$ and q be a prime power and $X_q^{(s)}$ as above. For every s, let $Y_q^{(s)}$ be the 2-skeleton of $X_q^{(s)}$, i.e., the 2-dimensional complex $Y_q^{(s)} = X_q^{(s)}(0) \cup X_q^{(s)}(1) \cup X_q^{(s)}(2)$. For any sufficiently large odd prime power q, the family $\{Y_q^{(s)}\}_{s \in \mathbb{N}, s>4}$ is a family of bounded degree cosystolic expanders.*

Technical details of the paper

In Section 1, we give the precise definitions and notations regarding (co)homology. In Section 2, we define the cone radius of a simplicial complex and prove that for symmetric simplicial complexes, the cone radius can be used to bound the coboundary expansion.

The other technical details of the paper are available in its Arxiv version [13]. Namely, in [13] the reader can find the following:

- A precise definition of the filling constants of a simplicial complex and a proof that the filling constants of the complex can be used to bound the cone radius.
- A review the idea of coset complexes and a proof that our assumption of strong symmetry combined with some extra assumptions on a complex imply that it is a coset complex.
- A bound on the first two filling constants for a coset complex in terms of algebraic properties of the presentation of the group and subgroups from which it arises.
- New examples of coboundaries expanders arising from coset complexes of unipotent groups.
- New examples of bounded degree cosystolic and topological expanders.
- A proof that the existence of a cone function is equivalent to the vanishing of (co)homology.

1 Homological and Cohomological definitions and notations

The aim of this section is to recall a few basic definitions regarding homology and cohomology of simplicial complexes that we will need below.

Let X be an n-dimensional simplicial complex. A simplicial complex X is called *pure* if every face in X is contained in some face of size $n+1$. The set of all k-faces of X is denoted $X(k)$, and we will be using the convention in which $X(-1) = \{\emptyset\}$.

We denote by $C_k(X) = C_k(X, \mathbb{F}_2)$ the $\mathbb{F}_2$-vector space with basis $X(k)$ (or equivalently, the $\mathbb{F}_2$-vector space of subsets of $X(k)$), and $C^k(X) = C^k(X, \mathbb{F}_2)$ the $\mathbb{F}_2$-vector space of functions from $X(k)$ to $\mathbb{F}_2$.

The *boundary map* $\partial_k : C_k(X, \mathbb{F}_2) \to C_{k-1}(X, \mathbb{F}_2)$ is:

$$\partial_k(\sigma) = \sum_{\tau \subset \sigma, |\tau| = |\sigma|-1} \tau,$$

where $\sigma \in X(k)$, and the *coboundary map* $d_k : C^k(X, \mathbb{F}_2) \to C^{k+1}(X, \mathbb{F}_2)$ is:

$$d_k(\phi)(\sigma) = \sum_{\tau \subset \sigma, |\tau| = |\sigma|-1} \phi(\tau),$$

where $\phi \in C^k$ and $\sigma \in X(k+1)$.

For $A \in C_k(X)$ and $\phi \in C^k(X)$, we denote

$$\phi(A) = \sum_{\tau \in A} \phi(\tau),$$

Thus, for $\phi \in C^k(X)$ and $A \in C_{k+1}(X)$

$$(d_k \phi)(A) = \phi(\partial_{k+1} A)$$

We sometimes refer to k-chains as subsets of $X(k)$, e.g., the 0-chain $\{u\} + \{v\}$ will be sometimes referred to as the set $\{\{u\}, \{v\}\}$. For $A \in C_k(X)$, we denote $|A|$ to be the size of A as a set.

Well known and easily calculated equations are:

$$\partial_k \circ \partial_{k+1} = 0 \text{ and } d_{k+1} \circ d_k = 0 \tag{1}$$

Thus, if we denote: $B_k(X) = B_k(X, \mathbb{F}_2) = \text{Image}(\partial_{k+1}) =$ the space of k-boundaries.
$Z_k(X) = Z_k(X, \mathbb{F}_2) = \text{Ker}(\partial_{k+1}) =$ the space of k-cycles.
$B^k(X) = B^k(X, \mathbb{F}_2) = \text{Image}(d_{k-1}) =$ the space of k-coboundaries.
$Z^k(X) = Z^k(X, \mathbb{F}_2) = \text{Ker}(d_k) =$ the space of k-cocycles.
We get from (1)

$$B_k(X) \subseteq Z_k(X) \subseteq C_k(X) \text{ and } B^k(X) \subseteq Z^k(X) \subseteq C^k(X).$$

Define the quotient spaces $\widetilde{H}_k(X) = Z_k(X)/B_k(X)$ and $\widetilde{H}^k(X) = Z^k(X)/B^k(X)$, the k-homology and the k-cohomology groups of X (with coefficients in $\mathbb{F}_2$).

2 Cone radius as a bound on coboundary expansion

Below, we define a generalized notion of diameter (or more precisely radius) of a simplicial complex. We will later show that in symmetric simplicial complexes a bound on this radius yields a bound on the coboundary expansion of the complex.

▶ **Definition 23** (Cone function). *Let X be a pure n-dimensional simplicial complex. Let $-1 \leq k \leq n-1$ be a constant and v be a vertex of X. A k-cone function with apex v is a linear function $\mathrm{Cone}_k^v : \bigoplus_{j=-1}^{k} C_j(X) \to \bigoplus_{j=-1}^{k} C_{j+1}(X)$ defined inductively as follows:*

1. *For $k=-1$, $\mathrm{Cone}_{-1}^v(\emptyset) = \{v\}$.*
2. *For $k \geq 0$, $\mathrm{Cone}_k^v|_{\bigoplus_{j=-1}^{k-1} C_j(X)}$ is a $(k-1)$-cone function with an apex v and for every $A \in C_k(X)$, $\mathrm{Cone}_k^v(A) \in C^{k+1}(X)$ is a $(k+1)$-chain that fulfills the equation*

$$\partial_{k+1} \mathrm{Cone}_k^v(A) = A + \mathrm{Cone}_k^v(\partial_k A).$$

▶ **Observation 24.** *By linearity, the condition that*

$$\partial_{k+1} \mathrm{Cone}_k^v(A) = A + \mathrm{Cone}_k^v(\partial_k A), \forall A \in C^k(X)$$

is equivalent to the condition:

$$\partial_{k+1} \mathrm{Cone}_k^v(\tau) = \tau + \mathrm{Cone}_k^v(\partial_k \tau), \forall \tau \in X(k).$$

▶ Remark 25. We note that by linearity, a k-cone function is needs only to be defined on k-simplices, but it gives us homological fillings for every k-cycle in X: for every $A \in Z_k(X)$,

$$\partial_{k+1} \mathrm{Cone}_k^v(A) = A + \mathrm{Cone}_k^v(\partial_k A) = A + \mathrm{Cone}_k^v(0) = A,$$

i.e., $\partial_{k+1} \mathrm{Cone}_k^v(A) = A$. This might be computationally beneficial for other needs (apart from the results of this paper), since usually there are exponentially more k-cycles than k-simplices.

▶ **Example 26** (0-cone example). Let X be an n-dimensional simplicial complex. Fix some vertex v in X. By definition, for every $\{u\} \in X(0)$, $\mathrm{Cone}_0^v(\{u\})$ is a 1-chain such that $\partial_0 \mathrm{Cone}_0^v(\{u\}) = \{u\} + \{v\}$.

If the 1-skeleton of X is connected, we can define $\mathrm{Cone}_0^v(\{u\})$ to be a 1-chain that consists of a sum of edges that form a path between $\{u\}$ and $\{v\}$. If the 1-skeleton of X is not connected, a 0-cone function does not exist: for $\{u\} \in X(0)$ that is not in the connected component of $\{v\}$, $\mathrm{Cone}_0^v(\{u\})$ cannot be defined. Assuming that the 1-skeleton of X is connected, we note that the construction of Cone_0^v is usually not unique: different choices of paths between $\{u\}$ and $\{v\}$ give different 0-cone functions.

▶ **Example 27** (1-cone example). Let X be an n-dimensional simplicial complex. Assume that the 1-skeleton of X is connected and define a 0-cone function as in the example above and define Cone_2^v on $C_0(X)$ as that 0-cone function. We note that for every $\{u,w\} \in X(1)$, $\{u,w\} + \mathrm{Cone}_0^v(\{u\}) + \mathrm{Cone}_0^v(\{w\})$ forms a closed path, i.e., a 1-cycle, in X. If $\widetilde{H}_1(X) = 0$, we can deduce that $\{u,w\} + \mathrm{Cone}_1^v(\{u\}) + \mathrm{Cone}_1^v(\{w\})$ is a boundary. Therefore, for every $\{u,w\} \in X(1)$, we can choose $\mathrm{Cone}_1^v(\{u,w\}) \in X(2)$ such that

$$\partial_2 \mathrm{Cone}_1^v = \{u,w\} + \mathrm{Cone}_1^v(\{u\}) + \mathrm{Cone}_1^v(\{w\}).$$

▶ **Definition 28** (Cone radius). *Let X be an n-dimensional simplicial complex, $-1 \leq k \leq n-1$ and v a vertex of X. Given a k-cone function Cone_k^v define the volume of Cone_k^v as*

$$\mathrm{Vol}(\mathrm{Cone}_k^v) = \max_{\tau \in X(k)} |\mathrm{Cone}_k^v(\tau)|.$$

Define the k-th cone radius of X to be

$$\mathrm{Crad}_k(X) = \min\{\mathrm{Vol}(\mathrm{Cone}_k^v) : \{v\} \in X(0), \mathrm{Cone}_k^v \text{ is a } k\text{-cone function}\}.$$

If k-cone functions do not exist, we define $\mathrm{Crad}(X) = \infty$.

▶ Remark 29. The reason for the name "cone radius" is that in the case where $k = 0$, $\mathrm{Crad}_0(X)$ is exactly the (graph) radius of the 1-skeleton of X. Indeed, for $k = 0$, choose $\{v\} \in X(0)$ such that for every $\{v'\} \in X(0)$,

$$\max_{\{u\} \in X(0)} \mathrm{dist}(v, u) \leq \max_{\{u\} \in X(0)} \mathrm{dist}(v', u),$$

where dist denotes the path distance. For such a $\{v\} \in X(0)$, define $\mathrm{Cone}_0^v(\{u\})$ to be the edges of a shortest path between v and u. By our choice of v, it follows that $\mathrm{Vol}(\mathrm{Cone}_0^v)$ is the radius of the one-skeleton of X and we leave it to the reader to verify that this choice gives $\mathrm{Crad}_0(X) = \mathrm{Vol}(\mathrm{Cone}_0^v)$.

The main result of this section is that in a symmetric simplicial complex X, the k-th cone radius gives a lower bound on $\mathrm{Exp}_b^k(X)$:

▶ **Theorem 30.** *Let X be a pure finite n-dimensional simplicial complex. Assume that X is strongly symmetric, i.e., that there is a group G of automorphisms of X acting transitively on $X(n)$. For every $0 \leq k \leq n-1$, if $\mathrm{Crad}_k(X) < \infty$, then $\mathrm{Exp}_b^k(X) \geq \frac{1}{\binom{n+1}{k+1}\mathrm{Crad}_k(X)}$.*

Theorem 30 stated above generalizes a result of of Lubotzky, Meshulam and Mozes [17] in which coboundary expansion was proven for symmetric simplicial complexes given that they are "building-like", i.e., that they have have sub-complexes that behave (in some sense) as apartments in a Bruhat-Tits building.

We note that the notion of a cone function is already evident in Gromov's original work [8]. Gromov considered what he called "random cones", which was a probability over a family of cone functions and show that the expectancy of the occurrence of a simplex in the support of this family bounds $\mathrm{Exp}_b^k(X)$ (see also the work of Kozlv and Meshulam [14, Theorem 2.5]). Using Gromov's terminology, in the proof of the Theorem above, we show that under the assumption of symmetry a single cone function yields a family of random cones and the needed expectancy is bounded by the cone radius. In the sake of completeness, we will not prove the Theorem without using Gromov's results.

In order to prove Theorem 30, we will need some additional lemmas.

▶ **Lemma 31.** *For $-1 \leq k \leq n-1$ and a k-cone function Cone_k^v with apex v. Define the contraction operator $\iota_{\mathrm{Cone}_k^v}$,*

$$\iota_{\mathrm{Cone}_k^v} : \bigoplus_{j=-1}^{k} C^{j+1}(X) \to \bigoplus_{j=-1}^{k} C^j(X)$$

as follows: for $\phi \in C^{j+1}(X)$ and $A \in C_j(X)$, we define

$$(\iota_{\mathrm{Cone}_k^v}\phi)(A) = \phi(\mathrm{Cone}_k^v(A)).$$

Then for every $\phi \in C^k(X)$,

$$\iota_{\mathrm{Cone}_k^v} d_k \phi = \phi + d_{k-1}\iota_{\mathrm{Cone}_k^v}\phi.$$

Proof. Let $A \in C_k(X)$, then

$$\iota_{\mathrm{Cone}_k^v} d_k \phi(A) = (d_k\phi)(\mathrm{Cone}_k^v(A)) =$$
$$\phi(\partial_{k+1}(\mathrm{Cone}_k^v(A))) = \phi(A + (\mathrm{Cone}_k^v(\partial_k A))) =$$
$$\phi(A) + (\iota_{\mathrm{Cone}_k^v}\phi)(\partial_k A) = \phi(A) + (d_{k-1}\iota_{\mathrm{Cone}_k^v}\phi)(A),$$

as needed. ◀

Naively, it might seem that this Lemma gives a direct approach towards bounding the coboundary expansion: if one could find is some constant $C = C(n, k, \mathrm{Crad}_k(X))$ such that $w(\iota_{\mathrm{Cone}_k^v} d_k\phi) \leq C w(d_k\phi)$, then for every ϕ,

$$\frac{w(d_k\phi)}{\min_{\psi \in B^k(X)} w(\phi+\psi)} \geq \frac{1}{C}\frac{w(\iota_{\mathrm{Cone}_k^v} d_k\phi)}{w(\phi + d_{k-1}\iota_{\mathrm{Cone}_k^v}\phi)} = \frac{1}{C}.$$

However, by Remark 13, we note that without symmetry, the existence of a k-cone function cannot give an effective bound on the coboundary expansion.

Our proof strategy below is to improve on this naive idea by using the symmetry of X: we will show that for a group G that acts on X, the group G also acts on k-cone functions and we will denote this action by ρ. We then show that when G acts transitively on $X(n)$, we can average the action on the k-cone function that realizes the cone radius and deduce that

$$\frac{1}{|G|}\sum_{g\in G} w(\iota_{\rho(g).\,\mathrm{Cone}_k^v} d_k\phi) \leq \binom{n+1}{k+1}\mathrm{Crad}_k(X) w(d_k\phi).$$

Thus, using an averaged version of the naive argument above will get a bound on the coboundary expansion.

We start by defining an action on k-cone functions. Assume that G is a group acting simplicially on X. For every $g \in G$ and every k-cone function Cone_k^v define

$$(\rho(g).\,\mathrm{Cone}_k^v)(A) = g.(\mathrm{Cone}_k^v(g^{-1}.A)), \forall A \in \bigoplus_{j=-1}^{k} C_j(X).$$

▶ **Lemma 32.** *For $g \in G$, $-1 \leq k \leq n-1$ and a k-cone function Cone_k^v with apex v, $\rho(g).\,\mathrm{Cone}_k^v$ is a k-cone function with apex $g.v$ and $\mathrm{Vol}(g.\,\mathrm{Cone}_k^v) = \mathrm{Vol}(\mathrm{Cone}_k^v)$. Moreover, ρ defines an action of G on the set of k-cone functions.*

Proof. If we show that $g.\,\mathrm{Cone}_k^v$ is a k-cone function the fact that $\mathrm{Vol}(g.\,\mathrm{Cone}_k^v) = \mathrm{Vol}(\mathrm{Cone}_k^v)$ will follow directly from the fact that G acts simplicially.

The proof that $\rho(g).\,\mathrm{Cone}_k^v$ is a k-cone function is by induction on k. For $k = -1$,

$$(\rho(g).\,\mathrm{Cone}_{v,-1})(\emptyset) = g.\,\mathrm{Cone}_{v,-1}(g^{-1}.\emptyset) = g.\,\mathrm{Cone}_{v,-1}(\emptyset) = g.\{v\} = \{g.v\},$$

then $\rho(g).\,\mathrm{Cone}_{v,-1}$ is a (-1)-cone function with an apex $g.v$.

Assume the assertion of the lemma holds for $k-1$. Thus, $\rho(g).\,\mathrm{Cone}_k^v|_{\bigoplus_{j=-1}^{k-1} C_j(X)}$ is a $(k-1)$-cone function with an apex $g.v$ and, by Observation 24, we are left to check that for every $\tau \in X(k)$,

$$\partial_{k+1}(\rho(g).\,\mathrm{Cone}_k^v)(\tau) = \tau + \rho(g).\,\mathrm{Cone}_k^v(\partial_k\tau).$$

Note that the G acts simplicially on X and thus the action of G commutes with the ∂ operator. Therefore, for every $\tau \in X(k)$,

$$\partial_{k+1}(\rho(g).\operatorname{Cone}_k^v)(\tau) = \partial_{k+1}(g.(\operatorname{Cone}_k^v(g^{-1}.\tau))) = g.\left(\partial_{k+1}\operatorname{Cone}_k^v(g^{-1}.\tau)\right) = g.\left(g^{-1}.\tau + \operatorname{Cone}_k^v(\partial_k g^{-1}.\tau)\right) = \tau + g.\operatorname{Cone}_k^v(g^{-1}.\partial_k\tau) = \tau + \rho(g).\operatorname{Cone}_k^v(\partial_k\tau).$$

The fact that ρ is an action is straight-forward and left for the reader. ◀

Applying our proof strategy above, will lead us to consider the constant $\theta(\eta)$ defined in the Lemma below.

▶ **Lemma 33.** *Assume that G is a group acting simplicially on X and that this action is transitive on n-simplices. Let $0 \le k \le n-1$ and assume that $\operatorname{Crad}_k(X) < \infty$. Fix Cone_k^v to be a k-cone function such that $\operatorname{Crad}_k(X) = \operatorname{Vol}(\operatorname{Cone}_k^v)$. For every $\eta \in X(k+1)$, denote*

$$\theta(\eta) = \frac{1}{w(\eta)|G|}\sum_{g\in G}\sum\{w(\tau) : \tau \in X(k), \eta \in (\rho(g).\operatorname{Cone}_k^v)(\tau))\}.$$

Then for every $\eta \in X(k+1)$, $\theta(\eta) \le \binom{n+1}{k+1}\operatorname{Crad}_k(X)$.

Proof. Fix some $\eta \in X(k+1)$. First, we note that G acts transitively on $X(n)$ and therefore $\bigcup_{gG_\eta}\{g.\sigma : \sigma \in X(n), \eta \subseteq \sigma\} = X(n)$. This yields that

$$|X(n)| \le \frac{|G|}{|G_\eta|}|\{\sigma \in X(n) : \eta \subseteq \sigma\}|,$$

and therefore $|G_\eta| \le |G|\binom{n+1}{k+1}w(\eta)$.

Second, we note that for every $g \in G$, and every $\eta \in X(k+1)$,

$$\eta \in (\rho(g).\operatorname{Cone}_k^v)(\tau) \Leftrightarrow \eta \in g.(\operatorname{Cone}_k^v(g^{-1}.\tau)) \Leftrightarrow g^{-1}.\eta \in \operatorname{Cone}_k^v(g^{-1}.\tau).$$

Thus,

$$\begin{aligned}
\theta(\eta) &= \frac{1}{w(\eta)|G|}\sum_{g\in G}\sum\{w(\tau) : \tau \in X(k), \eta \in (\rho(g).\operatorname{Cone}_k^v)(\tau))\} = \\
&\frac{1}{w(\eta)|G|}\sum_{g\in G}\sum\{w(\tau) : \tau \in X(k), g^{-1}.\eta \in \operatorname{Cone}_k^v(g^{-1}.\tau)\} = \\
&\frac{1}{w(\eta)|G|}\sum_{g\in G}\sum\{w(g^{-1}.\tau) : \tau \in X(k), g^{-1}.\eta \in \operatorname{Cone}_k^v(g^{-1}.\tau)\} = \\
&\frac{1}{w(\eta)|G|}\sum_{g\in G}\sum\{w(\tau) : \tau \in X(k), g^{-1}.\eta \in \operatorname{Cone}_k^v(\tau)\} = \\
&\frac{1}{w(\eta)|G|}\sum_{g\in G}\sum_{\tau\in X(k)}\sum_{g^{-1}.\eta\in\operatorname{Cone}_k^v(\tau)} w(\tau) = \\
&\frac{1}{w(\eta)}\sum_{\tau\in X(k)} w(\tau)\sum_{g\in G}\sum_{g^{-1}.\eta\in\operatorname{Cone}_k^v(\tau)}\frac{1}{|G|} \le \\
&\frac{1}{w(\eta)}\sum_{\tau\in X(k)} w(\tau)|\operatorname{Cone}_k^v(\tau)|\frac{|G_\eta|}{|G|} \le \\
&\frac{1}{w(\eta)}\sum_{\tau\in X(k)} w(\tau)\operatorname{Crad}_k(X)\binom{n+1}{k+1}w(\eta) = \\
&\binom{n+1}{k+1}\operatorname{Crad}_k(X)\sum_{\tau\in X(k)} w(\tau) = \binom{n+1}{k+1}\operatorname{Crad}_k(X),
\end{aligned}$$

as needed. ◀

We turn now to prove Theorem 30:

Proof. Assume that G is a group acting simplicially on X such that the action is transitive on $X(n)$. Let $0 \leq k \leq n-1$ and assume that $\mathrm{Crad}_k(X) < \infty$. For $\phi \in C^k(X)$, we denote

$$[\phi] = \{\phi + \psi : \psi \in B^k(X)\}, \text{ and } w([\phi]) = \min_{\phi' \in [\phi]} w(\phi').$$

Thus, we need to prove that for every $\phi \in C^k(X)$,

$$\frac{1}{\binom{n+1}{k+1}\mathrm{Crad}_k(X)} w([\phi]) \leq w(d_k\phi),$$

or equivalently,

$$|G|w([\phi]) \leq |G|w(d_k\phi)\left(\binom{n+1}{k+1}\mathrm{Crad}_k(X)\right).$$

Fix Cone_k^v to be a k-cone function such that $\mathrm{Crad}_k(X) = \mathrm{Vol}(\mathrm{Cone}_k^v)$. By Lemma 32, for every $g \in G$, $\rho(g).\mathrm{Cone}_k^v$ is a k-cone function.

In the notation of Lemma 31, for every $g \in G$, denote $\iota_g = \iota_{\rho(g).\mathrm{Cone}_k^v}$. By Lemma 31, for every $g \in G$, $w([\phi]) \leq w(\iota_g d_k \phi)$ and therefore

$$\begin{aligned}
|G|w([\phi]) &\leq \sum_{g\in G} w(\iota_g d_k\phi) = \sum_{g\in G}\sum \{w(\tau) : \tau \in \mathrm{supp}(\iota_g d_k\phi)\} = \\
&\sum_{g\in G}\sum \{w(\tau) : \tau \in X(k), d_k\phi(\rho(g).\mathrm{Cone}_k^v(\tau)) = 1\} \leq \\
&\sum_{g\in G}\sum \{w(\tau) : \tau \in X(k), \mathrm{supp}(d_k\phi) \cap \rho(g).\mathrm{Cone}_k^v(\tau) \neq \emptyset\} \leq \\
&\sum_{\eta\in\mathrm{supp}(d_k\phi)}\sum_{g\in G}\sum \{w(\tau) : \tau \in X(k), \eta \in \rho(g).\mathrm{Cone}_k^v(\tau)\} = \\
\sum_{\eta\in\mathrm{supp}(d_k\phi)} |G|w(\eta)\theta(\eta) &\leq^{\text{Lemma 33}} |G|w(d_k\phi)\left(\binom{n+1}{k+1}\mathrm{Crad}_k(X)\right),
\end{aligned}$$

as needed. ◀

The converse of Theorem 30 is also true, i.e., the existence of a cone function is equivalent to vanishing of (co)homology and thus to coboundary expansion (for a proof see [13]).

References

1. Nima Anari, Kuikui Liu, Shayan Oveis Gharan, and Cynthia Vinzant. Log-concave polynomials II: High-dimensional walks and an FPRAS for counting bases of a matroid, 2019. arXiv:1811.01816.
2. Daniel K. Biss and Samit Dasgupta. A presentation for the unipotent group over rings with identity. *J. Algebra*, 237(2):691–707, 2001. doi:10.1006/jabr.2000.8604.
3. Fan R. K. Chung. *Spectral graph theory*, volume 92 of *CBMS Regional Conference Series in Mathematics*. Published for the Conference Board of the Mathematical Sciences, Washington, DC; by the American Mathematical Society, Providence, RI, 1997.
4. Irit Dinur, Prahladh Harsha, Tali Kaufman, Inbal Livni Navon, and Amnon Ta Shma. List decoding with double samplers. In *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 2134–2153. SIAM, Philadelphia, PA, 2019. doi:10.1137/1.9781611975482.129.

5 Irit Dinur and Tali Kaufman. High dimensional expanders imply agreement expanders. In *58th Annual IEEE Symposium on Foundations of Computer Science—FOCS 2017*, pages 974–985. IEEE Computer Soc., Los Alamitos, CA, 2017.

6 Shai Evra and Tali Kaufman. Bounded degree cosystolic expanders of every dimension. In *STOC'16—Proceedings of the 48th Annual ACM SIGACT Symposium on Theory of Computing*, pages 36–48. ACM, New York, 2016. doi:10.1145/2897518.2897543.

7 Shai Evra, Tali Kaufman, and Gilles Zémor. Decodable quantum ldpc codes beyond the sqrtn distance barrier using high dimensional expanders. In *FOCS 2020*, 2020.

8 Mikhail Gromov. Singularities, expanders and topology of maps. Part 2: From combinatorics to topology via algebraic isoperimetry. *Geom. Funct. Anal.*, 20(2):416–526, 2010. doi:10.1007/s00039-010-0073-8.

9 Anna Gundert and Uli Wagner. On Laplacians of random complexes. In *Computational geometry (SCG'12)*, pages 151–160. ACM, New York, 2012. doi:10.1145/2261250.2261272.

10 Tali Kaufman and Alexander Lubotzky. High dimensional expanders and property testing. In *ITCS'14—Proceedings of the 2014 Conference on Innovations in Theoretical Computer Science*, pages 501–506. ACM, New York, 2014.

11 Tali Kaufman and Izhar Oppenheim. Construction of new local spectral high dimensional expanders. In *STOC'18—Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing*, pages 773–786. ACM, New York, 2018.

12 Tali Kaufman and Izhar Oppenheim. Simplicial complexes arising from elementary matrix groups and high dimensional expanders. *European Jouranl of Combinatorics. In press*, 2020. arXiv:1710.05304.

13 Tali Kaufman and Izhar Oppenheim. Coboundary and cosystolic expansion from strong symmetry, 2021. arXiv:1907.01259.

14 Dmitry N. Kozlov and Roy Meshulam. Quantitative aspects of acyclicity. *Res. Math. Sci.*, 6(4):Paper No. 33, 32, 2019. doi:10.1007/s40687-019-0195-z.

15 Nathan Linial and Roy Meshulam. Homological connectivity of random 2-complexes. *Combinatorica*, 26(4):475–487, 2006. doi:10.1007/s00493-006-0027-9.

16 Alexander Lubotzky. High dimensional expanders, 2017. arXiv:1712.02526.

17 Alexander Lubotzky, Roy Meshulam, and Shahar Mozes. Expansion of building-like complexes. *Groups Geom. Dyn.*, 10(1):155–175, 2016. doi:10.4171/GGD/346.

18 R. Meshulam and N. Wallach. Homological connectivity of random k-dimensional complexes. *Random Structures Algorithms*, 34(3):408–417, 2009. doi:10.1002/rsa.20238.

19 John Steenbergen, Caroline Klivans, and Sayan Mukherjee. A Cheeger-type inequality on simplicial complexes. *Adv. in Appl. Math.*, 56:56–77, 2014. doi:10.1016/j.aam.2014.01.002.

Maximum Matchings and Popularity

Telikepalli Kavitha
Tata Institute of Fundamental Research, Mumbai, India

Abstract

Let G be a bipartite graph where every node has a strict ranking of its neighbors. For any node, its preferences over neighbors extend naturally to preferences over matchings. A maximum matching M in G is a *popular max-matching* if for any maximum matching N in G, the number of nodes that prefer M is at least the number that prefer N. Popular max-matchings always exist in G and they are relevant in applications where the size of the matching is of higher priority than node preferences. Here we assume there is also a cost function on the edge set. So what we seek is a min-cost popular max-matching. Our main result is that such a matching can be computed in polynomial time.

We show a compact extended formulation for the popular max-matching polytope and the algorithmic result follows from this. In contrast, it is known that the popular matching polytope has near-exponential extension complexity and finding a min-cost popular matching is NP-hard.

2012 ACM Subject Classification Theory of computation → Design and analysis of algorithms

Keywords and phrases Bipartite graphs, Popular matchings, Stable matchings, Dual certificates

Digital Object Identifier 10.4230/LIPIcs.ICALP.2021.85

Category Track A: Algorithms, Complexity and Games

Related Version *Full Version*: `https://arxiv.org/abs/2011.03434`

Funding *Telikepalli Kavitha*: Supported by the Department of Atomic Energy, Government of India, under project no. RTI4001.

Acknowledgements Thanks to Yuri Faenza and the reviewers for helpful comments and suggestions on the presentation.

1 Introduction

We consider a matching problem in a bipartite graph $G = (A \cup B, E)$ on n nodes and m edges where every node has a strict ranking of its neighbors. The bipartite graph G need not be complete. This is a very well-studied model in two-sided matching markets. This model has been used to match students to schools and colleges [1, 4] and doctors to residencies in hospitals [6, 28].

The goal is to find an optimal matching in G. The classical notion of optimality in such an instance is *stability*. A matching M is stable if there is no edge that *blocks* M; an edge (a, b) blocks M if a and b prefer each other to their respective assignments in M. Stable matchings always exist in G and one such matching can be computed in linear time by the Gale-Shapley algorithm [17].

In several applications, along with node preferences, the definition of optimality may involve attributes such as size, e.g., consider the problem of assigning doctors to hospitals during a pandemic. For instance, during the Covid-19 pandemic in Mumbai, public hospitals were overwhelmed with a rising number of patients and were severely short-staffed; so doctors associated with private clinics were asked to also work in public hospitals[1] [3, 22]. We want the maximum number of doctors to get assigned to hospitals, i.e., we do not wish to

[1] In the doctors-hospitals setting, one typically seeks a many-to-one matching since a hospital may need several doctors. Here we focus on one-to-one matchings.

compromise at all on the size of our matching. Thus the size of the matching is of higher priority here than node preferences. We refer to [5] for more such applications: these include school placement [1] and the assignment of sailors to billets [29, 34]. So the set of admissible solutions in all these applications is the set of maximum matchings and among all maximum matchings, we wish to find a *best* matching as per node preferences.

Note that a stable matching need not have maximum size. It is known that all stable matchings match the same set of nodes [18]. When stable matchings are not admissible, a natural alternative would be to seek a maximum matching with the least number of blocking edges. However this is an NP-hard problem [5] and as shown there, this is NP-hard to approximate within $n^{1-\varepsilon}$, for any $\varepsilon > 0$.

Observe that stability empowers every edge with the "veto power" to block a matching. Thus stability is a very strong notion and the notion of *popularity* is a meaningful relaxation of stability that captures collective welfare. We say node u prefers matching M to matching N if u prefers its assignment in M to its assignment in N; being unmatched is the worst choice for any node. We can compare any pair of matchings M and N by holding an election between them where nodes are voters.

Let $\phi(M, N)$ be the number of nodes that prefer M to N and similarly, let $\phi(N, M)$ be the number of nodes that prefer N to M. We say N is more popular than M if $\phi(N, M) > \phi(M, N)$.

▶ **Definition 1.** *A matching M is popular if $\Delta(M, N) \geq 0$ for all matchings N in G, where $\Delta(M, N) = \phi(M, N) - \phi(N, M)$.*

Thus a matching M is popular if there is no matching that is more popular than M. Every stable matching is popular[2] [19]. One of the main merits of popularity is that it allows larger matchings compared to stable matchings. Since a stable matching is a maximal matching, its size is at least $|M_{\max}|/2$ (where $M_{\max}$ is a maximum matching in G) and there are easy examples where this bound is tight. It is known that there is always a popular matching of size at least $2|M_{\max}|/3$ and there are simple instances with no larger popular matching [23].

Since a popular matching need not be maximum, a natural alternative is to ask for a maximum matching M that is popular *within* the set of maximum matchings, i.e., no maximum matching is more popular than M. Since it is only maximum matchings that are admissible and we are not willing to replace a maximum matching with a smaller one, elections that involve *non-maximum* or inadmissible matchings are not relevant here. Hence a natural candidate for a best maximum matching is a *popular max-matching* defined below.

▶ **Definition 2.** *Call a maximum matching M in $G = (A \cup B, E)$ a popular max-matching if $\Delta(M, N) \geq 0$ for all maximum matchings N in G.*

Thus what we seek is a *weak Condorcet winner* [7, 27] in the voting instance where maximum matchings are candidates and nodes are voters. The relation "more popular than" is not transitive and weak Condorcet winners need not exist in every voting instance. The question of whether every instance admits a popular max-matching was considered in [23] where it was shown that popular max-matchings always exist in $G = (A \cup B, E)$ and one such matching can be computed in $O(mn)$ time.

[2] In an election between a stable matching S and any matching M, if node u prefers M to S then the node $M(u)$ has to prefer S to M, otherwise $(u, M(u))$ blocks S, which is forbidden. Hence $\phi(M, S) \leq \phi(S, M)$.

In several applications there is a cost function $c : E \to \mathbb{R}$ and for any matching M, its cost $c(M) = \sum_{e \in M} c(e)$. So among all popular max-matchings in G, what we seek is a *min-cost* popular max-matching. There are no previous algorithmic/hardness results known for this problem. We know that it is NP-hard to find a min-cost popular matching [13].

Finding a min-cost popular max-matching is a natural and interesting problem in discrete optimization. Solving this problem efficiently implies efficient algorithms for a whole host of popular max-matching problems such as finding one with forced/forbidden edges or one with max-utility or one with min-regret. In general, a cost function allows us to "access" the entire set of popular max-matchings; note that G may have more than 2^n such matchings [32].

Let $\mathcal{M}_G$ denote the popular max-matching polytope of G, i.e., this polytope is the convex hull of the edge incidence vectors of popular max-matchings in G. A compact description of $\mathcal{M}_G$ (or some extension[3] of it) implies a polynomial time algorithm to compute a min-cost popular max-matching. Our main result is that the polytope $\mathcal{M}_G$ has a compact extended formulation. So unlike the min-cost popular matching problem, interestingly and quite surprisingly, the min-cost popular max-matching problem is tractable.

▶ **Theorem 3.** *Given $G = (A \cup B, E)$ where nodes have strict preferences and $c : E \to \mathbb{R}$, a min-cost popular max-matching can be computed in polynomial time.*

So Theorem 3 shows that a natural variant of the min-cost popular matching problem (which is NP-hard) admits a polynomial time algorithm. We also consider *Pareto-optimality* – this is a far more relaxed notion than popularity that any reasonable matching in this domain should satisfy. If M is a matching that is not Pareto-optimal then there is a matching N such that no node is worse-off in N than in M and at least one node is better-off.

The *unpopularity factor* of M is defined as follows [26]: $u(M) = \max_{N \neq M} \frac{\phi(N,M)}{\phi(M,N)}$. Observe that a popular matching M satisfies $u(M) \leq 1$. A matching M is Pareto-optimal if $u(M) < \infty$. A maximum matching M that satisfies $u(M) < \infty$ is a Pareto-optimal max-matching. We show the following hardness result here.

▶ **Theorem 4.** *Given $G = (A \cup B, E)$ with strict preferences and edge costs in $\{0, 1\}$, it is NP-hard to compute a min-cost Pareto-optimal matching/max-matching in G. Moreover, it is NP-hard to approximate this within any multiplicative factor.*

1.1 Background

The notion of popularity was introduced by Gärdenfors [19] in 1975 where he observed that every stable matching is popular. Many algorithmic questions in popular matchings have been investigated in the last 10-15 years and we refer to [8] for a survey. In the domain of popular matchings with two-sided preferences (every node has a preference order ranking its neighbors), other than a handful of positive results [10, 20, 23, 25], most optimization problems have turned out to be NP-hard [13].

Computing a min-cost *quasi-popular* matching M, i.e., $u(M) \leq 2$, is also NP-hard [12]. Compact extended formulations for the *dominant* matching[4] polytope [10, 12] and the popular fractional matching polytope [24] are known but the popular matching polytope has near-exponential extension complexity [12].

[3] A polytope Q that linearly projects to a polytope P is an *extension* of P and a linear description of Q is an *extended formulation* for P. The minimum size of an extension of P is the *extension complexity* of P.

[4] These are popular matchings that are more popular than all larger matchings.

Though an $O(mn)$ time algorithm to find a popular max-matching in G is known [23], there are no previous results on its *optimization* variant, i.e., to find a min-cost or max-utility popular max-matching. It is common in this domain to have an efficient algorithm to find a max-size matching in some class – say, popular matchings [20, 23] or Pareto-optimal matchings for one-sided preferences (only nodes in A have preferences) studied in [2], however finding a min-cost matching in these classes is NP-hard [2, 13]; Theorem 4 also shows such a hardness result. Thus an efficient algorithm to find some popular max-matching was no guarantee on the tractability of the min-cost popular max-matching problem.

There are several polynomial time algorithms to compute a min-cost stable matching and some variants of this problem [11, 14, 15, 16, 21, 30, 31, 33]. Moreover, the stable matching polytope of G has a linear-size description in $\mathbb{R}^m$ [30]. Thus in contrast to stable matchings, the landscape of popular matchings has only a few positive results.

1.2 Our Techniques

An algorithm called the "$|A|$-level Gale-Shapley algorithm" was given in [23] to find a popular max-matching in $G = (A \cup B, E)$. As we show in Section 2, this algorithm is equivalent to running the Gale-Shapley algorithm in an auxiliary instance G^* with $|A|$ copies of each node in A. The proof in [23] can be easily adapted to show that there is a map from the set of stable matchings in G^* to the set of popular max-matchings in G (see Theorem 6). Our novel contribution here is to show that this map is *surjective*, i.e., every popular max-matching in G is the image of a stable matching in G^*.

Loosely speaking, the instance G^* is made up of $|A|$ copies of G and is inspired by an instance from [10] that is made up of two copies of G. So our instance G^* is much larger than the instance used in [10]. In order to realize a popular max-matching in G as the image of a stable matching in G^*, rather than be guided by blocking edges (as done in [10]), we need a "global handle" over the given popular max-matching, i.e., we seek a function from $A \cup B$ to $\{0, \ldots, |A| - 1\}$ that guides us in how to "place" this matching in the instance G^*.

LP-duality gives us such a handle in terms of dual certificates. Dual certificates for popular matchings are well-understood: these are in $\{0, \pm 1\}^n$ [24]. To show a compact extended formulation for $\mathcal{M}_G$ (unlike the popular matching polytope), finding the right dual certificates is crucial. Our proof consists of two parts: when G admits a perfect matching, it is the easy case. We use total unimodularity, complementary slackness, and the fact that G has a perfect matching to show that the given popular max-matching M has a dual certificate $\vec{\alpha}$ where $\alpha_a \in \{0, -2, -4, \ldots\}$ for $a \in A$ and $\alpha_b \in \{0, 2, 4, \ldots\}$ for $b \in B$. Such an $\vec{\alpha}$ can be neatly used to realize M as the image of a stable matching in G^* (see Theorem 8).

The general case. When G does not admit a perfect matching, things are more complicated. The primal LP will not be as simple as (LP1) whose constraints describe the perfect matching polytope. So we reduce the general case to the case when M is a perfect matching, i.e., we use the dual solution $\vec{\alpha} \in \{0, \pm 2, \pm 4, \ldots\}^{2n_0}$ that certifies M's optimality in the subgraph $G' = G \setminus \{\text{nodes not matched in } M\}$ on $2n_0 = 2|M|$ nodes.

We need to update $\vec{\alpha}$ so that it certifies M's optimality in the entire graph G. Our main technical novelty is in how we update $\vec{\alpha}$ using certain rules (see Theorem 9). Let $A' \cup B'$ be the node set of G' and let U be the set of nodes not matched in M.

We use the fact that M is a maximum matching to prove that our update procedure terminates with a dual certificate $\vec{\alpha}$ for M in G' where $\alpha_a \in \{0, -2, -4, \ldots, -2(n_0 - 1)\}$ for $a \in A'$ and $\alpha_b \in \{0, 2, 4, \ldots, 2(n_0 - 1)\}$ for $b \in B'$ such that the neighbors of nodes in U take

the highest possible α-values, i.e., (i) $\alpha_a = 0$ for $a \in A' \cap \mathsf{Nbr}(U)$ and (ii) $\alpha_b = 2(n_0 - 1)$ for $b \in B' \cap \mathsf{Nbr}(U)$. Roughly speaking, such an $\vec{\alpha}$ will *take care* of the edges of G missing in G' and will allow us to realize M as the image of a stable matching in G^* (see Theorem 10).

2 Popular Max-Matchings

In this section we first show a simple characterization of popular max-matchings. Then we show a method to construct matchings that satisfy this characterization. Let M be any matching in $G = (A \cup B, E)$. The following edge weight function wt_M will be useful here. For any $(a, b) \in E$:

$$\text{let } \mathsf{wt}_M(a,b) = \begin{cases} 2 & \text{if } (a,b) \text{ blocks } M; \\ -2 & \text{if } a \text{ and } b \text{ prefer their assignments in } M \text{ to each other;} \\ 0 & \text{otherwise.} \end{cases}$$

So $\mathsf{wt}_M(e) = 0$ for every $e \in M$. For any edge e, $\mathsf{wt}_M(e)$ is the sum of votes (each vote is in $\{0, \pm 1\}$) of the endpoints of e for each other versus their respective assignments in M.

For any cycle/path ρ in G, let $\mathsf{wt}_M(\rho) = \sum_{e \in \rho} \mathsf{wt}_M(e)$. Theorem 5 uses this edge weight function to characterize popular max-matchings. Recall that an alternating path (resp., cycle) with respect to matching M is a path (resp., cycle) whose alternate edges are in M.

▶ **Theorem 5.** *For any maximum matching M in G, M is a popular max-matching if and only if (1) there is no alternating cycle C wrt M such that $\mathsf{wt}_M(C) > 0$ and (2) there is no alternating path p with an unmatched node as an endpoint such that $\mathsf{wt}_M(p) > 0$.*

Proof. Let M be a popular max-matching. We need to show that conditions (1) and (2) given in the theorem statement hold. Suppose not. Then there exists either an alternating path with an unmatched node as an endpoint or an alternating cycle wrt M (call this path/cycle ρ) such that $\mathsf{wt}_M(\rho) > 0$. Since $\mathsf{wt}_M(e) \in \{0, \pm 2\}$, $\mathsf{wt}_M(\rho) \geq 2$.

Consider $N = M \oplus \rho$. This is a maximum matching in G and observe that $\Delta(N, M) \geq \mathsf{wt}_M(\rho) - 1$. We are subtracting 1 here to count for that endpoint of ρ (when ρ is a path) that is matched in M but will become unmatched in N. Since $\mathsf{wt}_M(\rho) \geq 2$, $\Delta(N, M) \geq 1$. So N is more popular than M; this is a contradiction to M's popularity within the set of maximum matchings. Thus conditions (1) and (2) have to hold.

To show the converse, suppose M is a maximum matching that obeys conditions (1) and (2). Consider the symmetric difference $M \oplus N$, where N is any maximum matching in G and let C be any alternating cycle here. We know from (1) that $\mathsf{wt}_M(C) \leq 0$. Let p be any alternating path in $M \oplus N$. Since M and N are maximum matchings, p is an alternating path with exactly one node not matched in M as an endpoint. We know from (2) that $\mathsf{wt}_M(p) \leq 0$. So we have $\Delta(N, M) \leq \sum_{\rho \in M \oplus N} \mathsf{wt}_M(\rho) \leq 0$. Thus no maximum matching is more popular than M. ◀

A new instance. We will now construct a new instance $G^* = (A^* \cup B^*, E^*)$ such that every stable matching in G^* maps to a maximum matching in G that satisfies properties (1) and (2) given in Theorem 5. As mentioned earlier, the structure of the instance G^* is inspired by an instance from [10] whose stable matchings map to dominant matchings in G.

We first describe the node sets A^* and B^*. Let $n_0 = |A|$. For every $a \in A$, the set A^* has n_0 copies of a: call them $a_0, \ldots, a_{n_0 - 1}$. So $A^* = \cup_{a \in A} \{a_0, \ldots, a_{n_0 - 1}\}$.

Let $B^* = \cup_{a \in A}\{d_1(a), \ldots, d_{n_0-1}(a)\} \cup \{\tilde{b} : b \in B\}$, where $\tilde{B} = \{\tilde{b} : b \in B\}$ is a copy of the set B. So along with nodes in $\tilde{B}$, the set B^* also contains $n_0 - 1$ nodes $d_1(a), \ldots, d_{n_0-1}(a)$ for each $a \in A$. These will be called *dummy* nodes. The purpose of $d_1(a), \ldots, d_{n_0-1}(a)$ is to ensure that in any stable matching in G^*, at most one node among $a_0, \ldots, a_{n_0-1}$ is matched to a neighbor in $\tilde{B}$.

The edge set. For each $(a, b) \in E$, the edge set E^* contains n_0 edges $(a_i, \tilde{b})$ for $0 \leq i \leq n_0 - 1$. For each $a \in A$ and $i \in \{1, \ldots, n_0 - 1\}$, E^* also has $(a_{i-1}, d_i(a))$ and $(a_i, d_i(a))$.

Preference orders. Let a's preference order in G be $b_1 \succ \cdots \succ b_k$. Then a_0's preference order in G^* is $\tilde{b}_1 \succ \cdots \succ \tilde{b}_k \succ d_1(a)$, i.e., it is analogous to a's preference order in G with $d_1(a)$ added as a_0's last choice.

- For $i \in \{1, \ldots, n_0 - 2\}$, the preference order of a_i in G^* is as follows: $d_i(a) \succ \tilde{b}_1 \succ \cdots \succ \tilde{b}_k \succ d_{i+1}(a)$. So a_i's top choice is $d_i(a)$ and last choice is $d_{i+1}(a)$.
- The preference order of a_{n_0-1} is $d_{n_0-1}(a) \succ \tilde{b}_1 \succ \cdots \succ \tilde{b}_k$.

For each $i \in \{1, \ldots, n_0 - 1\}$, the preference order of $d_i(a)$ is $a_{i-1} \succ a_i$. Since each of $a_0, \ldots, a_{n_0-2}$ and $d_1(a), \ldots, d_{n_0-1}(a)$ is a top choice neighbor for some node, every stable matching in G^* has to match all these nodes. So the only node among $a_0, \ldots, a_{n_0-1}, d_1(a), \ldots, d_{n_0-1}(a)$ that can possibly be left unmatched in a stable matching in G^* is a_{n_0-1}.

Consider any $b \in B$ and let its preference order in G be $a \succ \cdots \succ z$. Then the preference order of $\tilde{b}$ in G^* is

$$\underbrace{a_{n_0-1} \succ \cdots \succ z_{n_0-1}}_{\text{all subscript } n_0-1 \text{ neighbors}} \succ \underbrace{a_{n_0-2} \succ \cdots \succ z_{n_0-2}}_{\text{all subscript } n_0-2 \text{ neighbors}} \succ \cdots \succ \underbrace{a_0 \succ \cdots \succ z_0}_{\text{all subscript 0 neighbors}}$$

That is, $\tilde{b}$'s preference order in G^* is all its subscript $n_0 - 1$ neighbors, followed by all its subscript $n_0 - 2$ neighbors, so on, and finally, all its subscript 0 neighbors. For each $i \in \{0, \ldots, n_0 - 1\}$: within all subscript i neighbors, the order of preference for $\tilde{b}$ in G^* is the same as b's order of preference in G.

The set S'. For any stable matching S in G^*, define $S' \subseteq E$ to be the set of edges obtained by deleting edges in S that are incident to dummy nodes and replacing any edge $(a_i, \tilde{b}) \in S$ with the original edge $(a, b) \in E$. Since S matches at most one node among $a_0, \ldots, a_{n_0-1}$ to a neighbor in $\tilde{B}$, the set S' is a matching in G.

The proof of Theorem 6 is based on the proof of correctness of the $|A|$-level Gale-Shapley algorithm (from [23]) in the original instance G.

▶ **Theorem 6.** *Let S be a stable matching in G^*. Then S' is a popular max-matching in G.*

Proof. Partition the set A into $A_0 \cup \cdots \cup A_{n_0-1}$ where for $0 \leq i \leq n_0 - 2$: $A_i = \{a \in A : (a_i, \tilde{b}) \in S \text{ for some } \tilde{b} \in \tilde{B}\}$, i.e., a_i is matched in S to a neighbor in $\tilde{B}$. The left-out nodes in A, i.e., those in $A \setminus (A_0 \cup \cdots \cup A_{n_0-2})$, form the set A_{n_0-1} (see Fig. 1).

Similarly, partition B into $B_0 \cup \cdots \cup B_{n_0-1}$ where for $1 \leq i \leq n_0 - 1$: $B_i = \{b : (a_i, \tilde{b}) \in S \text{ for some } a \in A_i\}$, i.e., $\tilde{b}$'s partner in S is a subscript i node. Let $B_0 = B \setminus (B_1 \cup \cdots \cup B_{n_0-1})$ be the set of left-out nodes in B.

The following properties hold: (these are proved below)

1. $S' \subseteq \cup_{i=0}^{n_0-1}(A_i \times B_i)$. Moreover, S' restricted to each set $A_i \cup B_i$ is stable.
2. For any i and edge (a, b) where $a \in A_{i+1}, b \in B_i$: we have $\mathsf{wt}_{S'}(a, b) = -2$.

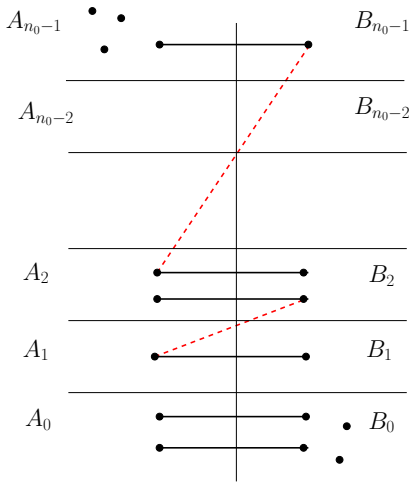


Figure 1 $A = A_0 \cup \cdots \cup A_{n_0-1}$ and $B = B_0 \cup \cdots \cup B_{n_0-1}$. The matching $S' \subseteq \cup_{i=0}^{n_0-1}(A_i \times B_i)$. All nodes unmatched in S' are in $A_{n_0-1} \cup B_0$. The dashed edges are blocking edges to S'.

3. G has no edge in $A_i \times B_j$ where $i \geq j+2$.
4. Any blocking edge to S' has to be in $A_i \times B_j$ where $i \leq j-1$.
5. All nodes that are unmatched in S' are in $A_{n_0-1} \cup B_0$.
6. S' is a maximum matching in G.

Properties 1–4 imply that for any alternating cycle C wrt S', $\mathtt{wt}_{S'}(C) \leq 0$. Similarly, properties 1–5 imply that for any alternating path p with one unmatched node as an endpoint, $\mathtt{wt}_{S'}(p) \leq 0$. We refer to [23, Theorem 2] for more details. Property 6 states that S' is a maximum matching in G. Hence S' is a popular max-matching in G (by Theorem 5). ◀

Properties 1-6. These six properties are proved below.

1. The inclusion $S' \subseteq \cup_{i=0}^{n_0-1}(A_i \times B_i)$ follows from the definition of the sets $B_0, \ldots, B_{n_0-1}$. Recall that for $1 \leq i \leq n_0 - 1$, B_i is the set of nodes b such that $(a_i, \tilde{b}) \in S$ for some $a \in A_i$. Also, B_0 contains all nodes b such that $(a_0, \tilde{b}) \in S$ for some $a \in A_0$. Thus $S' \subseteq \cup_{i=0}^{n_0-1}(A_i \times B_i)$.
   The stability of S' restricted to each set $A_i \cup B_i$ is by $\tilde{b}$'s preference order in G^*. Recall that within subscript i neighbors, the order of preference for $\tilde{b}$ in G^* is b's order of preference in G. Thus the stability of S in G^* implies the stability of S' restricted to $A_i \cup B_i$ for each i.
2. Let $a \in A_{i+1}$. Then $(a_{i+1}, d_{i+1}(a)) \notin S$. So it has to be the case that $(a_i, d_{i+1}(a)) \in S$. Recall that $d_{i+1}(a)$ is a_i's least preferred neighbor in G^*. So a_i prefers $\tilde{b}$ to its partner in S. Hence it follows from the stability of S in G^* that $\tilde{b}$ prefers its partner in S (this is a subscript i node z_i) to a_i, i.e., b prefers z to a.
   Since $\tilde{b}$ prefers subscript $i+1$ nodes to subscript i nodes, $\tilde{b}$ prefers a_{i+1} to its partner z_i in S. It follows from the stability of S in G^* that a_{i+1} has to prefer its partner $\tilde{w}$ in S to $\tilde{b}$, otherwise $(a_{i+1}, \tilde{b})$ would block S. Hence a prefers w to b. Thus $\mathtt{wt}_{S'}(a, b) = -2$.
3. Suppose $a \in A_i$ where $i \geq j+2$ and $b \in B_j$. So the edge $(a_{j+1}, d_{j+2}(a)) \in S$. Since $d_{j+2}(a)$ is a_{j+1}'s least preferred neighbor in G^*, the stability of S implies that $\tilde{b}$ prefers its partner in S to a_{j+1}. However $b \in B_j$ and so $\tilde{b}$'s partner in S is a subscript j node z_j. This contradicts $\tilde{b}$'s preference order that it prefers any subscript $j+1$ neighbor to a subscript j neighbor. Thus there is no edge (a, b) in G with $a \in A_i$ and $b \in B_j$ where $i \geq j+2$.

4. Property 4 follows from properties 1, 2, and 3 given above. Properties 2 and 3 tell us that there is no blocking edge in $A_i \times B_j$ where $i \geq j+1$. Property 1 tells us that there is no blocking edge in $A_i \times B_i$ for any i. So any blocking edge to S' has to be in $A_i \times B_j$ where $i \leq j-1$.
5. Property 5 follows from the definitions of the sets $A_0, \ldots, A_{n_0-2}$ and $B_1, \ldots, B_{n_0-1}$. For each $a \in A_i$ where $0 \leq i \leq n_0-2$: we have $(a_i, \tilde{b}) \in S$ for some $\tilde{b} \in \tilde{B}$ and thus $(a,b) \in S'$. Similarly, for each $b \in B_j$ where $1 \leq j \leq n_0-1$: we have $(a_j, \tilde{b}) \in S$ for some $a \in A_j$ and thus $(a,b) \in S'$. Hence all nodes unmatched in S' are in $A_{n_0-1} \cup B_0$.
6. Suppose S' is not a maximum matching in G. Then there is an augmenting path ρ with respect to S'. Let us refer to an edge e that satisfies $\mathsf{wt}_{S'}(e) = -2$ as a *negative* edge. The endpoints of a negative edge prefer their respective partners in S' to each other.
   We know from property 5 above that all the nodes in A that are unmatched in S' are in A_{n_0-1} and all the nodes in B that are unmatched in S' are in B_0. We also know that $S' \subseteq \cup_i (A_i \times B_i)$ (by property 1 above). Moreover, all the edges e in $A_{j+1} \times B_j$ are negative edges (by property 2) and there is no edge in $A_i \times B_j$ where $i \geq j+2$ (by property 3).
   Thus the path ρ starts in A_{n_0-1} at an unmatched node a and since there cannot be any negative edge incident to an unmatched node, all of a's neighbors have to be in B_{n_0-1}: this is because every edge e in $A_{n_0-1} \times B_{n_0-2}$ is a negative edge. The matched partners of a's neighbors are in A_{n_0-1}. Then the next node can be in B_{n_0-2} (this is by property 3) and its partner is in A_{n_0-2} and so on. Finally, there is no edge from A_1 to an unmatched node in B_0: this is because there is no negative edge incident to an unmatched node and we know all edges in $A_1 \times B_0$ are negative edges (by property 2).
   So the *shortest* alternating path ρ from an unmatched $a \in A_{n_0-1}$ to an unmatched $b \in B_0$ moves across sets as follows: $A_{n_0-1} - B_{n_0-1} - A_{n_0-1} - B_{n_0-2} - A_{n_0-2} - B_{n_0-3} - \cdots - A_1 - B_0 - A_0 - B_0$. This implies there are at least n_0+1 nodes in A. However $|A| = n_0$. So there is no such alternating path, i.e., there is no augmenting path with respect to S'. In other words, S' is a maximum matching in G. This finishes our proof of these six properties.

Theorem 6 shows that every stable matching in G^* maps to a popular max-matching in G. In fact, as we show next, *every* popular max-matching in G has to be realized in this manner. This is the tough part of the proof and as mentioned earlier, we will use LP-duality here. We will see that appropriate dual certificates capture a very useful feature of popular max-matchings.

3 Proving Surjectivity in a Special Case

In this section we consider the case when G admits a perfect matching. Let M be a popular perfect matching in G. So no perfect matching in G is more popular than M.

Consider the following linear program (LP1) that computes a max-weight (wrt wt_M) perfect matching in G. For any node u, let $\delta(u)$ be the set of edges incident to u.

$$\text{maximize} \quad \sum_{e \in E} \mathsf{wt}_M(e) \cdot x_e \qquad \text{(LP1)}$$

$$\text{subject to}$$

$$\sum_{e \in \delta(u)} x_e = 1 \quad \forall\, u \in A \cup B \qquad \text{and} \qquad x_e \geq 0 \quad \forall\, e \in E.$$

It follows from the definition of the function wt_M that the optimal value of (LP1) is $\max_N \Delta(N, M)$ where N is a perfect matching in G. So if M is a popular perfect matching then the optimal value of (LP1) is 0, which is $\Delta(M, M)$, i.e., the edge incidence vector of M is an optimal solution to (LP1). The linear program (LP2) is the dual of (LP1). Hence if M is a popular perfect matching then there exists a dual feasible $\vec{\alpha}$ such that $\sum_{u \in A \cup B} \alpha_u = 0$.

$$\text{minimize} \sum_{u \in A \cup B} y_u \qquad \text{(LP2)}$$

$$\text{subject to}$$

$$y_a + y_b \geq \mathsf{wt}_M(a,b) \quad \forall\, (a,b) \in E.$$

Let $\vec{\alpha}$ be an optimal solution to (LP2). Observe that there exists an integral optimal solution to (LP2) since the constraint matrix is totally unimodular. Thus we can assume that $\vec{\alpha} \in \mathbb{Z}^{2n_0}$, where $n_0 = |A| = |B|$.

▶ **Lemma 7.** *If M is a popular perfect matching in G then there exists an optimal solution $\vec{\alpha}$ to* (LP2) *such that $\alpha_a \in \{0, -2, -4, \ldots, -2(n_0-1)\}$ for all $a \in A$ and $\alpha_b \in \{0, 2, 4, \ldots, 2(n_0-1)\}$ for all $b \in B$.*

Proof. The dual feasibility constraints are $\alpha_a + \alpha_b \geq \mathsf{wt}_M(a,b)$ for all $(a,b) \in E$. For each edge $(a,b) \in M$: $\alpha_a + \alpha_b = \mathsf{wt}_M(a,b) = 0$ by complementary slackness. Since $\alpha_b = -\alpha_a$ for $(a,b) \in M$ and because $\mathsf{wt}_M(e) \in \{0, \pm 2\}$ for each edge e, we can assume that in the sorted order of distinct α-values taken by nodes in A, for any two consecutive values $\alpha_{a'}, \alpha_{a''}$, where $\alpha_{a'} > \alpha_{a''}$, we have $\alpha_{a'} - \alpha_{a''} = 2$.

Thus we can assume that $\alpha_a \in \{k, k-2, k-4, \ldots, k-2(n_0-1)\}$ for all $a \in A$ and $\alpha_b \in \{-k, -k+2, -k+4, \ldots, -k+2(n_0-1)\}$ for all $b \in B$, for some $k \in \mathbb{Z}$. Observe that k has no impact on the objective function $\sum_{u \in A \cup B} \alpha_u$. This is because $|A| = |B|$ and so k's and $-k$'s cancel each other out.

Let us update $\vec{\alpha}$ as follows: $\alpha_a = \alpha_a - k$ for every $a \in A$ and $\alpha_b = \alpha_b + k$ for every $b \in B$. The updated vector $\vec{\alpha}$ continues to be dual feasible since $\alpha_a + \alpha_b$, for any edge (a,b), is unchanged by this update. Thus there is an optimal solution $\vec{\alpha}$ to (LP2) such that $\alpha_a \in \{0, -2, \ldots, -2(n_0-1)\}$ for all $a \in A$ and $\alpha_b \in \{0, 2, \ldots, 2(n_0-1)\}$ for all $b \in B$. ◀

Let M be a popular perfect matching in G. In order to define a stable matching S in G^* such that $M = S'$ (the set S' is defined above Theorem 6), we will use the vector $\vec{\alpha}$ described in Lemma 7. Since M is perfect, we know that for any $a \in A$, there is an edge $(a,b) \in M$ for some neighbor b of a. Recall that $\alpha_a + \alpha_b = \mathsf{wt}_M(a,b) = 0$ by complementary slackness. We will include the edge $(a_i, \tilde{b})$ in S where $\alpha_a = -2i$ and $\alpha_b = 2i$. Thus we define S as follows:

$$S = \cup_{i=0}^{n_0-1} \{(a_i, \tilde{b}) : (a,b) \in M \text{ and } \alpha_a = -2i, \alpha_b = 2i\} \cup \{\text{necessary edges incident to dummy nodes in } G^*\}.$$

In more detail, the edges incident to dummy nodes that are present in S are as follows: for each $a \in A$, these edges are $(a_j, d_{j+1}(a))$ for $0 \leq j \leq i-1$ and $(a_j, d_j(a))$ for $i+1 \leq j \leq n_0-1$, where $\alpha_a = -2i$.

Since $(a_i, \tilde{b}) \in S$, all the n_0 nodes $a_0, \ldots, a_{n_0-1}$ and the dummy nodes $d_1(a), \ldots, d_{n_0-1}(a)$ corresponding to a in G^* are matched in S. This holds for every $a \in A$. Also every $\tilde{b} \in \tilde{B}$ is matched in S since M is a perfect matching in G. Thus S is a perfect matching in G^*. It is easy to check that $S' = M$. What we need to prove is the stability of S in G^*.

▶ **Theorem 8.** *The matching S is stable in G^*.*

Proof. We need to show there is no edge in G^* that blocks S. There is no blocking edge incident to a dummy node: this is because a dummy node $d_i(a)$ has only two neighbors and when $d_i(a)$ is matched in S to its second choice neighbor a_i, its top choice neighbor a_{i-1} prefers its partner in S to $d_i(a)$.

Let us now show that no node in $a_0, \ldots, a_{n_0-1}$ has a blocking edge incident to it, for any $a \in A$. Let $(a_i, \tilde{b}) \in S$ where $(a, b) \in M$. All of $a_{i+1}, \ldots, a_{n_0-1}$ are matched to their respective top choice neighbors $d_{i+1}(a), \ldots, d_{n_0-1}(a)$. So there is no blocking edge incident to any of $a_{i+1}, \ldots, a_{n_0-1}$.

All of $a_0, \ldots, a_{i-1}$ are matched to their last choice neighbors – these are the dummy nodes $d_1(a), \ldots, d_i(a)$, respectively. Consider any neighbor $w \in B$ of a. We need to show that $\tilde{w} \in \tilde{B}$ is matched in S to a neighbor preferred to all of $a_0, \ldots, a_{i-1}$. We have $\alpha_a + \alpha_w \geq \mathsf{wt}_M(a, w)$. Since $\alpha_a = -2i$ and $\mathsf{wt}_M(e) \geq -2$ for every edge e, it follows that $\alpha_w \geq 2i - 2$.

So $(z, w) \in M$ for some neighbor z of w such that $\alpha_z = -\alpha_w \leq -(2i-2)$. Equivalently, $(z_j, \tilde{w}) \in S$ where $j \geq i-1$. Thus there is no blocking edge between $\tilde{w}$ and any of $a_0, \ldots, a_{i-2}$ by $\tilde{w}$'s preference order in G^*. We will now show that $(a_{i-1}, \tilde{w})$ cannot be a blocking edge.

- If $j \geq i$ then by $\tilde{w}$'s preference order in G^*, $\tilde{w}$ prefers z_j to a_{i-1} and so $(a_{i-1}, \tilde{w})$ does not block S.
- If $j = i-1$ then $\mathsf{wt}_M(a, w) \leq \alpha_a + \alpha_w = -2i + 2i - 2 = -2$. So both a and w prefer their respective partners in M to each other. Thus $\tilde{w}$ prefers z_{i-1} to a_{i-1}. So $(a_{i-1}, \tilde{w})$ does not block S.

Finally, we need to show there is no blocking edge incident to a_i. By the above arguments, we only need to consider edges $(a_i, \tilde{w})$ where $(z_i, \tilde{w}) \in S$. So $\mathsf{wt}_M(a, w) \leq \alpha_a + \alpha_w = -2i + 2i = 0$. Hence either $(a, w) \in M$ or at least one of a, w prefers its partner in M to the other. So either $(a_i, \tilde{w}) \in S$ or at least one of $a_i, \tilde{w}$ prefers its partner in S to the other; thus the edge $(a_i, \tilde{w})$ does not block S. Hence S is a stable matching in G^*. ◀

4 The General Case

We showed that when G has a perfect matching, our mapping from the set of stable matchings in G^* to the set of popular max-matchings in G is surjective. Now we look at the general case, i.e., G need not have a perfect matching. Let M be a popular max-matching in G.

Let $U \subseteq A \cup B$ be the set of nodes left unmatched in M. Consider (LP3) that computes a max-weight perfect matching (with respect to wt_M) in the subgraph G' induced on $V = (A \cup B) \setminus U$. Let E' be the edge set of G'. For any $v \in V$, let $\delta'(v) = \delta(v) \cap E'$.

$$\text{maximize} \quad \sum_{e \in E'} \mathsf{wt}_M(e) \cdot x_e \qquad \text{(LP3)}$$

$$\text{subject to}$$

$$\sum_{e \in \delta'(v)} x_e = 1 \quad \forall\, v \in V \qquad \text{and} \qquad x_e \geq 0 \quad \forall\, e \in E'.$$

The optimal value of (LP3) is $\max_N \Delta(N, M)$ where N is a perfect matching in G'. Any perfect matching in G' is a maximum matching in G and since M is a popular max-matching in G, $\Delta(N, M) \leq 0$ for any perfect matching N in G'. Since $\Delta(M, M) = 0$, the edge incidence vector of M is an optimal solution to (LP3). The linear program (LP4) is the dual of (LP3).

$$\text{minimize} \sum_{u \in V} y_u \qquad \text{(LP4)}$$

$$\text{subject to}$$

$$y_a + y_b \;\geq\; \mathsf{wt}_M(a,b) \quad \forall\, (a,b) \in E'.$$

Since M is a popular max-matching, the optimal value of (LP3) is 0. So there exists an optimal solution $\vec{\alpha}$ to (LP4) such that $\sum_{u \in V} \alpha_u = 0$. Moreover, we can assume the following (see Lemma 7) where $A' = A \setminus U$ and $B' = B \setminus U$. Here $|A'| = |B'| = n_0$.

1. $\alpha_a \in \{0, -2, -4, \ldots, -2(n_0 - 1)\}$ for all $a \in A'$
2. $\alpha_b \in \{0, 2, 4, \ldots, 2(n_0 - 1)\}$ for all $b \in B'$.

For any $T \subseteq A \cup B$, let $\mathsf{Nbr}(T)$ be the set of neighbors in G of nodes in T. Theorem 9 is our main technical result here. Let $U_A = U \cap A$ and $U_B = U \cap B$.

▶ **Theorem 9.** *Let M be a popular max-matching in G and let U be the set of nodes left unmatched in M. There exists an optimal solution $\vec{\alpha}$ to* (LP4) *such that*

- $\alpha_a \in \{0, -2, \ldots, -2(n_0 - 1)\}$ *for* $a \in A'$ *and* $\alpha_b \in \{0, 2, \ldots, 2(n_0 - 1)\}$ *for* $b \in B'$
- *(i)* $\alpha_a = 0$ *for* $a \in \mathsf{Nbr}(U_B)$ *and (ii)* $\alpha_b = 2(n_0 - 1)$ *for* $b \in \mathsf{Nbr}(U_A)$.

Let us first finish our proof of surjectivity by assuming Theorem 9. Then we will prove Theorem 9. Let M be any popular max-matching in G. Corresponding to M, there is a vector $\vec{\alpha}$ as given in Theorem 9. We will use this vector $\vec{\alpha}$ to construct $S = \cup_{i=0}^{n_0 - 1} \{(a_i, \tilde{b}) : (a,b) \in M$ and $\alpha_a = -2i, \alpha_b = 2i\} \cup \{$necessary edges incident to dummy nodes in $G^*\}$.

The edges in S incident to dummy nodes are: $(a_j, d_{j+1}(a))$ for $0 \leq j \leq n_0 - 2$ for $a \in U_A$. For $a \in A \setminus U_A$, these are $(a_j, d_{j+1}(a))$ for $0 \leq j \leq i-1$ and $(a_j, d_j(a))$ for $i+1 \leq j \leq n_0 - 1$, where $\alpha_a = -2i$. It is easy to see that S is a matching in G^* and $S' = M$.

▶ **Theorem 10.** *The matching S is stable in G^*.*

Proof. Let E' be the edge set of G', where G' is the subgraph of G induced on $(A \cup B) \setminus U$. Consider any edge $(a_j, \tilde{w})$ in G^* where $(a,w) \in E'$ and $0 \leq j \leq n_0 - 1$. The proof of Theorem 8 shows that $(a_j, \tilde{w})$ does not block S.

Consider any edge (a,w) in $E \setminus E'$. Such an edge has a node in U as an endpoint. A useful observation is that every node in $\mathsf{Nbr}(U)$ has to be matched in M to some neighbor that it prefers to all its neighbors in U. Otherwise M would not be a popular max-matching.

- Suppose $a \in U_A$. So $w \in \mathsf{Nbr}(U_A)$ and $\alpha_w = 2(n_0 - 1)$ by property (ii) in Theorem 9. So $(z_{n_0 - 1}, \tilde{w}) \in S$ for some neighbor z that w prefers to a. Thus $(a_{n_0 - 1}, \tilde{w})$ does not block S. For $i \in \{0, \ldots, n_0 - 2\}$, none of the edges $(a_i, \tilde{w})$ can block S (by $\tilde{w}$'s preference order).
- Suppose $w \in U_B$. So $a \in \mathsf{Nbr}(U_B)$ and $\alpha_a = 0$ by property (i) in Theorem 9. Thus $(a_0, \tilde{b}) \in S$ for some neighbor b that a prefers to w. Hence $(a_0, \tilde{w})$ does not block S. Moreover, none of the edges $(a_i, \tilde{w})$ for $i \in \{1, \ldots, n_0 - 1\}$ can block S since $a_1, \ldots, a_{n_0 - 1}$ are matched to their respective top choice neighbors $d_1(a), \ldots, d_{n_0 - 1}(a)$.

Finally, no edge incident to a dummy node blocks S (by the same argument as given in the proof of Theorem 8). Hence S is a stable matching in G^*. ◀

Thus Theorem 9 allows us to show that for any popular max-matching M in G, there is a stable matching S in G^* such that $M = S'$. We will now prove Theorem 9.

Proof of Theorem 9. We know there is an optimal solution $\vec{\alpha}$ to (LP4) where $\alpha_a \in \{0, -2, \ldots, -2(n_0 - 1)\}$ for $a \in A'$ and $\alpha_b \in \{0, 2, \ldots, 2(n_0 - 1)\}$ for $b \in B'$. We now update $\vec{\alpha}$ so that it remains an optimal solution to (LP4) in the above format and it also satisfies properties (i) and (ii) given in the theorem statement.

Property (ii). Suppose the vector $\vec{\alpha} \in \{0, \pm 2, \ldots, \pm 2(n_0 - 1)\}^{2n_0}$ does not satisfy property (ii). So we have to update $\vec{\alpha}$ so that property (ii) is satisfied. First, we increase the α-values of the nodes in $\mathsf{Nbr}(U_A)$ to $2(n_0 - 1)$ and decrease the α-values of their partners in M to $-2(n_0 - 1)$. Now $\vec{\alpha}$ may no longer be a feasible solution to (LP4).

We use the following three update rules for all $a \in A'$ to make $\vec{\alpha}$ feasible again. Let $\alpha_a = -2i$ where $i \in \{0, \ldots, n_0 - 1\}$. Suppose there is some $(a, b) \in E'$ with $\alpha_a + \alpha_b < \mathsf{wt}_M(a, b)$. Let $M(b)$ be b's partner in M.

- *Rule 1.* If $\mathsf{wt}_M(a, b) = 0$ then update $\alpha_b = 2i$ and $\alpha_{M(b)} = -2i$.
- *Rule 2.* If $\mathsf{wt}_M(a, b) = -2$ then update $\alpha_b = 2(i - 1)$ and $\alpha_{M(b)} = -2(i - 1)$.
- *Rule 3.* If $\mathsf{wt}_M(a, b) = 2$ then update $\alpha_b = 2(i + 1)$ and $\alpha_{M(b)} = -2(i + 1)$.

At the onset, $\vec{\alpha}$ was a feasible solution to (LP4), so $\alpha_a + \alpha_b \geq \mathsf{wt}_M(a, b)$ for $(a, b) \in E'$. Then we moved the nodes in $\mathsf{Nbr}(U_A)$ and their partners in M to sets B_{n_0-1} and A_{n_0-1}, respectively, where $A_i = \{a \in A' : \alpha_a = -2i\}$ and $B_i = \{b \in B' : \alpha_b = 2i\}$ for all i. The subscript i will be called the *level* of nodes in $A_i \cup B_i$.

The nodes that moved to A_{n_0-1} have a lower α-value than earlier and it is these nodes that "pull" their neighbors upwards to higher levels as given by rules 1-3. Let a be a new node in level i and let b be a neighbor of a such that $\alpha_a + \alpha_b < \mathsf{wt}_M(a, b)$. Then b and $M(b)$ move to: (1) level i if $\mathsf{wt}_M(a, b) = 0$, (2) level $i - 1$ if $\mathsf{wt}_M(a, b) = -2$, else (3) level $i + 1$, i.e., if $\mathsf{wt}_M(a, b) = 2$.

In turn, the nodes in A' that have moved to these higher levels by rules 1-3 pull their neighbors and the partners of these neighbors upwards to higher levels by these rules. Thus we may get further new nodes in B_{n_0-1}, A_{n_0-1} and so on. While any of rules 1-3 is applicable, we apply that rule. So a rule may be applied many times to the same edge in E'.

Claim 11 (proved in Section 4.1) shows a useful property. We show in its proof that such a blocking edge creates a *forbidden* alternating cycle/path wrt M, as given in Theorem 5.

▷ Claim 11. By applying the above rules, suppose a node $v_0 \in A'$ moves to A_{n_0-1}. Then there is no blocking edge (e such that $\mathsf{wt}_M(e) = 2$) incident to v_0.

Applying rules 1-3 increases the α-values of some nodes in B' and it never decreases the α-value of any node in B'. The nodes in B' with increased α-values and their partners have moved to higher levels (see Fig. 1). This upwards movement of nodes has to terminate at level $n_0 - 1$. For the α-value of any $b \in B'$ to be increased beyond $2(n_0 - 1)$, we need a blocking edge (a, b) where $a \in A_{n_0-1}$ – this would cause rule 3 to be applied which would increase α_b to $2n_0$. However there is no such blocking edge (by Claim 11).

Since there are n_0 levels and because $|B'| = n_0$, there can be at most n_0^2 applications of these rules. When no rule is applicable, $\vec{\alpha}$ is a feasible solution to (LP4). Moreover, $\sum_{u \in V} \alpha_u$ is invariant under this update of α-values, since we maintain $\alpha_a + \alpha_b = 0$ for every $(a, b) \in M$. Hence $\vec{\alpha}$ is an optimal solution to (LP4). Thus for every popular max-matching M, there is an optimal solution $\vec{\alpha}$ to (LP4) in the desired format that satisfies property (ii).

Property (i). We now have an optimal solution $\vec{\alpha} \in \{0, \pm 2, \ldots, \pm 2(n_0 - 1)\}^{2n_0}$ to (LP4), where $\alpha_a \leq 0$ for all $a \in A'$ and $\alpha_b \geq 0$ for all $b \in B'$, such that $\alpha_b = 2(n_0 - 1)$ for all $b \in \mathsf{Nbr}(U_A)$. Suppose property (i) is not satisfied.

Then we increase α-values of certain nodes in A' – this moves these nodes *downwards* with respect to their level (see Fig. 1) and ensures that property (i) holds. First, we increase the α-values of the nodes in $\mathsf{Nbr}(U_B)$ to 0 and their partners also have α-values updated to 0. Now $\vec{\alpha}$ may no longer be a feasible solution to (LP4).

So we will use the following three update rules for all $b \in B'$. Let $\alpha_b = 2i$ where $i \in \{0, \ldots, n_0 - 1\}$. Suppose there is an edge $(a, b) \in E'$ such that $\alpha_a + \alpha_b < \mathsf{wt}_M(a, b)$. Let $M(a)$ be a's partner in M.

- *Rule 4.* If $\mathsf{wt}_M(a, b) = 0$ then update $\alpha_a = -2i$ and $\alpha_{M(a)} = 2i$.
- *Rule 5.* If $\mathsf{wt}_M(a, b) = -2$ then update $\alpha_a = -2(i + 1)$ and $\alpha_{M(a)} = 2(i + 1)$.
- *Rule 6.* If $\mathsf{wt}_M(a, b) = 2$ then update $\alpha_a = -2(i - 1)$ and $\alpha_{M(a)} = 2(i - 1)$.

Applying rules 4-6 increases the α-values of some nodes in A' and it never decreases the α-value of any node in A'. The nodes in A' with increased α-values and their partners have moved to lower levels. Moreover, the movement of nodes downwards has to stop at level 0 since no blocking edge can be incident to any node that moves to B_0 (analogous to Claim 11). While any of the above three rules is applicable, we apply that rule.

When no rule is applicable, $\vec{\alpha}$ is a feasible solution to (LP4). Since $\sum_{u \in V} \alpha_u = 0$, $\vec{\alpha}$ is an optimal solution to (LP4). So there is an optimal solution $\vec{\alpha}$ to (LP4) in the desired format that satisfies property (i).

Properties (i) and (ii). Note that we cannot claim straightaway that the above $\vec{\alpha}$ satisfies both property (i) and property (ii). This is because applying rules 4-6 may have caused $\alpha_b < 2(n_0 - 1)$ for some $b \in \mathsf{Nbr}(U_A)$. Claim 12 shows this is not possible.

▷ Claim 12. The above $\vec{\alpha}$ satisfies property (ii), i.e., $\alpha_b = 2(n_0 - 1)$ for $b \in \mathsf{Nbr}(U_A)$.

Claim 12 is proved in Section 4.1. So we have an optimal solution $\vec{\alpha}$ to (LP4) in the desired format that satisfies both property (i) and property (ii). ◀

4.1 Proofs of Claim 11 and Claim 12

The proofs of Claim 11 and Claim 12 use the fact that certain alternating cycles/paths are forbidden for popular max-matchings. These include the ones given in Theorem 5 and also augmenting paths (since M is a maximum matching).

Proof of Claim 11. Let v_0 be the first node that moves to A_{n_0-1} with a blocking edge incident to it. Recall our update procedure – we initially added nodes in $\mathsf{Nbr}(U_A)$ and their partners in M to B_{n_0-1} and A_{n_0-1}, respectively. Then we applied rules 1-3 in some order and this resulted in the node v_0 moving to A_{n_0-1}. Corresponding to these rules, we will construct an alternating path $p = v_0 - M(v_0) - v_1 - M(v_1) - v_2 - \cdots - v_k - M(v_k) - u$ between v_0 and some node $u \in U_A$.

The path p can be partitioned into $k + 1$ pairs of edges for some $k \geq 0$. For $0 \leq i \leq k - 1$: the i-th pair consists of the matching edge $(v_i, M(v_i))$ of weight 0 and the non-matching edge $e_i = (M(v_i), v_{i+1})$ where v_{i+1} is the node that pulled $M(v_i)$ and v_i to their current level due to the application of one of the above three rules. Rule 1 implies $\mathsf{wt}_M(e_i) = 0$ while rule 2 implies $\mathsf{wt}_M(e_i) = -2$ and rule 3 implies $\mathsf{wt}_M(e_i) = 2$.

Observe that rule 1 places $M(v_i)$ in the same level as v_{i+1} while rule 2 places $M(v_i)$ one level lower than v_{i+1} and rule 3 places $M(v_i)$ one level higher than v_{i+1}. The last pair of edges in p are $(v_k, M(v_k)) \in A_{n_0-1} \times B_{n_0-1}$ and $(M(v_k), u)$, where the latter edge has weight 0. The node v_0 is in level $n_0 - 1$ and the node $M(v_k)$ is also in level $n_0 - 1$. So v_0 and v_k are at the same level, hence the number of edges in p of weight -2 is exactly the same as the number of edges of weight 2, thus $\mathsf{wt}_M(p) = 0$.

Suppose there is a blocking edge (v_0, w). If w belongs to p, then it is easy to see that the alternating cycle C obtained by joining the endpoints of the v_0-w subpath in p with the edge (v_0, w) satisfies $\mathsf{wt}_M(C) \geq 2$. This contradicts Theorem 5 since M is a popular max-matching. Hence w does not belong to path p. So let us add the 2-edge path $M(w) - w - v_0$ as a prefix to the v_0-u path p and call this alternating path q: we have $\mathsf{wt}_M(q) = \mathsf{wt}_M(p) + \mathsf{wt}_M(v_0, w) = 2$. Since $\mathsf{wt}_M(q) > 0$ and the unmatched node u is an endpoint of q, this again contradicts Theorem 5. Hence there is no blocking edge incident to v_0. ◁

Proof of Claim 12. Suppose there is a node $w_1 \in \mathsf{Nbr}(U_A)$ such that $\alpha_{w_1} < 2(n_0 - 1)$. So there is some $u \in U_A$ such that $(u, w_1) \in E$ and though $\alpha_{w_1} = 2(n_0 - 1)$ just before we started applying rules 4-6, the application of these rules caused α_{w_1} to become less than $2(n_0 - 1)$.

Initially we added nodes in $\mathsf{Nbr}(U_B)$ and their partners in M to A_0 and B_0, respectively. Then we repeatedly applied rules 4-6 and this resulted in w_1 moving to a level lower than $n_0 - 1$. Corresponding to what caused w_1 to be "pulled" downwards, we will construct an alternating path $p = w_1 - M(w_1) - w_2 - M(w_2) - \cdots - w_r - M(w_r) - u'$ between w_1 and a node $u' \in U_B$.

The path p will consist of r pairs of edges for some $r \geq 1$. For $1 \leq i \leq r - 1$: the i-th pair of edges is $(w_i, M(w_i))$ and $(M(w_i), w_{i+1})$ where w_{i+1} is the node that pulled $M(w_i)$ and w_i to their current level due to the application of one of rules 4-6. The last pair of edges in p is $(w_r, M(w_r))$ and $(M(w_r), u')$, where $w_r \in B_0, M(w_r) \in A_0$, and $u' \in U_B$. Thus we have an alternating path p between w_1 and $u' \in U_B$.

By adding the edge (u, w_1) as a prefix to the path p, we get an augmenting path $u - w_1 - \cdots - M(w_r) - u'$ with respect to M. However there cannot be any augmenting path wrt M since M is a maximum matching in G. Thus $\vec{\alpha}$ satisfies property (ii). ◁

This finishes the proof of Theorem 9. So the stable matching polytope of G^* yields a compact extended formulation for the popular max-matching polytope of G (by Theorem 6 and Theorem 10). This formulation is described in Section 4.2.

Linear programming on this formulation with $\min \sum_{e \in E} c(e) \cdot x_e$ as the objective function computes a min-cost popular max-matching in G in polynomial time. Equivalently, we can compute a min-cost stable matching S in G^* and return the corresponding matching S' in G. It follows from Theorem 6 and Theorem 10 that S' is a min-cost popular max-matching in G. This proves Theorem 3 stated in Section 1.

4.2 An Extended Formulation for the Popular Max-Matching Polytope

For any node u in G^*, let $\{v' \succ_u v\}$ be the set of all neighbors of u in G^* that it prefers to v. Let $\delta^*(u)$ denote the set of edges incident to u in G^*.

Let $T = \cup_{a \in A}(\{a_0, \ldots, a_{n_0-2}\} \cup \{d_1(a), \ldots, d_{n_0-1}(a)\})$. Every node in T is a top choice neighbor for some node, so every node in T has to be matched in all stable matchings in G^*. The constraints given below describe the stable matching polytope of G^*, as shown in [30].

$$\sum_{w \succ_{a_i} \tilde{b}} x_{(a_i,w)} + \sum_{z \succ_{\tilde{b}} a_i} x_{(z,\tilde{b})} + x_{(a_i,\tilde{b})} \geq 1 \qquad \forall (a_i, \tilde{b}) \in E^*$$

$$\sum_{e \in \delta^*(u)} x_e = 1 \qquad \forall\, u \in T$$

$$\sum_{e \in \delta^*(u)} x_e \leq 1 \quad \forall u \in A^* \cup B^* \qquad \text{and} \qquad x_e \geq 0 \qquad \forall e \in E^*.$$

1. The topmost constraint captures the *stability* constraint for edge $(a_i, \tilde{b}) \in E^*$ where $(a, b) \in E$ and $0 \leq i \leq n_0 - 1$.
2. The constraint in the second line for $u = a_{i-1}$ captures the stability constraint for the edge $(a_{i-1}, d_i(a))$ and for $u = d_i(a)$ captures the stability constraint for the edge $(a_i, d_i(a))$.
3. The constraints in the third line capture that $\vec{x}$ belongs to the matching polytope of G^*.

Consider the equations $x_{(a,b)} = \sum_{i=0}^{n_0-1} x_{(a_i,\tilde{b})}$ for all $(a, b) \in E$. It follows from Theorem 6 and Theorem 10 that these m equations along with the constraints of the stable matching polytope of G^* given above describe an extended formulation for the popular max-matching polytope $\mathcal{M}_G$. So the extension complexity of the polytope $\mathcal{M}_G$ is $O(mn)$.

5 A Hardness Result

In this section we show that it is NP-hard to compute a min-cost Pareto-optimal matching and a min-cost Pareto-optimal max-matching in an instance $G = (A \cup B, E)$ with edge costs in $\{0, 1\}$.

Given a 3SAT formula ψ, we will build an instance G_ψ with edge costs in $\{0, 1\}$ such that G_ψ admits a Pareto-optimal matching of cost 0 if and only if ψ is satisfiable. Any Pareto-optimal matching of cost 0 would have to be a perfect matching in G_ψ. Hence this will prove the NP-hardness of both the min-cost Pareto-optimal matching problem and the min-cost Pareto-optimal max-matching problem.

Our reduction resembles a hardness reduction from [9] that showed the NP-hardness of deciding if an instance G has a stable matching M that is also dominant. As done in this reduction, we will first transform ψ so that every clause contains either only positive literals or only negative literals; moreover, there will be a single occurrence of each negative literal in the transformed ψ. This is easy to achieve:

- let $X_1, \ldots, X_n$ be the starting variables. For $i \in [n]$: replace all occurrences of $\neg X_i$ with the same variable X_{n+i} (a new one) and add the two clauses $(X_i \vee X_{n+i}) \wedge (\neg X_i \vee \neg X_{n+i})$ to capture $\neg X_i \equiv X_{n+i}$. Thus there are $2n$ variables in the transformed ψ.

We build the graph G_ψ as follows. There are two types of gadgets: those that correspond to positive clauses and those that correspond to negative clauses. Fig. 2 (resp., Fig. 3) shows how a positive (resp., negative) clause gadget looks like.

We now describe the preference lists of nodes in a positive clause $C_\ell = x \vee y \vee z$ (see Fig. 2). The nodes a_x, a'_x, b_x, b'_x occur in x's gadget and a_y, a'_y, b_y, b'_y occur in y's gadget and a_z, a'_z, b_z, b'_z occur in z's gadget: these gadgets are in the ℓ-th clause gadget C_ℓ. Every occurrence of a literal has a separate gadget.

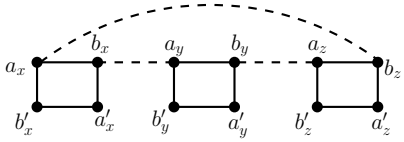


Figure 2 The clause gadget for a positive clause $C_\ell = x \vee y \vee z$. Every occurrence of a literal in ψ has a separate gadget. So we ought to use labels such as $a_{x,\ell}, b_{x,\ell}, \ldots$ here; for the sake of simplicity, we used the labels $a_x, b_x, \ldots$ here.

a_x	a'_x	a_y	a'_y	a_z	a'_z
b_z	b_x	b_x	b_y	b_y	b_z
b_x	b'_x	b_y	b'_y	b_z	b'_z
d'_x	–	d'_y	–	d'_z	–
b'_x	–	b'_y	–	b'_z	–

Here a_x's top choice is b_z, second choice b_x, third choice d'_x, fourth choice b'_x, and similarly for other nodes. For every occurrence of a positive literal x: there will be a pair of *consistency edges* – the pair (a_x, d'_x) and (b'_x, c_x) in Fig. 4 – between this gadget of x and the unique gadget of $\neg x$. In our preferences, the neighbors on consistency edges are marked in red.

b_x	b'_x	b_y	b'_y	b_z	b'_z
a_y	a'_x	a_z	a'_y	a_x	a'_z
a_x	c_x	a_y	c_y	a_z	c_z
a'_x	a_x	a'_y	a_y	a'_z	a_z

The preference lists of nodes that occur in a clause gadget with 2 positive literals will be totally analogous to the preference lists of nodes in a clause gadget with 3 positive literals.

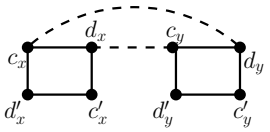


Figure 3 A clause gadget corresponding to a negative clause $D_k = \neg x \vee \neg y$; due to our transformation of ψ, every negative clause has only 2 literals.

We will now describe the preference lists of nodes in a negative clause k – the overall picture here is given in Fig. 3.

c_x	c'_x	c_y	c'_y
d_y	d_x	d_x	d_y
d_x	d'_x	d_y	d'_y
$b'_{x,i}$	–	$b'_{y,i'}$	–
$\cdots$	–	$\cdots$	–
$b'_{x,j}$	–	$b'_{y,j'}$	–
d'_x	–	d'_y	–

d_x	d'_x	d_y	d'_y
c_y	c'_x	c_x	c'_y
c_x	$a_{x,i}$	c_y	$a_{y,i'}$
c'_x	$\cdots$	c'_y	$\cdots$
–	$a_{x,j}$	–	$a_{y,j'}$
–	c_x	–	c_y

The nodes c_x, c'_x, d_x, d'_x and c_y, c'_y, d_y, d'_y occur in the gadgets of $\neg x$ and $\neg y$, respectively. The nodes $b'_{x,i}, \ldots, b'_{x,j}$ (resp., $b'_{y,i'}, \ldots, b'_{y,j'}$) in the preference lists above are the b'-nodes in the x-gadgets (resp., y-gadgets) in the various clauses that x (resp., y) occurs in. Similarly, $a_{x,i}, \ldots, a_{x,j}$ (resp., $a_{y,i'}, \ldots, a_{y,j'}$) are the a-nodes in the x-gadgets (resp., y-gadgets) in the

various clauses that x (resp., y) occurs in. The preference order among the b'-nodes and among the a-nodes in these lists is not important. The consistency edges between a gadget of x and the gadget of $\neg x$ are shown in Fig. 4.

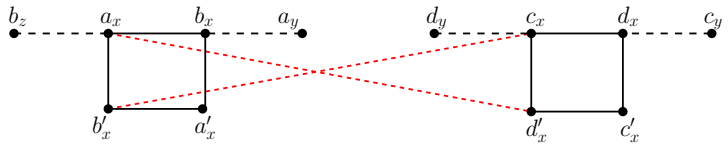


Figure 4 For the sake of simplicity, we use a_x, b_x, a'_x, b'_x to denote the 4 nodes in the gadget of x in the ℓ-th clause; c_x, d_x, c'_x, d'_x are the 4 nodes in the unique gadget of $\neg x$. The consistency edges are the red dashed edges.

Edge costs. For each edge e in G_ψ, we will set $\mathsf{cost}(e) \in \{0, 1\}$ as follows.

- For each variable $r \in \{X_1, \ldots, X_{2n}\}$: set $\mathsf{cost}(e) = 0$ where e is any of the 4 edges in any literal gadget $\langle a_r, b_r, a'_r, b'_r\rangle$ of r or any of the 4 edges in the gadget $\langle c_r, d_r, c'_r, d'_r\rangle$ of $\neg r$.
- For all other edges e, set $\mathsf{cost}(e) = 1$.

In particular, for any edge e in the consistency pair for any variable, we have $\mathsf{cost}(e) = 1$. In our figures, all dashed edges have cost 1 and all solid edges have cost 0.

Let M be a Pareto-optimal matching in G_ψ with $\mathsf{cost}(M) = 0$. So M has to use only cost 0 edges. Thus M is forbidden to use any edge other than the 4 edges in the gadget of any literal. Moreover, since M is Pareto-optimal, M cannot leave two adjacent nodes unmatched. Thus for $r \in \{X_1, \ldots, X_{2n}\}$:

1. From a gadget of r (say, on nodes a_r, b_r, a'_r, b'_r), either (i) $(a_r, b_r), (a'_r, b'_r)$ are in M or (ii) $(a_r, b'_r), (a'_r, b_r)$ are in M.
2. From the gadget of $\neg r$ (the nodes are c_r, d_r, c'_r, d'_r), either (i) $(c_r, d_r), (c'_r, d'_r)$ are in M or (ii) $(c_r, d'_r), (c'_r, d_r)$ are in M.

Thus any Pareto-optimal matching in G_ψ of cost 0 is a perfect matching. Lemma 13 will be useful to us.

▶ **Lemma 13.** *Let M be a Pareto-optimal matching in G_ψ. For any $r \in \{X_1, \ldots, X_{2n}\}$, both (a_r, b'_r) and (c_r, d'_r) cannot simultaneously be in M.*

Proof. The preferences of the nodes are set such that if both (a_r, b'_r) and (c_r, d'_r) are in M then both the non-matching edges (a_r, d'_r) and (b'_r, c_r) in the alternating cycle $\rho = a_r - (d'_r, c_r) - (b'_r, a_r) - d'_r$ are blocking edges to M. Consider $M \oplus \rho$ versus M. All the 4 nodes a_r, b'_r, c_r, d'_r prefer $M \oplus \rho$ to M while the other nodes are indifferent between $M \oplus \rho$ and M. Thus $\phi(M \oplus \rho, M) = 4$ and $\phi(M, M \oplus \rho) = 0$, so $u(M) = \infty$. This means M is not Pareto-optimal, a contradiction. Thus for any $r \in \{X_1, \ldots, X_{2n}\}$, we cannot have both (a_r, b'_r) and (c_r, d'_r) in M. ◀

Theorem 14 is our main result here.

▶ **Theorem 14.** *G_ψ has a Pareto-optimal matching M with $\mathsf{cost}(M) = 0$ if and only if ψ is satisfiable.*

Proof. Suppose G_ψ has a Pareto-optimal matching M with $\mathsf{cost}(M) = 0$. For any variable $r \in \{X_1, \ldots, X_{2n}\}$, consider the edges in $\neg r$'s gadget that are in M. If $(c_r, d'_r), (c'_r, d_r)$ are in M then set $r = \mathsf{false}$ else set $r = \mathsf{true}$.

Lemma 13 tells us that when we set r to false, the edges $(a_{r,i}, b_{r,i}), (a'_{r,i}, b'_{r,i})$ from r's gadget in the i-th clause have to be in M (where $a_{r,i}, b_{r,i}, a'_{r,i}, b'_{r,i}$ are the 4 nodes from r's gadget in the i-th clause).

▷ Claim 15. The above assignment satisfies ψ.

Claim 15 uses the Pareto-optimality of M to show that every clause has at least one literal set to true. Its proof is given after the proof of Theorem 14. Hence if G_ψ admits a Pareto-optimal matching M with $\mathsf{cost}(M) = 0$, then ψ is satisfiable.

The converse. We will now show that if ψ is satisfiable then there is a Pareto-optimal matching M in G_ψ such that $\mathsf{cost}(M) = 0$. There is a natural way of constructing the matching M – we will use the satisfying assignment for ψ to choose edges from each literal gadget. For any variable r, include the following edges in the matching M:

- if $r =$ true then take the edges $(c_r, d_r), (c'_r, d'_r)$ from $\neg r$'s gadget and the edges $(a_{r,i}, b'_{r,i})$, $(a'_{r,i}, b_{r,i})$ from r's gadget in clause i (for every clause i that r belongs to).
- if $r =$ false then take the edges $(c_r, d'_r), (c'_r, d_r)$ from $\neg r$'s gadget and the edges $(a_{r,i}, b_{r,i})$, $(a'_{r,i}, b'_{r,i})$ from r's gadget in clause i (for every clause i that r belongs to).

It is easy to see that $\mathsf{cost}(M) = 0$. Since M is a perfect matching, there is no alternating path ρ wrt M such that $\phi(M, M \oplus \rho) = 0$. This is because for every alternating path ρ wrt M, we have $|M \oplus \rho| < |M|$ and the nodes matched in M and unmatched in $M \oplus \rho$ prefer M to $M \oplus \rho$, so $\phi(M, M \oplus \rho) > 0$. Hence in order to prove M's Pareto-optimality, what we need to show is Claim 16.

▷ Claim 16. There is no alternating cycle ρ with respect to M such that $\phi(M \oplus \rho, M) > 0$ and $\phi(M, M \oplus \rho) = 0$.

The proof of Claim 16 is given below. This finishes the proof of Theorem 14. ◀

Proof of Claim 15. Suppose this assignment does not satisfy ψ. We have 3 cases here.

1. Let $C_i = x \vee y \vee z$. Suppose all the three variables x, y, z are in false state. Consider the following alternating cycle ρ wrt M:

   $$b_{z,i} - (a_{x,i}, b_{x,i}) - (a_{y,i}, b_{y,i}) - (a_{z,i}, b_{z,i}) - a_{x,i}.$$

   All non-matching edges in this alternating cycle, i.e., the edges $(b_{z,i}, a_{x,i})$, $(b_{x,i}, a_{y,i})$, $(b_{y,i}, a_{z,i})$, are blocking edges with respect to M. In the $M \oplus \rho$ versus M comparison, these 6 nodes $a_{x,i}, b_{x,i}, a_{y,i}, b_{y,i}, a_{z,i}, b_{z,i}$ prefer $M \oplus \rho$ to M while all other nodes in G_ψ are indifferent between $M \oplus \rho$ and M. Thus we have $\phi(M \oplus \rho, M) = 6$ and $\phi(M, M \oplus \rho) = 0$. Hence $u(M) = \infty$, contradicting the Pareto-optimality of M.
2. Let $C_j = x \vee y$, i.e., this is a positive clause with 2 literals. Suppose both x and y are in false state. Consider the following alternating cycle ρ wrt M:

   $$b_{y,j} - (a_{x,j}, b_{x,j}) - (a_{y,j}, b_{y,j}) - a_{x,j}.$$

   In the $M \oplus \rho$ versus M comparison, the 4 nodes $a_{x,j}, b_{x,j}, a_{y,j}, b_{y,j}$ prefer $M \oplus \rho$ to M while all the other nodes in G_ψ are indifferent between $M \oplus \rho$ and M. Thus $\phi(M \oplus \rho, M) = 4$ and $\phi(M, M \oplus \rho) = 0$. Hence $u(M) = \infty$, contradicting the Pareto-optimality of M.
3. Let $D_k = \neg x \vee \neg y$. Suppose both $\neg x$ and $\neg y$ are in false state. Consider the following alternating cycle ρ wrt M:

   $$d_y - (c_x, d_x) - (c_y, d_y) - c_x.$$

In the $M \oplus \rho$ versus M comparison, the 4 nodes c_x, d_x, c_y, d_y prefer $M \oplus \rho$ to M while all the other nodes in G_ψ are indifferent between $M \oplus \rho$ and M. So $\phi(M \oplus \rho, M) = 4$ and $\phi(M, M \oplus \rho) = 0$. Hence $u(M) = \infty$, contradicting the Pareto-optimality of M.

Thus every clause in ψ has at least one literal in true state. ◁

Proof of Claim 16. We need to show there is no alternating cycle ρ with respect to M such that $\phi(M \oplus \rho, M) > 0$ and $\phi(M, M \oplus \rho) = 0$. Every non-matching edge in such an alternating cycle ρ has to be a blocking edge wrt M.

First, we argue that every consistency edge is a *non-blocking* edge to M; say, this is a consistency edge corresponding to variable r in clause i. It follows from our construction of M that M contains either:

1. $(a_{r,i}, b'_{r,i}), (a'_{r,i}, b_{r,i})$ and $(c_r, d_r), (c'_r, d'_r)$ or
2. $(a_{r,i}, b_{r,i}), (a'_{r,i}, b'_{r,i})$ and $(c_r, d'_r), (c'_r, d_r)$.

- case 1: the node d'_r prefers $M(d'_r) = c'_r$ to $a_{r,i}$ and the node c_r prefers $M(c_r) = d_r$ to $b'_{r,i}$.
- case 2: the node $a_{r,i}$ prefers $M(a_{r,i}) = b_{r,i}$ to d'_r and the node $b'_{r,i}$ prefers $M(b'_{r,i}) = a'_{r,i}$ to c_r.

Thus in both cases, the consistency edges $(a_{r,i}, d'_r)$ and $(b'_{r,i}, c_r)$ are non-blocking edges to M. Let H be the subgraph of G_ψ obtained by preserving only the edges that are in M and also blocking edges wrt M. Thus no non-blocking edge (other than edges in M) is included in H – so no consistency edge belongs to H.

Since there are no consistency edges in H, any alternating cycle in H has to be contained within a single clause. We will now show there is no such cycle in H by using the fact that we constructed M using a satisfying assignment for ψ: thus every clause has at least one literal set to true.

Let $C = x \vee y \vee z$ and suppose $y = \mathsf{true}$ in ψ. Then (a_y, b'_y) and (a'_y, b_y) are in M, however the edge (a'_y, b'_y) is non-blocking wrt M and hence it is missing in H. Thus there is no alternating cycle in H that is contained within the clause C. Now consider a negative clause $D = \neg x \vee \neg y$ and suppose $x = \mathsf{false}$ in ψ. Then (c_x, d'_x) and (c'_x, d_x) are in M, however the edge (c'_x, d'_x) is non-blocking wrt M and it is missing in H. Thus there is no alternating cycle in H that is contained within the clause D.

Consider the 4 edges of any literal gadget (say, r) in G_ψ: if $(a_r, b_r) \in M$ then (a_r, b'_r) is a non-blocking edge wrt M and if $(a'_r, b_r) \in M$ then (a'_r, b'_r) is a non-blocking edge wrt M. Similarly, in the gadget of $\neg r$: if $(c_r, d_r) \in M$ then (c_r, d'_r) is a non-blocking edge wrt M and if $(c'_r, d_r) \in M$ then (c'_r, d'_r) is a non-blocking edge wrt M. Thus there is no alternating cycle wrt M in H. So there is no alternating cycle ρ in G_ψ such that $\phi(M \oplus \rho, M) > 0$ and $\phi(M, M \oplus \rho) = 0$. ◁

Since any Pareto-optimal matching in G_ψ of cost 0 is a perfect matching, Theorem 14 shows that the min-cost Pareto-optimal matching problem and the min-cost Pareto-optimal max-matching problem are NP-hard. Moreover, these problems are NP-hard to approximate to any multiplicative factor. Thus we have shown Theorem 4 stated in Section 1.

References

1 A. Abdulkadiroğlu and T. Sönmez. School choice: a mechanism design approach. *American Economic Review*, 93(3):729–747, 2003.

2 D. J. Abraham, K. Cechlŕová, D. F. Manlove, and K. Mehlhorn. Pareto optimality in house allocation problems. In *Proceedings of the 15th International Symposium on Algorithms and Computation (ISAAC)*, pages 3–15, 2004.

3 T. Barnagarwala. 1172 private doctors sign up to treat patients at govt hospitals in Mumbai. Indian Express, May 11, 2020.

4 S. Baswana, P. P. Chakrabarti, S. Chandran, Y. Kanoria, and U. Patange. Centralized admissions for engineering colleges in India. *INFORMS Journal on Applied Analytics*, 49(5):338–354, 2019.

5 P. Biro, D. F. Manlove, and S. Mittal. Size versus stability in the marriage problem. *Theoretical Computer Science*, 411:1828–1841, 2010.

6 Canadian Resident Matching Service. How the matching algorithm works. `http://carms.ca/algorithm.htm`.

7 M.-J.-A.-N. de C. (Marquis de) Condorcet. *Essai sur l'application de l'analyse à la probabilité des décisions rendues à la pluralité des voix*. L'Imprimerie Royale, 1785.

8 Á. Cseh. Popular matchings. Trends in Computational Social Choice, Ulle Endriss (ed.), 2017.

9 Á. Cseh, Y. Faenza, T. Kavitha, and V. Powers. Understanding popular matchings via stable matchings. `arXiv:1811.06897`.

10 Á. Cseh and T. Kavitha. Popular edges and dominant matchings. *Mathematical Programming*, 172(1):209–229, 2018.

11 P. Eirinakis, D. Magos, I. Mourtos, and P. Miliotis. Polyhedral aspects of stable marriage. *Mathematics of Operations Research*, 39(3):656–671, 2014.

12 Y. Faenza and T. Kavitha. Quasi-popular matchings, optimality, and extended formulations. In *Proceedings of the 31st Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 325–344, 2020.

13 Y. Faenza, T. Kavitha, V. Powers, and X. Zhang. Popular matchings and limits to tractability. In *Proceedings of the 30th ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 2790–2809, 2019.

14 T. Feder. A new fixed point approach for stable networks and stable marriages. *Journal of Computer and System Sciences*, 45(2):233–284, 1992.

15 T. Feder. Network flow and 2-satisfiability. *Algorithmica*, 11(3):291–319, 1994.

16 T. Fleiner. A fixed-point approach to stable matchings and some applications. *Mathematics of Operations Research*, 28(1):103–126, 2003.

17 D. Gale and L.S. Shapley. College admissions and the stability of marriage. *American Mathematical Monthly*, 69(1):9–15, 1962.

18 D. Gale and M. Sotomayor. Some remarks on the stable matching problem. *Discrete Applied Mathematics*, 11:223–232, 1985.

19 P. Gärdenfors. Match making: assignments based on bilateral preferences. *Behavioural Science*, 20:166–173, 1975.

20 C.-C. Huang and T. Kavitha. Popular matchings in the stable marriage problem. *Information and Computation*, 222:180–194, 2013.

21 R. W. Irving, P. Leather, and D. Gusfield. An efficient algorithm for the "optimal" stable marriage. *Journal of the ACM*, 34(3):532–543, 1987.

22 P. B. Jaiswal. Mumbai struggles to tackle increasing number of COVID-19 cases. The Week, May 6, 2020.

23 T. Kavitha. A size-popularity tradeoff in the stable marriage problem. *SIAM Journal on Computing*, 43(1):52–71, 2014.

24 T. Kavitha. Popular half-integral matchings. In *Proceedings of the 43rd International Colloquium on Automata, Languages, and Programming (ICALP)*, pages 22:1–22:13, 2016.

25 T. Kavitha. Popular matchings with one-sided bias. In *Proceedings of the 47th International Colloquium on Automata, Languages, and Programming (ICALP)*, pages 70:1–70:18, 2020.

26 M. McCutchen. The least-unpopularity-factor and least-unpopularity-margin criteria for matching problems with one-sided preferences. In *Proceedings of the 8th Latin American Symposium on Theoretical Informatics (LATIN)*, pages 593–604, 2008.

27 S. Merrill and B. Grofman. *A unified theory of voting: directional and proximity spatial models*. Cambridge University Press, 1999.

28 National Resident Matching Program. Why the Match? `http://www.nrmp.org/whythematch.pdf`.

29 P. A. Robards. Applying the two-sided matching processes to the United States Navy enlisted assignment process. Master's Thesis, Naval Postgraduate School, Monterey, Canada, 2001.

30 U. G. Rothblum. Characterization of stable matchings as extreme points of a polytope. *Mathematical Programming*, 54:57–67, 1992.

31 C.-P. Teo and J. Sethuraman. The geometry of fractional stable matchings and its applications. *Mathematics of Operations Research*, 23(4):874–891, 1998.

32 E. G. Thurber. Concerning the maximum number of stable matchings in the stable marriage problem. *Discrete Mathematics*, 248(1-3):195–219, 2002.

33 J. H. Vande Vate. Linear programming brings marital bliss. *Operations Research Letters*, 8(3):147–153, 1989.

34 W. Yang, J. A. Giampapa, and K. Sycara. Two-sided matching for the US Navy detailing process with market complication. Technical Report CMU-R1-TR-03-49, Robotics Institute, Carnegie Mellon University, 2003.

Automorphisms and Isomorphisms of Maps in Linear Time

Ken-ichi Kawarabayashi
National Institute of Informatics, Tokyo, Japan

Bojan Mohar
Department of Mathematics, Simon Fraser University, Burnaby, Canada
IMFM, Department of Mathematics, University of Ljubljana, Slovenia

Roman Nedela
Univeristy of West Bohemia, Pilsen, Czech Republic

Peter Zeman
Department of Applied Mathematics, Faculty of Mathematics and Physics,
Charles University, Prague, Czech Republic

Abstract

A map is a 2-cell decomposition of a closed compact surface, i.e., an embedding of a graph such that every face is homeomorphic to an open disc. An automorphism of a map can be thought of as a permutation of the vertices which preserves the vertex-edge-face incidences in the embedding. When the underlying surface is orientable, every automorphism of a map determines an angle-preserving homeomorphism of the surface. While it is conjectured that there is no "truly subquadratic" algorithm for testing map isomorphism for unconstrained genus, we present a linear-time algorithm for computing the generators of the automorphism group of a map, parametrized by the genus of the underlying surface. The algorithm applies a sequence of local reductions and produces a uniform map, while preserving the automorphism group. The automorphism group of the original map can be reconstructed from the automorphism group of the uniform map in linear time. We also extend the algorithm to non-orientable surfaces by making use of the antipodal double-cover.

2012 ACM Subject Classification Theory of computation → Design and analysis of algorithms

Keywords and phrases maps on surfaces, automorphisms, isomorphisms, algorithm

Digital Object Identifier 10.4230/LIPIcs.ICALP.2021.86

Category Track A: Algorithms, Complexity and Games

Related Version *Full Version*: `https://arxiv.org/abs/2008.01616`

Funding *Ken-ichi Kawarabayashi*: JSPS Kakenhi Grant Number JP18H05291 and 20H05965
Bojan Mohar: Supported in part by the NSERC Discovery Grant R611450 (Canada), and by the Research Project J1-2452 of ARRS (Slovenia).
Roman Nedela: Supported by Slovak Research and Development Agency under Grant No. APVV-19-0308 and by GAČR 20-15576S.
Peter Zeman: Supported by GAUK 1224120 and GAČR 20-15576S.

1 Introduction

The graph isomorphism problem asks whether or not two given graphs are isomorphic. It is one of the most fundamental problems in the theory of algorithms. It is probably the most notorious problem whose computational complexity is still a huge open question, even after Babai's recent quasipolynomial-time breakthrough [2]. While some complexity theoretic results indicate that this problem is unlikely NP-complete (if it was, the polynomial hierarchy would collapse to its second level, see [28]), no polynomial-time algorithm is known, even with extended resources like randomization or quantum computing.

On the other hand, there is a number of important classes of graphs on which the graph isomorphism problem is known to be solvable in polynomial time. These include graphs with bounded degree [23, 9], bounded eigenvalue multiplicity [3], bounded tree-width [22, 10], excluded small minors [11], etc.

In this paper, we are interested in planar graphs and, more generally, graphs of bounded genus. In 1966, Weinberg [30] gave a very simple quadratic algorithm for the graph isomorphism of planar graphs. This was improved by Hopcroft and Tarjan [16] to $\mathcal{O}(n \log n)$. Building, on this earlier work, Hopcroft and Wong [17] published in 1974 a paper, where they described a linear-time algorithm for isomorphism testing of planar graphs.

For graphs on surfaces of higher genus, the graph isomorphism problem seems much harder. This can be perhaps explained in the following way. We can rather easily reduce the problem to 3-connected graphs. For planar graphs, the famous result of Whitney [31] says that embeddings of 3-connected planar graphs in the plane are (combinatorially) unique. But for every simply connected surface, there exist 3-connected graphs with exponentially many embeddings. This makes an essential difference between planar graphs and graphs of higher genus.

For quite a long time it has been known that the isomorphism of bounded genus graphs can be solved in time $n^{\mathcal{O}(g)}$, where g is the genus of the underlying surface; see for example [27]. However, an interesting question is whether the result of Hopcroft and Wong [17] can be generalized also for the bounded genus graphs, i.e., whether the isomorphism problem for graphs of bounded genus can be solved in time $f(g) \cdot n$, for some computable function f. This motivates the study of the isomorphism problem for embedded graphs first.

By a *topological map* we mean a 2-cell decomposition of a closed compact surface, i.e., an embedding of a graph into a surface such that every face is homeomorphic to an open disc. An *isomorphism* of two maps is an isomorphism of the underlying graphs, which preserves the vertex-edge-face incidences. In particular, a map isomorphism induces a homeomorphism of the underlying surfaces. Our main result reads as follows.

▶ **Theorem 1.** *Let M_1 and M_2 be maps on a surface of genus g. The set of all isomorphisms* $\mathrm{Iso}(M_1, M_2)$ *from M_1 to M_2 can be determined in time $f(g) \cdot (\|M_1\| + \|M_2\|)$, where f is some computable function and $\|M\|$ denotes the size of the map M.*

In [21], two of the authors deal with a much weaker version of this problem, where only testing isomorphism is considered instead of constructing the whole set $\mathrm{Iso}(M_1, M_2)$. Recently, a linear-time algorithm was announced [19] for testing isomorphism of bounded genus graphs and the proposed approach heavily relies on our result. It should be also mentioned, that an algorithm with running time $n^{\mathcal{O}((\log g)^c)}$ for bounded genus graphs follows from [26], however, this result is based on completely different techniques.

Determining the set of all isomorphisms between two maps is closely related to finding the generators of the automorphism group $\mathrm{Aut}(M)$ of a map M, where an automorphism of M is just an isomorphism $M \to M$. More precisely, the set of all isomorphisms $M_1 \to M_2$ can be expressed as a composition $\psi \cdot \mathrm{Aut}(M_1)$ where $\psi : M_1 \to M_2$ is any isomorphism. Thus, our first result goes hand-in-hand with the following.

▶ **Theorem 2.** *Let M be a map on a surface of genus g. The generators of the automorphism group* $\mathrm{Aut}(M)$ *of M can be computed in time $f(g) \cdot \|M\|$, where f is some computable function and $\|M\|$ denotes the size of the map M.*

Colbourn and Booth [7] proposed a way to modify the Hopcroft-Wong algorithm [17] to compute the generators of the automorphism group of a spherical map. However, they state the following: "*We ... base our automorphism algorithms on the Hopcroft-Wong algorithm. Necessarily, we will only be able to sketch our procedure. A more complete description and a*

proof of correctness would require a more thorough analysis of the Hopcroft-Wong algorithm than has yet appeared in the literature." Sadly, the situation has not changed since, and the only available description of the Hopcroft-Wong algorithm is the extended abstract [17], which contains no proof of correctness and running time.[1] Our contribution also fills in this gap and we obtain much better insight into the Hopcroft-Wong algorithm by solving the problem in a much greater generality; see [20] as well.

Roughly speaking, the key idea of the Hopcroft-Wong algorithm is to try to apply contractions of edges to obtain two smaller isomorphic maps. In order to do this, edges must be chosen canonically, which is not always possible. Since Hopcroft and Wong consider only the spherical case, this situation occurs only in one special case. However, on the surfaces of higher genus, this situation is quite common and requires a completely different, more systematic, approach. As a consquence of considering the problem on the higher genus, our approach turnes out to be much simpler even for planar graphs than the approach originally proposed by Colbourn and Booth [7].

The Hopcroft-Wong algorithm reduces spherical maps to maps having the same degrees of vertices and also the same degrees of faces (e.g. Platonic solids). These maps are then treated separately. We, however, relax this condition and instead reduce our map to a map having the same cyclic vector of face sizes at each vertex (e.g. on sphere these also include Archimedean solids). The number of such maps is bounded for surfaces of genus $g > 1$, and for surfaces of genus $g \leq 1$ we give some special algorithms. This, surprisingly, allows a much more unified method of reducing the map, while preserving its automorphisms and isomorphisms.

Simultaneous conjugation problem. The problems of testing isomorphism of maps and computing the generators of the automorphism group of a map are related to the problem of *simultaneous conjugation*. In the latter problem, the input consists of two sets of permutations $\alpha_1, \ldots, \alpha_d$ and $\beta_1, \ldots, \beta_d$ on the set $\{1, \ldots, n\}$, each of which generates a transitive subgroup of the symmetric group. The goal is to find a permutation γ such that $\gamma\alpha_i\gamma^{-1} = \beta_i$, for $i = 1, \ldots, d$. Let us observe that this problem is a generalization of the map isomorphism problem. If α_1 and β_1 are involutions, $d = 2$, and the set $\{1, \ldots, n\}$ is identified with the set of darts of a map on a surface (see Section 2 for definitions), then this problem is exactly the map isomorphism problem. If further $\alpha_1 = \beta_1$ and $\alpha_2 = \beta_2$, we get the map automorphism problem.

Since mid 1970s it has been known that the simultaneous conjugation problem can be solved in time $\mathcal{O}(dn^2)$ [8, 15]. A faster algorithm, with running time $\mathcal{O}(n^2 \log d/\log n + dn \log n)$, was found only recently [6]. This implies an $\mathcal{O}(n^2/\log n)$ algorithm for the isomorphism and automorphism problems for maps of unrestricted genus. In complexity theory, this is not considered to be a "truly subquadratic" algorithm. This motivates the following conjecture.

▶ **Conjecture 3.** *There is no $\varepsilon > 0$ for which there is an algorithm for testing isomorphism of maps of unrestricted genus in time $\mathcal{O}(n^{2-\varepsilon})$.*

An interesting open subproblem is to prove a conditional "truly superlinear" lower bound for any of the mentioned problems. There has been some progress in the direction of providing a lower bound. In particular it is known that the communication complexity of the simultaneous conjugation problem is $\Omega(dn \log(n))$, for $d > 1$, and that under the decision tree model the search version of the simultaneous conjugation problem has lower bound of $\Omega(n \log n)$ [5].

[1] The PhD thesis of Wong also does not bring any new information compared to [17].

2 Preliminaries

A *map* M is an embedding $\iota\colon X \to S$ of a connected graph X to a closed connected compact surface S such that every connected component of $S \setminus \iota(X)$ is homeomorphic to an open disc. The connected components are called *faces*. By $V(M)$, $E(M)$, and $F(M)$ we denote the sets of vertices, edges, and faces of M, respectively. We put $v(M) := |V(M)|$, $e(M) := |E(M)|$, and $f(M) := |F(M)|$.

Recall that closed connected compact surfaces are characterized by two invariants: orientability and the Euler characteristic χ. For the orientable surfaces, the latter can be replaced by the *(orientable) genus* $g \geq 0$, which is the number of tori in the connected sum decomposition of the surface, and for the non-orientable surfaces by the *non-orientable genus* $\gamma \geq 1$, which is the number of real projective planes in the connected sum decomposition of the surface. The following is well-known.

▶ **Theorem 4** (Euler-Poincaré formula). *Let M be a map on a surface S. Then $v(M) - e(M) + f(M) = \chi(S) = 2 - 2g$ if S has genus g and $\chi(S) = 2 - \gamma$ if S has non-orientable genus γ.*

We give an algebraic description of a map, where a map is defined by means of three fixed-point-free involutions acting on *flags*. A flag is a triple representing a vertex-edge-face incidence. The involutions are simply instructions on how to join the flags together to form a map. There are several advantages: (i) in such a form, maps can be easily passed to an algorithm as an input, (ii) verifying whether a mapping is an automorphism reduces to checking three commuting rules, and (iii) group theory techniques can be applied to obtain results about maps. For more details see for example [18] and [13, Section 7.6].

Oriented maps. Even though our main concern is in general maps, a large part of our algorithm deals with maps on orientable surfaces, where the algebraic description is simpler. An *oriented map* is a map on an orientable surface with a fixed global orientation. Every oriented map can be combinatorially described as a triple (D, R, L). Here, D is the set of *darts*. By a dart we mean an edge endowed with one of two possible orientations. Hence, each edge gives rise to two darts. The permutation $R \in \mathrm{Sym}(D)$, called *rotation*, is the product $R = \Pi_{v \in V} R_v$, where each R_v cyclically permutes the darts originating at $v \in V$, following the chosen orientation around v. The *dart-reversing involution* $L \in \mathrm{Sym}(D)$ is an involution of D that, for each edge, swaps the two oppositely oriented darts arising from the edge.

Formally, a *combinatorial oriented map* is any triple $M = (D, R, L)$, where D is a finite non-empty set of *darts*, R is any permutation of darts, L is a fixed-point-free involution of D, and the group $\langle R, L\rangle \leq \mathrm{Sym}(D)$ is transitive on D. By the size $\|M\|$ of the map, we mean the number of darts $|D|$. We require transitivity because the maps are connected by definition.

The group $\langle R, L\rangle$ is called the *monodromy group* of M. The vertices, edges, and faces of M are in one-to-one correspondence with the cycles of the permutations R, L, $R^{-1}L$, respectively. By the phrase "a dart x is incident to a vertex v" we mean that $x \in R_v$. Similarly, "x is incident to a face f" means that x belongs to the boundary walk of f defined by the respective cycle of $R^{-1}L$. By the *degree of a face* we mean the length of its boundary walk. A face of degree d will be called a d-face. Note that each dart is incident to exactly one face. For convenience, we frequently use a shorthand notation $x^{-1} = Lx$, for $x \in D$. The *dual* of an oriented map $M = (D, R, L)$ is the oriented map $M^* = (D, R^{-1}L, L)$.

Apart from standard map theory references, we need to introduce labeled maps. A *planted tree* is a rooted tree embedded in the sphere, i.e., a planted tree is a spherical map having exactly one face. We say that a planted tree is *integer-valued* if an integer is assigned to each vertex. A *dart-labeling* of an oriented map $M = (D, R, L)$ is a mapping $\ell\colon D \to \mathcal{T}$, where $\mathcal{T}$ is the set of integer-valued planted trees. A *labeled oriented map* M is a 4-tuple (D, R, L, ℓ). The *dual map* is the map M^* defined as $M^* = (D, R^{-1}L, L, \ell)$.

Two labeled oriented maps $M_1 = (D_1, R_1, L_1, \ell_1)$ and $M_2 = (D_2, R_2, L_2, \ell_2)$ are *isomorphic*, in symbols $M_1 \cong M_2$, if there exists a bijection $\psi\colon D_1 \to D_2$, called an *orientation-preserving isomorphism* from M_1 to M_2, such that

$$\psi R_1 = R_2\psi, \quad \psi L_1 = L_2\psi, \quad \text{and} \quad \ell_1 = \ell_2\psi. \tag{1}$$

The *set of orientation-preserving isomorphisms* from M_1 to M_2 is denoted by $\mathrm{Iso}^+(M_1, M_2)$. The *orientation-preserving automorphism group* of M is the set $\mathrm{Aut}^+(M) := \mathrm{Iso}^+(M, M)$. The following statement, which can be easily seen for unlabeled maps, extends also to labeled maps.

▶ **Theorem 5.** *Let M_1 and M_2 be labeled oriented maps with sets of darts D_1 and D_2, respectively. For every $x \in D_1$ and every $y \in D_2$, there exists at most one isomorphism $M_1 \to M_2$ mapping x to y. In particular, $\mathrm{Aut}^+(M_1)$ is fixed-point-free on D_1.*

▶ **Corollary 6.** *Let M_1 and M_2 be labeled oriented maps with sets of darts D_1 and D_2, respectively. If $x \in D_1$ and $y \in D_2$, then it can be checked in time $\mathcal{O}(|D_1| + |D_2|)$ whether there is an isomorphism mapping x to y.*

Chirality. The *mirror image* of an oriented map $M = (D, R, L)$ is the oriented map $M^{-1} = (D, R^{-1}, L)$. Similarly, the *mirror image* of labeled oriented map $M = (D, R, L, \ell)$ is the map $M^{-1} = (D, R^{-1}, L, \ell^{-1})$, where $\ell^{-1}(x)$ is the mirror image of $\ell(x)$ for each $x \in D$.

An oriented map M is called *reflexible* if $M \cong M^{-1}$. Otherwise the maps M and M^{-1} form a *chiral pair*. For example, all the Platonic solids are reflexible. The set of *all isomorphisms* from M_1 to M_2 is defined as $\mathrm{Iso}(M_1, M_2) := \mathrm{Iso}^+(M_1, M_2) \cup \mathrm{Iso}^+(M_1, M_2^{-1})$. Similarly, we put $\mathrm{Aut}(M) := \mathrm{Iso}(M, M)$.

Maps on all surfaces. Let M be a map on any, possibly non-orientable, surface. In general, a *combinatorial non-oriented map* is a quadruple (F, λ, ρ, τ), where F is a finite non-empty set of *flags*, and $\lambda, \rho, \tau \in \mathrm{Sym}(F)$ are fixed-point-free[2] involutions such that $\lambda\tau = \tau\lambda$ and the group $\langle \lambda, \rho, \tau \rangle$ acts transitively on F. By the *size* $\|M\|$ of the map M we mean the number of flags $|F|$.

Each flag corresponds uniquely to a vertex-edge-face incidence triple (v, e, f). Geometrically, it can be viewed as the triangle defined by v, the center of e, and the center of f. The group $\langle \lambda, \rho, \tau \rangle$ is called the *non-oriented monodromy group* of M. The vertices, edges, and faces of M correspond uniquely to the orbits of $\langle \rho, \tau \rangle$, $\langle \lambda, \tau \rangle$, and $\langle \rho, \lambda \rangle$, respectively. Similarly, an *isomorphism* of two non-oriented maps M_1 and M_2 is a bijection $\psi\colon F_1 \to F_2$ which commutes with λ, ρ, τ. The even-word subgroup $\langle \rho\tau, \tau\lambda \rangle$ has index at most two in the monodromy group of M. If it is exactly two, the map M is called *orientable*. For every oriented map (D, R, L) it is possible to construct the corresponding non-oriented map

[2] It is possible to extend the theory to maps on surfaces with boundaries by allowing fixed points of λ, ρ, τ.

(F, λ, ρ, τ). Conversely, from an orientable non-oriented map (F, λ, ρ, τ) it is possible to construct two oriented maps (D^+, R, L) and (D^-, R^{-1}, L), where D^+ and D^- are the two orbits of the even word subgroup, and $L = \tau\lambda$, $R = \rho\tau$.

Test of orientability. For a non-oriented map $M = (F, \lambda, \rho, \tau)$, it is possible to test in linear time if M is orientable [12, 24]. The *barycentric subdivision* B of M is constructed by placing a new vertex in the center of every edge and face, and then joining the centers of faces with the incident vertices and with the center of the incident edges. The dual of B is a 3-valent map, i.e., every vertex is of degree 3.

▶ **Theorem 7.** *A map $M = (F, \lambda, \rho, \tau)$ is orientable if and only if the underlying 3-valent graph of the dual of the barycentric subdivision of M is bipartite.*

Light vertices. A map is called *face-normal*, if all its faces are of degree at least three. It is well-known that every face-normal map on the sphere or on the projective plane has a vertex of degree at most 5. The next theorem generalizes this for other surfaces.

▶ **Theorem 8.** *Let S be a closed compact surface with Euler characteristic $\chi(S) \leq 0$ and let M be a face-normal map on S. Then there is a vertex of valence at most $6(1 - \chi(S))$.*

Proof. A bound for maximum degree is achieved by a triangulation, thus we may assume that M is a triangulation. We have $f = 2e/3$. By plugging this in the Euler-Poincaré formula and using the Handshaking lemma, we obtain $3v - \overline{d}v/2 = 3\chi(S)$, where $\overline{d}$ is the average degree. By manipulating the equality, we get $\overline{d} - 6 = -6\chi(S)/v$. Since $\chi(S) \leq 0$, the right hand side is maximized for $v = 1$. We conclude that $\overline{d} \leq 6(1 - \chi(S))$. ◀

A vertex is called *light* if it is of minimum degree, otherwise it is called *heavy*.

Uniform and homogeneous maps. Given a map on an orientable surface, the cyclic vector of degrees of faces incident with a vertex v, induced by the chosen global orientation, is called the *local type* of v. A map is *uniform*[3] if the local types of all vertices are the same. A map is *homogeneous* of type $\{k, \ell\}$ if every vertex is of degree k and every face is of degree ℓ.

A *dipole* is a 2-vertex spherical map dual to a spherical cycle. *A bouquet* is a one-vertex map that is a dual of a *planted star* (a tree with at most one vertex of degree > 1).

▶ **Example 9.** The face-normal uniform spherical maps are: the 5 Platonic solids, the 13 Archimedean solids, pseudo-rhombicuboctahedron, prisms, antiprisms, and cycles of length at least 3. It easily follows from Euler's formula that the spherical homogeneous maps are the 5 Platonic solids, cycles, and dipoles.

3 Overview of the algorithm

We provide a high-level overview of the whole algorithm determining the automorphism group of a map. The input consists of a non-oriented map given by the quadruple $N = (F, \lambda, \rho, \tau)$.

First, using Theorem 7, we test whether N is orientable or not. If the map is orientable, then we know that the underlying surface is orientable and we fix a global orientation of the surface. We construct two oriented maps $M = (D, R, L)$ and $M^{-1} = (D, R^{-1}, L)$ representing N.

[3] In [1] Babai uses the term semiregular instead of uniform.

We start by determining $\mathrm{Aut}^+(M)$. On the map M, we perform a sequence of elementary local reductions (Section 4). There are two types of reductions: normalization and elimination of vertices of minimum degree. The normalization is of the highest priority and its purpose is to ensure that the resulting map is face-normal. In a face-normal map, it is guaranteed by Theorem 8 that there is a vertex of small degree. The second elementary reduction replaces a vertex of minimum degree by a polygon connecting its higher-degree neighbours and reconnecting the other incident edges (see Figure 3). These two reductions are applied until we are left with a map which has all vertices of degree k. Now, we observe that our reductions do not really depend on the degrees of vertices, but rather on some vertex-labelling (not related to dart labelling) which is linearly ordered. At this stage we can no longer distinguish vertices based on their degreee. We refine the procedure by using the local types instead of degrees. Note that the local types can be linearly ordered. It follows from Theorem 8 that the number of local types sufficient to consider is bounded. Thus, our reduction can be applied in the same way, but instead of degrees we use local types. The result is a labeled face-normal uniform oriented map $M' = (D', R', L', \ell')$ with $\mathrm{Aut}^+(M) \cong \mathrm{Aut}^+(M')$ and $D' \subseteq D$; for more details see Section 4.

The number of face-normal uniform oriented map M' on a surface of genus $g > 1$ is bounded by a fucntion of g (Proposition 13), which means that a brute-force approach is sufficient to determine $\mathrm{Aut}^+(M')$. For the case of sphere and torus, the problem is non-trivial since there are infinite families of face-normal uniform maps and a special treatment is necessary; for more details see Section 5. Now, since $\mathrm{Aut}^+(M)$ acts fixed-point-freely on D and $D' \subseteq D$, there is a unique way to extend $\mathrm{Aut}^+(M')$ to $\mathrm{Aut}^+(M)$. Finally, to construct $\mathrm{Aut}(M)$, we run the whole algorithm again to determine $\mathrm{Iso}(M, M^{-1})$.

If the map is N is non-orientable, we construct its oriented antipodal double-cover $\widetilde{M} = (D, R, L) = (F, \rho\tau, \tau\lambda)$. We show that $\mathrm{Aut}(N) \leq \mathrm{Aut}^+(\widetilde{M})$, and therefore, we can again apply our algorithm to determine $\mathrm{Aut}^+(\widetilde{M})$. Here, the most difficult part is to determine $\mathrm{Aut}(N)$ within $\mathrm{Aut}^+(\widetilde{M})$. For the case of projective plane and Klein bottle the problem is highly non-trivial and a special treatment is again needed, while for the other cases, again, a brute force approach is sufficient; for more details see Section 6.

4 From oriented maps to uniform oriented maps

In this section, we describe a set of elementary reductions defined on labeled oriented maps, given by a quadruple (D, R, L, ℓ), in detail. The output of each elementary reduction is always a quadruple (D', R', L', ℓ'), satisfying $D' \subseteq D$, $v(M') + e(M') < v(M) + e(M)$, and $\mathrm{Aut}^+(M') \cong \mathrm{Aut}^+(M)$. If none of the reductions apply, the map is a uniform oriented map. The procedure defines a function which assigns to a given oriented map M a unique labeled oriented map U with $\mathrm{Aut}^+(M) \cong \mathrm{Aut}^+(U)$. Since the darts of U form a subset of the darts of M, by semiregularity, every generator of $\mathrm{Aut}^+(U)$ can be extended to a generator of $\mathrm{Aut}^+(M)$ in linear time. We deal with the uniform oriented maps in Section 5.

After every elementary reduction, to ensure that $\mathrm{Aut}^+(M') = \mathrm{Aut}^+(M)$, we need to define a new labeling ℓ'. To this end, in the whole section, we assume that we have an injective function $\mathbf{Label}\colon \mathbb{N} \times \bigcup_{k=1}^{\infty} \mathcal{T}^k \to \mathcal{T}$, where $\mathcal{T}$ is the set of all integer-valued planted trees. Moreover, we assume that the root of $\mathbf{Label}(t, T_1, \ldots, T_k)$ contains the integer t, corresponding to the current step of the reduction procedure. After every elementary reduction, this integer is increased by one; see the full version for more details.

Even though we defined our reductions only based on the minimum degree, it can be easily seen that we are only using the fact that natural numbers are linearly ordered. Thus, our reduction really works with any vertex lables, which are linearly ordered. In particular,

if we replace degrees with local types together with a natural lexicographic linear ordering, our reductions are well-defined. The consequence is thet every irreducible map with respect to these reductions is a face-normal uniform map.

Normalization. By Theorem 8, there is always a light vertex in a face-normal map. The purpose of the following reduction is to remove faces of degree one and two. This reduction is of the highest priority and it is applied until the map is one of the following: (i) face-normal, (ii) bouquet, (iii) dipole. In the cases (ii) and (iii), the whole reduction procedure stops with a uniform map. In the case (i), the reduction procedure continues with further reductions. We describe the reduction formally.

For technical reasons we split the reduction into two parts: deletion of loops, denoted by $\mathbf{Loops}(M)$, and replacement of a dipole by an edge, denoted by $\mathbf{Dipoles}(M)$.

Reduction Loops. If $M = (D, R, L, \ell)$ with $v(M) > 1$ contains loops, we remove them. Let $\mathcal{L}$ be the list of all maximal sequences of darts of the form $s = \{x_1, x_1^{-1}, \dots, x_k, x_k^{-1}\}$, where $Rx_i = x_i^{-1}$, for $i = 1, \dots, k$, $Rx_i^{-1} = x_{i+1}$ for $i = 1, \dots, k-1$, and $Rx_k^{-1} \neq x_1$. By definition, $R^{-1}Lx_i = x_i$, hence x_i bounds a 1-face, for $i = 1, \dots, k-1$; see Figure 1. Moreover, for each such sequence s, all the darts x_i are incident to the same vertex $v \in V(M)$. We say that the unique vertex v with $R_v = (x_0, x_1, x_1^{-1}, \dots, x_k, x_k^{-1}, x_{k+1}, \dots)$ is *incident* to s. We call the darts x_0 and x_{k+1} the *bounding darts* of the sequence s.

The new map $M' = (D'; R', L', \ell') =: \mathbf{Loops}(M)$ is defined as follows. First, we put $D' := D \setminus \bigcup_{s \in \mathcal{L}} s$, and $L' := L_{\restriction D'}$. Let $s = \{x_1, x_1^{-1}, \dots, x_k, x_k^{-1}\} \in \mathcal{L}$ with bounding darts x_0 and x_{k+1}. If v is incident to s, then we put $R'_v := (x_0, x_{k+1}, \dots)$, else we put $R'_v := R_v$. Moreover, we put $\ell'(x_0) := \mathbf{Label}(t, a_0, \dots, a_k)$ and $\ell'(x_{k+1}) := \mathbf{Label}(t, a_{k+1}, b_k, \dots, b_1)$, where t is the current step, $a_i = \ell(x_i)$, for $i = 0, \dots, k+1$, and $b_i = \ell(x_i^{-1})$, for $i = 1, \dots, k$. For every $x \in D'$ which is not a bounding dart in M, we put $\ell'(x) := \ell(x)$. We obtain a well-defined map M' with no faces of valence one; see Figure 1.

▶ **Lemma 10.** *Let $M_i = (D_i, R_i, L_i, \ell_i)$, $i = 1, 2$ where $D_1 \cap D_2 = \emptyset$, be labeled oriented maps. Let $M'_1 := \mathbf{Loops}(M_1)$ and $M'_2 := \mathbf{Loops}(M_2)$. Then $\mathrm{Iso}^+(M_1, M_2)_{\restriction D'_1} = \mathrm{Iso}^+(M'_1, M'_2)$. In particular, $\mathrm{Aut}^+(M_1)_{\restriction D'_1} = \mathrm{Aut}^+(M'_1)$.*

Reduction Dipoles. If $M = (D, R, L, \ell)$ with $v(M) > 2$ contains dipoles. Let $\mathcal{L}$ be the list of all maximal sequences $s = (x_1, \dots, x_k)$ of darts, $k > 1$, satisfying $Rx_i = x_{i+1}$, $(R^{-1}L)^2 x_i = x_i$, and either $Rx_k \neq x_1$ or $Rx_1^{-1} \neq x_k^{-1}$; see Figure 2. Let $s^{-1} := (x_k^{-1}, \dots, x_1^{-1}) \in \mathcal{L}$ be the *inverse* sequence. There are vertices u and v such that $R_u = (y_1, s, y_2, \dots)$ and

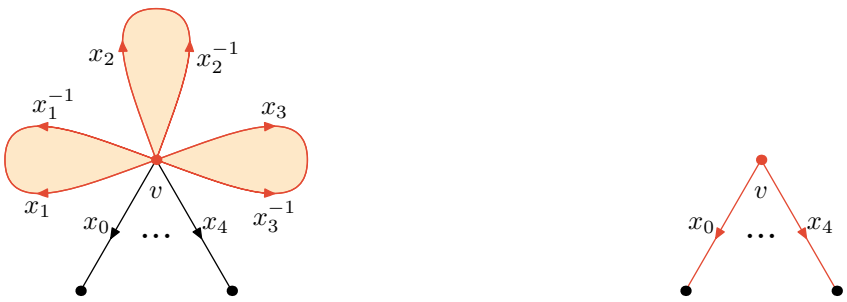


Figure 1 A sequence of darts $x_1, x_1^{-1}, x_2, x_2^{-1}, x_3, x_3^{-1}$ with bounding darts x_0 and x_4.

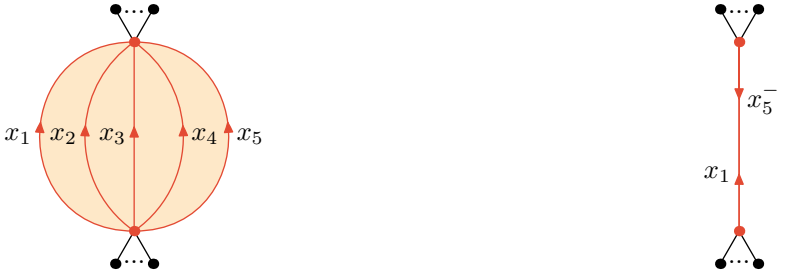


Figure 2 A sequence of darts $x_1, \dots, x_5$ forming a dipole.

$R_v = (z_1, s^{-1}, z_2, \dots)$, for some $y_1, y_2, z_1, z_2 \in D$. At least one of the sets $\{y_1, y_2\}$, $\{z_1, z_2\}$ is non-empty since otherwise $v(M) = 2$ and M is a dipole. We say that u, v are *incident* to s, s^{-1}, respectively; see Figure 2

The new map $M' = (D', R', L', \ell') =: \mathbf{Dipoles}(M)$ is defined as follows. First, we put

$$D' := D \setminus \bigcup_{(x_1,\dots,x_k)\in\mathcal{L}} \{x_2, \dots, x_k\} \cup \{x_1^{-1}, \dots, x_{k-1}^{-1}\}.$$

Let $s = (x_1, \dots, x_k) \in \mathcal{L}$. If u and v are incident to s and s^{-1}, respectively, then we put $R'_u := (y_1, x_1, y_2 \dots)$ and $R'_v := (z_1, x_k^{-1}, z_2, \dots)$, else we put $R'_u := R_u$. Next, we put $L'x_1 := x_k^{-1}$, $L'x_k^{-1} := x_1$, and $L'x := Lx$ if $x \notin s \in \mathcal{L}$. Finally, we put $\ell'(x_1) := \mathbf{Label}(t, a_1, \dots, a_k)$ and $\ell'(x_k^{-1}) := \mathbf{Label}(t, b_k, \dots, b_1)$, where t is the current step, $a_i = \ell(x_i)$ and $b_i = \ell(x_i^{-1})$, for $i = 1, \dots, k$. We put $\ell'(x) := \ell(x)$ for $x \notin s \in \mathcal{L}$. We obtain a well-defined map M' with no 2-faces; see Figure. 2.

▶ **Lemma 11.** *Let $M_i = (D_i, R_i, L_i, \ell_i)$, $i = 1, 2$ where $D_1 \cap D_2 = \emptyset$, be labeled oriented maps. Let $M'_1 := \mathbf{Dipoles}(M_1)$ and $M'_2 := \mathbf{Dipoles}(M_2)$. Then $\mathrm{Iso}^+(M_1, M_2)_{\restriction D'_1} = \mathrm{Iso}^+(M'_1, M'_2)$. In particular, $\mathrm{Aut}^+(M_1)_{\restriction D'_1} = \mathrm{Aut}^+(M'_1)$.*

Face-normal maps. The input is a labeled face-normal oriented map $M = (D, R, L, \ell)$ and a list $\mathcal{L}$ of all light vertices of degree d which have at least one heavy neighbour. For every vertex $v \in \mathcal{L}$, we denote by $u_0, \dots, u_{k-1}$, for some $1 \le k \le d$, the cyclic sequence of all heavy neighbours of v, following the prescribed orientation of the underlying surface. Denote by $x_0, x_1, \dots, x_{k-1}$ the darts based at $u_0, u_1, \dots, u_{k-1}$, joining u_j to v for $j = 0, \dots, k-1$. Let $R_{u_i} = (y_i, x_i, z_i, \dots)$, for $i = 0, \dots, k-1$, and let

$$R_v = (x_0^{-1}, A_0, x_1^{-1}, A_1, \dots, x_{k-1}^{-1}, A_{k-1}),$$

where each A_i is a (possibly empty) sequence of darts.

The new map $M' = (D', R', L', \ell') =: \mathbf{Delete}(M)$ is defined as follows. We set $D' := D$ and $L' := L$. For a heavy vertex w with no light neighbour, we have $R'_w := R_w$. If $v \in \mathcal{L}$, with the above notation, we set $R'_{u_i} := (y_i, A_i, x_i, x_{i-1}^{-1}, z_i, \dots)$. Moreover, we set $\ell'(x_i) := \mathbf{Label}(t, \ell(x_i))$ and $\ell'(x_i^{-1}) := \mathbf{Label}(t, \ell(x_i^{-1}))$, where t is the current step number; see Figure 3.

▶ **Lemma 12.** *Let $M_i = (D_i, R_i, L_i, \ell_i)$, $i = 1, 2$ where $D_1 \cap D_2 = \emptyset$, be labeled oriented maps. Let $M'_1 := \mathbf{Delete}(M_1)$ and $M'_2 := \mathbf{Delete}(M_2)$. Then $\mathrm{Iso}^+(M_1, M_2) = \mathrm{Iso}^+(M'_1, M'_2)$. In particular, $\mathrm{Aut}^+(M_1) = \mathrm{Aut}^+(M'_1)$.*

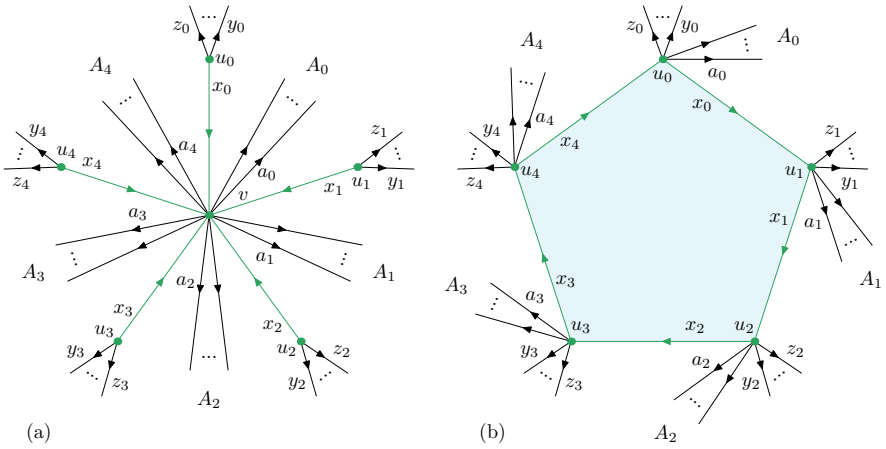


Figure 3 An example of the reduction deleting a vertex.

Proof. Let $\psi\colon M_1 \to M_2$ be an isomorphism. We prove that ψ is also an isomorphism of M_1' and M_2'. We check the commuting rules (1) for ψ. We have $L_i' = L_i$, for $i = 1, 2$, so $L_1'\psi = \psi L_2'$. For R_1' and R_2', we need to check the commuting rules only at $x_i, x_i^{-1}, y_i, a_i \in D_1'$, for $i = 0, \ldots, k-1$, where a_i is the last dart in the sequence A_i. We have

$$\psi R_1' x_i = \psi R_1^{-1} L_1 x_i = R_2^{-1} L_2 \psi x_i = R_2' \psi x_i,$$

$$\psi R_1' x_i^{-1} = \psi R_1 L_1 x_i^{-1} = R_2 L_2 \psi x_i^{-1} = R_2' \psi x_i^{-1}.$$

It remains to check the commuting rules at each y_i and a_i. Note that if A_i is empty there is nothing to check. We have

$$\psi R_1' y_i = \psi R_1 L_1 R_1 y_i = R_2 L_2 R_2 \psi y_i = R_2' \psi y_i.$$

Further, using the relations $R_1' a_i = x_i = L_1 R_1^q a_i$, for some $q > 0$, we get

$$\psi R_1' a_i = \psi x_i = \psi L_1 R_1^q a_i = L_2 R_2^q \psi a_i = R_2' \psi a_i.$$

Putting it together, we proved that $\psi R_1' = R_2' \psi$. Clearly, $\ell_1'(x_i) = \ell_2'(\psi x_i)$ if and only if $\ell_1(x_i) = \ell_2(\psi x_i)$. Similarly for x_i^{-1}.

For the converse, we assume that $\psi R_1' = R_2' \psi$ and $\psi L_1' = L_2' \psi$ and we prove $\psi R_1 = R_2 \psi$ and $\psi L_1 = L_2 \psi$. Similarly as above, we need to check the commuting rules for $x_i, x_i^{-1}, y_i, a_i \in D_1$.

- By the definition of M_1' and M_2', we have $R_1 x_i = z_i = (R_1')^2 x_i$. Since **Label** is injective, we have $R_2 \psi x_i = \psi z_i = (R_2')^2 \psi x_i$. Using these relations, we get

$$\psi R_1 x_i = \psi (R_1')^2 x_i = (R_2')^2 \psi x_i = R_2 \psi x_i.$$

- By the definition of M_1' and M_2', we have $R_1 x_i^{-1} = R_1'^m L_1' x_i^{-1}$, for some m. Since **Label** is injective, we have $R_2 \psi x_i^{-1} = R_2'^m L_2' \psi x_i^{-1}$. Using these relations, we get

$$\psi R_1 x_i^{-1} = \psi R_1'^m L_1 x_i^{-1} = R_1'^m L_2' \psi x_i^{-1} = R_2 \psi x_i^{-1}.$$

- By the definition of M_1' and M_2', we have $R_1 y_i = x_i = R_1'^m y_i$, for some m. Since **Label** is injective, $R_2 \psi y_i = \psi x_i = R_2'^m \psi y_i$. Using these relations, we get

$$\psi R_1 y_i = \psi R_1'^m y_i = R_2'^m \psi y_i = R_2 \psi y_i.$$

- By the definition of M_1' and M_2', we have $R_1 a_i = L_1' R_1'^{-1} L_1' R_1' a_i$. Since **Label** is injective, $R_2 \psi a_i = L_2' R_2'^{-1} L_2' R_2' \psi a_i$. Using these relations, we get

$$\psi R_1 a_i = \psi L_1' R_1'^{-1} L_1' R_1' a_i = L_2' R_2'^{-1} L_2' R_2' \psi a_i = R_2 \psi a_i.$$

Putting it togehter, we proved that $\psi R_1 = R_2 \psi$, which implies that ψ is an isomorphism $M_1 \to M_2$. This completes the proof. ◀

5 Irreducible maps on orientable surfaces

In this section, we provide an algorithm computing the automorphism group of irreducible oriented maps, with fixed Euler characteristic, in linear time. The proof splits into three parts: maps of negative Euler characteristic, maps on the sphere, and maps on torus.

Surfaces of negative Euler characteristic. If the Euler characteristic χ is negative, the irreducible maps are exactly all the uniform face-normal maps. We prove that the number of uniform face-normal maps is bounded by a function of χ. Therefore, generators of the automorphism group can be computed by a brute force approach. Note that the following lemma does not require the underlying surface to be orientable, it only requires χ to be negative.

▶ **Proposition 13.** *The number of edges of a uniform face-normal map on a closed compact surface S with Euler characteristic $\chi(S) < 0$ is bounded by a function of $\chi(S)$.*

Proof. Babai noted in [1, Theorem 3.3] that the Hurwitz Theorem (see, e.g. [4] or [12]) implies that the number of vertices of a uniform map M on S is at most $84|\chi(S)|$. By Theorem 8, the degree of a vertex of M is bounded by a function of $\chi(S)$ as well. Therefore, the number of edges is also bounded by a function of $\chi(S)$ and the theorem follows. ◀

▶ **Corollary 14.** *Let $M = (D, R, L)$ be a uniform face-normal map $M = (D, R, L)$ on an orientable surface S with $\chi(S) < 0$. Then $\mathrm{Aut}(M)$ can be computed in time $f(\chi(S))|D|$, for some computable function f.*

Sphere. By the definition of the reductions in Section 4, the irreducible spherical maps are the five Platonic maps, 13 Archimedean maps, pseudo-rhombicuboctahedron, prisms, antiprisms, cycles, dipoles, and bouquets.

In the first three cases, the automorphism group can be computed by a brute force approach. We show that for (labeled) prisms, antiprisms, dipoles and bouquets, the problem can be reduced to computing the automorphism group of a vertex-labeled cycle.

▶ **Theorem 15** ([17])**.** *If $M = (D, R, L)$ is an irreducible spherical map, then the generators of $\mathrm{Aut}(M)$ can be computed in time $\mathcal{O}(|D|)$.*

Torus. The toroidal irreducible maps are uniform face-normal maps. The universal covers of uniform toroidal maps are uniform tilings (infinite maps with finite vertex and face degrees) of the Euclidean plane. There are 12 of such tilings; see [14, page 63]. The corresponding local types are $(3,3,3,3,3,3)$, $(4,4,4,4)$, $(6,6,6)$, $2 \times (3,3,3,3,6)$, $(3,3,3,4,4)$, $(3,3,4,3,4)$, $(3,4,6,4)$, $(3,6,3,6)$, $(3,12,12)$, $(4,6,12)$, and $(4,8,8)$. One type occurs in two forms, one of the respective tilings is the mirror image of the other. Each of these tilings T gives rise to an infinite family of toroidal uniform maps as follows. It is well-known that $\mathrm{Aut}^+(T)$ is

isomorphic either to the triangle group $\Delta(4,4,2)$ or to $\Delta(6,3,2)$. Each of these contains an infinite subgroup H of translations generated by two shifts. Every finite uniform toroidal map of the prescribed local type can be constructed as the quotient T/K, where K is a subgroup of H of finite index.

First, our algorithm reduces a uniform map to one of the two homogeneous types $\{4,4\}$ and $\{6,3\}$, while preserving the automorphism group. Then, the algorithm computes the generators of the automorphism groups of a labeled homogeneous toroidal map M of type $\{4,4\}$ or $\{3,6\}$. For technical reasons, we transform the dart-labelling to a vertex-labelling of M. These transformations can be done easily by, for a given vertex, encoding the lables of the outgoing darts into the vertex. The following lemma describes some important properties of $\mathrm{Aut}^+(M)$.

▶ **Lemma 16** ([29]). *Let M be a toroidal map of type $\{4,4\}$ or $\{6,3\}$. The orientation-preserving automorphism group of a labeled map M is a semidirect product $T \rtimes H$, where T is a direct product of two cyclic groups, and $|H| \leq 6$. Moreover, the action of T is regular on the vertices of M.*

Since the order of H is bounded by a constant, it takes linear time to check whether every element of H is a label-preserving automorphism. The main difficulty is to find T. The subgroup T is generated by α and β, where α is the horizontal, and β is the vertical shift by the unit distance. Now the meaning of the parameters r, s, t is the following: $|\alpha| = r$, $\alpha^t = \beta^s$, and s is the least power of β such that $\beta^s \in \langle\alpha\rangle$. The following lemma shows that T can always be written as a direct product of two cyclic groups.

▶ **Lemma 17.** *There exists δ and γ such that $T = \langle\delta\rangle \times \langle\gamma\rangle$. Moreover, δ and γ can be computed in time $\mathcal{O}(rs)$.*

Lemma 17 can be viewed as a transformation of the shifted grid $\mathcal{G}$ to the orthogonal grid $\mathcal{G}^\perp$. Note that the underlying graph may change, but both $\mathcal{G}$ and $\mathcal{G}^\perp$ are Cayley graphs based on the group T, therefore, the vertex-labeling naturally transfers. Thus, we may assume that $t = 0$ and $T = \langle\alpha\rangle \times \langle\beta\rangle \cong \mathbb{Z}_r \times \mathbb{Z}_s$. We need to compute generators of the label-preserving subgroup of T.

From now on, we assume that we are given a *cyclic orthogonal grid* $\mathcal{G}$ of size rs, which is graph with vertices identified with $(i,j) \in G$, where $G = \mathbb{Z}_r \times \mathbb{Z}_s$. For every (i,j), there is an edge between (i,j) and $(i+1 \mod r, j)$, and between (i,j) and $(i, j+1 \mod s)$. Moreover, we are given an integer-labeling ℓ of the vertices of $\mathcal{G}$. Clearly, $\mathcal{G}$ determines the *ℓ-preserving subgroup* H of G, namely

$$H = \{(x,y) : \forall (i,j) \in G, \ell(i,j) = \ell(i+x, j+y)\}.$$

The goal is to find the generators of H in time $\mathcal{O}(rs)$.

We give a description of any subgroup of the direct product of G that is suitable for our algorithm. First, we define four important mappings. The two *projections* $\pi_1 \colon G \to \mathbb{Z}_r$ and $\pi_2 \colon G \to \mathbb{Z}_s$ are defined by $\pi_1(x,y) = x$ and $\pi_2(x,y) = y$, respectively. The two *inclusions* $\iota_1 \colon \mathbb{Z}_r \to G$ and $\iota_2 \colon \mathbb{Z}_s \to G$ are defined by $\iota_1(x) = (x,0)$ and $\iota_2(y) = (0,y)$, respectively.

▶ **Lemma 18.** *Let $G = \mathbb{Z}_r \times \mathbb{Z}_s$ for $r, s \geq 1$, and let H be a subgroup of G. Then there are $a, c \in \mathbb{Z}_r$ and $b \in \mathbb{Z}_s$ such that*

$$H = \{(ia + jc, jb) : i, j \in \mathbb{Z}\} = \langle (a,0), (c,b) \rangle,$$

where $\langle a\rangle = \iota_1^{-1}(H)$, $\langle b\rangle = \pi_2(H)$, and $c < a$ is the minimum integer such that $(c,b) \in H$.

This description suggests an algorithm to find the generators of the given subgroup H of $\mathbb{Z}_r \times \mathbb{Z}_s$. In our setting, the subgroup H is given on the input by a labeling function ℓ, defined on the vertices of the $r \times s$ orthogonal grid. The subgroup H is the ℓ-preserving subgroup of $\mathbb{Z}_r \times \mathbb{Z}_s$.

To compute the generators of H, it suffices, by Lemma 18, to determine $a, c \in \mathbb{Z}_r$ and $b \in \mathbb{Z}_s$ such that $\langle a \rangle = \iota_1^{-1}(H)$, $\langle b \rangle = \pi_2(H)$, and c is the smallest integer such that $(c, b) \in H$. Then $H = \langle (a, 0), (c, b) \rangle$.

▶ **Lemma 19.** *There is an $\mathcal{O}(rs)$-time algorithm which computes the integers a, b, c such that $\iota_1^{-1}(H) = \langle a \rangle$, $\pi_2(H) = \langle b \rangle$ and $c < a$ is the smallest integer such that $(c, b) \in H$.*

The results of this subsection are summarized by the following.

▶ **Theorem 20.** *If $M = (D, R, L, \ell)$ is a uniform face-normal labeled toroidal map, then the generators of $\mathrm{Aut}(M)$ can be computed in time $\mathcal{O}(|D|)$.*

6 Non-orientable surfaces

For a map M on a non-orientable surface S, we reduce the problem of computing the generators of $\mathrm{Aut}(M)$ to the problem of computing the generators of $\mathrm{Aut}^+(\widetilde{M})$, for some orientable map $\widetilde{M}$. In particular, the map $\widetilde{M}$ is the antipodal double cover of M.

Given a map $M = (F, \lambda, \rho, \tau)$ on a non-orientable surface of genus γ, we define the antipodal double cover $\widetilde{M} = (D, R, L)$ by setting $D := F$, $R := \rho\tau$, and $L := \tau\lambda$. Since M is non-orientable, we have $\langle R, L \rangle = \langle \lambda, \rho, \tau \rangle$, so $\langle R, L \rangle$ is transitive and $\widetilde{M}$ is well-defined. For more details on this construction see [25]. We note that $\widetilde{\chi} = 2\chi$, where $\widetilde{\chi}$ and χ is the Euler characteristic of the underlying surface of $\widetilde{M}$ and M, respectively.

▶ **Lemma 21.** *We have $\mathrm{Aut}(M) \leq \mathrm{Aut}^+(\widetilde{M})$.*

Proof. Let $\varphi \in \mathrm{Aut}(M)$. Then we have $R^\varphi = (\rho\tau)^\varphi = \rho^\varphi\tau^\varphi = \rho\tau = R$ and $L^\varphi = (\tau\lambda)^\varphi = \tau^\varphi\lambda^\varphi = \tau\lambda = L$. ◀

▶ **Lemma 22.** *We have $\mathrm{Aut}(M) = \{\varphi \in \mathrm{Aut}^+(\widetilde{M}) : \varphi\tau = \tau\varphi\}$.*

Proof. Let $\varphi \in \mathrm{Aut}^+(\widetilde{M})$. We have $\varphi R\varphi^{-1} = R$ and $\varphi L\varphi^{-1} = L$. By plugging in $R = \rho\tau$ and $L = \tau\lambda$, we obtain

$$\varphi(\rho\tau)\varphi^{-1} = \rho\tau \quad \text{and} \quad \varphi(\tau\lambda)\varphi^{-1} = \lambda\tau.$$

From there, by rearranging the left-hand sides of the equations, we get

$$(\varphi\rho\varphi^{-1})(\varphi\tau\varphi^{-1}) = \varphi(\rho\tau)\varphi^{-1} = \rho\tau \quad \text{and} \quad (\varphi\tau\varphi^{-1})(\varphi\lambda\varphi^{-1}) = \varphi(\tau\lambda)\varphi^{-1} = \tau\lambda.$$

Finally, we obtain

$$\varphi\rho\varphi^{-1} = \rho\tau(\varphi\tau\varphi^{-1}) \quad \text{and} \quad \varphi\lambda\varphi^{-1} = (\varphi\tau\varphi^{-1})\tau\lambda.$$

If $\varphi \in \mathrm{Aut}(M)$, then, in particular, it commutes with τ. On the other hand, if φ commutes with τ, then the last two equations imply that it also must commute with ρ and λ, i.e., $\varphi \in \mathrm{Aut}(M)$. ◀

The previous lemmas are key and suggest an approach for computing the generators of the automorphism group of M. In particular, it is necessary to check which automorphisms of $\widetilde{M}$ commute with τ. The cases when the underlying surface is the projective plane, or the Klein bottle, must be treated separately.

▶ **Theorem 23.** *Let $M = (F, \lambda, \rho, \tau)$ be a map on a non-orientable a non-orietable surface of genus γ. Then it is possible to compute the generators of $\mathrm{Aut}(M)$ in time $f(\gamma)|F|$.*

References

1 L. Babai. Vertex-transitive graphs and vertex-transitive maps. *Journal of graph theory*, 15(6):587–627, 1991.

2 L. Babai. Graph isomorphism in quasipolynomial time. In *Proceedings of the forty-eighth annual ACM symposium on Theory of Computing*, pages 684–697. ACM, 2016.

3 L. Babai, D. Y. Grigoryev, and D. M. Mount. Isomorphism of graphs with bounded eigenvalue multiplicity. In *Proceedings of the fourteenth annual ACM symposium on Theory of computing*, pages 310–324. ACM, 1982.

4 N. L. Biggs and A. T. White. *Permutation groups and combinatorial structures*, volume 33. Cambridge University Press, 1979.

5 A. Brodnik, A. Malnič, and R. Požar. Lower bounds on the simultaneous conjugacy problem in the symmetric group. In *4th annual Mississippi Discrete Mathematics Workshop*, 2015.

6 A. Brodnik, A. Malnič, and R. Požar. Fast permutation-word multiplication and the simultaneous conjugacy problem. *arXiv preprint*, 2019. `arXiv:1907.07889`.

7 C. J. Colbourn and K. S. Booth. Linear time automorphism algorithms for trees, interval graphs, and planar graphs. *SIAM Journal on Computing*, 10(1):203–225, 1981.

8 M. Fontet. Calcul de centralisateur d'un groupe de permutations. *Bull. Soc. Math. France Mém.,(49-50)*, pages 53–63, 1977.

9 M. Grohe, D. Neuen, and P. Schweitzer. A faster isomorphism test for graphs of small degree. In *2018 IEEE 59th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 89–100, 2018.

10 M. Grohe, D. Neuen, P. Schweitzer, and D. Wiebking. An improved isomorphism test for bounded-tree-width graphs. *ACM Trans. Algorithms*, 16(3):34:1–34:31, 2020.

11 M. Grohe, D. Wiebking, and D. Neuen. Isomorphism testing for graphs excluding small minors. In *61st IEEE Annual Symposium on Foundations of Computer Science (FOCS)*, pages 625–636, 2020.

12 J. L. Gross and T. W. Tucker. *Topological graph theory*. Dover, 2001.

13 J. L. Gross, J. Yellen, and P. Zhang. *Handbook of graph theory*. Chapman and Hall/CRC, 2013.

14 B. Grünbaum and G. C. Shephard. *Tilings and patterns*. Freeman, 1987.

15 Christoph M Hoffmann. Subcomplete generalizations of graph isomorphism. *Journal of Computer and System Sciences*, 25(3):332–359, 1982.

16 J. E. Hopcroft and R. Endre. Tarjan. A V log V algorithm for isomorphism of triconnected planar graphs. *J. Comput. Syst. Sci.*, 7(3):323–331, 1973.

17 J. E. Hopcroft and J. K. Wong. Linear time algorithm for isomorphism on planar graphs. In *Proc. 6th Annual ACM Symp. on Theory of Computing.*, 1974.

18 G. A. Jones and D. Singerman. Theory of maps on orientable surfaces. *Proceedings of the London Mathematical Society*, 3(2):273–307, 1978.

19 K. Kawarabayashi. Graph isomorphism for bounded genus graphs in linear time. *arXiv preprint*, 2015. `arXiv:1511.02460`.

20 K. Kawarabayashi, P. Klavík, B. Mohar, R. Nedela, and P. Zeman. Isomorphisms of maps on the sphere. In *Polytopes and Discrete Geometry*, volume 764 of *Contemporary Mathematics*, pages 125–147. American Mathematical Society, 2021.

21 K. Kawarabayashi and B. Mohar. Graph and map isomorphism and all polyhedral embeddings in linear time. In *Proceedings of the fortieth annual ACM symposium on Theory of computing*, pages 471–480. ACM, 2008.

22 D. Lokshtanov, M. Pilipczuk, M. Pilipczuk, and S. Saurabh. Fixed-parameter tractable canonization and isomorphism test for graphs of bounded treewidth. *SIAM Journal on Computing*, 46(1):161–189, 2017.

23 E. M. Luks. Isomorphism of graphs of bounded valence can be tested in polynomial time. *Journal of computer and system sciences*, 25(1):42–65, 1982.

24 B. Mohar and C. Thomassen. *Graphs on surfaces*, volume 10. Johns Hopkins University Press, 2001.

25 R. Nedela and M. Škoviera. Exponents of orientable maps. *Proceedings of the London Mathematical Society*, 75(1):1–31, 1997.

26 D. Neuen. Hypergraph isomorphism for groups with restricted composition factors. In *47th International Colloquium on Automata, Languages, and Programming, (ICALP)*, volume 168 of *LIPIcs*, pages 88:1–88:19, 2020.

27 I. N. Ponomarenko. The isomorphism problem for classes of graphs closed under contraction. *Journal of Soviet Mathematics*, 55(2):1621–1643, 1991.

28 U. Schöning. Graph isomorphism is in the low hierarchy. *Journal of Computer and System Sciences*, 37(3):312–323, 1988.

29 Ondrej Šuch. Vertex-transitive maps on a torus. *Acta Math. Univ. Comenianae*, 80(1):1–30, 2011.

30 L. Weinberg. A simple and efficient algorithm for determining isomorphism of planar triply connected graphs. *IEEE Transactions on Circuit Theory*, 13(2):142–148, 1966.

31 H. Whitney. 2-isomorphic graphs. *American Journal of Mathematics*, 55(1):245–254, 1933.

Lower Bounds on Dynamic Programming for Maximum Weight Independent Set

Tuukka Korhonen
Department of Computer Science, University of Helsinki, Finland

Abstract

We prove lower bounds on pure dynamic programming algorithms for maximum weight independent set (MWIS). We model such algorithms as tropical circuits, i.e., circuits that compute with max and + operations. For a graph G, an MWIS-circuit of G is a tropical circuit whose inputs correspond to vertices of G and which computes the weight of a maximum weight independent set of G for any assignment of weights to the inputs. We show that if G has treewidth w and maximum degree d, then any MWIS-circuit of G has $2^{\Omega(w/d)}$ gates and that if G is planar, or more generally H-minor-free for any fixed graph H, then any MWIS-circuit of G has $2^{\Omega(w)}$ gates. An MWIS-formula is an MWIS-circuit where each gate has fan-out at most one. We show that if G has treedepth t and maximum degree d, then any MWIS-formula of G has $2^{\Omega(t/d)}$ gates. It follows that treewidth characterizes optimal MWIS-circuits up to polynomials for *all* bounded degree graphs and H-minor-free graphs, and treedepth characterizes optimal MWIS-formulas up to polynomials for all bounded degree graphs.

2012 ACM Subject Classification Theory of computation → Graph algorithms analysis

Keywords and phrases Maximum weight independent set, Treewidth, Tropical circuits, Dynamic programming, Treedepth, Monotone circuit complexity

Digital Object Identifier 10.4230/LIPIcs.ICALP.2021.87

Category Track A: Algorithms, Complexity and Games

Related Version *Full Version*: `https://arxiv.org/abs/2102.06901`

Funding This work has been financially supported by Academy of Finland (grant 322869).

Acknowledgements I wish to thank Prafullkumar Tale for suggesting to generalize the result from planar graphs to H-minor-free graphs. I also thank Matti Järvisalo, Andreas Niskanen, and anonymous reviewers for helpful comments.

1 Introduction

In this paper, we prove lower bounds for *tropical circuits* computing the weight of a maximum weight independent set (MWIS) of a graph. A tropical circuit is a circuit with Max and Plus operations as gates. In particular, we consider *MWIS-circuits* of graphs. An MWIS-circuit of a graph G is a tropical circuit whose inputs correspond to the vertices of G and which computes the weight of a maximum weight independent set of G for any assignment of weights to the inputs. An MWIS-formula is an MWIS-circuit where each gate has fan-out at most one.

Our motivation for proving lower bounds for MWIS-circuits is that many algorithmic techniques for maximum weight independent set implicitly build an MWIS-circuit of the input graph, and therefore the running time of any algorithm resulting from such a technique is bounded from below by the minimum size of an MWIS-circuit of the graph. Examples of algorithmic techniques that build MWIS-circuits are dynamic programming over different kinds of decompositions of graphs [3, 8, 16] and dynamic programming over potential maximal cliques [10, 17, 29]. Examples of algorithmic techniques that build MWIS-formulas are branching [20, 34] and maximal independent set enumeration [32].

1.1 Our Results

We prove unconditional lower bounds for sizes of MWIS-circuits and MWIS-formulas parameterized by graph parameters treewidth and treedepth, respectively. The lower bounds are exponential in treewidth and treedepth, and therefore well-known algorithms yield matching upper bounds for them [3, 18]. We emphasize that our lower bounds are not worst-case bounds over graph classes, but instead hold for each individual graph.

MWIS-Circuits and Treewidth

First we characterize optimal MWIS-circuits of bounded degree graphs.

▶ **Theorem 1.** *Let G be any graph with treewidth w and maximum degree d. Any MWIS-circuit of G has $2^{\Omega(w/d)}$ gates.*

Theorem 1 is optimal up to a factor d in the sense that for each pair w, d we can construct a graph with treewidth $\Omega(w)$ and maximum degree $O(d)$ that admits an MWIS-formula with $d2^{w/d}$ gates.

Then we extend the result to some graphs that may have high-degree vertices. A graph H is a *minor* of a graph G if it can be obtained from G by vertex deletions, edge deletions, and edge contractions. If H can be obtained by only vertex deletions and edge contractions, then it is an *induced minor*.

▶ **Theorem 2.** *Let G be any graph that contains an induced minor with treewidth w and maximum degree d. Any MWIS-circuit of G has $2^{\Omega(w/(d4^d))}$ gates.*

In Theorem 2 it is essential to require an induced minor instead of a minor because a complete graph with n vertices admits an MWIS-circuit of size $O(n)$, but contains all n-vertex graphs as minors.

A graph G is H-minor-free if it does not contain the graph H as a minor.

▶ **Corollary 3.** *Let H be any fixed graph and G any H-minor-free graph with treewidth w. Any MWIS-circuit of G has $2^{\Omega(w)}$ gates.*

Proof. For any fixed H, every H-minor-free graph of treewidth w contains an $\Omega(w) \times \Omega(w)$-grid as an induced minor [13][1]. An $\Omega(w) \times \Omega(w)$-grid has treewidth $\Omega(w)$ and maximum degree 4, so the result follows from Theorem 2. ◀

Corollary 3 implies a $2^{\Omega(w)}$ lower bound for all planar graphs because planar graphs are K_5-minor-free [27].

The following corollary follows from Theorem 1, Corollary 3, constant-factor treewidth approximation in $2^{O(w)}n^{O(1)}$ time [36], and dynamic programming over a tree decomposition [3].

▶ **Corollary 4.** *There is an algorithm which, given a bounded degree or H-minor-free graph G whose smallest MWIS-circuit has τ gates, constructs an MWIS-circuit of G with $\tau^{O(1)}$ gates in $\tau^{O(1)}$ time.*

In particular, a property analogous to automatizability of proof systems [6] holds for MWIS-circuits on bounded degree graphs and H-minor-free graphs.

[1] The stated result in [13] is that such a grid is a minor, but the same proof works directly to show that the constructed grid minor is also an induced minor. In particular, the proof in [13] does not use any edge deletions, and the corresponding result for bounded-genus graphs that it depends on [14] is already stated in terms of contraction to a graph that can be turned into a grid by removing vertices without decreasing treewidth by more than a constant factor.

MWIS-Formulas and Treedepth

We characterize optimal MWIS-formulas of bounded degree graphs.

▶ **Theorem 5.** *Let G be any graph with treedepth t and maximum degree d. Any MWIS-formula of G has $2^{\Omega(t/d)}$ gates.*

Again, Theorem 5 is optimal up to a factor d by the same construction as Theorem 1. As formulas can be thought of as bounded space analogies of circuits, Theorem 5 gives further evidence (in addition to e.g. [9, 26, 35]) supporting that while treewidth is the right parameter for CSP-like problems when equipped with unlimited space, treedepth is the right parameter when dealing with bounded space.

Obtaining a constant-factor single-exponential time parameterized approximation algorithm for treedepth is a well-known open problem [12], so while we know that the converse of Theorem 5 existentially holds in bounded degree graphs, we currently do not know how to construct such MWIS-formulas without having the treedepth decomposition as an input.

1.2 Techniques

Our main circuit complexity tool is an adaptation of a circuit decomposition lemma used in e.g. [22, 23, 37]. In particular, we show that this lemma can be adapted so that given an MWIS-circuit with τ gates of a graph with treewidth w it extracts a family of τ vertex separators each of size $\Omega(w)$. Once this family has been extracted, the main challenge for proving Theorem 1 is to show that if this family of separators is too small, there exists an independent set that intersects all of the separators. For this we use the lopsided Lovász Local Lemma [15], though we note that more elementary arguments would suffice to prove the theorem with a worse dependency on d. To extend the result from bounded degree graphs to H-minor-free graphs we use the minor model of the bounded degree induced minor with high treewidth to further control the structure of these separators.

For MWIS-formulas parameterized by treedepth t we similarly extract a family of 2τ vertex sets each of size $\Omega(t)$ from a τ-gate MWIS-formula, showing that if an independent set intersects all of these vertex sets the formula cannot compute it. The same application of the Local Lemma is used to prove that such an independent set indeed exists in low degree graphs if τ is too small. The argument for extracting the family from the formula is more ad-hoc than the argument for circuits.

1.3 Related Work

The convention of modeling dynamic programming algorithms as tropical circuits originates from the recent works of Jukna [24, 25], although some earlier results in monotone arithmetic circuit complexity apply also to tropical circuits [23]. In general, tropical circuit lower bounds imply lower bounds for monotone arithmetic circuits, but not necessarily the other way around [24]. In addition to the works of Jukna, the other works explicitly giving lower bounds for tropical circuits or formulas that we are aware of are [30, 31]. We are not aware of prior works on lower bounds for tropical circuits or formulas considering maximum weight independent set or the graph parameters treewidth or treedepth.

There are multiple worst-case hardness results related to different formulations of the independent set polynomial. In [7] it was shown that the multivariate independent set polynomial is VNP-complete. The univariate independent set polynomial is #P-hard to evaluate at every non-zero rational point [5], and more fine-grainedly its evaluation has $2^{\Omega(n/\log^3 n)}$ worst-case complexity assuming #ETH [21].

Chvátal has shown that a certain proof system for maximum independent set which naturally corresponds to branching algorithms requires exponential size proofs on almost all graphs that have the number of edges linear in the number of vertices [11].

Multiple worst-case lower bounds of form $n^{\Omega(w)}$ in limited models of computation for graph homomorphism problems of a pattern graph with treewidth w to a graph with n vertices are known [4, 26, 28]. In particular, recently it was shown that the worst-case monotone arithmetic circuit complexity of homomorphism polynomial is $\Theta(n^{w+1})$, and the worst-case monotone arithmetic formula complexity is $\Theta(n^t)$, where t is the treedepth of the pattern graph [26].

Recently, a lower bound of $2^{\Omega(w)}$ was shown for DNNF-compilation of monotone CNFs with primal treewidth w and bounded degree and arity, applying to all such CNFs [2]. We note that after the acceptance of this paper, we became aware of a reduction from MWIS-circuits to DNNFs that allows to prove a weaker version of our Theorem 1 via the result of [2]. In particular, the techniques of [2] yield an exponent of form $\Omega(w/2^d)$ instead of the best possible $\Omega(w/d)$ given in Theorem 1.

1.4 Organization

In Section 2 we present preliminaries on graph theory, define MWIS-circuits and prove simple lemmas on them, and discuss the lopsided Lovász Local Lemma and prove a lemma using it. In Section 3 we prove the lower bounds for MWIS-circuits parameterized by treewidth, i.e., Theorems 1 and 2. In Section 4 we prove the lower bound for MWIS-formulas parameterized by treedepth, i.e., Theorem 5. In Section 5 we give the construction that shows the optimality of Theorems 1 and 5 up to a factor d. We conclude and discuss future work in Section 6.

2 Preliminaries

2.1 Graphs

The vertex set of a graph G is denoted by $V(G)$ and the edge set by $E(G)$. The set of neighbors of a vertex v is denoted by $N(v)$ and the neighborhood of a vertex set X by $N(X) = \bigcup_{v\in X} N(v) \setminus X$. Closed neighborhoods are denoted by $N[v] = N(v) \cup \{v\}$ and $N[X] = N(X) \cup X$. The subgraph $G[X]$ induced by a vertex set $X \subseteq V(G)$ has $V(G[X]) = X$ and $E(G[X]) = \{\{u, v\} \in E(G) \mid u \in X \wedge v \in X\}$. We also use $G \setminus X = G[V(G) \setminus X]$ to denote induced subgraphs. An independent set of G is a vertex set I such that $G[I]$ has no edges. In particular, an empty set is an independent set.

A tree decomposition of a graph G is a tree T whose each vertex $i \in V(T)$ corresponds to a bag $B_i \subseteq V(G)$, satisfying that

1. $V(G) = \bigcup_{i\in V(T)} B_i$,
2. for each $\{u, v\} \in E(G)$ there is a bag B_i with $\{u, v\} \subseteq B_i$, and
3. for each $v \in V(G)$ the subtree of T induced by bags containing v is connected.

The width of a tree decomposition is $\max |B_i| - 1$ and the treewidth $tw(G)$ of a graph G is the minimum width over its tree decompositions.

A treedepth decomposition of a graph G is a rooted forest F with vertex set $V(F) = V(G)$, satisfying for each $\{u, v\} \in E(G)$ that u and v have an ancestor-descendant relation in F. The depth of F is the maximum number of vertices on a simple path from a root to a leaf in F. The treedepth $td(G)$ of a graph G is the minimum depth over its treedepth decompositions. Note that $tw(G) + 1 \leq td(G)$.

2.2 MWIS-Circuits

We start by giving a formal definition of a tropical circuit. Our definition is non-standard in that it does not allow any other input constants than 0, which we can w.l.o.g. assume in the context of maximum weight independent set. For a comprehensive treatment of tropical circuits and their relations to monotone Boolean and monotone arithmetic circuits see [24].

▶ **Definition 6.** A tropical circuit over variables X is a directed acyclic graph with in-degree of each vertex either 0 or 2. The vertices are called gates, the in-degree of a gate is called fan-in, and the out-degree of a gate is called fan-out. Each gate with fan-in 0 is labeled with a variable $x_i \in X$ or the constant 0 and each gate with fan-in 2 is labeled with either max or $+$. One gate is designated as the output gate. A tropical formula is a tropical circuit where each gate has fan-out at most 1.

With an assignment of real numbers to the variables X, a tropical circuit outputs a number computed by the output gate by natural semantics, i.e., a gate labeled with a variable x_i computes the value of x_i, a gate labeled with 0 computes 0, a gate labeled with $+$ computes the sum of the values computed by its children, and a gate labeled with max computes the maximum of the values computed by its children. In particular, a tropical circuit computes a tropical polynomial in the variables X over the tropical $(\mathbb{R} \cup \{-\infty\}, \max, +)$ semiring. In the tropical semiring max corresponds to addition and $+$ corresponds to multiplication, with $-\infty$ as the zero and 0 as the unit. We will refer to max as addition and to $+$ as multiplication.

We define an *MWIS-polynomial* with the following simple lemma.

▶ **Lemma 7.** *Let G be a graph. A tropical circuit over variables $V(G)$ computes the weight of a maximum weight independent set of G for any assignment of real weights to the inputs if and only if for the tropical polynomial f computed by the circuit it holds that*

1. *each monomial of f is of form $v_1 \cdot \ldots \cdot v_l$, where $\{v_1, \ldots, v_l\}$ is an independent set of G and*
2. *for each independent set $\{v_1, \ldots, v_l\}$ of G there is a monomial $v_1 \cdot \ldots \cdot v_l$ in f, including the empty independent set corresponding to the empty product 0.*

Proof. For the if-direction, (1) guarantees that the value computed by the circuit is at most the weight of a maximum weight independent set and (2) guarantees that the value is at least the weight of a maximum weight independent set.

For the only if-direction, if some monomial would not be multilinear, i.e., include a factor v^2 for some vertex v, the output would be incorrect when assigning weight 1 to v and 0 to other vertices. If some monomial would be of form $v_1 \cdot \ldots \cdot v_l$, where $\{v_1, \ldots, v_l\}$ is not an independent set the output would be incorrect when assigning weight 1 to those v_i and 0 to others. Finally, if the output polynomial would not include $v_1 \cdot \ldots \cdot v_l$ as a monomial for some independent set $\{v_1, \ldots, v_l\}$ then the circuit would be incorrect when assigning weight 1 to vertices of this independent set and -1 to others. ◀

An MWIS-polynomial of a graph G is a polynomial f satisfying (1) and (2) in Lemma 7. An MWIS-circuit of G is a tropical circuit that computes an MWIS-polynomial of G. An MWIS-formula of G is an MWIS-circuit of G that is a tropical formula.

We note that requiring the circuit to work for all real weights is not a strong assumption: Any MWIS-circuit that works for weights $\{0, 1\}$ can be turned into an MWIS-circuit that works for weights $\mathbb{R} \cup \{-\infty\}$ by replacing each input variable v_i by $\max(v_i, 0)$. In particular, the weight of an empty independent set is 0, so negative weights will never be used. Our assumption that the only constant available to the circuit is 0 can be justified by noting

that if an output monomial would contain a positive constant the circuit would be incorrect on the all-zero input, and that if an output monomial would contain a negative constant it should also occur without the constant. In particular, any other constants than 0 could be replaced by 0.

Next we make some simple observations on the structure of MWIS-circuits.

▶ **Definition 8.** A partial MWIS-polynomial is a polynomial f satisfying (1) in Lemma 7. A partial MWIS-circuit is a tropical circuit computing a partial MWIS-polynomial.

Note that by monotonicity of $(\max, +)$ computations we can assume that each gate of an MWIS-circuit computes a partial MWIS-polynomial and therefore each subcircuit is a partial MWIS-circuit.

▶ **Definition 9.** Let f be a partial MWIS-polynomial. We denote by $\mathrm{Sup}(f)$ the support of f, that is, the variables that occur in the monomials of f.

We also use $\mathrm{Sup}(g)$ for a gate g to denote the support of the polynomial computed by the gate. Note that each monomial of f corresponds to an independent set of $G[\mathrm{Sup}(f)]$.

The following property is the basis for proving lower bounds for MWIS-circuits.

▶ **Lemma 10.** *Let $f = g \cdot h$ be a partial MWIS-polynomial of a graph G. The sets $N[\mathrm{Sup}(g)]$ and $\mathrm{Sup}(h)$ are disjoint.*

Proof. If there was a vertex $v \in \mathrm{Sup}(g) \cap \mathrm{Sup}(h)$ then f would contain a monomial with a factor v^2. If there was a vertex $v \in \mathrm{Sup}(g)$ and $u \in \mathrm{Sup}(h)$ with $\{u, v\} \in E(G)$, then there would be a monomial in f containing a factor $u \cdot v$. ◀

We will say that a partial MWIS-polynomial f or a circuit computing f *computes* an independent set I if f contains the monomial $\prod_{v_i \in I} v_i$. In particular, an MWIS-polynomial computes every independent set.

2.3 Lopsided Lovász Local Lemma

The lopsided Lovász Local Lemma [15] (see [1] for the general version) is a method for showing that there is a non-zero probability that none of the events in a collection of events hold. In particular, we use it to show that independent sets satisfying certain requirements exist.

▶ **Definition 11.** Let $\mathcal{E}_1, \ldots, \mathcal{E}_n$ be events in a probability space. A graph Γ is a negative dependency graph of the events if its vertices are $V(\Gamma) = \{\mathcal{E}_1, \ldots, \mathcal{E}_n\}$ and for all events $\mathcal{E}_i$ and subsets $J \subseteq V(\Gamma) \setminus N(\mathcal{E}_i)$ it holds that $\Pr[\bigcup_{j \in J} \mathcal{E}_j \mid \mathcal{E}_i] \geq \Pr[\bigcup_{j \in J} \mathcal{E}_j]$.

In words, the negative dependency graph should capture all negative correlations between the events.

▶ **Proposition 12** ([1])**.** *Let $\mathcal{E}_1, \ldots, \mathcal{E}_n$ be a collection of events with a negative dependency graph Γ. If there exists real numbers $x_1, \ldots, x_n$ with $0 < x_i < 1$ such that for each i it holds that $\Pr[\mathcal{E}_i] \leq x_i \prod_{\mathcal{E}_j \in N(\mathcal{E}_i)} (1 - x_j)$, then $\Pr[\bigcap_{i=1}^{n} \overline{\mathcal{E}_i}] > 0$.*

2.4 Hitting Vertex Sets with Independent Sets

We prove a lemma which captures our use of the Local Lemma in Theorems 1 and 5. We spell out the constants to emphasize that they are not particularly high, although noting that a more careful proof could improve them a bit.

▶ **Lemma 13.** *Let G be a graph with maximum degree d and $\mathcal{F}$ a family of vertex subsets of G, each member of $\mathcal{F}$ containing at least k vertices. If $6|\mathcal{F}| \leq e^{k/(6d)}$, then there exists an independent set of G that intersects all sets in $\mathcal{F}$.*

Proof. We assume $d \geq 2$ as the lemma is easy to verify for $d \leq 1$. We use the Local Lemma to construct such an independent set. We let each vertex be in the independent set with probability $p = 1/(2d)$. Our bad events are $\mathcal{E}_e$ for each each edge e indicating that both endpoints of e are selected in the independent set, and $\mathcal{E}_A$ for each $A \in \mathcal{F}$ indicating that the independent set does not intersect A. The negative dependency graph is a bipartite graph connecting $\mathcal{E}_e$ to $\mathcal{E}_A$ if at least one of the endpoints of e is in A. In particular, note that the edge events $\mathcal{E}_e$ have non-negative correlation with each other and the vertex set events $\mathcal{E}_A$ also have non-negative correlation with each other. For all edge events $\mathcal{E}_e$ we choose $x_e = 1/(3d^2+1)$ and for all vertex set events $\mathcal{E}_A$ we choose $x_A = 1/(5|\mathcal{F}|+1)$. Now, by Proposition 12, it suffices to verify that

$$\Pr[\mathcal{E}_e] = p^2 \leq x_e(1-x_A)^{|\mathcal{F}|} \tag{1}$$

and

$$\Pr[\mathcal{E}_A] = (1-p)^{|A|} \leq x_A(1-x_e)^{|A|d} \tag{2}$$

hold whenever $6|\mathcal{F}| \leq e^{|A|/6d}$.

For (1), a lower bound for the right hand side is $e^{-1/5}/(3d^2+1)$, which can be verified to be greater than $p^2 = 1/(4d^2)$ when $d \geq 2$. For (2), an upper bound for the left hand side is $e^{-|A|/(2d)}$, and a lower bound for the right hand side is $x_A e^{-|A|d/(3d^2)}$, implying that (2) holds if $e^{-|A|/(2d)}e^{|A|/(3d)} \leq x_A$. This simplifies to $e^{-|A|/(6d)} \leq x_A \Leftrightarrow e^{|A|/(6d)} \geq 5|\mathcal{F}|+1$. ◀

3 Treewidth and MWIS-Circuits

In this section we prove lower bounds for MWIS-circuits parameterized by treewidth, i.e., Theorems 1 and 2.

We use a witness of high treewidth due to Robertson-Seymour treewidth approximation algorithm [36]. A separation of a graph G is an ordered triple of vertex sets (A,S,B) such that A,S,B are disjoint, $A \cup S \cup B = V(G)$, and no vertex of A is adjacent to a vertex of B. The order of a separation (A,S,B) is $|S|$. A separation (A,S,B) is a balanced separation of a vertex set $X \subseteq V(G)$ if $|A \cap X| \leq 2|X|/3$ and $|B \cap X| \leq 2|X|/3$.

▶ **Lemma 14** ([36])**.** *If a graph G has treewidth at least $4k$, then there is a vertex set $X \subseteq V(G)$ such that any balanced separation of X in G has order at least k.*

The next lemma is our main tool to connect circuit complexity with treewidth. This lemma is an adaptation of a classical circuit decomposition lemma (e.g. Theorem 1 in [22], Lemma 3 in [37]). In our applications the vertex set X will be the set given by Lemma 14.

▶ **Lemma 15.** *Let G be a graph and $X \subseteq V(G)$ with $|X| \geq 2$. If there is an MWIS-circuit of G with τ gates, then we can write an MWIS-polynomial of G as $g_1 \cdot h_1 + \ldots + g_\tau \cdot h_\tau$, where for all i it holds that $|Sup(g_i) \cap X| \leq 2|X|/3$ and $|Sup(h_i) \cap X| \leq 2|X|/3$.*

Proof. Let $f + e$ be an MWIS-polynomial of G, where f can be computed by a tropical circuit with τ gates. (The term e is here for the induction argument. In the first step we can assume it to be empty.) We will show that there is an MWIS-polynomial $f' + g \cdot h + e$ of G, where f' can be computed by a tropical circuit with $\tau - 1$ gates, and $|\mathrm{Sup}(g) \cap X| \leq 2|X|/3$ and $|\mathrm{Sup}(h) \cap X| \leq 2|X|/3$. The lemma follows from this by induction.

If $|\mathrm{Sup}(f) \cap X| \leq 2|X|/3$ we are done. Otherwise, we traverse the circuit computing f down starting from the output gate, always choosing the one of the two child gates whose support has larger intersection with X, until we reach a gate v computing a polynomial f_v with $|X|/3 \leq |\mathrm{Sup}(f_v) \cap X| \leq 2|X|/3$. Let $f_{v=-\infty}$ be the polynomial computed by the circuit when the value of the gate v is set to $-\infty$. Now we can write an MWIS-polynomial of G as $f_{v=-\infty} + f_v \cdot g + e$, for example by letting g be an MWIS-polynomial of $G \setminus N[\mathrm{Sup}(f_v)]$. Now, we observe that $f_{v=-\infty}$ can be computed by a circuit with $\tau - 1$ gates. We also observe that the supports of f_v and g cannot intersect, and therefore $|\mathrm{Sup}(g) \cap X| \leq 2|X|/3$. ◀

3.1 Proof of Theorem 1

Now we complete the proof of Theorem 1 by putting Lemmas 13, 14, and 15 together.

▶ **Lemma 16.** *Let G be a graph with maximum degree d and treewidth at least $4k$. Any MWIS-circuit of G has at least $e^{k/(6d)}/6$ gates.*

Proof. Suppose there is an MWIS-circuit of G with τ gates. By Lemma 14 there is a vertex set $X \subseteq V(G)$ that does not admit a balanced separation of order less than k. By Lemma 15 we can write an MWIS-polynomial of G as $g_1 \cdot h_1 + \ldots + g_\tau \cdot h_\tau$, where for all i it holds that $|\mathrm{Sup}(g_i) \cap X| \leq 2|X|/3$ and $|\mathrm{Sup}(h_i) \cap X| \leq 2|X|/3$. Now, by Lemma 10 each multiplication $g_i \cdot h_i$ defines a balanced separation $(\mathrm{Sup}(g_i), V(G) \setminus \mathrm{Sup}(g_i \cdot h_i), \mathrm{Sup}(h_i))$ of X. The order of such a separation is $|V(G) \setminus \mathrm{Sup}(g_i \cdot h_i)|$, and therefore $|V(G) \setminus \mathrm{Sup}(g_i \cdot h_i)| \geq k$. Note that $g_i \cdot h_i$ does not compute an independent set I if I intersects $V(G) \setminus \mathrm{Sup}(g_i \cdot h_i)$. Therefore, by letting $\mathcal{F}$ be the collection of vertex sets $\{V(G) \setminus \mathrm{Sup}(g_1 \cdot h_1), \ldots, V(G) \setminus \mathrm{Sup}(g_\tau \cdot h_\tau)\}$, Lemma 13 shows that if $6\tau \leq e^{k/(6d)}$ we can construct an independent set that is not computed by any of the multiplications, contradicting the assumption that we have an MWIS-circuit. ◀

3.2 Proof of Theorem 2

An *induced minor model* of a graph H in a graph G is a function $f : V(H) \to 2^{V(G)} \setminus \{\emptyset\}$, where $2^{V(G)}$ denotes the power set of $V(G)$, satisfying that

1. the sets $f(u)$ and $f(v)$ are disjoint for $u \neq v$,
2. for each $v \in V(H)$ the induced subgraph $G[f(v)]$ is connected, and
3. $\{u, v\} \in E(H)$ if and only if $N(f(u))$ intersects $f(v)$.

A graph G contains a graph H as an induced minor if and only if there is an induced minor model of H in G. For $v \in V(H)$ we call the induced subgraphs $G[f(v)]$ *clusters*.

First, we ensure that the maximum degree of each cluster is bounded.

▶ **Lemma 17.** *Let G be a graph that contains a graph H with maximum degree d as an induced minor. There is an induced minor model f of H in G such that the maximum degree of each cluster $G[f(v)]$ is at most d.*

Proof. Consider an induced minor model f of H in G and a cluster $G[f(v)]$ for some $v \in V(H)$. Because the degree of H is at most d, we can assign the cluster a set of at most d terminal vertices whose connectivity should be preserved in order to satisfy that f is an induced minor model of H in G. Now, we can remove from the cluster any vertices as long as the terminals stay connected. In particular, if there is a vertex u with degree $> d$ in $G[f(v)]$, then we can consider the shortest paths from u to the terminals, and remove from $G[f(v)]$ the vertices of $N(u) \cap G[f(v)]$ that do not participate in the shortest paths. This makes the degree of u in $G[f(v)]$ at most d. ◀

We also need the following lemma.

▶ **Lemma 18.** *Let I be an independent set selected uniformly at random from the set of all independent sets of a graph G with maximum degree d. For all $v \in V(G)$ it holds that* $\Pr[v \in I] \geq 1/2^{d+1}$.

Proof. For any set $J \subseteq N(v)$ it holds that $\Pr[I \cap N(v) = J] \leq \Pr[I \cap N(v) = \emptyset]$ because we can map any independent set I with $I \cap N(v) = J$ into an independent set $I \setminus N(v)$. Therefore $\Pr[I \cap N(v) = \emptyset] \geq 1/2^d$, so by observing that $\Pr[v \in I \mid I \cap N(v) = \emptyset] \geq 1/2$ we get $\Pr[v \in I] \geq 1/2^{d+1}$. ◀

Next we finish the proof with similar arguments as in the proof of Theorem 1, but with a different kind of construction of the independent set with the Local Lemma. In this case the constants involved appear to be impractical.

▶ **Lemma 19.** *Let G be a graph that contains a graph H with maximum degree d and treewidth $4k$ as an induced minor. Any MWIS-circuit of G has $2^{\Omega(k/(d4^d))}$ gates.*

Proof. Let f be the induced minor model of H in G. First, by Lemma 17 we can assume that the maximum degree of each cluster $G[f(v)]$ is at most d. Now, by Lemma 14 we let X' be a vertex set of H that has no balanced separation of order less than k. Then we let X be a vertex set of G created by mapping each $v \in X'$ to an element of $f(v)$. For each balanced separation (A, S, B) of X in G, the set S must intersect at least k different clusters, because otherwise we could map it into a balanced separation of X' of order $< k$ in H. Therefore, by assuming that G has an MWIS-circuit with τ gates and applying Lemma 15 with the set X we write an MWIS-polynomial of G as $g_1 \cdot h_1 + \ldots + g_\tau \cdot h_\tau$, observing that for each i the set $S_i = V(G) \setminus \mathrm{Sup}(g_i \cdot h_i)$ intersects at least k different clusters. Now it remains to show that if τ is too small we can construct an independent set of G that intersects S_i for all i.

By removing vertices from each S_i we can assume that S_i contains only vertices in clusters, and moreover contains exactly one vertex from each cluster that it intersects. We use the Local Lemma to construct the independent set. First we select each cluster independently with probability $p = 1/(4d2^d)$, and then for each selected cluster $G[f(v)]$ we select an independent set uniformly at random from the set of all independent sets of $G[f(v)]$. By Lemma 18 each vertex of G that is in some cluster will appear in the independent set with probability at least $p/2^{d+1}$. Vertices in different clusters appear in it independently of each other.

Now our bad events are $\mathcal{E}_{\{u,v\}}$ for all $\{u, v\} \in E(H)$ indicating that both clusters $G[f(u)]$ and $G[f(v)]$ have been selected and $\mathcal{E}_i$ for each S_i indicating that the set S_i does not intersect the independent set. Our negative dependency graph has edges connecting each $\mathcal{E}_{\{u,v\}}$ to each $\mathcal{E}_i$ such that S_i intersects $f(u)$ or $f(v)$. It also has all edges between all events $\mathcal{E}_i$ because $\mathcal{E}_i$ and $\mathcal{E}_j$ can be negatively correlated if S_i and S_j intersect a common cluster.

For edges $\{u, v\} \in E(H)$ we let $x_{\{u,v\}} = 1/(15d^2 4^d + 1)$ and for sets S_i we choose $x_i = 1/(20\tau + 1)$. Now it suffices to verify that

$$\Pr[\mathcal{E}_{\{u,v\}}] = p^2 \leq x_{\{u,v\}}(1 - x_i)^\tau \tag{3}$$

and

$$\Pr[\mathcal{E}_i] \leq (1 - p/2^{d+1})^{|S_i|} \leq x_i(1 - x_i)^\tau (1 - x_{\{u,v\}})^{|S_i|d} \tag{4}$$

hold whenever $30\tau \leq e^{7|S_i|/(120d4^d)}$. We also assume that $d \geq 3$ since if $d \leq 2$ then the treewidth of H is at most 2.

For (3), a lower bound for the right hand side is $e^{-1/20}/(15d^2 4^d + 1)$, which is greater than $p^2 = 1/(16d^2 4^d)$ when $d \geq 2$. For (4), a lower bound for the right hand side is $x_i e^{-1/20} e^{-|S_i|d/(15d^2 4^d)}$ and an upper bound for the left hand side is $e^{-|S_i|/(8d4^d)}$, so it holds whenever $e^{-|S_i|/(8d4^d)} \leq x_i e^{-1/20} e^{-|S_i|d/(15d^2 4^d)}$ holds, which we can simplify to $e^{|S_i|(1/(15d4^d)-1/(8d4^d))} \leq x_i e^{-1/20}$, and finally to $e^{-7|S_i|/(120d4^d)} \leq 1/(20\tau+1)e^{-1/20}$, which holds whenever $30\tau \leq e^{7|S_i|/(120d4^d)}$. ◀

4 Treedepth and MWIS-Formulas

For treedepth we are not aware of linear high-treedepth witnesses similar to what Lemma 14 is for treewidth. However, it turns out that we can use very basic properties of treedepth decompositions to establish the connection to formula complexity.

Recall that we denote the treedepth of a graph G with $td(G)$. The following properties follow from the definition of treedepth.

▶ **Proposition 20.** *Let G be a graph with treedepth $td(G)$. It holds that*
1. *$td(G \setminus \{v\}) \geq td(G) - 1$ for any $v \in V(G)$ and*
2. *$td(G)$ is the maximum of $td(G[C])$ over the connected components C of G.*

For our proof we need to introduce two definitions on MWIS-formulas. We start by defining *typical* independent sets of a partial MWIS-formulas.

▶ **Definition 21.** Let F be a partial MWIS-formula of a graph G. An independent set I of G is a typical independent set of F if for each multiplication gate g with $td(G[\mathrm{Sup}(g)]) \geq td(G)/2$ it holds that I intersects a connected component C of $G[\mathrm{Sup}(g)]$ with $td(G[C]) = td(G[\mathrm{Sup}(g)])$.

Note that by the property 2 of Proposition 20 such component indeed exists.

We also define the *separator* $\mathrm{Sep}(g)$ of a gate g. Note that an MWIS-formula forms a tree rooted at the output gate, so we will use standard tree terminology (parent, child, ancestor, descendant).

▶ **Definition 22.** The separator of the output gate o is $\mathrm{Sep}(o) = V(G) \setminus \mathrm{Sup}(o)$. The separator of a gate g whose parent p is a multiplication gate is $\mathrm{Sep}(g) = \mathrm{Sep}(p)$. The separator of a gate g whose parent p is a sum gate is $\mathrm{Sep}(g) = \mathrm{Sep}(p) \cup \mathrm{Sup}(p) \setminus \mathrm{Sup}(g)$.

With the definitions of typical independent sets and separators of gates, we can state the following lemma which will be applied with Lemma 13 to prove our lower bound.

▶ **Lemma 23.** *Let G be a graph with $td(G) \geq 2$ and F a partial MWIS-formula of G. If I is a typical independent set of F and intersects $Sep(g)$ for each gate g with $|Sep(g)| \geq td(G)/2$, then F does not compute I.*

Proof. Let F be such a formula and I such an independent set. We say that a gate g of F is redundant if F computes I if and only if F without g computes I. First, note that all gates g such that I intersects $\mathrm{Sep}(g)$ are redundant because by the definition of separator there is an ancestor gate g' of g with a sum gate parent p such that none of the monomials M contributed from g' to p have $M = \prod_{v_i \in I \cap \mathrm{Sup}(p)} v_i$, implying that g' is redundant and thus all of its descendants are redundant.

Now, we prove by induction starting from the leaves that every gate g of F for which $|\mathrm{Sep}(g)| + td(G[\mathrm{Sup}(g)]) \geq td(G)$ holds is redundant. First, for all such gates g with $td(G[\mathrm{sup}(g)]) \leq td(G)/2$, including all leaves, we have that $|\mathrm{Sep}(g)| \geq td(G)/2$, making g redundant by our definition of I. For a sum gate g and its child c we have by property 1

of Proposition 20 that $td(G[\mathrm{Sup}(c)]) \geq td(G[\mathrm{Sup}(g)]) - |\mathrm{Sup}(g) \setminus \mathrm{Sup}(c)|$, rearranging to $td(G[\mathrm{Sup}(c)] \geq td(G[\mathrm{Sup}(g)]) - |\mathrm{Sep}(c)| + |\mathrm{Sep}(g)|$, and finally to $td(G[\mathrm{Sup}(c)]) + |\mathrm{Sep}(c)| \geq td(G[\mathrm{Sup}(g)]) + |\mathrm{Sep}(g)|$. This implies that if $|\mathrm{Sep}(g)| + td(G[\mathrm{Sup}(g)]) \geq td(G)$ then both children of g are redundant, making g redundant. For a multiplication gate g with $td(G[\mathrm{Sup}(g)]) \geq td(G)/2$ it follows from the typicality assumption that there is a child c of g with $td(G[\mathrm{Sup}(c)]) + |\mathrm{Sep}(c)| = td(G[\mathrm{Sup}(g)]) + |\mathrm{Sep}(g)|$ such that g is redundant if c is redundant. Therefore the induction works, and because $|\mathrm{Sep}(o)| + td(G[\mathrm{Sup}(o)]) \geq td(G)$ holds for the output gate o, the output gate is redundant and therefore the formula does not compute I. ◀

Now the only thing left to complete the proof of Theorem 5 is to show that if a formula has less than $2^{\Omega(td(G)/d)}$ gates then we can construct an independent set that is typical for the formula and intersects $\mathrm{Sep}(g)$ whenever $|\mathrm{Sep}(g)| \geq td(G)/2$. For an independent set to be typical it suffices that it intersects $\mathrm{Sup}(g)$ for all gates g with $|\mathrm{Sup}(g)| \geq td(G)/2$. Therefore it suffices to apply Lemma 13 with $\mathcal{F}$ consisting of $\mathrm{Sep}(g)$ for all $|\mathrm{Sep}(g)| \geq td(G)/2$ and $\mathrm{Sup}(g)$ for all $|\mathrm{Sup}(g)| \geq td(G)/2$. This yields a lower bound of $e^{td(G)/(12d)}/12$ for the number of gates.

5 Optimality of Theorems 1 and 5

We show that for each pair w, d we can construct a graph with treewidth $\Omega(w)$ and maximum degree $O(d)$ that admits an MWIS-formula with $d2^{w/d}$ gates.

If $d > w$ then a d-clique does the job. Otherwise, we take a bounded degree expander with w/d vertices, having treewidth $\Omega(w/d)$, constructible by e.g. [19]. We replace each vertex of the expander with a d-clique (which will be referred to as cluster) such that each vertex of a cluster is connected to each vertex of the clusters of the adjacent vertices. We denote the constructed graph with $G_{w,d}$

▶ **Proposition 24.** *The graph $G_{w,d}$ has treewidth $\Omega(w)$, maximum degree $O(d)$, and admits an MWIS-formula with $d2^{w/d}$ gates.*

Proof. The maximum degree is at most $(d+1)$ times the maximum degree of the original bounded degree expander. The treewidth is $\Omega(w)$ because if a balanced separator contains one vertex from a cluster it must contain all vertices of the cluster.

Note that by a simple recursion any n-vertex graph admits an MWIS-formula with at most 2^n gates, so the original expander admits an MWIS-formula with $2^{w/d}$ gates. We can construct an MWIS-formula of $G_{w,d}$ by taking the MWIS-formula of the original expander and replacing each leaf corresponding to a vertex v with a d-gate construction computing the maximum over the vertices of the cluster of v. ◀

6 Conclusions and Future Work

We investigated the tropical circuit complexity of maximum weight independent set. Our initial motivation for this was the fact that lower bounds for tropical circuits imply lower bounds for many actual algorithmic techniques for maximum weight independent set that are widely used in both theory and practice. We showed that in bounded degree graphs optimal MWIS-circuits are characterized by treewidth and optimal MWIS-formulas are characterized by treedepth. We generalized the result for MWIS-circuits to apply beyond bounded degree

graphs, to a graph class that includes all planar graphs, and more generally all H-minor-free graphs. The constants hidden by the Ω-notation in Theorems 1 and 5 are somewhat practical even though we did not specifically optimize them. For example, Theorem 1 shows that any MWIS-circuit of the 5000×5000-grid has at least 10^{21} gates.

We identify some technical barriers for extending the results. First, we note that Lemma 13 is not effective in graphs with maximum degree higher than k: If $|N(v)| \geq k$, we can add $N(v)$ to $\mathcal{F}$ to force the independent set to avoid v, essentially forcing us to work with $G \setminus \{v\}$. Indeed an example of a graph with high treewidth and no small MWIS-circuits for which Lemma 13 is unsuitable is a clique with each edge subdivided. In some cases, including H-minor-free graphs and the subdivided clique, this barrier can be circumvented with Theorem 2 by using a bounded degree induced minor with high treewidth. We also note that our proofs do not exploit the fact that the separators given by Lemma 15 are balanced beyond just the size bound.

The subdivided clique does not exclude any fixed graph as a minor, so the fact that Theorem 2 works also for proving a lower bound for it seems to indicate that Theorem 2 is more powerful than what is captured by Corollary 3. We are in fact not aware of graph families for which a $2^{\Omega(w)}$ lower bound can be proved but Theorem 2 does not apply.

An interesting general direction for future work could be to prove Corollary 4 for as large graph classes as possible, starting by extending the $2^{\Omega(w)}$ lower bound as far as possible. In particular, H-topological-minor-free graphs generalize both bounded degree and H-minor-free graphs [33], so proving a $2^{\Omega(w)}$ lower bound for them seems like a natural next step. Even more generally, it could be that such a lower bound could even apply to all bounded degeneracy graphs. We hope that this line of work will lead to new insights on the structure of independent sets that could even be useful for positive results on algorithms for maximum weight independent set.

References

1 Noga Alon and Joel H. Spencer. *The Probabilistic Method, Third Edition*. Wiley-Interscience series in discrete mathematics and optimization. Wiley, 2008.

2 Antoine Amarilli, Florent Capelli, Mikaël Monet, and Pierre Senellart. Connecting knowledge compilation classes and width parameters. *Theory of Computing Systems*, 64(5):861–914, 2020. doi:10.1007/s00224-019-09930-2.

3 Stefan Arnborg and Andrzej Proskurowski. Linear time algorithms for NP-hard problems restricted to partial k-trees. *Discrete Applied Mathematics*, 23(1):11–24, 1989. doi:10.1016/0166-218X(89)90031-0.

4 Per Austrin, Petteri Kaski, and Kaie Kubjas. Tensor network complexity of multilinear maps. In Avrim Blum, editor, *Proceedings of the 10th Innovations in Theoretical Computer Science Conference*, volume 124 of *LIPIcs*, pages 7:1–7:21. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019. doi:10.4230/LIPIcs.ITCS.2019.7.

5 Markus Bläser and Christian Hoffmann. On the complexity of the interlace polynomial. In Susanne Albers and Pascal Weil, editors, *Proceedings of the 25th Annual Symposium on Theoretical Aspects of Computer Science*, volume 1 of *LIPIcs*, pages 97–108. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, Germany, 2008. doi:10.4230/LIPIcs.STACS.2008.1337.

6 Maria Luisa Bonet, Toniann Pitassi, and Ran Raz. On interpolation and automatization for Frege systems. *SIAM Journal on Computing*, 29(6):1939–1967, 2000. doi:10.1137/S0097539798353230.

7 Irénée Briquel and Pascal Koiran. A dichotomy theorem for polynomial evaluation. In Rastislav Královic and Damian Niwinski, editors, *Proceedings of the 34th International Symposium on Mathematical Foundations of Computer Science*, volume 5734 of *Lecture Notes in Computer Science*, pages 187–198. Springer, 2009. doi:10.1007/978-3-642-03816-7_17.

8 Binh-Minh Bui-Xuan, Jan Arne Telle, and Martin Vatshelle. Boolean-width of graphs. *Theoretical Computer Science*, 412(39):5187–5204, 2011. doi:10.1016/j.tcs.2011.05.022.

9 Li-Hsuan Chen, Felix Reidl, Peter Rossmanith, and Fernando Sánchez Villaamil. Width, depth, and space: Tradeoffs between branching and dynamic programming. *Algorithms*, 11(7):98, 2018. doi:10.3390/a11070098.

10 Maria Chudnovsky, Marcin Pilipczuk, Michał Pilipczuk, and Stéphan Thomassé. On the maximum weight independent set problem in graphs without induced cycles of length at least five. *SIAM Journal on Discrete Mathematics*, 34(2):1472–1483, 2020. doi:10.1137/19M1249473.

11 V. Chvátal. Determining the stability number of a graph. *SIAM Journal on Computing*, 6(4):643–662, 1977. doi:10.1137/0206046.

12 Wojciech Czerwinski, Wojciech Nadara, and Marcin Pilipczuk. Improved bounds for the excluded-minor approximation of treedepth. In Michael A. Bender, Ola Svensson, and Grzegorz Herman, editors, *Proceedings of the 27th Annual European Symposium on Algorithms*, volume 144 of *LIPIcs*, pages 34:1–34:13. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019. doi:10.4230/LIPIcs.ESA.2019.34.

13 Erik D. Demaine and MohammadTaghi Hajiaghayi. Linearity of grid minors in treewidth with applications through bidimensionality. *Combinatorica*, 28(1):19–36, 2008. doi:10.1007/s00493-008-2140-4.

14 Erik D. Demaine, MohammadTaghi Hajiaghayi, and Dimitrios M. Thilikos. The bidimensional theory of bounded-genus graphs. *SIAM J. Discret. Math.*, 20(2):357–371, 2006. doi:10.1137/040616929.

15 Paul Erdős and Joel Spencer. Lopsided Lovász local lemma and latin transversals. *Discrete Applied Mathematics*, 30(2-3):151–154, 1991. doi:10.1016/0166-218X(91)90040-4.

16 Fedor V. Fomin and Petr A. Golovach. Subexponential parameterized algorithms and kernelization on almost chordal graphs. In Fabrizio Grandoni, Grzegorz Herman, and Peter Sanders, editors, *Proceedings of the 28th Annual European Symposium on Algorithms*, volume 173 of *LIPIcs*, pages 49:1–49:17. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPIcs.ESA.2020.49.

17 Fedor V. Fomin, Ioan Todinca, and Yngve Villanger. Large induced subgraphs via triangulations and CMSO. *SIAM Journal on Computing*, 44(1):54–87, 2015. doi:10.1137/140964801.

18 Eugene C. Freuder and Michael J. Quinn. Taking advantage of stable sets of variables in constraint satisfaction problems. In *Proceedings of the 9th International Joint Conference on Artificial Intelligence*, pages 1076–1078. Morgan Kaufmann, 1985. URL: http://ijcai.org/Proceedings/85-2/Papers/082.pdf.

19 Ofer Gabber and Zvi Galil. Explicit constructions of linear-sized superconcentrators. *Journal of Computer and System Sciences*, 22(3):407–420, 1981. doi:10.1016/0022-0000(81)90040-4.

20 Peter Gartland and Daniel Lokshtanov. Independent set on P_k-free graphs in quasi-polynomial time. In *Proceedings of the 61st IEEE Annual Symposium on Foundations of Computer Science*, pages 613–624. IEEE, 2020. doi:10.1109/FOCS46700.2020.00063.

21 Christian Hoffmann. Exponential time complexity of weighted counting of independent sets. In Venkatesh Raman and Saket Saurabh, editors, *Proceedings of the 5th International Symposium on Parameterized and Exact Computation*, volume 6478 of *Lecture Notes in Computer Science*, pages 180–191. Springer, 2010. doi:10.1007/978-3-642-17493-3_18.

22 Laurent Hyafil. On the parallel evaluation of multivariate polynomials. *SIAM Journal on Computing*, 8(2):120–123, 1979. doi:10.1137/0208010.

23 Mark Jerrum and Marc Snir. Some exact complexity results for straight-line computations over semirings. *Journal of the ACM*, 29(3):874–897, 1982. doi:10.1145/322326.322341.

24 Stasys Jukna. Lower bounds for tropical circuits and dynamic programs. *Theory of Computing Systems*, 57(1):160–194, 2015. doi:10.1007/s00224-014-9574-4.

25 Stasys Jukna. Tropical complexity, sidon sets, and dynamic programming. *SIAM Journal on Discrete Mathematics*, 30(4):2064–2085, 2016. doi:10.1137/16M1064738.

26 Balagopal Komarath, Anurag Pandey, and C. S. Rahul. Graph homomorphism polynomials: Algorithms and complexity. *CoRR*, abs/2011.04778, 2020. arXiv:2011.04778.

27 Casimir Kuratowski. Sur le probleme des courbes gauches en topologie. *Fundamenta mathematicae*, 15(1):271–283, 1930.

28 Yuan Li, Alexander A. Razborov, and Benjamin Rossman. On the AC^0 complexity of subgraph isomorphism. *SIAM Journal on Computing*, 46(3):936–971, 2017. doi:10.1137/14099721X.

29 Daniel Lokshantov, Martin Vatshelle, and Yngve Villanger. Independent set in P_5-free graphs in polynomial time. In Chandra Chekuri, editor, *Proceedings of the 25th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 570–581. SIAM, 2014. doi:10.1137/1.9781611973402.43.

30 Meena Mahajan, Prajakta Nimbhorkar, and Anuj Tawari. Computing the maximum using (min, +) formulas. In Kim G. Larsen, Hans L. Bodlaender, and Jean-François Raskin, editors, *Proceedings of the 42nd International Symposium on Mathematical Foundations of Computer Science*, volume 83 of *LIPIcs*, pages 74:1–74:11. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2017. doi:10.4230/LIPIcs.MFCS.2017.74.

31 Meena Mahajan, Prajakta Nimbhorkar, and Anuj Tawari. Shortest path length with bounded-alternation (min, +) formulas. *International Journal of Advances in Engineering Sciences and Applied Mathematics*, 11(1):68–74, 2019. doi:10.1007/s12572-018-0229-6.

32 J. W. Moon and L. Moser. On cliques in graphs. *Israel Journal of Mathematics*, 3(1):23–28, 1965.

33 Jaroslav Nesetril and Patrice Ossona de Mendez. On nowhere dense graphs. *European Journal of Combinatorics*, 32(4):600–617, 2011. doi:10.1016/j.ejc.2011.01.006.

34 Patric R. J. Östergård. A fast algorithm for the maximum clique problem. *Discrete Applied Mathematics*, 120(1-3):197–207, 2002. doi:10.1016/S0166-218X(01)00290-6.

35 Michał Pilipczuk and Marcin Wrochna. On space efficiency of algorithms working on structural decompositions of graphs. *ACM Transactions on Computation Theory*, 9(4):18:1–18:36, 2018. doi:10.1145/3154856.

36 Neil Robertson and P. D. Seymour. Graph minors. II. Algorithmic aspects of tree-width. *Journal of Algorithms*, 7(3):309–322, 1986. doi:10.1016/0196-6774(86)90023-4.

37 L. G. Valiant. Negation can be exponentially powerful. *Theoretical Computer Science*, 12:303–314, 1980. doi:10.1016/0304-3975(80)90060-2.

Sorting Short Integers

Michal Koucký
Computer Science Institute, Charles University, Prague, Czech Republic

Karel Král
Computer Science Institute, Charles University, Prague, Czech Republic

Abstract

We build boolean circuits of size $\mathcal{O}(nm^2)$ and depth $\mathcal{O}(\log(n) + m\log(m))$ for sorting n integers each of m-bits. We build also circuits that sort n integers each of m-bits according to their first k bits that are of size $\mathcal{O}(nmk(1 + \log^*(n) - \log^*(m)))$ and depth $\mathcal{O}(\log^3(n))$. This improves on the results of Asharov et al. [3] and resolves some of their open questions.

2012 ACM Subject Classification Theory of computation → Sorting and searching

Keywords and phrases sorting, small integers, boolean circuits

Digital Object Identifier 10.4230/LIPIcs.ICALP.2021.88

Category Track A: Algorithms, Complexity and Games

Funding Michal Koucký and Karel Král were supported by the Grant Agency of the Czech Republic under the grant agreement no. 19-27871X. Karel Král was also partially supported by the Charles University project SVV–2020–260578.

Acknowledgements The authors are grateful for insightful discussions with Mike Saks on sorting and to Veronika Slívová for her insights and comments regarding the first versions of this paper. The authors thank Igor Sergeev for pointing us to the paper of Kospanov [6].

1 Introduction

Sorting undoubtedly plays a central role in computer science. Great many problems can be solved using sorting as a subcomponent. There are many practical variants of sorting based either on what we sort (integers, rational numbers, strings, etc.) or how we sort (in parallel, in distributed fashion, in external memory, etc.). Despite lots of research there are still many basic questions about sorting unanswered.

The classical comparison based sorting takes time $\mathcal{O}(n\log(n))$ when sorting n integers. Well known lower bound postulates that this is optimal for comparison based sorting. However, this is a great over-simplification and the picture is much more nuanced: sorting integers from a domain of size M can be done using binary search trees in time $\mathcal{O}(n\log|M|)$, thus sorting for example m-bit integers only needs $\mathcal{O}(nm)$ comparisons. Such an algorithm can be implemented on a pointer machine, for example. In the RAM model, with the word size m we can sort even faster: When $m = O(\log(n))$ one can sort in time $\mathcal{O}(n)$ using radix sort, and when $m = \Omega(\log^3(n))$ one can also sort in linear time using the algorithm of Andersson [2]. When $m = O(\log^3(m))$ one can sort in expected time $\mathcal{O}\left(n\sqrt{\log\frac{m}{\log(n)}}\right)$ and linear space using the algorithm of Han and Thorup [4]. It is an easy exercise to design Turing machines that sort m-bit integers in time $\mathcal{O}(nm^2)$.

In many cryptographic applications there is an interest in oblivious algorithms, algorithms in which the sequence of the operations is independent of the processed data. Sorting plays an important role in construction of oblivious RAM. An oblivious comparison based parallel model of computation intended for sorting are *sorting networks*. Numbers in a sorting network are thought of as signals which can only be compared. The seminal paper by Ajtai,

Komlós, and Szemerédi [1] gives an asymptotically optimal sorting network of logarithmic depth and thus having $\mathcal{O}(n \log(n))$ comparators matching the comparison based lower bound. The AKS network has immense applications in theoretical computer science, and we use it in this paper, too.

Another oblivious model of computation heavily used throughout theoretical computer science are boolean circuits. One can turn the AKS sorting network into a circuit of size $\mathcal{O}(nm \log(n))$ and depth $\mathcal{O}(\log(m) \log(n))$ (see Section 4). However, when building boolean circuits for sorting it is not clear whether one can take any advantage of some of the faster algorithms for RAM or Turing machines as simulating random access memory or Turing machine tapes by circuits requires substantial overhead. Asharov et al. [3] asked the question whether one can sort m-bit integers in time $o(nm \log(n))$ when $m = o(\log(n))$. They provide an answer to this question by constructing circuits for sorting m-bit integers of size $\mathcal{O}(nm^2(1 + \log^*(n) - \log^*(m))^{2+\varepsilon})$ and polynomial depth, for any $\varepsilon > 0$. We improve their results: We build boolean circuits for sorting m-bit integers of size $\mathcal{O}(nm^2)$ and depth $\mathcal{O}(\log(n) + m \log(m))$. Pending some unexpected breakthrough this size seems optimal. The depth is provably optimal whenever $m = O(\log(n)/\log\log(n))$.

Asharov et al. [3] solve even a more general problem as their circuits partially sort n numbers each of m bits by their first k bits using a circuit of size $\mathcal{O}(nmk(1 + \log^*(n) - \log^*(m))^{2+\varepsilon})$. We improve on this result as well by presenting circuits that sort m-bit integers according to their first k bits of size $\mathcal{O}(nmk(1 + \log^*(n) - \log^*(m)))$ and depth $\mathcal{O}(\log^3(n))$. Our small circuits of poly-logarithmic depth answer some of the open questions of Asharov et al. [3]. In a work subsequent to ours, Lin and Shi [7] get circuits of depth $\mathcal{O}(\log(n) + \log(k))$ and size $\mathcal{O}(nkm \cdot \mathrm{poly}(\log^*(n) - \log^*(m)))$ whenever $n > 2^{4k+7}$. They use substantially different approach. We state our results in the next section.

1.1 Our Results

We provide a family of boolean circuits that sort m-bit strings. Our circuits are smaller than the circuits directly derived from the AKS sorting network, and they improve on the result of Asharov et al. [3]. Our circuits achieve optimal logarithmic depth whenever $m \log(m) \leq \log(n)$. Pending some unexpected breakthrough, their size seems also optimal.

▶ **Theorem 1.** *For any integers $n, m \geq 1$ there is a size $\mathcal{O}(nm^2)$ and depth $\mathcal{O}(\log(n) + m \log(m))$ circuit that sorts n integers of m bits each.*

For $m \geq \Omega(\log(n))$, the existence of such a circuit directly follows from AKS sorting networks. Our contribution is the construction of such circuits for $m \leq o(\log(n))$. Our construction also uses a sorting network as a building block. We use the AKS sorting network as one of our primitives but in principle, we could use any sorting network or sorting circuit. In particular, we could use any circuit sorting n numbers of $\log(n)$ bits each in our construction. Any improvement of asymptotic complexity of sorting of $\log(n)$-bit numbers would give us improved complexity of sorting short numbers.

The main idea behind our construction is to *compress* the input by computing the number of occurrences of each m-bit integer. This gives a vector of 2^m integers, each of size $\mathcal{O}(\log(n))$. Decompressing this vector back gives the sorted input. Combining the counting and decompressing circuit gives us a circuit that sorts. The main technical lemma is our counting circuit which is of independent interest.

▶ **Lemma 2.** *For any integers $n, m \geq 1$ where $m \leq \log(n)/10$ there is a circuit*

$$FAST\_COUNT_{n,m} \colon \{0,1\}^{n \cdot m} \to \{0,1\}^{\lceil 1+\log(n) \rceil 2^m}$$

which given a sequence of n strings of m bits each outputs the number of occurrences of each possible m-bit string among the inputs, that is for input $x_1, x_2, \ldots, x_n \in \{0,1\}^m$ it outputs $n_{0^m}, n_{0^{m-1}1}, \ldots, n_{1^m}$ where for each string $y \in \{0,1\}^m$, $n_y \in \{0,1\}^{\lceil 1+\log(n) \rceil}$ represents $|\{j \in [n] \mid x_j = y\}|$ in binary. The size of the circuit $FAST\_COUNT_{n,m}$ is $\mathcal{O}(nm^2)$ and depth $\mathcal{O}(\log(n) + m\log(m))$.

We also provide a family of boolean circuits which sort the input integers by their first k bits only. One can view this as sorting (key, value) pairs, where keys have k bits and values have $m - k$ bits. For the special case of $k = 1$ (that is partially sorting the numbers by a single bit) the problem is equivalent to routing in *super-concentrators* (see Section 1.2), and we use super-concentrators of Pippenger [8] as our building block. We get size improvement over the result of Asharov et al. [3] while achieving also poly-logarithmic depth.

▶ **Theorem 3.** *For any integers $n, m, k \geq 1$ where $k \leq m$ and $k \leq \log(n)/11$ there is a circuit*

$$SORT_{n,m,k} \colon \{0,1\}^{nm} \to \{0,1\}^{nm}$$

which partially sorts n numbers each of m bits by their first k bits. The circuit $SORT_{n,m,k}$ has size $\mathcal{O}(knm(1 + \log^*(n) - \log^*(m)))$ and depth $\mathcal{O}(\log^3(n))$.

1.2 Our Techniques

One can take AKS sorting networks and turn them into circuits of size $\mathcal{O}(nm\log(n))$ and depth $\mathcal{O}(\log(m)\log(n))$. For $m = o(\log(n))$ this is sub-optimal as shown by Asharov et al. [3]. Asharov et al. show how to reduce the problem of sorting m-bit integers according to the first k bits into the problem of sorting m-bit integers according to just single bit. Sorting according to single bit is essentially equivalent to routing in super-concentrators.

Super-concentrators have been studied originally by Valiant with the aim of proving circuit lower bounds. A super-concentrator is a graph with two disjoint subsets of vertices $A, B \subseteq V(G)$, called inputs and outputs, with the property that for any set $S \subseteq A$ and $T \subseteq B$ of the same size there is a set of vertex disjoint paths from each vertex of S to some vertex of T. Pippenger [8] constructs super-concentrators with a linear number of edges and an algorithm that on input describing S and T outputs the list of edges forming the disjoint paths between S and T. This can be turned into a circuit of size $\mathcal{O}(n\log(n))$ and depth $\mathcal{O}(\log^2(n))$.

The result of Pippenger [8] can be used to build a circuit sorting by one bit, but the circuit will be larger than we want (see Corollary 18.) Thus, Asharov et al. [3] used the technique of Pippenger rather than his result to design a circuit sorting by one bit, and iterate it to sort by k bits. Our technique differs substantially from that of Asharov et al. yet, we use the circuits from AKS networks and from Pippenger's super-concentrators as black box.

To sort m-bit integers for $2^m \ll n$ our approach is to count the number of occurrences of each number in the input. This compresses the input from nm bits into $2^m \log(n)$ bits. We can then decompress the vector back to get the desired output. So the main challenge is to construct counting (compressing) circuits of size $\mathcal{O}(nm^2)$. Interestingly, we use the sorting circuits derived from AKS networks to do that. But to avoid the size blow-up we don't use them on all of the integers at once but on blocks of integers of size 2^{8m}. Then the $\mathcal{O}(\log(n))$ overhead of the circuits turns into the acceptable $\mathcal{O}(m)$ overhead. Each sorted

block is then subdivided into parts of size 2^{2m}. Clearly, most parts in each block will be monochromatic, they will contain copies of the same integer. There will be at most 2^m non-monochromatic parts. We *move* the parts within a block to one side using another application of the AKS sorting circuit. Then we can afford to build a fairly expensive counting circuit for the small fraction of non-monochromatic parts, while cheaply counting the monochromatic parts. Summing the results by linear size circuit gives us the desired compression. Our decompression essentially mirrors the compression.

We also design a circuit to sort according to a single bit improving the parameters of Asharov et al. [3]. We take the circuit of Pippenger as basis and apply it iteratively to larger and larger blocks of inputs. Again we start from blocks of size $2^{O(m)}$, and increase the size of the blocks exponentially at each iteration. We use Pippenger's circuit to sort each block by the bit. When we split the block into parts, only one will be monochromatic. Merging multiple blocks into one gives a mega-block with only a small fraction of non-monochromatic parts. These non-monochromatic parts can be separated from monochromatic ones, re-sorted, and re-partitioned to give only one non-monochromatic part in the mega-block. Each part takes on the role of an "m"-bit integer in the next iteration. Iterating this process leads to the desired result.

To sort according to the first k bits we use the one-bit sorting similarly to Asharov et al. [3]. Thanks to our efficient sorting circuits for m-bit integers to sort the k-bit keys, we can avoid the use of median finding circuits.

Organization

In the next section we review our notation. We provide basic construction tools including naïve constructions of counting and decompression circuits in Section 3. In Section 4 we recall basic facts on AKS sorting networks and related sorting circuits. In Section 5 we prove our main result by constructing efficient counting and decompression circuits. Finally, we provide a construction of partial sorting circuits for Theorem 3 in Section 6.

2 Notation

In this paper $\mathbb{N}$ denotes the set of natural numbers, and for $1 \leq a \leq b \in \mathbb{N}$, $[a,b] = \{a, a+1, \ldots, b\}$ and $[a] = \{1, \ldots, a\}$. All logarithms are base two unless stated otherwise. For $m \in \mathbb{N}$, $\{0,1\}^m$ is the set of all binary strings of length m. A string $x \in \{0,1\}^m$, $x = x_1 x_2 \cdots x_m$, represents the number $\sum_{j\in[m]} x_j 2^{m-j}$ in binary, and we often identify the string with that number. (As the same integer has multiple binary representations differing in the number of leading zeroes, the number of leading zeroes should be clear from the context.) The most significant bit of $x = x_1 x_2 \cdots x_m$ is x_1 and the least significant bit of x is x_m. Symbol $\circ$ denotes the concatenation of two strings. For strings $x, y \in \{0,1\}^m$, $x \oplus y$ denotes the bit-wise XOR of x and y, $x \wedge y$ denotes the bit-wise AND, and $x \vee y$ the bit-wise OR.

We assume the reader is familiar with boolean circuits (see for instance the book of Jukna [5]). We assume boolean circuits consist of gates computing binary AND and OR, and unary gates computing negation. For us, boolean circuits might have multiple outputs so a circuit with n inputs and m outputs computes a function $f\colon \{0,1\}^n \to \{0,1\}^m$. We usually index a circuit family by multiple integral parameters. Inputs and outputs of boolean circuits are often interpreted as sequences of substrings, e.g., a circuit $C_{n,m}\colon \{0,1\}^{nm} \to \{0,1\}^{nm}$ is viewed as taking n binary strings of length m as its input, and similarly for its output. We say a circuit family $(C_n)_{n\in\mathbb{N}}$ is uniform, if there is an algorithm that on input 1^n outputs the description of the circuit C_n in time polynomial in n.

3 Preliminaries

Here we review some of the circuits for basic primitives that we will use in our later constructions. Most of them are well known facts but for the others we provide proofs for the sake of completeness.

▶ **Lemma 4** (Addition). *There is a uniform family of boolean circuits* $ADD_m\colon \{0,1\}^{2m} \to \{0,1\}^{m+1}$ *that given* $x, y \in \{0,1\}^m$ *representing two numbers in binary outputs their sum* $x + y \in \{0,1\}^{m+1}$. *The circuit* ADD_m *has size* $\Theta(m)$ *and depth* $\Theta(\log(m))$.

▶ **Lemma 5** (Subtraction). *There is a uniform family of boolean circuits* $SUB_m\colon \{0,1\}^{2m} \to \{0,1\}^m$ *that given* $x, y \in \{0,1\}^m$ *representing two numbers in binary outputs the absolute value of their difference* $|x - y| \in \{0,1\}^m$. *The circuit* SUB_m *has size* $\Theta(m)$ *and depth* $\Theta(\log(m))$.

▶ **Lemma 6** (Summation). *There is a uniform family of boolean circuits*

$$SUM_{n,m}\colon \{0,1\}^{n\cdot m} \to \{0,1\}^{\lceil \log(n) \rceil + m}$$

that given $x_1, x_2, \ldots, x_n \in \{0,1\}^m$ *interpreted as* n *numbers, each of* m *bits, outputs their sum* $\sum_{j=1}^{n} x_j$. *The circuit* $SUM_{n,m}$ *has size* $\Theta(nm)$ *and depth* $\Theta(\log(n) + \log(m))$.

Proof. We sketch the construction following the technique of Wallace [9]. Given three numbers $x, y, z \in \{0,1\}^k$ in constant depth and using $\Theta(k)$ gates we can compute $p, q \in \{0,1\}^{k+1}$ such that $x + y + z = p + q$. Here, p is the coordinate-wise addition without carry, i.e., $0 \circ (x \oplus y \oplus z)$, and q is the carry, i.e., $((x \wedge y) \vee (x \wedge z) \vee (y \wedge z)) \circ 0$. Thus as long as there are at least three numbers to sum we can use this to transform x, y, z which takes $3k$ bits into p, q which take $2k + 2$ bits and continue summing those. Doing this in parallel for disjoint triples of summants after $\mathcal{O}(\log_{3/2}(n)) = \mathcal{O}(\log(n))$ rounds we are left with just two numbers and we sum those using Lemma 4. ◀

▶ **Lemma 7** (Comparator). *There is a uniform family of boolean circuits*

$$SWITCH_m\colon \{0,1\}^{2m} \to \{0,1\}^{2m}$$

that given two numbers $x, y \in \{0,1\}^m$ *outputs these two numbers sorted as integers, i.e.,* $\min(x, y) \circ \max(x, y)$. *The size of the circuit* $SWITCH_m$ *is* $\Theta(m)$ *and depth is* $\Theta(\log(m))$.

Technique similar to the proof of the next lemma will be used also later in the proofs of Lemma 2 and Lemma 16 in order to achieve smaller circuit size. The main idea is to split inputs into smaller blocks and process the blocks independently by smaller circuits.

▶ **Lemma 8** (Binary to unary). *There is a uniform family of boolean circuits*

$$ONES_b\colon \{0,1\}^{b+1} \to \{0,1\}^{2^b}$$

such that for any number $x \in \{0,1\}^{b+1}$ *represented in binary the output consists of* x *ones followed by* $2^b - x$ *zeroes, provided* $x \leq 2^b$. *The circuit* $ONES_b$ *has size* $\Theta(2^b)$ *and depth* $\Theta(\log(b))$.

Proof. We first show how to construct a uniform family of boolean circuits (ONES'$_b$) which computes the same function, has the same size but depth $\mathcal{O}(b)$. Then we use ONES'$_{\log(b)}$ to construct the desired circuit ONES$_b$.

The main idea of the construction of ONES'$_b$ is to recursively split the number x into two numbers x_L, x_R which describe how many bits set to one there should be in the first and the second half of the output.

Each of the two numbers x_L, x_R will be represented by b bits with the convention that if the most significant bit is equal to one then the number is a power of two (corresponding to all output bits in this part of the output set to one). We recursively split the numbers x_L, x_R in the same fashion until the numbers are represented by a single bit each at which point they will represent the output bits. We set

$$x_L = \min(2^{b-1}, x)$$
$$x_R = \min(2^{b-1}, \max(0, x - 2^{b-1}))$$

note that if the number x is represented by $b+1$ bits ($x \in \{0,1\}^{b+1}$) then the numbers x_L, x_R can be represented by b bits ($x_L, x_R \in \{0,1\}^b$) as both of them represent at most half of x. Given $x \in \{0,1\}^{b+1}$ we can compute the maximum and minimum defining x_L, x_R by inspecting the two most significant bits of x:

- If the most significant bit of x is set to one (thus $x \geq 2^b$) we set $x_L = x_R = x/2$ each a power of two with the most significant bit set to one (and represented by a binary string 10^{b-1}).
- If the most significant bit of x is set to zero and the second most significant bit is set to one, then x_L will be set to the binary number 10^{b-1} and x_R will be $x - x_L$ (a copy of x without the second most significant bit of x).
- If the two most significant bits of x are equal to zero then $x_L = x$ (represented by one less bit than x) and $x_R = 0$.

See Figure 1 for an example of splitting of x into x_L, x_R.

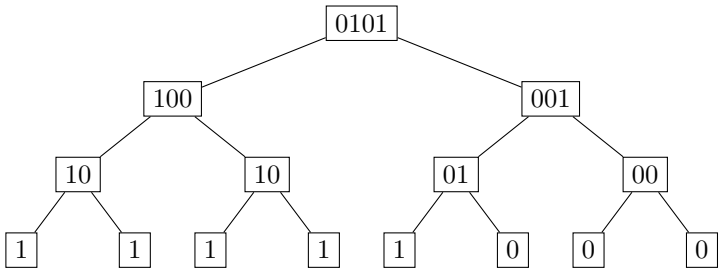


Figure 1 An example of splitting numbers where $b = 3$. The input number $x = 5$ is represented as 0101 and is split into $x_L = 100, x_R = 001$ which are themselves split recursively. The bottom nodes form the output.

Thus we can compute the transformation $x \mapsto (x_L, x_R)$ where $x \in \{0,1\}^{b+1}$ and $x_L, x_R \in \{0,1\}^b$ using a circuit of size $\Theta(b)$ and depth $\Theta(1)$. Then each of the numbers x_L, x_R is again split into two, etc. until we get single bit numbers which represent the final output. The depth of the circuit ONES'$_b$ is $\Theta(b)$ as each splitting can be done in constant depth. If the circuit splitting $b+1$ bits into two b-bit numbers has size $s(b) \leq cb + d$, for some universal constants c and d, then the circuit ONES'$_b$ has size:

$$\begin{aligned} s(b+1) + 2s(b) + 4s(b-2) + \ldots + 2^b s(1) &= \sum_{j=0}^{b} 2^j s(b-j) \\ &\leq \sum_{j=0}^{b} 2^j c(b-j) + 2^j d \\ &\leq c\left(2^{b+2} - b - 1\right) + 2^{b+1} d \\ &= O(2^b) \end{aligned}$$

To build the circuit ONES_b of depth $\mathcal{O}(\log(b))$ we proceed as follows. For any $y > 1$ we denote the largest power of two that is at most y by $\ell(y) = \max\left\{2^j \mid j \in \mathbb{N}, 2^j \leq y\right\}$. We divide the output bits into blocks of $\ell(b)$ bits and for each block $j \in \left[\frac{2^b}{\ell(b)}\right]$ of output bits with positions $[(j-1)\ell(b)+1, j\ell(b)]$ (counting positions from one) we compute if it should be constant (that is either constant zero when $x \leq (j-1)\ell(b)$ or constantly equal to one when $x > j\ell(b)$). This check for constant values can be done in each block by a circuit of size $\Theta(b)$ and depth $\Theta(\log(b))$. We compute $\mathrm{ONES'}_{\log(\ell(b))}$ with the input being the $\log(\ell(b))$ least significant bits of x. This circuit is of size $\mathcal{O}(b)$ and depth $\mathcal{O}(\log(b))$. In each block if the block should not be monochromatic then we use the output of that circuit as the output of the block, otherwise we use the appropriate constant one or zero copied $\ell(b)$-times as the output of the block. ◀

We will need a primitive that counts the number of occurrences of each string in the input. A counting similar to Lemma 9 appears in Appendix A of the paper of Asharov et al. [3]. The construction of the counting circuit is rather straightforward, we just compare each input string x_j with a given string y getting an indicator bit set to one for equality and to zero for inequality and then sum the indicator bits.

▶ **Lemma 9** (Count). *There is a uniform family of boolean circuits $COUNT_{n,m}\colon \{0,1\}^{n\cdot m} \to \{0,1\}^{2^m\lceil 1+\log(n)\rceil}$ that given $x_1, x_2, \ldots, x_n \in \{0,1\}^m$ counts the number of occurrences of each $y \in \{0,1\}^m$ among the inputs, i.e., the circuit outputs $n_{0^m}, n_{0^{m-1}1}, \ldots, n_{1^m}$ where for each $y \in \{0,1\}^m$, n_y represents in binary $|\{j \in [n] \mid y = x_j\}|$ using $\lceil 1 + \log(n)\rceil$ bits. The size of the circuit $COUNT_{n,m}$ is $\mathcal{O}(nm2^m)$ and depth $\mathcal{O}(\log(n) + \log(m))$.*

Proof. For each $y \in \{0,1\}^m$ we build a sub-circuit computing the number of times y occurs among the inputs $x_1, \ldots, x_n$. This is done by comparing y to each x_i in parallel, $i \in [n]$, to get an indicator bit whether they are equal. We obtain n_y by summing up the indicator bits using the circuit $\mathrm{SUM}_{n,1}$ of size $\Theta(n)$ and depth $\Theta(\log(n))$ from Lemma 6. Comparing y to x_i can be done by a circuit of size $\mathcal{O}(m)$ and depth $\mathcal{O}(\log(m))$. So we get n_y using a circuit of size $\Theta(nm)$ and depth $\Theta(\log(n) + \log(m))$. Doing this for each $y \in \{0,1\}^m$ in parallel we get a circuit of size $\Theta(nm2^m)$ and depth $\Theta(\log(n) + \log(m))$. ◀

We will need also an inverse operation for the counting. To construct a circuit that decompresses the counts we would like to first compute the interval where a given string x should appear and then get indicator bits for this interval. We can compute the interval using prefix sums of the counts. To get the indicator bits for the interval we utilize the circuit from Lemma 8 which outputs a given number of bits set to one followed by bits set to zero.

▶ **Lemma 10** (Decompress). *There is a uniform family of boolean circuits*

$$DECOMPRESS_{n,m}\colon \{0,1\}^{\lceil 1+\log(n)\rceil 2^m} \to \{0,1\}^{n\cdot m}$$

that decompresses its input *that is on input numbers* $n_{0^m}, n_{0^{m-1}1}, \ldots, n_{1^m}$*, each represented in binary by* $\lceil 1+\log(n)\rceil$ *bits, where* $\sum_{x\in\{0,1\}^m} n_x = s \le n$*, outputs the string*

$$(\underbrace{0\cdots0}_{m})^{n_{0\cdots0}} \circ (\underbrace{0\cdots0}_{m-1}1)^{n_{0\cdots01}} \circ (\underbrace{0\cdots0}_{m-2}10)^{n_{0\cdots010}} \circ (\underbrace{0\cdots0}_{m-2}11)^{n_{0\cdots011}} \circ \cdots \circ (\underbrace{1\cdots1}_{m})^{n_{1\cdots1}} \circ (0^m)^{n-s}.$$

When $s > n$ *the output might be arbitrary. The size of the circuit* $DECOMPRESS_{n,m}$ *is* $\mathcal{O}(nm2^m + 2^{2m}\log(n))$ *and depth* $\mathcal{O}(m + \log\log(n))$*.*

Proof. Given $n_{0^m}, n_{0^{m-1}1}, \ldots, n_{1^m}$ we can compute the total sum $s = \sum_{x\in\{0,1\}^m} n_x$ and for each $y \in \{0,1\}^m$, the number p_y of binary strings before the first occurrence of y, i.e., $p_y = \sum_{x\in\{0,1\}^m\colon x<y} n_x$. Each of the numbers p_y can be computed using the circuit $\mathrm{SUM}_{y,\lceil 1+\log(n)\rceil}$ from Lemma 6 of size $\mathcal{O}(2^m\log(n))$ and depth $\mathcal{O}(m+\log\log(n))$. Similarly for s. Thus we can get all numbers p_y in parallel by a circuit of size $\mathcal{O}(2^{2m}\log(n))$. A given string $y \in \{0,1\}^m$, $y \neq 1^m$, should appear at each position $j \in [p_y+1, p_{y+1}]$. Let $I_y \in \{0,1\}^n$ be the indicator vector of positions where y should appear in the output. We can use $\mathrm{ONES}_{\lceil 1+\log(n)\rceil}(p_y) \oplus \mathrm{ONES}_{\lceil 1+\log(n)\rceil}(p_{y+1})$ to calculate I_y for each $y \neq 1^m$. For $y = 1^m$, $I_y = \mathrm{ONES}_{\lceil 1+\log(n)\rceil}(p_y) \oplus \mathrm{ONES}_{\lceil 1+\log(n)\rceil}(s)$. The size of $\mathrm{ONES}_{\lceil 1+\log(n)\rceil}$ is $\Theta(n)$. As there are 2^m different y's, we need a circuit of size $\Theta(n2^m)$ and depth $\Theta(\log\log(n))$ to calculate all I_y's.

If $x_1, x_2, \ldots, x_n$ are the output integers, for each output position $j \in [n]$, we calculate the k-bit of x_j as

$$\bigvee_{y\in\{0,1\}^m} ((I_y)_j \wedge y_k)$$

To compute all these ORs we need a circuit of total size $\Theta(nm2^m)$ and depth $\Theta(m)$. ◀

4 Sorting Circuits from AKS Sorting Networks

In this section we recall the construction of circuits for sorting from the Ajtai-Komlós-Szemerédi sorting networks. They will serve as the basic primitive for our later constructions.

Sorting networks

Sorting networks model parallel algorithms that sort values using only comparisons. A sorting network consists of n wires and s comparators. The wires extend from left to right in parallel. Each wire carries an integer from left to right. Any two wires can be connected by a comparator at any point along their length. The comparator swaps the values carried along the two wires if the higher wire carries a higher value at that point otherwise it has no effect. The sorting network should be such when we input arbitrary integers to the wires on the left, the integers always exit in sorted order from top to bottom. The *depth* of a sorting network is the maximum number of comparators a value can encounter on its way. A figure of a small sorting network is given in Figure 2. For a formal definition see, e.g., [1]. Observe that if the depth of a sorting network is d and the number of inputs is n then there are at most $s \le nd$ comparators. Ajtai, Komlós and Szemerédi [1] established the existence of sorting networks of logarithmic depth.

▶ **Theorem 11** (AKS [1]). *For any integer* $n \ge 1$*, there is a sorting network for* n *integers of depth* $\mathcal{O}(\log(n))$*.*

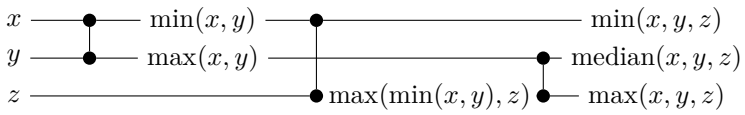


Figure 2 An example of a sorting network with three inputs (the horizontal lines), three comparators (the vertical lines), and depth three. The inputs on the left are numbers x, y, z and after each comparator we noted what is on the horizontal line. Note that the bottom most output is $\max(\max(x,y), \max(\min(x,y),z)) = \max(x,y,z)$ and the middle one is $\min(\max(x,y), \max(\min(x,y),z))$ which is the median.

Sorting circuits

Here we give a precise definition of sorting by a circuit. First we consider a circuit sorting n integers, each of them m bits long.

▶ **Definition 12** (Sort). *Let $n, m \in \mathbb{N}$, and $(C_{n,m})$ be a family of boolean circuits. We say that the* circuit $C_{n,m} \colon \{0,1\}^{nm} \to \{0,1\}^{nm}$ sorts its input *interpreted as n integers $x_1, x_2, \ldots, x_n$ each represented by m bits if it outputs $y_1, y_2, \ldots, y_n \in \{0,1\}^m$ such that:*

1. *The outputs are sorted: For any $i < j \in [n]$, $y_i \leq y_j$.*
2. *The inputs and outputs form the same multiset: For each $j \in [n]$, $|\{i \in [n] \mid y_i = x_j\}| = |\{i \in [n] \mid x_i = x_j\}|$.*

An immediate consequence of the existence of AKS sorting networks is the existence of shallow sorting circuits, since by Lemma 7, each comparator can be replaced by a small circuit:

▶ **Corollary 13.** *There is a family of boolean circuits $AKS_{n,m} \colon \{0,1\}^{n \cdot m} \to \{0,1\}^{n \cdot m}$ that on an input $x_1, x_2, \ldots, x_n \in \{0,1\}^m$ sorts these numbers. The size of the circuit $AKS_{n,m}$ is $\mathcal{O}(nm \log(n))$ and depth $\mathcal{O}(\log(n) \log(m))$.*

We also need circuits that sort the n input integers, each of m bits, by the k most significant bits where $k < m$. Such sorting can be thought of as sorting (key, value) pairs, where keys are k-bit long and values $(m-k)$-bit long. Formally it can be defined as follows:

▶ **Definition 14** (Partial Sort). *Let $n, m, k \in \mathbb{N}$, be such that $k < m$, and let $(C_{n,m,k})$ be a family of boolean circuits. We say that the* circuit $C_{n,m,k} \colon \{0,1\}^{nm} \to \{0,1\}^{nm}$ partially sorts by the first k bits its input *interpreted as n integers $x_1, x_2, \ldots, x_n$ each represented by m bits if it outputs $y_1, y_2, \ldots, y_n \in \{0,1\}^m$ such that:*

1. *The outputs are partially sorted: For any $i < j$, $(y_i)_1 (y_i)_2 \cdots (y_i)_k \leq (y_j)_1 (y_j)_2 \cdots (y_j)_k$.*
2. *The inputs and outputs form the same multiset: For each $j \in [n]$, $|\{i \in [n] \mid y_i = x_j\}| = |\{i \in [n] \mid x_i = x_j\}|$.*

Using a circuit of size $\mathcal{O}(m)$ and depth $\mathcal{O}(\log(k))$ implementing a comparator which swaps two m-bit integers based only on the first k bits we get the following variant of the previous corollary.

▶ **Corollary 15.** *There is a family of boolean circuits $PARTIAL\_AKS_{n,m,k} \colon \{0,1\}^{n \cdot m} \to \{0,1\}^{n \cdot m}$, for $k \leq m$ and $k \leq \log(n)$, that on input $x_1, x_2, \ldots, x_n \in \{0,1\}^m$ partially sorts these numbers according to their k most significant bits. That is if y_i, y_j are two output numbers where $i < j$ then we have $\lfloor y_i / 2^{m-k} \rfloor \leq \lfloor y_j / 2^{m-k} \rfloor$. The size of the circuit $PARTIAL\_AKS_{n,m,k}$ is $\mathcal{O}(nm \log(n))$ and depth $\mathcal{O}(\log(n) \log(k))$.*

5 Sorting n Binary Strings of Length m

Here we present a sorting circuit for short numbers. The construction consists of two circuits. The first circuit counts the number of occurrences of various strings (as stated in Lemma 2) and the second circuit decompresses these counts. Both of these constructions use heavily the following technique: we divide the problem into blocks which can be efficiently sorted using the AKS-based circuit. These blocks will be of size between $2^{O(m)}$ and $n/2^{O(m)}$ where m is the binary length of the input integers.

Thus when we sort the numbers inside each block and subdivide the block into parts, then by the pigeon-hole principle, most of the parts will be monochromatic (containing copies of a single string only). We can then separately count the strings in monochromatic parts (count the first string and then multiply that by the length of the part) and in the non-monochromatic parts (there are not that many strings in total in non-monochromatic parts). However a priori we do not know which parts will be monochromatic and which will be not. To save on circuitry we use sorting (on whole parts) to move the non-monochromatic parts aside. We build the (expensive) counting circuits only for non-monochromatic parts.

Proof of Lemma 2. For the sake of simplicity let us assume that n is a power of two so, it is divisible by 2^{8m}. (By our assumption $n \geq 2^{10m}$, thus if n is not a power of two take the circuit for the closest power of two larger than n and feed ones for the extra input bits.) We partition the input into $n/2^{8m}$ blocks each consisting of 2^{8m} numbers. We sort each block by the circuit $\mathrm{AKS}_{2^{8m},m}$ of size $\mathcal{O}(2^{8m}m\log(2^{8m})) = \mathcal{O}(2^{8m}m^2)$ and depth $\mathcal{O}(m\log(m))$ as given in Corollary 13 . Thus for this phase we need a circuit of total size $\mathcal{O}(nm^2)$.

Then we subdivide each block into 2^{6m} parts each consisting of 2^{2m} numbers. Observe that most of these parts are monochromatic: a part is monochromatic if it contains 2^{2m} copies of a single m-bit number. We can upper bound the number of non-monochromatic parts by 2^m. We can add a single indicator bit to each part indicating whether this part is monochromatic. As the parts are sorted it is enough to compare the first and last number in each part and set the bit to 1 if the numbers are equal and to 0 otherwise. We sort the parts prefixed by their indicator bit using the circuit $\mathrm{PARTIAL\_AKS}_{2^{6m},1+m2^{2m},1}$ from Corollary 15 to move all non-monochromatic parts to the front of each block. Thus the total size of the circuit sorting parts inside each block is $\mathcal{O}\left(\frac{n}{2^{8m}}(2^{6m})(1+m2^{2m})6m\right) = \mathcal{O}(nm^2)$ and depth $\mathcal{O}(m)$. We call the first 2^m parts of each block *potentially non-monochromatic*. The other parts are *definitely monochromatic*.

From each definitely monochromatic part we take the first m-bit number and we count them. This can be done by the circuit $\mathrm{COUNT}_{\frac{n}{2^{8m}}(2^{6m}-2^m),m}$ from Lemma 9 of size $\mathcal{O}\left(\left(\frac{n}{2^{2m}} - \frac{n}{2^{7m}}\right)m2^m\right) \leq \mathcal{O}(nm)$ and depth $\mathcal{O}(\log(n)+\log(m))$. By multiplying each count by 2^{2m} (that is by appending $2m$ zeroes) we get the number of occurrences of each number in the definitely monochromatic parts.

As there are relatively few (exactly $\frac{n}{2^{8m}}2^m2^{2m}$) numbers overall in potentially non-monochromatic parts we can use the circuit $\mathrm{COUNT}_{n/2^{5m},m}$ from Lemma 9 to count those numbers by a circuit of size $\mathcal{O}\left(\frac{n}{2^{5m}}m2^m\right) \leq \mathcal{O}(nm)$ and depth $\mathcal{O}(\log(n)+\log(m))$.

Thus we get two vectors of counts for numbers in potentially non-monochromatic and definitely monochromatic blocks. Finally, we add the two vectors of 2^m numbers each consisting of at most $\lceil 1+\log(n)\rceil$ bits to get the resulting counts. This uses a circuit of size $\mathcal{O}(m2^m) = O(n)$ and depth $\mathcal{O}(\log\log(n))$. Thus, the overall size of the circuit is $\mathcal{O}(nm^2)$ and depth $\mathcal{O}(\log(n)+m\log(m))$. ◀

▶ **Lemma 16.** *For integers $n, m \geq 1$ such that $m \leq \log(n)/11$, there is a family of boolean circuits*

$$FAST\_DECOMPRESS_{n,m} \colon \{0,1\}^{\lceil 1+\log(n)\rceil 2^m} \to \{0,1\}^{n\cdot m}$$

that decompresses its input as in Lemma 10. The size of $FAST\_DECOMPRESS_{n,m}$ is $\mathcal{O}(nm^2)$ and its depth is $\mathcal{O}(m\log(m) + \log\log(n))$.

The construction of the decompression circuit mirrors the counting circuit albeit it is somewhat simpler with a different choice of parameters. We separately decompress monochromatic blocks (by decompressing just a single string from each block and then creating the right number of copies) and the strings from non-monochromatic blocks (as there are not many of those). We then use partial sorting to rearrange the blocks in the proper order to construct a sorted sequence.

Proof. For the sake of simplicity let us assume that n is a power of two and let us set $k = n/2^{8m}$. (Thus k is an integer.) We will think of the output as partitioned into 2^{8m} blocks of size k. As in the proof of Lemma 10 we compute the prefix sums

$$p_x = \sum_{y\in\{0,1\}^m \colon y<x} n_y \qquad \text{for each } x \in \{0,1\}^m$$

and we set $p_{2^m} = n$. (Here, we identify m-bit strings x and y with integers they represent.) We can compute each p_x using the circuit $\mathrm{SUM}_{2^m,1+\log(n)}$, thus computing all of them using a circuit of size $\mathcal{O}(\log(n)2^{2m}) \leq \mathcal{O}(n)$ (by the assumption $m \leq \log(n)/11$) and depth $\mathcal{O}(m+\log\log(n))$. Thus the string $x \in \{0,1\}^m$ should appear at output positions $[p_x+1, p_{x+1}]$. For any $x \in \{0,1\}^m$ we set:

$$r_x = ((k - (p_x \mod k)) \mod k) + (p_{x+1} \mod k)$$
$$q_x = \frac{n_x - r_x}{k}$$

The meaning is that if we partition the output into blocks of k consecutive numbers, then for any $x \in \{0,1\}^m$ the number r_x tells the number of times the string x appears in non-monochromatic blocks. (These occurrences are located in at most two non-monochromatic blocks.) The number q_x tells us in how many monochromatic blocks the string $x \in \{0,1\}^m$ appears. Observe that q_x is an integer. Since n is a power of two, so is k, furthermore, k is fixed for given n and m, and thus computing mod k and division by k corresponds to selecting appropriate bits from the binary representation of numbers. All numbers p_x, q_x and r_x are integers represented by $1 + \log(n)$ bits. Hence, each q_x and r_x can be computed from n_x and p_x by one circuit $\mathrm{ADD}_{1+\log(n)}$ and two $\mathrm{SUB}_{1+\log(n)}$. The circuit computing values q_x and r_x for all x has total size $\mathcal{O}(2^m\log(n))$ and depth $\mathcal{O}(\log\log(n))$.

The following holds:

$$n_x = kq_x + r_x$$
$$\sum_{x\in\{0,1\}^m} q_x = \sum_{x\in\{0,1\}^m} \frac{n_x - r_x}{k} \leq n/k = 2^{8m}$$
$$\sum_{x\in\{0,1\}^m} r_x \leq 2k2^m = 2n/2^{7m}$$

We use circuit $\mathrm{DECOMPRESS}_{2^{8m},m}(q_{0^m}, q_{0^{m-1}1}, \ldots, q_{1^m})$ from Lemma 10 of size $\mathcal{O}\left(m2^{9m}\right)$ and depth $\mathcal{O}\left(m\right)$ to decompress monochromatic blocks. We then just copy each resulting number k times to create sorted monochromatic blocks. Last $2^{8m} - \sum_{x\in\{0,1\}^m} q_x$ blocks contain zero padding corresponding to the numbers in non-monochromatic blocks. They will be merged with the non-monochromatic blocks obtained next.

In order to properly match the non-monochromatic blocks to the padded zeroes we adjust the count r_{0^m}:

$$r'_{0^m} = \left(2n/2^{7m}\right) - \sum_{x\in\{0,1\}^m:\, x\neq 0^m} r_x$$

using circuit $\mathrm{SUM}_{2^m,1+\log(n)}$ and $\mathrm{SUB}_{1+\log(n)}$ of size $\mathcal{O}(n)$ and depth $\mathcal{O}(m+\log\log(n))$. We use the circuit $\mathrm{DECOMPRESS}_{2n/2^{7m},m}(r'_{0^m}, r_{0^{m-1}1}, \ldots, r_{1^m})$ from Lemma 10 to decompress the non-monochromatic blocks. The circuit is of size

$$\mathcal{O}\left(\left(2n/2^{7m}\right)m2^m + 2^{2m}\log\left(2n/2^{7m}\right)\right) \leq \mathcal{O}\left(nm/2^{6m}\right)$$

and of depth $\mathcal{O}(m+\log\log(n))$. (Here, we used our assumption $m \leq \log(n)/11$, to bound $n \geq 2^{11m}$ and $2^{2m} \leq n^{3/4}/2^{6m}$.)

Finally, we compute the bit-wise OR of the last 2^{m+1} blocks of the output from the previous step (monochromatic decompression) with the current output (non-monochromatic decompression). This way we get a sequence of n numbers partitioned into blocks where each block corresponds to one of the blocks in the desired output. However, we still need to rearrange the blocks in the proper order. We will use partial sorting of the whole blocks to do that.

For a given block let x be the first number in that block. We prefix the block by a number $2x$ (represented by $m+1$ bits) if the block is monochromatic or the number $2x+1$ if the block is non-monochromatic. To determine whether the block is monochromatic we compare for equality the first and last number inside the block. We do this for each block. Thus each block of k numbers is prefixed by an $m+1$ bit number. Computing these prefixes requires a circuit of total size $\mathcal{O}(2^{8m}m) = O(n)$ and depth $\mathcal{O}(\log(m))$. We then use the $\mathrm{PARTIAL\_AKS}_{2^{8m},(m+1)+km,m+1}$ circuit of size $\mathcal{O}(nm^2)$ and depth $\mathcal{O}(m\log(m))$ to sort the blocks. Finally, we ignore the $m+1$ bit prefixes of each block to get the desired output. ◀

Proof of Theorem 1. This is just a combination of Lemma 2 with Lemma 16. ◀

Observe that the proofs of Lemma 2 and Lemma 16 do not depend on using specifically the AKS sorting. In particular for the case of Lemma 2 if there is a circuit that sorts input numbers that is linear in the number of input bits then there is a linear size circuit that counts these numbers.

6 Partial Sorting by the First k Bits in Poly-logarithmic Depth

Here we design a family of boolean circuits that partially sorts by the first k bits out of m bits which is asymptotically smaller than $\mathrm{PARTIAL\_AKS}_{n,m,k}$. We will need super-concentrators for our construction.

A directed acyclic graph $G = (V, E, A, B)$, where V is the set of vertices, E is the set of directed edges, and A and B are disjoint subsets of vertices of the same size, is a *super-concentrator* if the following hold: The vertices in A (*inputs*) have in-degree zero, vertices in B (*outputs*) have out-degree zero, and for any $S \subseteq A$ and for any $T \subseteq B$: $|S| = |T|$ there is a set of pairwise vertex disjoint paths connecting each vertex from S to some vertex in T.

We parametrize the super-concentrator by the number of input vertices n, and we measure its size by the number of edges. We want the graph to have as few edges as possible. The depth of the super-concentrator is the number of edges on the longest directed path.

Pippenger [8] shows a construction of super-concentrators of linear size and logarithmic depth. He constructs a family of super-concentrators S_n for n being the number of inputs, where the in-degree and out-degree of each vertex is bounded by some universal constant, the number of edges is linear in n, and the depth is $\mathcal{O}(\log(n))$. Moreover there are finite automatons which for any $S \subset A, T \subset B$: $|S| = |T|$ when put on the vertices of the super-concentrator find the set of vertex disjoint paths from S to T in $\mathcal{O}(\log(n))$ iterations, each taking $\mathcal{O}(\log(n))$ steps, for the total number of $\mathcal{O}(n)$ steps of the automatons. We describe this construction using the language of circuits. The circuit on input of characteristic vector of S and T computes the set of $|T|$ vertex disjoint paths connecting S and T. The circuit outputs the characteristic vector of the set of edges participating in the paths.

▶ **Theorem 17** (Pippenger [8]). *There is a family of super-concentrators S_n as described above and boolean circuits $ROUTE_n\colon \{0,1\}^{2n} \to \{0,1\}^{|S_n|}$ of size $\mathcal{O}(n\log(n))$ and depth $\mathcal{O}(\log^2(n))$ that on input characteristic vector of any set $T \subseteq [n]$ and characteristic vector of any $S \subseteq [n]$ where $|T| = |S|$, outputs the characteristic vector of edges that form $|T|$ vertex disjoint paths between S and T.*

By routing m bits along each path in the super-concentrator we can use the above circuit to build a circuit that partially sorts m-bit integers by their most significant bit.

▶ **Corollary 18.** *There is a family of boolean circuits*

$$PIPPENGER\_SORT_{n,m,1}\colon \{0,1\}^{n\cdot m} \to \{0,1\}^{n\cdot m}$$

that on input $x_1, x_2, \ldots, x_n \in \{0,1\}^m$ partially sort these numbers according to their first most significant bit. The size of the circuit $PIPPENGER\_SORT_{n,m,1}$ is $\mathcal{O}(nm + n\log(n))$ and depth $\mathcal{O}(\log^2(n))$.

Proof. We give a sketch of the proof. First, we will use the graph S_n to get all inputs starting with one to the proper place. Then, using the same construction we will move all inputs starting by 0 to the proper place. We transform the graph S_n into a circuit by replacing each vertex of in-degree d by a *routing gadget* (circuit) which takes d m-bit inputs together with d control bits, one bit for each of the m-bit inputs, and outputs the bit-wise OR of inputs for which their control bit is set to 1. Such a routing gadget of size $\mathcal{O}(dm)$ and depth $\mathcal{O}(\log(d))$ can be easily constructed. If (u, v) is the j-th incoming edge of v in S_n, we connect the j-th block of m input bits of the routing gadget corresponding to v to the output of the routing gadget of u. The routing gadgets of input vertices of S_n are connected directly to the appropriate inputs of the sorting circuit. The routing gadget will be used with at most single control bit set to one, thus it will route the corresponding input.

It remains to calculate paths that will route the integers starting with 1 in the above circuit in the desired way. For that, we calculate the sum s of the most significant bits by which we are sorting using $\mathrm{SUM}_{n,1}$ from Lemma 6, we expand it back using $\mathrm{ONES}_{\lceil\log(n)\rceil+1}(s)$, and reverse it to get the characteristic vector of a set T, where we want to route to. Together with the most significant bits of each input integer (which form the characteristic vector of S from which we route) we feed this as an input to ROUTE_n. The output bits of ROUTE_n are connected to the appropriate control bits of our routing gadgets. The sorted output will be obtained as the output of the n routing gadgets corresponding to the output vertices of S_n.

The size of the ROUTE_n is $\mathcal{O}(n \log(n))$ and the total size of the circuits implementing the routing gadgets is $\mathcal{O}(mn)$. These two terms dominate the overall size of the circuit. The depth of the circuit is dominated by the depth of the ROUTE_n. ◀

We can use the above circuit in an iterative fashion to build a smaller circuit for the same primitive.

▶ **Lemma 19.** *There is a family of boolean circuits* $ITERATIVE\_SORT_{n,m,1}\colon \{0,1\}^{n \cdot m} \to \{0,1\}^{n \cdot m}$ *that on input* $x_1, x_2, \ldots, x_n \in \{0,1\}^m$ *partially sort these numbers according to their first most significant bit. The size of the circuit* $ITERATIVE\_SORT_{n,m,1}$ *is* $\mathcal{O}(nm(1 + \log^*(n) - \log^*(m)))$ *and its depth is* $\mathcal{O}(\log^2(n))$.

Proof. Assume $m \leq \log(n)/11$ otherwise use Corollary 18. We will build the circuit iteratively using the circuit from Corollary 18 for blocks of various sizes. We will start with small blocks of items and we will iteratively sort larger and larger number of items organized into mostly monochromatic blocks. Without loss of generality we assume that m is a power of two, and we will ignore the rounding issues. We will have two parameters m_i and $n_i = 2^{3m_i}$, where $m_0 = m$ and $m_{i+1} = 2^{m_i}$ for $i \geq 0$. At iteration i, all the items will be partitioned into *parts* of consecutive numbers, each part will be either *monochromatic* containing all zeros, all ones, or it will be *mixed*. (Here we refer to the most significant bits of the numbers in the part.) For each part we will maintain two indicator bits which of the three possibilities occurs: an indicator which is one if the block is mixed, and another *color* indicator which specifies the highest order bit of the integers if the block is monochromatic. (For the latter we could use the first bit of the first integer in the part.) At each iteration $i > 0$, m_i will denote the number of items in each part. n_i/m_i consecutive parts form a *block*, so each block contains n_i items. The blocks partition the input. We will maintain an invariant that the fraction of mixed parts in each block is at most $2/m_i^3$.

At iteration 0 we apply $\mathrm{PIPPENGER\_SORT}_{n_0,m,1}$ to consecutive blocks of n_0 input integers. Afterwards, the block is partitioned into parts of size m_1 and for each part we determine its status by comparing the most significant bits of the first and last integer in the part. It is clear that each block of size n_0 contains at most one mixed part. As the number of parts in the block is m_1^3, the fraction of mixed parts in each block is at most $2/m_1^3$, and this is also true for blocks of size n_1.

At iteration $i > 0$, we divide the current sequence of parts of size m_i into blocks containing n_i/m_i parts, and we proceed in three steps:

Step 1. Sort the parts in each block using $\mathrm{PIPPENGER\_SORT}_{n_i/m_i,2+m_i \cdot m,1}$ according to the mixed indicator. Hence, all the mixed parts will move to the end of the block. There are at most $2n_i/m_i^3$ mixed parts in each block, the remaining parts must be monochromatic.

Step 2. In each block, sort all the m-bit integers in the last $2n_i/m_i^3$ parts according to their most significant bit using $\mathrm{PIPPENGER\_SORT}_{2n_i/m_i^2,m,1}$. This sorts together all the integers in the mixed parts (and perhaps few other parts). Repartition them into parts of m_i consecutive numbers and determine their indicator bits. Only one of the parts should be mixed at this point. Swap it with the last part in the block. (We provide details of the swap later.)

Step 3. In each block, sort all the parts except for the last one according to their color indicator using $\mathrm{PIPPENGER\_SORT}_{(n_i/m_i)-1,2+m_i \cdot m,1}$. This moves all the parts of color 0 to the front. Repartition all the numbers in the block into parts of m_{i+1} consecutive integers and determine their indicator bits, where the last part is marked as mixed. At

most two of the new parts should be mixed at this point. Notice, that out of m_{i+1}^3 parts in each block, at most two are marked as mixed so the invariant applies. We can move to the next iteration.

We iterate the algorithm until $m_i \geq \log(n)/4$. Once $m_i \geq \log(n)/4$, the number of integers in mixed parts is at most $2n/m_i^2 \leq O(n/\log^2(n))$, remaining items are in monochromatic parts. At this point we cannot form a block of size n_i, but we can still perform the same type of actions as in Steps 1-3: We can bring the monochromatic parts forward as in Step 1, sort the last $32n/\log^2(n)$ integers belonging to the mixed parts, move the remaining mixed part to the end, sort the monochromatic parts and swap the mixed part with the first monochromatic part of color 1.

To swap a single mixed part with the last part we can copy the mixed part into a buffer by AND-ing every part bit-wise with the indicator whether that is the mixed part, and OR-ing all the results together. This copies the mixed part into a buffer. In a similar fashion we can copy the last part into the now unused part by letting each part bit-wise copy to its place either its original content or the content of the last part, again conditioning on an appropriate indicator bit. Hence, the swap can be implemented by a circuit of size proportional to the total size of the parts and depth logarithmic in the number of parts.

Now we will bound the total size of the circuit we constructed. Step 1 requires n/n_i circuits of size $\mathcal{O}(n_i m + n_i/m_i \log(n_i/m_i)) = \mathcal{O}(n_i m)$, as $\log(n_i) = O(m_i)$, and of depth at most $\mathcal{O}(\log^2(n_i))$. Step 2 requires n/n_i sorting circuits of size $\mathcal{O}(mn_i/m_i^2 + 2n_i/m_i^2 \log(2n_i/m_i^2)) = \mathcal{O}(n_i)$ and of depth at most $\mathcal{O}(\log^2(n_i))$, together with a circuit of total linear size $\mathcal{O}(n)$ to recalculate the parts and do the swaps. The last step requires the same amount of circuitry as the first step.

Hence, each step requires circuits of total size $\mathcal{O}(nm)$. The same goes for the initial sort at iteration 0, and the final sorts at the end. As there are at most $\log^*(n) - \log^*(m)$ iterations, the resulting size is $\mathcal{O}(nm(\log^*(n) - \log^*(m)))$. Each step requires a circuit of depth $\mathcal{O}(\log^2(n_i))$, recall that by our choice $n_i = 2^{4m_i}$, thus $\log(n_i) = 4m_i$. Since $m_{i+1} = 2^{m_i}$ and for each i we have $m_i \leq \log(n)/4$, thus the total depth is dominated by the last iteration where we use a circuit of depth $\mathcal{O}(\log^2(n))$. ◀

Proof of Theorem 3. We assume that $k \leq \log(n)/11$ otherwise we can use Corollary 15 to sort the elements. Without loss of generality we assume n is a power of two. We think of the input as organized into an array. We extract the first k bits (*key*) from each input element and we sort the keys using the circuit from Theorem 1 of size $\mathcal{O}(nk^2)$ and depth $\mathcal{O}(\log(n) + k\log(k))$.

We will build recursively a circuit that will sort the input array of n elements according to the first k bits when the input is augmented with the array of sorted keys. Now our goal is to split the input array into two equal sized parts L and R where all elements in L are less or equal to elements in R when comparing only the keys.

To do that we take the *median*, the $n/2$-th element among the keys, and we partition the array according to it. We split the input array into three arrays L, M, and R of length n with elements less than, equal to, and greater than the median, resp., and we mark the unused elements as *dummy* using an extra bit associated to each element. We sort L and M so that all non-dummy elements are to the left and R so that all non-dummy elements are to the right. We use three circuits ITERATIVE_SORT$_{n,m+1,1}$ to do that. Now, we flip the first half of elements in M, i.e., swap the i-th element with the element in position $(n/2) - i + 1$, and we replace the dummy elements in the first half of L by the corresponding elements in M.

By one application of ITERATIVE_SORT$_{n,m+1,1}$ we move all the remaining non-dummy elements in M to the left, and we merge those elements with the second half of R. We discard the second and first half of L and R, respectively. (They contain only dummy elements.)

If the highest order bit of the median is set to 0 then all the elements in L have the highest order bit set to 0, otherwise all the elements in R have the highest order bit set to 1. In either case we reduced the problem to one problem of sorting half of the elements according to $k-1$ bits and the other half according to k-bits. We recursively build a circuit to sort SORT$_{n/2,m,k-1}$ and SORT$_{n/2,m,k}$ when the input is augmented with the sorted array of keys. We pass to each of the sorting sub-circuits the appropriate sub-problem and we re-route the results from them to form the final output.

Not counting the two sub-circuits SORT$_{n/2,m,k-1}$ and SORT$_{n/2,m,k}$, this step requires four copies of the circuit ITERATIVE_SORT$_{n,m+1,1}$ and additional $\mathcal{O}(nm)$ gates to do the moves and element comparison with the median. Denote the size of this part of the circuit by $L_m(n) = \mathcal{O}(nm(1+\log^*(n)-\log^*(m)))$. The depth of the resulting circuit to perform all those operations is $\mathcal{O}(\log^2(n))$ as the move operations are done in parallel (again, not counting the depth of SORT$_{n/2,m,k-1}$ and SORT$_{n/2,m,k}$). If we denote by $S_{m,k}(n)$ the size of the circuit SORT$_{n,m,k}$ we get the following recurrence:

$$\begin{aligned}
S_{m,k}(1) &= \mathcal{O}(m)\\
S_{m,1}(n) &= \mathcal{O}(nm(1+\log^*(n)-\log^*(m)))\\
S_{m,k}(n) &\leq L_m(n) + S_{m,k-1}\left(\frac{n}{2}\right) + S_{m,k}\left(\frac{n}{2}\right)
\end{aligned}$$

when we iterate the recurrence:

$$\begin{aligned}
S_{m,k}(n) &= L_m(n) + S_{m,k-1}(n/2) + S_{m,k}(n/2)\\
&= L_m(n) + S_{m,k-1}(n/2) + L_m(n/2) + S_{m,k-1}(n/4) + S_{m,k}(n/4)\\
&= L_m(n) + S_{m,k-1}(n/2) + L_m(n/2)\\
&\quad + S_{m,k-1}(n/4) + L_m(n/4) + S_{m,k-1}(n/8) + S_{m,k}(n/8)\\
&= \dots\\
&= (L_m(n) + L_m(n/2) + \dots + L_m(1))\\
&\quad + (S_{m,k-1}(n/2) + S_{m,k-1}(n/4) + \dots + S_{m,k-1}(1)) + S_{m,k}(1)\\
&\leq L_m(2n) + S_{m,k-1}(n) + \mathcal{O}(m)
\end{aligned}$$

which gives us

$$\begin{aligned}
S_{m,k}(n) &= kL_m(2n) + (k-1)S_{m,k}(1) + S_{m,1}(n)\\
&= kL_m(2n) + \mathcal{O}(nm(1+\log^*(n)-\log^*(m)))\\
&= \mathcal{O}(knm(1+\log^*(n)-\log^*(m)))
\end{aligned}$$

To bound the depth $D_{m,k}(n)$ we use the following recurrence:

$$\begin{aligned}
D_{m,k}(1) &= \mathcal{O}(1)\\
D_{m,k}(n) &\geq D_{m,k-1}(n)\\
D_{m,1}(n) &= \mathcal{O}(\log^2(n))\\
D_{m,k}(n) &= \mathcal{O}(\log^2(n)) + \max\left(D_{m,k}(n/2) + D_{m,k-1}(n/2)\right)\\
&\leq \mathcal{O}(\log^2(n)) + D_{m,k}(n/2)\\
&\leq \mathcal{O}(\log^3(n))
\end{aligned}$$

◀

7 Conclusion

We have provided improved sorting circuits. Our technique used in the proof of Theorem 1 can be viewed as information compression and decompression. This technique might prove useful for other related problems. We list some open problems:

- Most of our circuits are uniform. The non-uniform part is due to the use of the AKS circuits and Pippenger's super-concentrators. Can one make uniform circuits of the same size?
- Kospanov [6] shows that there is a family of sorting circuits with depth $\mathcal{O}(\log(n)+\log(m))$ and size $\mathcal{O}(mn^2)$ that sorts n numbers each of m bits. Is there a circuit family for sorting with circuits of depth $\mathcal{O}(\log(n)+\log(m))$ and size $\mathcal{O}(nm^2)$? In other words can we get rid of the $m\log(m)$ factor in the circuit depth from Theorem 1 while keeping the $\mathcal{O}(nm^2)$ size?
- Is it possible to partially sort n numbers of m bits each by their first bit using a circuit of size $\mathcal{O}(nm)$ and depth $\mathcal{O}(\log(n))$?

References

1 Miklós Ajtai, János Komlós, and Endre Szemerédi. Sorting in $c\log(n)$ parallel steps. *Combinatorica*, 3(1):1–19, 1983.

2 Arne Andersson, Torben Hagerup, Stefan Nilsson, and Rajeev Raman. Sorting in linear time? *Journal of Computer and System Sciences*, 57(1):74–93, 1998.

3 Gilad Asharov, Wei-Kai Lin, and Elaine Shi. Sorting short keys in circuits of size $o(n\log n)$. In *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 2249–2268. SIAM, 2021.

4 Yijie Han and Mikkel Thorup. Sorting integers in $O(n\sqrt{\log\log n})$ expected time and linear space. In *IEEE Symposium on Foundations of Computer Science (FOCS'02)*, 2002.

5 Stasys Jukna. *Boolean function complexity: advances and frontiers*, volume 27. Springer Science & Business Media, 2012.

6 E. S. Kospanov. Scheme realization of the sorting problem. *Diskretnyi Analiz i Issledovanie Operatsii*, 1(1):13–19, 1994.

7 Wei-Kai Lin and Elaine Shi. Optimal sorting circuits for short keys. *arXiv preprint*, 2021. `arXiv:2102.11489`.

8 Nicholas Pippenger. Self-routing superconcentrators. *Journal of Computer and System Sciences*, 52(1):53–60, 1996.

9 Christopher S. Wallace. A suggestion for a fast multiplier. *IEEE Transactions on electronic Computers*, (1):14–17, 1964.

Improving Gebauer's Construction of 3-Chromatic Hypergraphs with Few Edges

Jakub Kozik
Theoretical Computer Science Department, Faculty of Mathematics and Computer Science, Jagiellonian University, Kraków, Poland

Abstract

In 1964 Erdős proved, by randomized construction, that the minimum number of edges in a k-graph that is not two colorable is $O(k^2\,2^k)$. To this day, it is not known whether there exist such k-graphs with smaller number of edges. Known deterministic constructions use much larger number of edges. The most recent one by Gebauer requires $2^{k+\Theta(k^{2/3})}$ edges. Applying a derandomization technique we reduce that number to $2^{k+\widetilde{\Theta}(k^{1/2})}$.

2012 ACM Subject Classification Mathematics of computing → Hypergraphs; Mathematics of computing → Probabilistic algorithms; Theory of computation → Pseudorandomness and derandomization

Keywords and phrases Property B, Hypergraph Coloring, Deterministic Constructions

Digital Object Identifier 10.4230/LIPIcs.ICALP.2021.89

Category Track A: Algorithms, Complexity and Games

Related Version `https://arxiv.org/abs/2102.11674`

Funding *Jakub Kozik*: This work was partially supported by Polish National Science Center (2016/21/B/ST6/02165).

1 Introduction

In 1964 Erdős proved in [3] that $(1+o(1))\frac{e\ln(2)}{4}k^2 2^k$ edges are sufficient to build a k-graph[1] which is not two colorable. To this day that result provides the best known upper bound for the minimum number of edges in such hypergraph. The Erdős' bound results from the fact that random k-graph with that number of edges, built on a set of $k^2/2$ vertices can not be colored properly with two colors with high probability. In some sense this value is not far from being optimal since a straightforward probabilistic argument shows that any k-graph with at most 2^{k-1} edges is two colorable. Using random recoloring method, introduced by Beck in [1], Radhakrishnan and Srinivasan in [8] extended this result to k-graphs with at most $\Theta(\sqrt{k/\log(k)}2^k)$ edges (see also [2] for a simplified proof).

The best known deterministic construction of a k-graph that is not two colorable has been obtained by Gebauer [5]. It requires $2^{k+\Theta(k^{2/3})}$ edges. It is also the first construction in which the number of edges is $2^{k+o(k)}$. The main result of the current paper is an upgrade of this construction that allows to cut down the number of edges to $2^{k+\Theta((k\log(k))^{1/2})}$.

Within the whole paper, log(.) stands for binary logarithm. We are only concerned with vertex two coloring of hypergraphs. However, just like in the construction from [5], the presented method naturaly generalizes to any fixed number of colors and gives analogous improvement. Vertex coloring is *proper* if no edge is monochromatic. Following common convention we use colors *red* and *blue*.

[1] i.e. k-uniform hypergraph

2 Gebauer's construction

We start with recalling the construction of [5], as we are going to modify it. The whole procedure is parameterized by $t = t(k)$ that takes value roughly k^α for some optimized positive $\alpha < 1$. It it convenient to organize the vertices of the constructed hypergraph into a rectangular matrix $\mathbb{M}$ of $2t-1$ rows and s columns. In particular, the vertex set of the hypergraphs has $(2t-1)\cdot s$ elements. The value of s will be a subject of optimization and in the final construction we are going to chose $s = \lceil \mathrm{e}\,(k/t)\,2^t \rceil$. Slightly abusing the notation, we use $\mathbb{M}$ for both the matrix and the set of vertices. We use the same convention for submatrices of $\mathbb{M}$.

2.1 Preliminary choice of rows

As the vertices of constructed hypergraph are identified with the entries of $\mathbb{M}$, vertex coloring can be seen as assigning colors to the entries of the matrix. A color is *dominating* in a row if at least half of its entries are colored with it (there can be two dominating colors). A matrix for which one of the colors is dominating in all rows will be called *consistently dominated*. The main part of the construction is designed to work with a consistently dominated submatrix of $\mathbb{M}$ with t rows. Wlog we always assume that red is the dominating color in such a matrix.

Let $\mathcal{M}$ denote the set of submatrices of $\mathbb{M}$ built of every t rows. For every $M \in \mathcal{M}$ we apply the *main construction* described in the next section. The construction outputs hypergraph H_M. The union of the edge sets of these hypergraphs forms the edge set of the resulting hypergraph. For every coloring of $\mathbb{M}$ at least one submatrix $M \in \mathcal{M}$ is consistently dominated. The main construction guarantees that in such a case, H_M contains a monochromatic edge.

2.2 Main construction

Let $M \in \mathcal{M}$, recall that M has t rows. Our goal is to build a hypergraph H_M on the vertex set M such that for every consistently dominated coloring of M, there exists a monochromatic edge in H_M. For $(\sigma_1, \ldots, \sigma_t) \in [s]^t$, we denote by $M(\sigma_1, \ldots, \sigma_t)$ matrix M in which for every $i \in [t]$, the i-th row has been cyclically shifted by σ_i. The construction proceeds as follows.

For every
1. *sequence of shifts* $\sigma \in [s]^t$,
2. and *set of indices* $I \subset [s]$ of size k/t,

add to H_M an edge built from all elements of the columns of $M(\sigma)$ with indices in I.

Note that the edges of H_M are of size k as required.

Let us fix a consistently dominated coloring of M. We assume wlog that red is the dominating color of the rows. When the sequence of shifts is chosen randomly, the probability that some fixed column is red is at least 2^{-t}. As a consequence, for $s \geqslant (k/t)\,2^t$ the expected number of red columns is at least k/t. In particular, for some sequence of shifts, there exists a set of k/t red columns. Hence the edge built for these shifts and columns is monochromatic.

2.3 Counting

We have

$$\binom{2t-1}{t} < 2^{2t}$$

choices for the subset of rows in the preliminary step. Then, in the main construction, every sequence of t elements of $[s]$ and a subset of k/t elements of $[s]$ is used to build an edge. The number of choices is

$$s^t \cdot \binom{s}{k/t} \leqslant s^t \cdot \left(\frac{\mathrm{e}s}{k/t}\right)^{k/t}.$$

For $s = (k/t)\, 2^t$ (we assume for simplicity that it is an integer) we obtain

$$(k/t)^t\, 2^{t^2} \cdot \mathrm{e}^{k/t}\, 2^k = 2^{t\log(k/t)+t^2+k/t\log(\mathrm{e})+k}.$$

The total number of edges is smaller than

$$2^{2t+t\log(k/t)+t^2+k/t\log(\mathrm{e})+k}.$$

Finally we choose t so that the above exponent is minimized. That happens for $t = \Theta(k^{1/3})$. In the end we obtain that the total number of edges is $2^{k+\Theta(k^{2/3})}$.

3 Improved construction

We modify only the main construction. Recall that we work with matrix M with t rows. For a fixed consistently dominating coloring of M, sequence of shifts $\sigma \in [s]^t$ is called *good* if $M(\sigma)$ contains at least $s\, 2^{-t}$ red columns. The set of good sequences for a coloring $\mathcal{C}$ of M is denoted by $\mathcal{G}(\mathcal{C})$.

If we fix a consistently dominating coloring of M and choose the sequence of shifts $\sigma \in [s]^t$ uniformly at random, the expected number of red columns in $M(\sigma)$ is $s\, 2^{-t}$. That observation was used to justify that there exists a good sequence. However, it also suggests that a large number of shift sequences might be good. For the constructed hypergraph not to be two colorable, it is sufficient that for every consistently dominated coloring of M, at least one such sequence is used in the main construction.

We apply derandomization techniques to construct relatively small set of sequences of shifts that can be used in the main construction instead of $[s]^t$. For a family of sets $\mathcal{F}$, a set that intersects every element of that family is called a *hitting set* for $\mathcal{F}$. In these terms we are looking for a small hitting set for family $\mathcal{G}_M = \{\mathcal{G}(\mathcal{C}) : \mathcal{C} \text{ is a consistently dominating coloring of } M\}$.

3.1 Sequential choice of shifts

We start with estimating the size of the set of good shift sequences. While it is not directly used in our construction, it provides a good opportunity to introduce some tools. It will also allow us to derive a probabilistic argument that small hitting sets actually exist.

The property of being good is generalized to prefixes in the straightforward way – sequence of shifts $(\sigma_1, \ldots, \sigma_i)$ is *good* if the matrix trimmed to the first i rows and shifted according to the sequence, has at least $s\, 2^{-i}$ red columns.

Suppose that $(\sigma_1, \ldots, \sigma_i)$ is good. We want to estimate the number of possible choices of σ_{i+1} for which $(\sigma_1, \ldots, \sigma_i, \sigma_{i+1})$ is good as well. If the coloring of the $(i+1)$-th row was "random", then about half of the choices would be right, and almost all of the choices would be almost right. That property does not hold in the worst case scenario and hence we are going to work with relaxed definitions.

For $\varepsilon > 0$, a sequence of shifts $(\sigma_1, \ldots, \sigma_i)$ is *ε-good* if the number of red columns in the shifted matrix trimmed to the first i rows is at least $s\left(\frac{1-\varepsilon}{2}\right)^i$. Then, every ε-good sequence of shifts of length t gives a shifted matrix with at least

$$s\frac{(1-\varepsilon)^{t-1}}{2^t}$$

red columns. For $s \geqslant \mathrm{e}\,(k/t)\,2^t$ and $\varepsilon = 1/t$, the number of red columns is at least k/t as needed. In the modified construction we set s to $\lceil \mathrm{e}\,(k/t)\,2^t \rceil$.

We also define $\mathcal{G}^\varepsilon(\mathcal{C})$ as the set of ε-good sequences for a coloring $\mathcal{C}$ of M and $\mathcal{G}^\varepsilon_M$ as $\{\mathcal{G}^\varepsilon(\mathcal{C}) : \mathcal{C}$ is a consistently dominating coloring of $M\}$.

The following proposition is used to derive a lower bound for the number of ε-good sequences. It is formulated in more general terms that needed here, but we are going to use it again later. For a set $A \subset [s]$ and a number x, set $A + x$ is defined as the set A shifted cyclically within $[s]$ by x, formally $A + x = \{(a - 1 + x)(\mod s) + 1 : a \in A\}$.

▶ **Proposition 1.** *For any positive $\varepsilon < 1$ and sets $A, B \subset [s]$, let $\alpha = |B|/s$, there exist at least*

$$\frac{\varepsilon}{1-(1-\varepsilon)\alpha}\,\alpha s$$

elements $x \in [s]$ for which $|(A + x) \cap B| \geqslant (1-\varepsilon)\alpha|A|$.

Proof. Let random variable X denote the size of $(A+x)\cap B$, when $x \in [s]$ is chosen uniformly at random. By the fact that $|B| = \alpha s$ and linearity of expectation we obtain

$$\mathbb{E}(X) = \alpha|A|.$$

From the definition of X, we get also

$$X \leqslant |A|.$$

We can observe now that a distribution that minimizes $\Pr[X > (1-\varepsilon)\alpha|A|]$ and satisfies the above conditions, is supported only by values $(1-\varepsilon)\alpha|A|$ and $|A|$. There is only one such distribution that satisfies $\mathbb{E}(X) = \alpha|A|$. Straightforward calculations give

$$\Pr[X > (1-\varepsilon)\alpha|A|] \geqslant \frac{\varepsilon\alpha}{1-(1-\varepsilon)\alpha}.$$ ◀

For $|B| \geqslant s/2$ we get that there exist at least

$$\frac{2\varepsilon}{1+\varepsilon}\,s/2$$

elements $x \in [s]$ for which $|(A + x) \cap B| \geqslant (1-\varepsilon)|A|/2$.

Applying the proposition iteratively, we obtain that the number of ε-good sequences of length j is at least

$$\left(\frac{\varepsilon}{1+\varepsilon}\,s\right)^j.$$

(For a fixed j, and some ε-good sequence σ of length $j-1$, let A be the set of indices of the red columns in the matrix trimmed to the first $j-1$ rows and shifted according to σ, and B be the set of indices of red entries of the j-th row.)

For $j = t$ we get a lower bound for the number of ε-good sequences. Once we have that bound, a typical application of the probabilistic method (along the lines of the proof from [3]) allows to proof that there exists a hitting set for $\mathcal{G}_M^\varepsilon$ of size $2^{O(t\log(t))}$. That argument is briefly described below.

Recall that, for a fixed consistently dominated coloring of matrix M with t rows, the volume of ε-good sequences is at least

$$p = \left(\frac{\varepsilon}{1+\varepsilon}\right)^t.$$

The volume is exactly the probability that uniformly random sequence is ε-good. Let S be a set built from m uniformly and independently sampled random sequences from $[s]^t$. (Since, the sequences are sampled with repetitions, it may happen that $|S| < m$.) The following formula upperbounds the expected number of consistently dominated colorings of M, for which the set of ε-good sequences is not hit by S

$$2^{st} \cdot (1-p)^m < \exp(st\ln(2) - mp).$$

Therefore, whenever $st\ln(2) - mp \leqslant 0$, some set of m sequences hits all the sets of ε-good sequences for consistently dominating colorings. For $s = \lceil \mathrm{e}\,(k/t)\,2^t \rceil$ and $\varepsilon = 1/t$ it is sufficient to take m of the order $2^{O(t\log(t))}$ to satisfy the inequality. As a consequence there exists a hitting set for $\mathcal{G}_M^\varepsilon$ of size $2^{O(t\log(t))}$.

3.2 Expanders for hitting sets

Linial, Luby, Saks and Zuckerman [6] worked on deterministic constructions of small hitting sets for combinatorial rectangles. We summarize in this section, their results that are relevant for our developments. We follow closely their definitions.

Graph $G = (V, E)$ is an (m, Δ, α)-*expander* if it has m vertices, maximum degree Δ and for any $A \subset V$, the fraction of vertices in $V - A$ that have a neighbor in A is at least $\alpha|A|/m$. For a fixed graph G let W_r denote the set of walks in G of length r. Let $W_{r,d}$ be the set of subsequences of elements of W_r of length d (not necessarily subsequences of consecutive elements). Set $R \subset [m]^d$ is a *combinatorial rectangle* if it is of a form $R_1 \times \ldots \times R_d$ for some $R_1, \ldots, R_d \subset [m]$. The volume of rectangle R, denoted as $vol(R)$, is defined as $|R|/m^d$.

▶ **Lemma 2** ([6]). *Let m, d be positive integers and R be a rectangle in $[m]^d$. Suppose G is an (m, Δ, α)-expander with $1/2 > \alpha > 0$. If $r = 1 + (4/\alpha)(d + \log(1/vol(R)))$, then $W_{r,d}$ contains a point from R.*

The above lemma implies that a specific set of sequences $W_{r,d}$ hits every combinatorial rectangle in $[m]^d$ of sufficiently large volume.

The following rough estimations for the size of $W_{r,t}$ will be sufficient for our needs. We have

$$|W_r| \leqslant m(\Delta+1)^r$$

and

$$|W_{r,d}| < 2^r|W_r| \leqslant m\,(2(\Delta+1))^r.$$

Lemma 2 leaves some space for the choice of expander graph. Authors of [6] used the construction of Margulis [7] (see also [4]) which allows to build an expander with $\Delta = 8$ and $\alpha = (2-\sqrt{3})/4$. A minor inconvenience is that the construction requires the number

of vertices to be a perfect square. However, as observed already in [6], we can consider the rectangles of our interest as subsets of a larger space $[m']^d$, and apply the lemma in that space. For every m we can choose number m' that is a perfect square and satisfies $m \leqslant m' \leqslant 2m$. While that change affects the volumes of rectangles, they get smaller at most by a factor of 2^{-d}. For our purposes this cost is negligible.

When we are interested in rectangles of volume at least $\mathcal{V}$, Lemma 2 instructs to take

$$r = r(d, \mathcal{V}) = 1 + (4/\alpha)(d + \log(2^d/\mathcal{V})).$$

For some specific constant $\hat{C}$ and for all positive d and $\mathcal{V}$ we have

$$r(d, \mathcal{V}) \leqslant \hat{C}(d + \log(1/\mathcal{V})).$$

▶ **Corollary 3.** *There exists a constant $C > 0$ such that, for every integers m, d, and $\mathcal{V} > 0$ there exists an efficiently constructible subset of $[m]^d$ of size at most*

$$m \cdot 2^{C(d+\log(1/\mathcal{V}))},$$

that intersects every combinatorial rectangle in $[m]^d$ of volume at least $\mathcal{V}$.

We apply that result, to construct a small hitting set for $\mathcal{G}^{\varepsilon}_M$. That set is then used in the modified main construction instead of the set of all shift sequences.

3.3 Under false assumption

Unfortunately, for a fixed consistently dominating coloring of M, the set of good or ε-good shift sequences does not need to form a combinatorial rectangle. It is instructive to pretend for a moment that it does. We assume (falsely) in this subsection that $\mathcal{G}^{\varepsilon}_M$ contains only combinatorial rectangles.

By the discussion that follows Proposition 1, for every consistently dominating coloring of M, the set of ε-good shift sequences has volume at least

$$\nu = \left(\frac{\varepsilon}{2(1+\varepsilon)}\right)^t.$$

By Corollary 3 there exists a hitting set HS for all rectangles of volume ν of size $s \cdot 2^{C(t+\log(1/\nu))}$. For $\varepsilon = 1/t$ and $s = \lceil \mathrm{e}\ (k/t)\ 2^t \rceil$, the size of HS is at most $2^{2Ct\log(t)}$ (assuming that t is sufficiently large). Note that in the original construction all possible shift sequences were used. Using set HS instead of $[s]^t$ and choosing $t = (k\log(k))^{1/2}$, the total number of edges becomes

$$2^{k+O((k\log(k))^{1/2})}.$$

3.4 Decomposing good shift sequences

We showed in Section 3.1 that, for every consistently dominating coloring of M, the set of ε-good shift sequences is large. While, in general, it does not have a structure of combinatorial rectangle, in some sense it can be decomposed into a small number of such. We start by altering the way that the sequences of shifts are represented. For the clarity of the exposition we assume that t is a power of 2.

Let T be a rooted plane complete binary tree with t leaves[2]. A subtree rooted at some internal node of T consists of that node and all its descendants. A node of T is *at level* j if its distance to the set of leaves is j. Let S_j be the set of inner nodes at level j. Note that $|S_j| = t\ 2^{-j}$, we denote that value by d_j. For $h = \log(t)$, the tree has $h+1$ levels with all the leaves on level 0.

We associate leaves of T with rows of M in such a way that the i-th leaf from the left, corresponds to the i-th row. Inner nodes of the tree are going to be labeled by elements of $[s]$. These labels represent the relative shifts between neighboring rows of M. For an inner node v, if l is the rightmost leaf of the left subtree of v and r is the leftmost leaf of the right subtree of v, then the label of v describes how row r is shifted wrt l.

Labeling of a subtree rooted at node v is ε-good, if for r being the number of descendant leaves of v, the submatrix of the rows that correspond to these leaves, shifted according to the labels of the inner nodes of the subtree, has at least $s\ ((1-\varepsilon)/2)^r$ red columns. Note that ε-good labellings of the whole tree correspond to ε-good sequences (up to a cyclic shift of the whole matrix, which is clearly redundant in the original construction).

We order the nodes of S_j from left to right and represent labellings of the nodes of S_j as elements of $[s]^{d_j}$. We are going to work bottom up and label inner nodes in groups consisting of the nodes of the same level. A labeling of T is *ε-good up to level j* if all the subtrees rooted at level at most j are ε-good. In all the places where we use this definition, it can be assumed that the labeling is undefined for the nodes of higher levels. Suppose that τ is a labeling of T that is ε-good up to level $j-1$. Then, a sequence of labels $\sigma \in [s]^{d_j}$ is called *an ε-good level j extension (of τ)* if the labeling τ in which the labels of the nodes of level j has been set to σ is ε-good up to level j.

▶ **Proposition 4.** *Suppose, that a labelling of T is ε-good up to level $j-1$. Then, the set of its ε-good level j extensions forms a combinatorial rectangle of volume at least*

$$\nu_j = \left(\varepsilon\ ((1-\varepsilon)/2)^{-2^{j-1}}\right)^{d_j}.$$

Proof. Fix j and suppose that labeling τ is ε-good up to level $j-1$. We want to assign labels to the nodes of S_j in such a way that all the subtrees rooted at depth j are ε-good shift trees as well. Note that for any pair of distinct nodes of level j, the property of the corresponding subtrees of being ε-good shift trees is determined by disjoint sets of rows of the underlying matrix. That justify that the set of ε-good level j extensions forms a combinatorial rectangle.

Let v be a node of S_j and let A and B be the sets of indices of red columns respectively in the shifted submatrices corresponding to the left and right subtrees of v. By the assumptions we know that both these sets have cardinality at least

$$s\ ((1-\varepsilon)/2)^{-2^{j-1}}.$$

We need to estimate the number of $x \in [s]$ for which the set $A \cap (B+x)$ has cardinality at least

$$s\ ((1-\varepsilon)/2)^{-2^{j}}.$$

Proposition 1 gives that there exist at least

$$\varepsilon\ ((1-\varepsilon)/2)^{-2^{j-1}}\ s$$

[2] i.e. all the internal nodes of T have two children (left and right) and all the leaves are of the same distance from the root

such values. We obtain that the volume of combinatorial rectangle of ε-good level j extensions is at least

$$\left(\varepsilon\left((1-\varepsilon)/2\right)^{-2^{j-1}}\right)^{d_j}$$ ◀

By Corollary 3, there exists a set HS_j of cardinality

$$s \cdot 2^{C(d_j+\log(1/\nu_j))},$$

that is a hitting set for the family of ε-good level j extensions for labellings that are ε-good up to level $j-1$. That implies the following proposition.

▶ **Proposition 5.** *Set $HS = HS_1 \times \ldots \times HS_h$ is a hitting set for the family of sets of ε-good labellings of T.*

It remains to estimate the size of HS. We have

$$\begin{aligned}|HS| &\leqslant \prod_{j=1\ldots h} s \cdot 2^{C(d_j+\log(1/\nu_j))} \\ &= s^{\log(t)} \cdot 2^{C\sum_{j=1\ldots h}(d_j+\log(1/\nu_j))} \\ &< s^{\log(t)} \cdot 2^{Ct} \cdot 2^{C\sum_{j=1\ldots h}\log(1/\nu_j)},\end{aligned}$$

and

$$\begin{aligned}\sum_{j=1\ldots h}\log(1/\nu_j) &= \sum_{j=1\ldots h} d_j(\log(1/\varepsilon) + 2^{j-1}\log(2/(1-\varepsilon))) \\ &< t\log(1/\varepsilon) + t\sum_{j=1\ldots h}\log(4) \quad (\text{for } \varepsilon < 1/2) \\ &= t\cdot\log(1/\varepsilon) + 2t\cdot\log(t)\end{aligned}$$

Therefore, for our parametrization (i.e. $s = \lceil \mathrm{e}\,(k/t)\,2^t \rceil$ and $\varepsilon = 1/t$), and for all sufficiently large t we get

$$|HS| \leqslant 2^{4t\log(t)}.$$

3.5 Modified main construction

Let HS be the set from Proposition 5. As we already observed labellings of T correspond to shift sequences up to a cyclic shift of the whole matrix. For a labeling τ let $\sigma(\tau)$ be a shift sequence that is compatible with τ. Observe, that if τ is an ε-good labeling, then $\sigma(\tau)$ is ε-good shift sequence. Recall that we chose $s = \mathrm{e}\,(k/t)\,2^t$ so that if σ is an ε-good sequence for some consistently dominated coloring of M, then $M(\sigma)$ has at least k/t red columns. The modified main construction proceeds as follows.

`For every`
1. `labeling of the tree` $\tau \in HS$`,`
2. `and set of indices` $I \subset [s]$ `of size` k/t`,`

`add to` H_M `an edge build from all elements of the columns of` $M(\sigma(\tau))$ `with indices in` I`.`

By Proposition 5 for every consistently dominated coloring of M, at least one ε-good labeling τ is used in the construction. Then, for every such coloring, matrix M shifted according to $\sigma(\tau)$ has at least k/t red columns. As a consequence at least one of the edges of H_M is monochromatic.

Counting

Just like in the original construction, we have less than 2^{2t} choices for the subset of rows in the preliminary step. Then, in the modified main construction, we use every sequence of HS with every subset of k/t elements of $[s]$ to build an edge. The number of choices is smaller than

$$2^{4t\log(t)} \cdot \binom{s}{k/t} < 2^{4t\log(t)} \cdot \left(\frac{\mathrm{e}s}{k/t}\right)^{k/t}.$$

Substituting the value of s we obtain a value that is smaller than

$$2 \cdot 2^{4t\log(t)} \cdot \mathrm{e}^{2k/t} 2^k = 2^{1+4t\log(t)+(2k/t)\log(\mathrm{e})+k}.$$

The bound is multiplied by 2 to compensate for the ceiling in the definition of s. Taking into account preliminary choices of rows, the total number of edges is smaller than

$$2^{2t+1+4t\log(t)+(2k/t)\log(\mathrm{e})+k}.$$

For $t = (k/\log(k))^{1/2}$, the total number of edges becomes $2^{k+\Theta((k\log(k))^{1/2})}$.

References

1 József Beck. On 3-chromatic hypergraphs. *Discrete Mathematics*, 24(2):127–137, 1978. doi:10.1016/0012-365X(78)90191-7.

2 Danila D. Cherkashin and Jakub Kozik. A note on random greedy coloring of uniform hypergraphs. *Random Structures & Algorithms*, 47(3):407–413, 2015. doi:10.1002/rsa.20556.

3 Paul Erdős. On a combinatorial problem. II. *Acta Mathematica Academiae Scientiarum Hungaricae*, 15:445–447, 1964.

4 Ofer Gabber and Zvi Galil. Explicit constructions of linear-sized superconcentrators. *J. Comput. System Sci.*, 22(3):407–420, 1981. Special issued dedicated to Michael Machtey. doi:10.1016/0022-0000(81)90040-4.

5 Heidi Gebauer. On the construction of 3-chromatic hypergraphs with few edges. *Journal of Combinatorial Theory. Series A*, 120(7):1483–1490, 2013. doi:10.1016/j.jcta.2013.04.007.

6 Nathan Linial, Michael Luby, Michael Saks, and David Zuckerman. Efficient construction of a small hitting set for combinatorial rectangles in high dimension. *Combinatorica*, 17(2):215–234, 1997. doi:10.1007/BF01200907.

7 G. A. Margulis. Explicit constructions of expanders. *Problemy Peredači Informacii*, 9(4):71–80, 1973.

8 Jaikumar Radhakrishnan and Aravind Srinivasan. Improved bounds and algorithms for hypergraph 2-coloring. *Random Structures & Algorithms*, 16(1):4–32, 2000. doi:10.1002/(SICI)1098-2418(200001)16:1<4::AID-RSA2>3.3.CO;2-U.

SoS Certification for Symmetric Quadratic Functions and Its Connection to Constrained Boolean Hypercube Optimization

Adam Kurpisz
Department of Mathematics, ETH Zürich, Switzerland

Aaron Potechin
Department of Computer Science, University of Chicago, IL, USA

Elias Samuel Wirth
Institute of Mathematics, TU Berlin, Germany

Abstract

We study the rank of the Sum of Squares (SoS) hierarchy over the Boolean hypercube for Symmetric Quadratic Functions (SQFs) in n variables with roots placed in points $k-1$ and k. Functions of this type have played a central role in deepening the understanding of the performance of the SoS method for various unconstrained Boolean hypercube optimization problems, including the Max Cut problem. Recently, Lee, Prakash, de Wolf, and Yuen proved a lower bound on the SoS rank for SQFs of $\Omega(\sqrt{k(n-k)})$ and conjectured the lower bound of $\Omega(n)$ by similarity to a polynomial representation of the n-bit OR function.

Leveraging recent developments on Chebyshev polynomials, we refute the Lee–Prakash–de Wolf–Yuen conjecture and prove that the SoS rank for SQFs is at most $O(\sqrt{nk}\log(n))$.

We connect this result to two constrained Boolean hypercube optimization problems. First, we provide a degree $O(\sqrt{n})$ SoS certificate that matches the known SoS rank lower bound for an instance of Min Knapsack, a problem that was intensively studied in the literature. Second, we study an instance of the Set Cover problem for which Bienstock and Zuckerberg conjectured an SoS rank lower bound of $n/4$. We refute the Bienstock–Zuckerberg conjecture and provide a degree $O(\sqrt{n}\log(n))$ SoS certificate for this problem.

2012 ACM Subject Classification Theory of computation → Semidefinite programming; Theory of computation → Convex optimization

Keywords and phrases symmetric quadratic functions, SoS certificate, hypercube optimization, semidefinite programming

Digital Object Identifier 10.4230/LIPIcs.ICALP.2021.90

Category Track A: Algorithms, Complexity and Games

Funding *Adam Kurpisz*: supported by SNSF project PZ00P2_174117.
Aaron Potechin: supported in part by NSF grant CCF-2008920.

1 Introduction

Semialgebraic proof systems, also called certificates of nonnegativity, are systematic methods to prove nonnegativity of polynomials over semialgebraic sets. One of the most successful approaches for constructing theoretically efficient algorithms for polynomial optimization problems is the Sum of Squares (SoS) certificate [17, 38, 39, 46],

For a wide variety of combinatorial optimization problems, SoS provides the best available algorithms [1, 14, 5, 19, 34]. The strength of this method has also come to light for Max CSP [32] and problems in robust estimation [21], dictionary learning [3, 45], tensor completion and decomposition [4, 20, 41], and problems arising from statistical physics [13].

However, the SoS algorithm also admits certain weaknesses. It is known to struggle with solving certain combinatorial optimization problems, e.g., [7, 10, 18, 26, 49]. In a seminal example, Grigoriev showed that a $\Omega(n)$ degree SoS certificate is needed to detect a simple integrality argument for the Knapsack problem [15], see also [16, 24, 31]. A degree $n^{\Omega(\varepsilon)}$ SoS algorithm was proved to be unable to asymptotically certify an upper bound smaller than 2 times the optimal value for Sherrington-Kirkpatric Hamiltonian [23, 13]. Moreover, the degree $\Omega(\sqrt{n})$ SoS hierarchy was proved to have problems scheduling unit size jobs on a single machine to minimize the number of late jobs, see [27], even though the problem is known to be solvable in polynomial time using the Moore-Hodgson algorithm [36]. Finally, various examples where the SoS hierarchy fares very badly have been shown for the planted clique [2, 35] and Max CSP problems [22, 48].

The discrepancy between the excellent performance of the SoS hierarchy and its severe weaknesses has been studied extensively throughout the last decade. Thus, a natural question arises: what factors determine the difficulty of solving a problem for the SoS method?

A prominent example that was studied through the lens of this question is the Max Cut problem, which not only lies at the center of SoS research but was also one of the first problems for which lower bounds of the SoS rank were studied. Grigoriev proved that SoS needs at least degree $\lfloor \frac{n}{2} \rfloor$ to certify the size of the maximum cut in an odd clique of n vertices [15], for alternative proofs see also [16, 24, 31]. In a breakthrough paper nearly two decades later, Parrilo showed that the Grigoriev lower bound is tight by proving that every n-variate polynomial of degree 2, nonnegative over Boolean hypercube has an SoS certificate of degree at most $\lceil \frac{n}{2} \rceil$, see [12]. Subsequently, the analog of the results by Grigoriev and Parrilo for higher degree symmetric functions recently appeared in [25, 44], respectively.

Many of the problem instances with large lower bounds of the SoS rank target known limitations of the SoS method such as an issue with dealing with integrality constraints. Indeed, certifying the size of the maximum cut in a clique can be transformed into the problem of proving nonnegativity of the *Symmetric Quadratic Function* (SQF) of the form $q_{\lceil \frac{n}{2} \rceil}(\mathbf{x})$ over the Boolean hypercube, where, throughout this paper, $q_k : \{0,1\}^n \to \mathbb{R}$ is a multivariate polynomial of the form

$$q_k(\mathbf{x}) := (|\mathbf{x}| - k)(|\mathbf{x}| - k + 1). \tag{1.1}$$

The optimization of degree 2 polynomials over the Boolean hypercube plays a central role in Theoretical Computer Science. This claim is supported by the fact that high degree optimization problems attracted limited attention, especially since solving an NP-complete problem can be reduced in polynomial time to proving nonnegativity of a degree-4 even form [37]. Moreover, if an SQF has a complex root with a corresponding conjugate root, the polynomial is globally nonnegative and admits an SoS certificate of degree 2. Similarly, there exists an SoS certificate of nonnegativity of degree 2 for SQFs over the Boolean hypercube if the roots are real and placed outside the interval $[0, n]$. Hence, the only interesting case is when the roots are real and located within some interval $[k-1, k]$ for $k \in \{1, \ldots, n\}$.

Finding an SoS representation of the symmetric function q_k has gained significant attention in the SoS community. However, up to this day, the exact SoS rank for q_k is not known. The most recent result towards a characterization of the SoS rank of q_k provides a lower and upper bound of the SoS degree that approximates the function q_k with SoS polynomials in l_1 and l_∞ norm [33]. However, since finding an exact SoS certificate is at least as difficult as providing an approximate SoS representation, the result implies that for $k \geqslant 2$, q_k does not admit an SoS certificate of degree smaller than $\Omega\left(\sqrt{k(n-k)}\right)$. Moreover, in [33], Lee, Prakash, de Wolf, and Yuen conjectured that the lower bound of the SoS approximate representation with error

at most ε in the l_∞ norm is expected to be $\Omega\left(\sqrt{k(n-k)}+\sqrt{n\log(1/\varepsilon)}\right)$. They support the conjecture by arguing about similarity with approximating n-bit OR functions [40, 50]. This conjecture, if true, would imply a lower bound on the exact SoS certificate for SQFs of $\Omega(n)$, even for small, constant values of k. Proving this conjecture is left as an open question in [33]. In this paper, we refute the Lee–Prakash–de Wolf–Yuen (LPdWY) conjecture. We show that certifying SQFs is easier than representing n-bit OR functions. More specifically, we prove the following theorem.

▶ **Theorem 1.** *For any $k \in \{2,\dots,\lceil\frac{n}{2}\rceil\}$, there exists a degree $O(\sqrt{nk}\log(n))$ SoS certificate of nonnegativity for the Boolean function q_k as in* (1.1).

We motivate the research on the SoS degree of the SQFs q_k by connecting it to two combinatorial optimization problems. We first consider the instance of the Min Knapsack (MK) problem. For $P \geqslant 2$, the problem is defined as:

$$\text{MK:} \qquad \min \sum_{i\in[n]} x_i \qquad \text{s.t.} \quad \sum_{i\in[n]} x_i \geqslant \frac{1}{P}, \qquad \mathbf{x}\in\{0,1\}^n. \tag{1.2}$$

For $P=2$, the problem was previously considered by Cook and Dash [11]. They proved that the Lovasz-Schrijver hierarchy rank is n. For the Sherali-Adams hierarchy, Laurent proved that the rank is also equal to n and raised the open question to find the rank for the SoS hierarchy [30]. For $n=2$, they also proved that the SoS rank is 2, but the discussion of general n was left as an open question. Currently, it is known that the SoS rank of the MK problem falls within $\Omega(\sqrt{n})$ and $\lceil\frac{n+4\lceil\sqrt{n}\rceil}{2}\rceil$, see [28]. In this paper, we prove an upper bound on the SoS rank for the MK problem.

▶ **Theorem 2.** *The SoS rank for the MK problem is $\Omega(\sqrt{n}\log(P))$.*

The existing lower bound for general P (see Lemma 14 of [28]) is $\Omega(\sqrt{n\log(P)})$, so this is tight when P is constant, though for larger P there is a gap of $O(\sqrt{\log(P)})$.

We also consider the following instance of the Set Cover (SC) problem:

$$\text{SC:} \qquad \min \sum_{i\in[n]} x_i \qquad \text{s.t.} \quad \sum_{i\in[n]\setminus\{j\}} x_i \geqslant 1 \qquad \forall j\in[n], \qquad \mathbf{x}\in\{0,1\}^n. \tag{1.3}$$

This instance was considered in [8] and it is known that the SoS hierarchy cannot solve this problem with a degree smaller than $\Omega(\sqrt{n})$ [28]. In [8], Bienstock and Zuckerberg raised the question of what the actual SoS rank of this polytope is, conjecturing that, based on numerical experiments, the SoS rank is at least $\frac{n}{4}$. In this paper, using the SoS certificate for SQFs in Theorem 1, we refute the Bienstock–Zuckerberg conjecture and provide a nearly tight SoS rank for the SC problem:

▶ **Theorem 3.** *The SoS rank for the SC problem is at most $O(\sqrt{n}\log(n))$.*

2 Preliminaries

For $n\in\mathbb{N}$, let $[n]=\{1,\dots,n\}$. For $\mathbf{x}\in\mathbb{R}^n$, let $\mathbb{R}[\mathbf{x}]=\mathbb{R}[x_1,\dots,x_n]$ be the ring of *n-variate real polynomials*. For a set of polynomials $\mathcal{G}\subseteq\mathbb{R}[x]$, the corresponding *semialgebraic set* is

$$\mathcal{G}_+ \;:=\; \{\mathbf{x}\in\mathbb{R}^n \mid g(\mathbf{x})\geqslant 0 \text{ for all } g\in\mathcal{G}\}\subseteq\mathbb{R}^n.$$

Throughout this paper, we consider optimization problems on the Boolean hypercube $\{0,1\}^n$ and therefore, for $\mathcal{H}:=\{\pm(x_1^2-x_1),\dots,\pm(x_n^2-x_n)\}$, we assume that $\mathcal{G}$ is of the form

$$\mathcal{G} \;:=\; \mathcal{H}\cup\{g_1,\dots,g_m:\; g_i\in\mathbb{R}[\mathbf{x}] \text{ for all } \; i\in[m]\},$$

where $m \in \mathbb{N}_{>0}$. This implies that $\mathcal{G}_+ \subseteq \{0,1\}^n$. Moreover, define the *cone of nonnegative polynomials with respect to a given semialgebraic set, $\mathcal{G}_+$,* as

$$\mathcal{K}(\mathcal{G}_+) \;:=\; \{f \in \mathbb{R}[\mathbf{x}] \mid f(\mathbf{x}) \geqslant 0 \text{ for all } \mathbf{x} \in \mathcal{G}_+\}.$$

For given $f \in \mathbb{R}[\mathbf{x}]$ and $\mathcal{G} \subseteq \mathbb{R}[\mathbf{x}]$, define the corresponding *Constrained Polynomial Optimization Problem* (CPOP) as

$$f^* \;:=\; \min\{f(\mathbf{x}) \mid \mathbf{x} \in \mathcal{G}_+\} \;=\; \max\{\lambda \in \mathbb{R} \mid f - \lambda \in \mathcal{K}(\mathcal{G}_+)\}.$$

Generally, since CPOP is NP-hard, it is desirable to find a proper subset that is a good inner approximation of $\mathcal{K}(\mathcal{G}_+)$ such that the corresponding program is computationally *tractable*.

The *SoS method* approximates the cone $\mathcal{K}(\mathcal{G}_+)$ by using the set of *sum of square polynomials*. We define the set of finite sum of squares polynomials as $\Sigma := \{s \mid s = \sum_{i=1}^k s_i^2, s_i \in \mathbb{R}[\mathbf{x}]\ \forall i \in [k], k \in \mathbb{N}_{>0}\}$ and let $\Sigma_{n,d} := \{s \mid s = \sum_{i=1}^k s_i^2, s_i \in \mathbb{R}[\mathbf{x}] \wedge \deg(s_i) \leqslant d\ \forall i \in [k], k \in \mathbb{N}_{>0}\}$ denote the polynomials which are sums of squares of polynomials of degree at most d. We define the *hierarchy of certificates of nonnegativity depending on $d, n \in \mathbb{N}$* as

$$\Sigma_{n,d}^{\mathcal{G}} := \left\{ s_0 + \sum_{i=1}^m s_i g_i \mid s_i \in \Sigma_{n,d},\ g_i \in \mathcal{G}\ \forall i \in [m] \text{ and } s_0 \in \Sigma_{n,2\left\lceil \frac{2d+\deg(\mathcal{G})}{2} \right\rceil} \right\},$$

where $\deg(\mathcal{G}) = \max\{\deg(g) \mid g \in \mathcal{G}\}$. The *degree d SoS certificate* for f being nonnegative over $\mathcal{G}_+$ is $f \in \Sigma_{n,d}^{\mathcal{G}}$. Moreover, throughout the paper we say that a multivariate polynomial f is *a degree d SoS modulo Boolean axioms* if $f \in \Sigma_{n,d}^{\mathcal{H}}$. The *degree d SoS program* for CPOP is

$$f_\Sigma^d \;:=\; \max\{\lambda \in \mathbb{R} \mid f - \lambda \in \Sigma_{n,d}^{\mathcal{G}}\} \tag{2.1}$$

and is called *exact* if $f_\Sigma^d = f^*$. The smallest degree d such that the degree d SoS program is exact is called the *SoS rank*. Over the Boolean hypercube, the degree d SoS program can be solved via a *semidefinite program* (SDP) of size $O(m \sum_{k=0}^d \binom{n}{k})$. Moreover, the degree n SoS program is exact, see, e.g., [6, 29, 30].

Throughout this paper, we often encounter the following type of multivariate polynomials.

▶ **Definition 4.** *A polynomial $f : \{0,1\}^n \to \mathbb{R}$ is* symmetric *if there exists a univariate polynomial $\tilde{f} : \mathbb{R} \to \mathbb{R}$ such that $f(\mathbf{x}) = \tilde{f}\left(\sum_{i=1}^n x_i\right)$ for all $\mathbf{x} \in \{0,1\}^n$.*

With this in mind, let $|\mathbf{x}| := \sum_{i=1}^n x_i$ for any $\mathbf{x} \in \{0,1\}^n$. To prove SoS rank upper bounds, we consider symmetric multivariate polynomials over $\{0,1\}^n$ as univariate polynomials over $[0,n]$ and apply one of the many results on SoS certificates for univariate polynomials.

▶ Remark 5. Throughout this paper, we make frequent use of the fact that SoS certificates for polynomials over $[0,n]$ translate to SoS certificates for symmetric polynomials over $\{0,1\}^n$. More formally, if a univariate polynomial $\tilde{f} : \mathbb{R} \to \mathbb{R}$ has an univariate SoS certificate of degree d on $[0,n]$, then the multivariate polynomial $f : \{0,1\}^n \to \mathbb{R}$ such that $f(\mathbf{x}) := \tilde{f}(|\mathbf{x}|)$ has a degree d SoS certificate of nonnegativity over the Boolean hypercube.

In this paper, we use the following theorem to prove the SoS rank for univariate polynomials.

▶ **Theorem 6** ([9, Theorem 3.72])**.** *Let $a < b$. Then the univariate polynomial $p(x)$ is nonnegative on $[a,b]$ if and only if it can be written as*

$$\begin{cases} p(x) = s(x) + (x-a)(b-x) \cdot t(x) & \text{if } \deg(p) \text{ is even,} \\ p(x) = (x-a) \cdot s(x) + (b-x) \cdot t(x) & \text{if } \deg(p) \text{ is odd,} \end{cases}$$

where s, t are sum of squares. In the first case, we have $\deg(p) = 2d$, $\deg(s) \leqslant 2d$, and $\deg(t) \leqslant 2d-2$. In the second, $\deg(p) = 2d+1$, $\deg(s) \leqslant 2d$, and $\deg(t) \leqslant 2d-2$.

Finally, throughout the paper we use degree-d Chebyshev polynomials of the first type, which were used is several applications for bounds of sum of squares ranks, i.e., [28, 47, 42]. We frequently use the following lemma.

▶ **Lemma 7.** *Let $n, d \in \mathbb{N}$ such that $d \leqslant n$. Then,*

1. *For all $c \in [0, n]$, $T_d^2\left(-1-\frac{c}{n}\right) \geqslant \frac{1}{4}\left(-1-\sqrt{\frac{2c}{n}}\right)^{2d}$ and $T_d^2\left(-1-\frac{c}{n}\right) \leqslant \left(-1-2\sqrt{\frac{2c}{n}}\right)^{2d}$. Moreover, for constant c and n big enough, $T_d^2\left(-1-\frac{c}{n}\right) \leqslant \left(-1-\sqrt{\frac{2c+1}{n}}\right)^{2d}$.*
2. *For all $c \in (n, \infty)$, $T_d^2\left(-1-\frac{c}{n}\right) \leqslant \left(-1-3\frac{c}{n}\right)^{2d}$.*

Proof. It holds that:

1. Consider the characterization of Chebyshev polynomials given in [43, Equation 1.12]: $T_d(x) = \frac{1}{2}\left(\left(x-\sqrt{x^2-1}\right)^d + \left(\sqrt{x^2-1}+x\right)^d\right)$. For $x = -1-\frac{c}{n}$ and $c \in [0, n]$, we have $T_d^2\left(-1-\frac{c}{n}\right) \geqslant \frac{1}{4}\left(\left(-1-\frac{c}{n}\right)-\sqrt{\left(-1-\frac{c}{n}\right)^2-1}\right)^{2d} \geqslant \frac{1}{4}\left(-1-\sqrt{\frac{2c}{n}}\right)^{2d}$ and
$$\begin{aligned} T_d^2\left(-1-\frac{c}{n}\right) &\leqslant \left(\left(-1-\frac{c}{n}\right)-\sqrt{\left(-1-\frac{c}{n}\right)^2-1}\right)^{2d} \\ &\leqslant \left(\left(-1-\frac{c}{n}\right)-\sqrt{\frac{2c}{n}+\frac{c^2}{n^2}}\right)^{2d} \\ &\leqslant \left(-1-\sqrt{\frac{c}{n}}-\sqrt{\frac{2c}{n}+\frac{c}{n}}\right)^{2d} \leqslant \left(-1-2\sqrt{\frac{2c}{n}}\right)^{2d}. \end{aligned} \tag{2.2}$$
Moreover, we have $T_d^2\left(-1-\frac{c}{n}\right) \leqslant \left(\left(-1-\frac{c}{n}\right)-\sqrt{\left(-1-\frac{c}{n}\right)^2-1}\right)^{2d} \leqslant \left(-1-\sqrt{\frac{2c+1}{n}}\right)^{2d}$, where the last inequality holds for n large compared to c.
2. For $x = -1-\frac{c}{n}$ and $c \in (n, \infty)$, we have
$$\begin{aligned} T_d^2\left(-1-\frac{c}{n}\right) &\leqslant \left(\left(-1-\frac{c}{n}\right)-\sqrt{\left(-1-\frac{c}{n}\right)^2-1}\right)^{2d} \\ &\leqslant \left(\left(-1-\frac{c}{n}\right)-\sqrt{\frac{2c}{n}+\frac{c^2}{n^2}}\right)^{2d} \\ &\leqslant \left(-1-\frac{c}{n}-\sqrt{\frac{2c^2}{n^2}+\frac{c^2}{n^2}}\right)^{2d} \leqslant \left(-1-3\frac{c}{n}\right)^{2d}. \end{aligned} \tag{2.3}$$

◀

3 SoS rank for SQFs

In this section, we refute the LPdWY conjecture stated in [33] by proving Theorem 1. To prove Theorem 1, it is sufficient to prove the following theorem.

▶ **Theorem 8.** *For all $n \in \mathbb{N}$ and all $k \in [n]$, there exists a polynomial $s(x)$ of degree $O(\sqrt{kn}\log(n))$ such that*

1. *$s\left(\sum_{i=1}^n x_i\right)$ is a sum of squares (modulo the Boolean axioms).*
2. *For all $x \in [0, n]$, $(x-k+1)(x-k) - s(x) \geqslant 0$.*

Indeed, by Theorem 8 and Theorem 6, there exist sum of squares polynomials s, s_1 and s_2 of degree $O(\sqrt{kn}\log(n))$ s.t.

$$(x-k+1)(x-k) = s(x) + s_1(x) + s_2(x)x(n-x).$$

We now make the following observations:

1. By Theorem 8, $s(\sum_{i=1}^n x_i)$ is a sum of squares polynomial modulo the Boolean axioms.
2. $s_1(\sum_{i=1}^n x_i)$, $s_2(\sum_{i=1}^n x_i)$ are sum of squares polynomials.
3. $\sum_{i=1}^n x_i = \sum_{i=1}^n x_i^2 - \sum_{i=1}^n \left(x_i^2 - x_i\right)$ is a sum of squares polynomial modulo the Boolean axioms.
4. $n - \sum_{i=1}^n x_i = \sum_{i=1}^n (1-x_i) = \sum_{i=1}^n \left((x_i-1)^2 - \left(x_i^2 - x_i\right)\right)$ is a sum of squares polynomial modulo the Boolean axioms.

Putting everything together, the multivariate polynomial $q_k(\mathbf{x})$ has an $O(\sqrt{kn}\log(n))$ SoS certificate modulo the Boolean axioms of the form

$$q_k(\mathbf{x}) = s\left(\sum_{i=1}^n x_i\right) + s_1\left(\sum_{i=1}^n x_i\right) + s_2\left(\sum_{i=1}^n x_i\right)\left(\sum_{i=1}^n x_i\right)\left(n - \sum_{i=1}^n x_i\right).$$

Before we prove Theorem 8, we make the following observation which shows that our upper bound for $q_k(x)$ applies for any symmetric quadratic function with roots in $[k-1,k]$.

▶ **Corollary 9.** *For any $k \in \{1,\dots,\lceil n/2\rceil\}$ and any $a \leqslant b \in [k-1,k]$, a polynomial $f_k := (x-a)(x-b)$ admits an SoS certificate over the Boolean hypercube of degree at most the degree of an SoS certificate over the Boolean hypercube for polynomial q_k.*

Proof. We have $f_k(x) \geqslant \left((k-a)(b-k+1) + (k-b)(a-k+1)\right) q_k(x)$ as

$$\begin{aligned}
&(|x|-a)(|x|-b)\\
&= \left((k-a)(|x|-k+1) + (a-k+1)(|x|-k)\right)\left((k-b)(|x|-k+1) + (b-k+1)(|x|-k)\right)\\
&= (k-a)(k-b)(|x|-k+1)^2 + (a-k+1)(b-k+1)(|x|-k)^2\\
&+ \left((k-a)(b-k+1) + (k-b)(a-k+1)\right)(|x|-k+1)(x-|k|)
\end{aligned}$$

and invoke Theorem 1 to conclude the proof. ◀

3.1 Proof of Theorem 8

We construct $s(x)$ in two steps. We first construct a polynomial $s_1(x)$ which is a sum of squares (modulo the Boolean axioms), is less than or equal to $(x-k+1)(x-k)$ on the interval $[0,2k-1]$, and is not too large on the interval $[2k-1,n]$. We then construct a polynomial $s_2(x)$ which is a sum of squares, is less than or equal to 1 on the intervals $[0,k-1]$ and $[k,2k-1]$, is greater than or equal to 1 on the interval $[k-1,k]$, and is very small on the interval $[2k-1,n]$. We then take $s(x) = s_1(x)s_2(x)$. More precisely, we have the following conditions on s_1 and s_2:

1. $s_1\left(\sum_{i=1}^n x_i\right)$ is a sum of squares (modulo the Boolean axioms) and $s_2(x)$ is a sum of squares.
2. For all $x \in [k-1,k]$, $\frac{s_1(x)}{(x-k+1)(x-k)} \geqslant 1$ and $s_2(x) \geqslant 1$.
3. For all $x \in [0,k-1] \cup [k,2k-1]$, $\frac{s_1(x)}{(x-k+1)(x-k)} \leqslant 1$ and $s_2(x) \leqslant 1$.
4. For all $x \in [2k-1,n]$, $\left|\frac{s_1(x)}{(x-k+1)(x-k)}\right| \leqslant n^{40k}$ and $s_2(x) \leqslant n^{-40k}$.
5. $s_1(x)$ has degree $O(k)$ and $s_2(x)$ has degree $O(\sqrt{nk}\log(n))$.

▶ **Proposition 10.** *If $s_1(x)$ and $s_2(x)$ satisfy the above conditions and we take $s(x) = s_1(x)s_2(x)$ then $s\left(\sum_{i=1}^n x_i\right)$ is a sum of squares (modulo the Boolean axioms) and for all $x \in [0, n]$, $(x-k+1)(x-k) - s(x) \geqslant 0$.*

Proof. We make the following observations:

1. Since $s_1\left(\sum_{i=1}^n x_i\right)$ is a sum of squares (modulo the Boolean axioms) and $s_2(x)$ is a sum of squares, the product $s\left(\sum_{i=1}^n x_i\right) = s_1\left(\sum_{i=1}^n x_i\right) s_2\left(\sum_{i=1}^n x_i\right)$ is a sum of squares (modulo the Boolean axioms).
2. For all $x \in [0, k-1] \cup [k, 2k-1]$, since $(x-k+1)(x-k) \geqslant 0$, $\frac{s_1(x)}{(x-k+1)(x-k)} \leqslant 1$, and $0 \leqslant s_2(x) \leqslant 1$,
$$(x-k+1)(x-k) - s(x) = (x-k+1)(x-k)\left(1 - s_2(x)\frac{s_1(x)}{(x-k+1)(x-k)}\right) \geqslant 0.$$
3. For all $x \in [k-1, k]$, since $(x-k+1)(x-k) \leqslant 0$, $\frac{s_1(x)}{(x-k+1)(x-k)} \geqslant 1$, and $s_2(x) \geqslant 1$,
$$(x-k+1)(x-k) - s(x) = (x-k+1)(x-k)\left(1 - s_2(x)\frac{s_1(x)}{(x-k+1)(x-k)}\right) \geqslant 0.$$
4. For all $x \in [2k-1, n]$, since $(x-k+1)(x-k) \geqslant 0$, $\left|\frac{s_1(x)}{(x-k+1)(x-k)}\right| \leqslant n^{40k}$ and $|s_2(x)| \leqslant n^{-40k}$,
$$(x-k+1)(x-k) - s(x) = (x-k+1)(x-k)\left(1 - s_2(x)\frac{s_1(x)}{(x-k+1)(x-k)}\right) \geqslant 0. \quad ◀$$

Thus, we have an SoS proof of degree $O(\sqrt{kn}log(n))$ that $(|x|-k+1)(|x|-k) \geqslant 0$.

3.1.1 Constructing the polynomial $s_1(x)$

We now construct the polynomial $s_1(x)$.

▶ **Lemma 11.** *For $n \in \mathbb{N}$ and all $k \in [n]$, there exists a polynomial $s_1(x)$ such that*

1. *$s_1\left(\sum_{i=1}^n x_i\right)$ has a degree $O(k)$ sum of squares (modulo the Boolean axioms) certificate.*
2. *For all $x \in [k-1, k]$, $\frac{s_1(x)}{(x-k+1)(x-k)} \geqslant 1$.*
3. *For all $x \in [0, k-1] \cup [k, 2k-1]$, $\frac{s_1(x)}{(x-k+1)(x-k)} \leqslant 1$.*
4. *For all $x \in [2k-1, n]$, $\left|\frac{s_1(x)}{(x-k+1)(x-k)}\right| \leqslant n^{40k}$.*

Proof. For $k = 1$, we can take $s_1(x) = x(x-1)$ so we can assume that $n \geqslant k \geqslant 2$. For $k \geqslant 2$, we use the following construction.[1]

▶ **Definition 12.** *For all natural numbers $k \geqslant 2$, define $g_k(x)$ to be the polynomial*
$$g_k(x) = x^{16k}(x-2k+1)^{16k} \prod_{i \in \{0,\ldots,2k-1\} \setminus \{k-1,k\}} (x-i).$$

▶ **Definition 13.** *Given a natural number n and $k \in \{2, 3, \ldots, n\}$, we define $s_1(x)$ as follows:*

1. *If k is odd, then we define $s_1(x) = \frac{g_k(x)}{g_k(k-1)}(x-k+1)(x-k)$.*
2. *If k is even, then we define $s_1(x) = -\frac{g_k(x)(x+1)(x-2k)}{g_k(k-1)k(k+1)}(x-k+1)(x-k)$.*

[1] Definitions 12 and 13 are only used in the current section, Section 3.

We verify the desired properties. We first show that $s_1\left(\sum_{i=1}^n x_i\right)$ is a sum of squares (modulo the Boolean axioms). If k is odd, then since $g_k(k-1) > 0$, $\prod_{i=0}^{2k-1}\left(\left(\sum_{i=1}^n x_i\right) - i\right)$ is a sum of squares (modulo the Boolean axioms), and by [33, Lemma 4.4],

$$s_1\left(\sum_{i=1}^n x_i\right) = \frac{\left(\sum_{i=1}^n x_i\right)^{16k}\left(\left(\sum_{i=1}^n x_i\right) - 2k + 1\right)^{16k}}{g_k(k-1)} \prod_{i=0}^{2k-1}\left(\left(\sum_{i=1}^n x_i\right) - i\right)$$

is a sum of squares (modulo the Boolean axioms). If k is even, then since $g_k(k-1) < 0$, $\left(\sum_{i=1}^n x_i\right) + 1$ and $\prod_{i=0}^{2k}\left(\left(\sum_{i=1}^n x_i\right) - i\right)$ are sum of squares (modulo the Boolean axioms),

$$s_1\left(\sum_{i=1}^n x_i\right) = -\frac{\left(\sum_{i=1}^n x_i\right)^{16k}\left(\left(\sum_{i=1}^n x_i\right) - 2k + 1\right)^{16k}}{g_k(k-1)k(k+1)}\left(\left(\sum_{i=1}^n x_i\right) + 1\right)\prod_{i=0}^{2k}\left(\left(\sum_{i=1}^n x_i\right) - i\right)$$

is a sum of squares (modulo the Boolean axioms). Finally, to argue about the degree, note that by [33, Lemma 4.4], $\prod_{i=0}^{2k-1}\left(\left(\sum_{i=1}^n x_i\right) - i\right)$ has a sum of squares (modulo the Boolean axioms) certificate of degree $2k$ and thus, for all k, $s_1\left(\sum_{i=1}^n x_i\right)$ has a sum of squares (modulo the Boolean axioms) certificate of degree $O(k)$.

For the fourth property, observe that for $x \in [0, n]$, every term in the numerator (except for $(x+1)$ when k is even) has magnitude at most n, every term in the denominator has magnitude at least 1, and there are less than $40k$ terms in the numerator.

The second and third properties follow immediately from the following lemma.

▶ **Lemma 14.** *For all natural numbers $k \geqslant 2$, $g_k(x)$ satisfies the following properties:*

1. *For all $x \in [0, 2k-1]$, $g_k(2k-1-x) = g_k(x)$.*
2. *For all $x \in [k-1, k]$, $\frac{g_k(x)}{g_k(k-1)} \geqslant 1$.*
3. *For all $x \in [0, k-1] \cup [k, 2k-1]$, $\left|\frac{g_k(x)}{g_k(k-1)}\right| \leqslant 1$.*

Proof. Since the first and second properties hold for every term in the product $g_k(x) = (-1)^{k-1}\left(x(x-2k+1)\right)^{16k}\left(\prod_{i=0}^{k-2}(x-i)(2k-1-x-i)\right)$, they hold for $g_k(x)$ as well.

By symmetry, it suffices to show the third property for $x \in [0, k-1]$. For $x \in \{0, 1, \ldots, k-2\}$, $g_k(x) = 0$ and for $x \in (k-2, k-1)$, the third property holds for every term in this product, so it holds for $g_k(x)$ as well. To show that the third property holds for $x \in [0, k-2]\backslash\{0, 1, \ldots, k-2\}$, we compare $g_k(x-m)$ and $g_k(x)$, where $x \in (k-2, k-1)$ and $m \in \{0, 1, \ldots, k-2\}$. For this, we decompose $g_k(x)$ as $g_k(x) = a_k(x)b_k(x)^{16k}$, where $a_k(x) = \prod_{i \in \{0,\ldots,2k-1\}\backslash\{k-1,k\}}(x-i)$ and $b_k(x) = x(2k-1-x)$.

▶ **Lemma 15.** *Let $a_k(x) = \prod_{i \in \{0,\ldots,2k-1\}\backslash\{k-1,k\}}(x-i) = \left(\prod_{i=0}^{k-2}(x-i)\right)\left(\prod_{i=k+1}^{2k-1}(x-i)\right)$. For all $x \in (k-2, k-1)$ and all $m \in \{1, \ldots, k-2\}$, $\left|\frac{a_k(x-m)}{a_k(x)}\right| \leqslant e^{\frac{16m^2}{k}}$.*

Proof. Observe that

$$\begin{aligned}
\left|\frac{a_k(x-m)}{a_k(x)}\right| &= \left|\frac{\prod_{j=1}^m (x-k+2-j)}{\prod_{j=1}^m (x+1-j)} \cdot \frac{\prod_{j=1}^m (x-2k+1-j)}{\prod_{j=1}^m (x-k-j)}\right| \\
&= \left|\frac{\prod_{j=1}^m (k-2-x+j)}{\prod_{j=1}^m (k-x+j)} \cdot \frac{\prod_{j=1}^m (2k-x-1+j)}{\prod_{j=1}^m (x-m+j)}\right| \\
&\leqslant \left|\prod_{j=1}^m \left(\frac{k+1+j}{k-2-m+j}\right)\right|.
\end{aligned}$$

We distinguish between two cases.

1. If $m \leqslant \frac{3k}{4} - 1$, observe that

$$\left|\prod_{j=1}^{m}\left(\frac{k+1+j}{k-2-m+j}\right)\right| = \prod_{j=1}^{m}\left(1+\frac{m+3}{k-2-m+j}\right)$$
$$\leqslant \prod_{j=1}^{m}\left(1+\frac{m+3}{k-m-1}\right) \leqslant \prod_{j=1}^{m} e^{\frac{m+3}{k-m-1}} = e^{\frac{m(m+3)}{(k-m+1)}} \leqslant e^{\frac{16m^2}{k}}.$$

2. If $m > \frac{3k}{4} - 1$, then $m \geqslant \frac{3k}{4} - \frac{3}{4} \geqslant \frac{3k}{8}$ (as $k \geqslant 2$). Thus,

$$\left|\prod_{j=1}^{m}\left(\frac{k+1+j}{k-2-m+j}\right)\right| \leqslant \prod_{j=1}^{k-2}\left(\frac{k+1+j}{j}\right) = \frac{(2k-1)!}{(k-2)!(k+1)!} \leqslant 2^{2k-1} \leqslant e^{\frac{16m^2}{k}}. \quad \blacktriangleleft$$

▶ **Lemma 16.** *Let $b_k(x) = x(2k-1-x)$. For $x \in (k-2, k-1)$ and $m \in [k-2]$, $\left|\frac{b_k(x-m)}{b_k(x)}\right| \leqslant e^{-\frac{m^2}{k^2}}$.*

Proof. Observe that

$$\frac{b_k(x-m)}{b_k(x)} = \frac{(x-m)(2k-1+m-x)}{x(2k-1-x)} = \frac{x(2k-1-x)-(2k-1-2x)m-m^2}{x(2k-1-x)}$$
$$\leqslant 1 - \frac{m^2}{x(2k-1-x)} \leqslant 1 - \frac{m^2}{k^2} \leqslant e^{-\frac{m^2}{k^2}}. \quad \blacktriangleleft$$

▶ **Corollary 17.** *For all $x \in (k-2, k-1)$ and $m \in \{1, \ldots, k-2\}$, $\left|\frac{g_k(x-m)}{g_k(x)}\right| \leqslant 1$.*

Proof. By Lemmas 15 and 16, $\left|\frac{g_k(x-m)}{g_k(x)}\right| = \left|\frac{a_k(x-m)}{a_k(x)}\right| \left|\frac{b_k(x-m)}{b_k(x)}\right|^{16k} \leqslant e^{\frac{16m^2}{k}} \left(e^{-\frac{m^2}{k^2}}\right)^{16k} = 1.$ $\blacktriangleleft$

$\blacktriangleleft$

$\blacktriangleleft$

3.1.2 Constructing the polynomial $s_2(x)$

We now construct the polynomial $s_2(x)$.

▶ **Lemma 18.** *For all $n \in \mathbb{N}$ and all $k \in [n]$, there exists a polynomial $s_2(x)$ of degree $O(\sqrt{kn}log(n))$ satisfying the following properties:*

1. *$s_2(x)$ is a sum of squares.*
2. *For all $x \in [k-1, k]$, $s_2(x) \geqslant 1$.*
3. *For all $x \in [0, k-1] \cup [k, 2k-1]$, $s_2(x) \leqslant 1$.*
4. *For all $x \in [2k-1, n]$, $s_2(x) \leqslant n^{-40k}$.*

Proof.

▶ **Lemma 19.** *For $C := e^{8\sqrt{3}}$ and $k \in \{0, \ldots, \lceil n/2 \rceil\}$, $H_k = T^2_{\sqrt{\frac{n}{k}}}\left(2\frac{x}{n} - 1 - 2\frac{2k-1}{n}\right)$ satisfies the following properties:*

1. *For all $x \in [2k-1, n]$, $H_k(x) \leqslant 1$.*
2. *For all $k \in [0, 2k-1]$, $H_k'(x) < 0$.*
3. *$H_k(0) \leqslant C$.*
4. *$H_k(k) \geqslant 1.5$.*

Proof. Note that $H_k(x) = T^2_{\sqrt{\frac{n}{k}}}\left(2\frac{x}{n} - 1 - 2\frac{2k-1}{n}\right)$. Hence, $H_k(2k-1) = T^2_{\sqrt{\frac{n}{k}}}(-1) = 1$ and $H_k(n) = T^2_{\sqrt{\frac{n}{k}}}\left(1 - 2\frac{2k-1}{n}\right) \leqslant 1$, which implies the first property. We prove Properties (2) and (3). By Lemma 7, for k such that $4k-2 \leqslant n$, we have

$$H_k(0) = T^2_{\sqrt{\frac{n}{k}}}\left(-1 - \frac{4k-2}{n}\right) \leqslant \left(1 + \sqrt{\frac{32k-16}{n}}\right)^{2\sqrt{\frac{n}{k}}} \leqslant e^{2\sqrt{\frac{32k-16}{k}}} \leqslant e^{8\sqrt{3}}$$

and for k such that $4k-2 \geqslant n$, by Lemma 7, for $c \geqslant n$, we have

$$H_k(0) = T^2_{\sqrt{\frac{n}{k}}}\left(-1 - \frac{4k-2}{n}\right) \leqslant \left(1 + \frac{12k}{n}\right)^{2\sqrt{\frac{n}{k}}} \leqslant \left(1 + \sqrt{\frac{12k}{n}}\right)^{4\sqrt{\frac{n}{k}}} \leqslant e^{4\sqrt{\frac{12k}{k}}} \leqslant e^{8\sqrt{3}}.$$

Moreover, by Lemma 7 we have

$$H_k(k) = T^2_{\sqrt{\frac{n}{k}}}\left(-1 - \frac{2k-2}{n}\right) \geqslant \frac{1}{4}\left(1 + \sqrt{\frac{4k-4}{n}}\right)^{2\sqrt{\frac{n}{k}}} \geqslant \frac{1}{4}\left(1 + \sqrt{\frac{2k}{n}}\right)^{2\sqrt{\frac{n}{k}}},$$

where the last inequality holds because $k \geqslant 2$. Finally, since $n \geqslant 2k$,

$$\frac{1}{4}\left(1 + \sqrt{\frac{2k}{n}}\right)^{2\sqrt{\frac{n}{k}}} \geqslant \frac{1}{4}2^{2\sqrt{\frac{n}{k}}\sqrt{\frac{2k}{n}}} = \frac{1}{4}2^{2\sqrt{2}} \geqslant 1.5.$$ ◀

▶ **Lemma 20.** *For any constants a, b, C such that $1.5 \leqslant a < b < C$, there is a sum of squares polynomial $p_{a,b,C}(x)$ of degree at most $8\lceil C^2 \rceil$ such that the following hold:*

1. *For all $x \in [a, b]$, $p_{a,b,C}(x) \geqslant 1$.*
2. *For all $x \in [0, 1]$, $|p_{a,b,C}(x)| \leqslant \frac{1}{2}$.*
3. *For all $x \in [0, a] \cup [b, C]$, $|p_{a,b,C}(x)| \leqslant 1$.*

Proof. We can take the polynomial

$$p_{a,b,C}(x) = \left(1 - \frac{(x-a)(x-b)}{C^2}\right)^{4\lceil C^2 \rceil}.$$

We now make the following observations:

1. For all $x \in [a, b]$, $1 - \frac{(x-a)(x-b)}{C^2} \geqslant 1$ so $p_{a,b,C}(x) \geqslant 1$.
2. For all $x \in [0, 1]$, $|1 - \frac{(x-a)(x-b)}{C^2}| \leqslant 1 - \frac{1}{4C^2}$ so $|p_{a,b,C}(x)| \leqslant \left(1 - \frac{1}{4C^2}\right)^{4\lceil C^2 \rceil} \leqslant \frac{1}{2}$.
3. For all $x \in [0, a] \cup [b, C]$, $|1 - \frac{(x-a)(x-b)}{C^2}| \leqslant 1$ so $|p_{a,b,C}(x)| \leqslant 1$. ◀

We construct the polynomial $s_2(x)$. For $k \in \{2, \ldots, \lceil n/2 \rceil\}$, let $s_2(x) := p_{a,b,C}\left(H_k(x)\right)^{40\lceil k \log(n) \rceil}$, where $a = H_k(k)$, $b = H_k(k-1)$, and $C = e^{8\sqrt{3}}$ is the constant given by Lemma 19.

▶ **Lemma 21.** *For any $k \in \{2, \ldots, \lceil n/2 \rceil\}$, $s_2(x)$ satisfies the properties in Lemma 18.*

Proof. We make the following observations:

1. For all $x \in [0, k-1] \cup [k, 2k-1]$, $H_k(x) \in [0, H_k(k)] \cup [H_k(k-1), C]$ so $|p_{a,b,C}(H_k(x))| \leqslant 1$ and thus $s_2(x) = p_{a,b,C}\left(H_k(x)\right)^{40\lceil k \log(n) \rceil} \leqslant 1$.
2. For all $x \in [k-1, k]$, $H_k(x) \in [H_k(k), H_k(k-1)]$ so $p_{a,b,C}(H_k(x)) \geqslant 1$ and thus $s_2(x) = p_{a,b,C}\left(H_k(x)\right)^{40\lceil k \log(n) \rceil} \geqslant 1$.
3. For all $x \in [2k-1, n]$, $H_k(x) \in [0, 1]$ so $|p_{a,b,C}(H_k(x))| \leqslant 1$ and thus, $s_2(x) = p_{a,b,C}\left(H_k(x)\right)^{40\lceil k \log(n) \rceil} \leqslant n^{-40k}$. ◀

◀

4 SoS rank upper bound for the MK problem via SQF certification

In this section, we prove an upper bound of $O(\sqrt{n}\log(P))$ on the SoS rank for the MK problem, which, together with the lower bound presented in [28], constitutes proof of Theorem 2.

We first discuss the necessary properties a candidate SoS certificate for the MK problem has to satisfy. A degree d SoS certificate for the MK problem is of the form $\sum_{i\in[n]} x_i - 1 = s_0(\mathbf{x}) + s_1(\mathbf{x})\left(\sum_{i\in[n]} x_i - \frac{1}{P}\right)$, where s_0, s_1 are SoS polynomials of degree $2d+2$ and $2d$, respectively. Through permutation of indices, the existence of an SoS certificate for the MK problem implies the existence of an SoS certificate such that s_1 is symmetric, that is, there exists $\tilde{s}_1 : \mathbb{R} \to \mathbb{R}$ such that $s_1(\mathbf{x}) = \tilde{s}_1(|\mathbf{x}|)$ for all $\mathbf{x} \in \{0,1\}^n$. Since s_0 is globally nonnegative, $\tilde{s}_1$ needs to satisfy

$$|\mathbf{x}| - 1 \geqslant \tilde{s}_1(|\mathbf{x}|)\left(|\mathbf{x}| - \frac{1}{P}\right) \qquad \text{for all } \mathbf{x} \in \{0,1\}^n. \tag{4.1}$$

Thus, $\tilde{s}_1(0) \geqslant P$, $\tilde{s}_1(1) = 0$, and $\tilde{s}_1(x) \leqslant \frac{x-1}{x-\frac{1}{P}}$ for $x \in \{2,\dots,n\}$.

We will construct a sum of squares polynomial $\tilde{s}_1$ which satisfies the following slightly stronger conditions:

1. $\tilde{s}_1(0) > P$
2. For all $x \in [1,2]$, $\tilde{s}_1(x) \leqslant \frac{x-1}{2}$
3. For all $x \in [2,n]$, $\tilde{s}_1(x) \leqslant \frac{1}{2}$

We will then observe that these conditions imply that

$$\tilde{s}_0(|x|) = |x| - 1 - \tilde{s}_1(|x|)\left(|x| - \frac{1}{P}\right)$$

is positive for all $x \in \{0\} \cup (1,n]$ which is sufficient to show that $\tilde{s}_0(x)$ is a sum of squares modulo the Boolean constraints.

A polynomial $T_{2\sqrt{n}}(\frac{x-1+r_0}{n} - 1)$, where r_0 is the smallest root of the polynomial $T_{2\sqrt{n}}(\frac{x}{n} - 1)$, which for $P = 2$ satisfies similar requirements was constructed in [28, Lemma 15] using properties of Chebyshev polynomials.

To obtain our polynomial $\tilde{s}_1(x)$, we generalize this construction using three parameters, the degree d of the Chebyshev polynomial, a scaling factor α, and an even power m.

▶ **Definition 22.** *Given an $\alpha > 0$, a natural number d, and an even natural number m, define $\tilde{s}_{\alpha,d,m}(x) := \alpha T_d\left(\frac{x-1+r_0}{n} - 1\right)^m$, where r_0 is the smallest root of the polynomial $T_d\left(\frac{x}{n} - 1\right)$.*

▶ **Lemma 23.** $r_0 \leqslant \frac{\pi^2 n}{4d^2}$.

Proof. Observe that $T_d(x) = \cos(d\cos^{-1}(x))$ so the first zero of $T_d(x)$ is $\cos\left(-\pi + \frac{\pi}{2d}\right) \leqslant -1 + \frac{\pi^2}{4d^2}$. Thus, the first zero of $T_d\left(\frac{x}{n} - 1\right)$ is at most $\frac{\pi^2 n}{4d^2}$. ◀

▶ **Lemma 24.** *For $d > \frac{\pi}{2}\sqrt{n}$ the polynomial $\tilde{s}_{\alpha,d,m}(x)$ satisfies the following properties:*

1. *For all $x \in [1,n]$, $\tilde{s}_{\alpha,d,m}(x) \leqslant \min\left\{\frac{\alpha d^2}{n}(x-1), \alpha\right\}$.*
2. $\tilde{s}_{\alpha,d,m}(0) \geqslant \alpha\left(\frac{1}{4}\left(1 + \sqrt{\frac{2(1-r_0)}{n}}\right)^d\right)^m$.

Proof. For the first statement, observe that by the Markov Brothers' Theorem, since $|T_d(x)| \leqslant 1$ for all $x \in [-1,1]$, $|T_d'(x)| \leqslant d^2$ for all $x \in [-1,1]$. This implies that $\left|T_d'\left(\frac{x-1+r_0}{n} - 1\right)\right| \leqslant \frac{d^2}{n}$ for all $x \in [1-r_0, 2n+1-r_0]$. Since $T_d\left(\frac{x-1+r_0}{n} - 1\right) = 0$, when $x = 1$, $\left|T_d\left(\frac{x-1+r_0}{n} - 1\right)\right| \leqslant \min\{\frac{d^2(x-1)}{n}, 1\}$ for all $x \in [1,n]$, which implies the result.

For the second statement, by Lemma 7, if $0 \leqslant c \leqslant n$ then $|T_d(-1-\frac{c}{n})| \geqslant \frac{1}{4}\left(1+\sqrt{\frac{2c}{n}}\right)^d$. Applying this lemma with $c = 1 - r_0$, the result follows. ◀

▶ **Corollary 25.** *If the conditions*

1. $d \geqslant 3\sqrt{n}$,
2. $\alpha \leqslant \frac{n}{2d^2} \leqslant \frac{1}{2}$,
3. $m > \frac{\ln(P) - ln(\alpha)}{d\ln\left(1+\sqrt{\frac{2(1-r_0)}{n}}\right) - \ln(4)}$,

are satisfied, then the following properties hold:

1. $\tilde{s}_{\alpha,d,m}(0) > P$.
2. *For all* $x \in [1,2]$, $\tilde{s}_{\alpha,d,m}(x) \leqslant \frac{x-1}{2}$.
3. *For all* $x \in [2,n]$, $\tilde{s}_{\alpha,d,m}(x) \leqslant \frac{1}{2}$.

Thus, $(x-1) - \tilde{s}_{\alpha,d,m}(x)(x - \frac{1}{P}) > 0$ *whenever* $x \in \{0\} \cup (1,n]$.

Proof. The first statement follows from algebraic manipulations provided that $\frac{1}{4}\left(1+\sqrt{\frac{2(1-r_0)}{n}}\right)^d \geqslant 1$. To confirm that this holds, observe that $r_0 \leqslant \frac{\pi^2 n}{4d^2} \leqslant \frac{1}{2}$. Thus,

$$\left(1+\sqrt{\frac{2(1-r_0)}{n}}\right)^d \geqslant \left(1+\frac{1}{\sqrt{n}}\right)^d \geqslant 2^{\frac{d}{\sqrt{n}}} \geqslant 8.$$

For the second and third statements, we use the facts that for all $x \in [1,n]$, $\tilde{s}_{\alpha,d,m}(x) \leqslant \frac{\alpha d^2}{n}(x-1)$ and $\tilde{s}_{\alpha,d,m}(x) \leqslant \alpha$, respectively.

To show that $(x-1) - \tilde{s}_{\alpha,d,m}(x)(x - \frac{1}{P}) > 0$ whenever $x \in \{0\} \cup (1,n]$, we make the following observations:

1. For $x = 0$, $-1 - \tilde{s}_{\alpha,d,m}(0)(-\frac{1}{P}) > -1 - P\left(-\frac{1}{P}\right) = 0$.
2. For $x \in (1,2]$, $(x-1) - \tilde{s}_{\alpha,d,m}(x)(x - \frac{1}{P}) \leqslant (x-1) - \frac{x-1}{2}\left(x - \frac{1}{P}\right) > 0$.
3. For $x \in [2,n]$, $(x-1) - \tilde{s}_{\alpha,d,m}(x)(x - \frac{1}{P}) \leqslant (x-1) - \frac{1}{2}\left(x - \frac{1}{P}\right) > 0$. ◀

We now confirm that

$$\tilde{s}_0 = (x-1) - \tilde{s}_{\alpha,d,m}(x)\left(x - \frac{1}{P}\right)$$

is a sum of squares modulo the Boolean axioms. To see this, observe that since $\tilde{s}_0(x) > 0$ for $x \in \{0\} \cup (1,n]$, $\tilde{s}_0(x)$ must have an even number of roots in $(0,1]$ and no other roots in $[0,n]$. Thus, we can write

$$\tilde{s}_0(x) = p\prod_{i=1}^{l}(x-a_i)(x-b_i)$$

for some polynomial p which is positive on $[0,n]$ and some real roots $a_1, \ldots, a_l, b_1, \ldots, b_l \in (0,1]$. Since p is positive on $[0,n]$, p is a sum of squares modulo the Boolean axioms. By Corollary 9, since $|x|(|x|-1)$ is a sum of squares modulo the Boolean axioms, for each $i \in [l]$, $(x-a_i)(x-b_i)$ is also a sum of squares modulo the Boolean axioms. Thus, $\tilde{s}_0(x)$ is a sum of squares modulo the Boolean axioms.

Finally, we observe that we can satisfy the required conditions on d, α, and m by taking $d = \lceil 3\sqrt{n} \rceil$, $\alpha = \frac{1}{2d^2} \approx \frac{1}{18n}$, and $m = O(\log(P))$, which gives a sum of squares certificate of degree $O(\sqrt{n}\log(P))$.

5 SoS rank upper bound for the SC problem via SQF certification

In this section, we refute the Bienstock–Zuckenberg conjecture for the SC problem. We provide a degree $O(\sqrt{n}\log(n))$ SoS certificate for the SC problem on the Boolean hypercube, thus proving Theorem 3. For this proof, we use the SoS rank for certifying SQFs for $k = 2$ in Theorem 1. We present an alternative direct proof in Section 6.

We begin this section with a discussion on the properties necessary for an SoS polynomial s to even be considered as a possible candidate for an SoS certificate for the SC problem. An SoS certificate for the SC problem is of the form $\sum_{i\in[n]} x_i - 2 = s_0(\mathbf{x}) + \sum_{i\in[n]} s_i(\mathbf{x})g_i(\mathbf{x})$, where $g_i(\mathbf{x}) = \left(\sum_{j\in[n]\setminus\{i\}} x_j - 1\right)$. As opposed to the discussion in Section 4, an SoS certificate for the SC problem not only has multiple constraints but also displays a certain type of asymmetry, which is present in the formulation of the polynomials g_i for $i \in [n]$. One could hope to abuse this asymmetry by constructing different SoS polynomials $s_i \in \Sigma_{n,d}$ for certain $d \in [n]$, but for this proof, we proceed in a similar fashion as for the MK problem and instead construct only one symmetric SoS polynomial $s : \{0,1\}^n \to \mathbb{R}$ and look for the certificate of the form $\sum_{i\in[n]} x_i - 2 = s_0(\mathbf{x}) + \sum_{i\in[n]} s(\mathbf{x})g_i(\mathbf{x})$. Through permutation of indices, the existence of an SoS certificate for the SC problem implies the existence of an SoS certificate such that s is symmetric, that is, there exists an $\tilde{s} : \mathbb{R} \to \mathbb{R}$ such that $s(\mathbf{x}) = \tilde{s}(|\mathbf{x}|)$ for all $\mathbf{x} \in \{0,1\}^n$. As for the MK problem, we are interested in the requirements that polynomial $\tilde{s}$ needs to satisfy such that s constitutes part of an SoS certificate for the SC problem. Let $g(\mathbf{x}) := \sum_{i\in[n]} g_i(\mathbf{x}) = (n-1)(\sum_{i=1}^n x_i) - n$ and note that g is a symmetric polynomial; there exists a univariate polynomial $\tilde{g}$ such that $\tilde{g}(|\mathbf{x}|) = g(\mathbf{x})$ for all $\mathbf{x} \in \{0,1\}^n$. Since s_0 is globally nonnegative, this implies that s needs to satisfy

$$\begin{aligned} |\mathbf{x}| - 2 &\geqslant \tilde{s}(|\mathbf{x}|)\left(|\mathbf{x}|(|\mathbf{x}|-2) + (n-|\mathbf{x}|)(|\mathbf{x}|-1)\right) \\ &= \tilde{s}(|\mathbf{x}|)((n-1)|\mathbf{x}| - n) = \tilde{s}(|\mathbf{x}|)\tilde{g}(|\mathbf{x}|) \qquad \text{for all } \mathbf{x} \in \{0,1\}^n. \end{aligned} \tag{5.1}$$

This implies that $\tilde{s}(0) \geqslant \frac{2}{n}$, $\tilde{s}(1) \geqslant 1$, $\tilde{s}(2) = 0$ and $\tilde{s}(x) \leqslant \frac{x-2}{3(n-1)x-n}$ for all $x \in \{3, 4, \ldots, n\}$. We will construct a sum of squares polynomial $\tilde{s}(x)$ which satisfies the following slightly stronger conditions:

1. $\tilde{s}(x) \geqslant 1$ for all $x \in [0,1]$.
2. For all $x \in [1,2)$, $\frac{\tilde{s}(x)}{x-2} < 0$ and $\frac{\tilde{s}(x)}{x-2}$ is increasing.
3. $\tilde{s}(x) \leqslant \frac{(x-2)}{2n}$ for all $x \in [2,3]$.
4. $\tilde{s}(x) \leqslant \frac{1}{2n}$ for all $x \in [3,n]$.

We will then observe that these conditions imply that $\tilde{s}_0(x) = x - 2 - \tilde{s}(x)((n-1)x - n)$ is positive for $x \in [0,1) \cup (2,n]$ and has exactly two zeros in the interval $[1,2]$, one of which is $x = 2$. We can then use Theorem 1 and Corollary 9 to show that $\tilde{s}_0$ is a sum of squares of degree $\deg(\tilde{s}) + O(\sqrt{n}\log(n))$ modulo the Boolean axioms.

▶ **Lemma 26.** *For $d = 3\sqrt{n}$, $\alpha = \frac{1}{18n}$, and $m = 2\lceil \log_2(\sqrt{18n}) \rceil$ the polynomial $\tilde{s}(x) = \tilde{s}_{\alpha,d,m}(x-1)$ satisfies the following properties:*

1. *$\tilde{s}(x) \geqslant 1$ for all $x \in [0,1]$.*
2. *For all $x \in [1,2)$, $\frac{\tilde{s}(x)}{x-2} < 0$ and $\frac{\tilde{s}(x)}{x-2}$ is increasing.*
3. *$\tilde{s}(x) \leqslant \frac{(x-2)}{2n}$ for all $x \in [2,3]$.*
4. *$\tilde{s}(x) \leqslant \frac{1}{2n}$ for all $x \in [3,n]$.*

Proof. For the first statement, just as in the proof of Corollary 25, $r_0 \leqslant \frac{\pi^2 n}{4d^2} \leqslant \frac{1}{2}$. Thus,

$$\left(1+\sqrt{\frac{2(1-r_0)}{n}}\right)^d \geqslant \left(1+\frac{1}{\sqrt{n}}\right)^d \geqslant 2^{\frac{d}{\sqrt{n}}} \geqslant 8$$

Hence, by Lemma 24, $\tilde{s}(1) = \tilde{s}_{\alpha,d,m}(0) \geqslant \alpha 2^m \geqslant 1$. Since $deg(\tilde{s})$ is even, all roots of $\tilde{s}$ are real and the smallest root of $\tilde{s}$ is 2, $\tilde{s}$ is positive and decreasing when $x < 2$ so $\tilde{s}(x) \geqslant 1$ whenever $x \in [0,1]$, as needed.

For the second statement, observe that since $\deg(\tilde{s})$ is even, all roots of $\tilde{s}$ are real and the smallest root of $\tilde{s}$ is 2, $\frac{\tilde{s}(x)}{x-2}$ is negative and increasing whenever $x < 2$.

For the third statement, observe that by Lemma 24, for all $x \in [2,3]$, $\tilde{s}(x) = \tilde{s}_{\alpha,d,m}(x-1) \leqslant \alpha\frac{d^2}{n}(x-2) \leqslant \frac{x-2}{2n}$.

For the fourth statement, observe that by Lemma 24, for all $x \in [3,n]$, $\tilde{s}(x) = \tilde{s}_{\alpha,d,m}(x-1) \leqslant \alpha < \frac{1}{2n}$. ◀

▶ **Corollary 27.** *For $d = 3\sqrt{n}$, $\alpha = \frac{1}{n}$, $m = 2\lceil\log_2(n)\rceil$, and $\tilde{s}(x) = \tilde{s}_{\alpha,d,m}(x-1)$ the polynomial $\tilde{s}_0(x) = x-2-\tilde{s}(x)((n-1)x-n)$ is positive for $x \in [0,1)\cup(2,n]$ and has exactly two zeros in the interval $[1,2]$, one of which is $x = 2$.*

Proof. We make the following observations:

1. For all $x \in [0,1)$,

$$\tilde{s}_0(x) = x-2-\tilde{s}(x)((n-1)x-n) \geqslant x-2-((n-1)x-n) = (n-2)(1-x) > 0.$$

2. For all $x \in [1,2]$, $\frac{\tilde{s}_0}{x-2} = 1-((n-1)x-n)\frac{\tilde{s}}{x-2}$. When $x \in [\frac{n}{n-1},2]$, $((n-1)x-n)\frac{\tilde{s}}{x-2} \leqslant 0$ so $\frac{\tilde{s}}{x-2} > 0$. When $x \in [1,\frac{n}{n-1})$, both $((n-1)x-n)$ and $\frac{\tilde{s}}{x-2}$ are negative and increasing so $((n-1)x-n)\frac{\tilde{s}}{x-2}$ is positive and decreasing and thus $\frac{\tilde{s}_0}{x-2}$ is increasing. Since $\frac{\tilde{s}_0(1)}{1-2} \leqslant 0$ and $\frac{\tilde{s}_0(\frac{n}{n-1})}{\frac{n}{n-1}-2} > 0$, $\frac{\tilde{s}_0(x)}{x-2}$ must have exactly one zero in the interval $[1,\frac{n}{n-1}]$.
3. For all $x \in (2,3]$, $\tilde{s}_0(x) = x-2-\tilde{s}(x)((n-1)x-n) \geqslant x-2-\frac{(n-1)x-n}{2n}(x-2) > 0$.
4. For all $x \in [3,n]$, $\tilde{s}_0(x) = x-2-\tilde{s}(x)((n-1)x-n) \geqslant x-2-\frac{(n-1)x-n}{2n} > \frac{x}{2}-\frac{3}{2} \geqslant 0$.

◀

▶ **Corollary 28.** *$\tilde{s}_0(|x|)$ is a sum of squares of degree $O(\sqrt{n}\log(n))$ modulo the Boolean axioms.*

Proof. Since $\tilde{s}_0(x) = x-2-\tilde{s}(x)((n-1)x-n)$ is positive for $x \in [0,1)\cup(2,n]$ and has exactly two zeros in the interval $[1,2]$, one of which is $x = 2$, we can write

$$\tilde{s}_0(x) = \tilde{p}(x-a)(x-2),$$

for some $a \in [1,2)$ where $\tilde{p}(x)$ is positive for has no real roots in the interval $[0,n]$. Since $\tilde{p}(x)$ is positive and has no real roots in the interval $[0,n]$, $\tilde{p}(|x|)$ is a sum of squares modulo the Boolean axioms. By Theorem 1 and Corollary 9, $(x-a)(x-2)$ is a sum of squares of degree $O(\sqrt{n}\log(n))$ modulo the Boolean axioms. ◀

Thus, there exists a degree $O(\sqrt{n}\log(n))$ SoS certificate of nonnegativity for the SC problem.

6 Alternative Proof for the SoS rank upper bound for the SC problem

In this section, we provide an alternative proof of Theorem 3. More precisely, we prove an $O(\sqrt{n}\log(n))$ upper bound on the SoS rank for the SC problem without using Theorem 1.

By the problem formulation, Definition (1.2), and Equation (2.1), the SoS rank for the SC Problem is the smallest d for which there exist SoS polynomials $s_0 \in \Sigma_{n,2d+2}$ and $s_i \in \Sigma_{n,2d}$ for $i \in [n]$ such that

$$\sum_{i=1}^{n} x_i - 2 = s_0(\mathbf{x}) + \sum_{i=1}^{n} s_i \left(\sum_{\substack{j=1 \\ j\neq i}}^{n} x_j - 1 \right).$$

Equivalently, it is the smallest positive integer d such that $\sum_{i=1}^{n} x_i - 2 \in \Sigma^{\mathcal{G}}_{n,d}$.

To prove the SoS rank upper bound for the SC problem, we define the polynomials $h_1(\mathbf{x}) := |\mathbf{x}| - 1$ and $h_2(\mathbf{x}) := |\mathbf{x}| \left(|\mathbf{x}| - 2\right)$ and require the following lemma, in which we use the asymmetry inherent to the constraints of the SC problem.

▶ **Lemma 29.** *For polynomials h_1, h_2 it holds that $h_1(\mathbf{x}) \in \Sigma^{\mathcal{G}}_{n,0}$ and $h_2(\mathbf{x}) \in \Sigma^{\mathcal{G}}_{n,1}$.*

Proof. Consider the first polynomial, h_1, and note that

$$\sum_{i=1}^{n} x_i - 1 = \frac{1}{n-1} \sum_{i=1}^{n} \left(\sum_{\substack{j=1 \\ j\neq i}}^{n} x_j - 1 \right) + \frac{1}{n-1} \in \Sigma^{\mathcal{G}}_{n,0}.$$

Polynomial h_2 can be written as

$$\begin{aligned} \sum_{j=1}^{n} x_j \left(\sum_{i=1}^{n} x_i - 2 \right) &= \sum_{j=1}^{n} \left(x_j \left(\sum_{i=1}^{n} x_i - x_j - 1 \right) + (x_j^2 - x_j) \right) \\ &= \sum_{j=1}^{n} \left(x_j^2 \left(\sum_{\substack{i=1 \\ i\neq j}}^{n} x_i - 1 \right) - (x_j^2 - x_j) \left(\sum_{\substack{i=1 \\ i\neq j}}^{n} x_i - 1 \right) + (x_j^2 - x_j) \right) \in \Sigma^{\mathcal{G}}_{n,1}. \end{aligned}$$

◀

Although Lemma 29 uses asymmetry in the constraints of the SC problem, both h_1 and h_2 are symmetric polynomials. We can thus define polynomials $\tilde{h}_1,\ \tilde{h}_2 : \mathbb{R} \to \mathbb{R}$ such that $\tilde{h}_1(|\mathbf{x}|) = h_1(\mathbf{x})$, and $\tilde{h}_2(|\mathbf{x}|) = h_2(\mathbf{x})$, respectively. We are working towards a proof of the existence of polynomials $p_1, p_2 : \mathbb{R} \to \mathbb{R}$ such that

$$(x-2) - p_1(x)\tilde{h}_1(x) - p_2(x)\tilde{h}_2(x) \geqslant 0 \qquad \text{for all } x \in [0, n]. \tag{6.1}$$

6.1 Construction of polynomials $\mathbf{p_1}, \mathbf{p_2}$

We consider necessary, but not sufficient requirements that the polynomials p_1 and p_2 have to satisfy, that is, $p_1(2)\tilde{h}_1(2) + p_2(2)\tilde{h}_2(2) = 0$, $\left[p_1\tilde{h}_1 + p_2\tilde{h}_2\right]'(2) = 1$, and $\left[p_1\tilde{h}_1 + p_2\tilde{h}_2\right]''(2) < 0$. It is easy to check that these requirements are satisfied if p_1 has a double root at $x = 2$, $p_2(2) = 1/2$, and $1 + 4p_2'(2) + 2\frac{p_1(2)}{(x-2)^2} < 0$. We use these guidelines to construct polynomials

$$\begin{aligned} p_1(x) &:= \frac{1}{2n^2 c_1} (x-2)^2 \, T^2_{2\sqrt{n}\log(n)} \left(2\frac{x-2}{n} - 1 \right), \\ p_2(x) &:= \frac{1}{2nc_2} \, T^2_{2\sqrt{n}\log(n)} \left(2\frac{x-3}{n} - 1 \right), \end{aligned} \tag{6.2}$$

where c_1 and c_2 are constants equal to $\frac{1}{2n^2}T^2_{2\sqrt{n}\log(n)}\left(-\frac{2}{n}-1\right)$ and $\frac{1}{n}\,T^2_{2\sqrt{n}\log(n)}\left(-\frac{2}{n}-1\right)$, respectively, such that $p_1(1)=1$ and $p_2(2)=1/2$.

▶ **Lemma 30.** *There exists $C\in\mathbb{N}$ such that for $n\geqslant C$, the polynomial p_1 satisfies the following properties:*

1. $p_1(x)\geqslant 4$ *for* $x\in[0,\frac{1}{2}]$.
2. $p_1(x)\leqslant\left(-0.9(x-1)+1\right)(x-2)^2$ *for* $x\in[1,2]$.
3. $p_1(x)\leqslant\frac{1}{2n^2}(x-2)^2$ *for* $x\in[2,n]$.

Proof. Since $p_1(x)$ is decreasing for $x\leqslant 1$, to prove Property (1) it is enough to show that $p_1(\frac{1}{2})\geqslant 4$. By Lemma 7 and for sufficiently big n, it holds

$$p_1(1/2)=\frac{\frac{9}{4}\,T^2_{2\sqrt{n}\log(n)}\left(\frac{-3}{n}-1\right)}{T^2_{2\sqrt{n}\log(n)}\left(-\frac{2}{n}-1\right)}\geqslant\frac{1}{2}\left(\frac{1+\sqrt{\frac{6}{n}}}{1+\sqrt{\frac{5}{n}}}\right)^{4\sqrt{n}\log(n)}.$$

Since $\frac{1}{2}\left(\frac{1+\sqrt{\frac{6}{n}}}{1+\sqrt{\frac{5}{n}}}\right)^{4\sqrt{n}\log(n)}\geqslant 4$ for $n\geqslant 32$ and by monotonicity, Property (1) is satisfied.

To prove Property (3), note that for every $x\in[2,n]$ and $d\in\mathbb{N}$ we have $T^2_d(2\frac{x-2}{n}-1)\leqslant 1$ and for every $n\geqslant 2$, by Lemma 7, we have $c_1=\frac{1}{2n^2}\,T^2_{2\sqrt{n}\log(n)}\left(-\frac{2}{n}-1\right)\geqslant\frac{1}{2n^2}\frac{1}{4}\left(1+\sqrt{\frac{4}{n}}\right)^{4\sqrt{n}\log(n)}\geqslant 1$.

To prove Property (2), we show that $\frac{1}{2n^2}T^2_{2\sqrt{n}\log(n)}\left(2\frac{x-2}{n}-1\right)\leqslant\left(-0.9(x-1)+1\right)$ for every $x\in[1,2]$. By construction, it is satisfied for $x=1$ and by Property (3), it is satisfied for $x=2$. Since the function $T^2_{2\sqrt{n}\log(n)}\left(2\frac{x-2}{n}-1\right)$ is convex in the interval $[1,2]$, the property is satisfied for $x\in[1,2]$. ◀

▶ **Lemma 31.** *There exists a constant $C\in\mathbb{N}$ such that for $n\geqslant C$, the polynomial p_2 satisfies the following properties:*

1. $p_2(x)\geqslant 4$ *for* $x\in[0,1]$.
2. $p_2(2)=\frac{1}{2}$.
3. $p_2'(x)\leqslant -1$ *for* $x\in[1,2]$.
4. $p_2(x)\leqslant -0.45(x-2)+\frac{1}{2}$ *for* $x\in[2,3]$.
5. $p_2(x)\leqslant\frac{1}{2n}$ *for* $x\in[3,n]$.

Proof. Since $p_2(x)$ is decreasing for $x\leqslant 1$, to prove Property (1), it is enough to show that $p_2(1)\geqslant 4$. For sufficiently big n we get:

$$p_2(1):=\frac{\frac{1}{2}\,T^2_{2\sqrt{n}\log(n)}\left(-\frac{4}{n}-1\right)}{T^2_{2\sqrt{n}\log(n)}\left(-\frac{2}{n}-1\right)}\geqslant\frac{1}{8}\left(\frac{-1-\sqrt{\frac{8}{n}}}{-1-\sqrt{\frac{5}{n}}}\right)^{4\sqrt{n}\log(n)}.$$

Since $\frac{1}{8}\left(\frac{-1-\sqrt{\frac{8}{n}}}{-1-\sqrt{\frac{5}{n}}}\right)^{4\sqrt{n}\log(n)}\geqslant 4$ for $n\geqslant 13$ and by monotonicity, Property (1) is satisfied.

Property (2) is satisfied by construction.

Since for every $x\in[3,n]$ and $d\in\mathbb{N}$ we have $\left|T_d(2\frac{x-3}{n}-1)\right|\leqslant 1$ and for every $n\geqslant 2$, by Lemma 7, we have $c_2=\frac{1}{n}\,T^2_{2\sqrt{n}\log(n)}\left(-\frac{2}{n}-1\right)\geqslant\frac{1}{n}\frac{1}{4}\left(-1-\sqrt{\frac{4}{n}}\right)^{4\sqrt{n}\log(n)}\geqslant 1$, Property (5) is satisfied.

Since $p_2(x)$ is convex for $x \in [1,2]$, to prove Property (3), it is enough to show $p_2^{'}(2) \leqslant -1$. Note that $\frac{\partial T_d(x)}{\partial x} = dU_{d-1}(x)$ and $\frac{\partial T_d^2(x)}{\partial x} = 2dT_d(x)U_{d-1}(x)$, where $U_d(x)$ is a Chebyshev polynomial of the second type. Thus,

$$p_2^{'}(x) = \frac{4\log(n)T_{2\sqrt{n}\log(n)}\left(\frac{2(x-3)}{n} - 1\right)U_{2\sqrt{n}\log(n)-1}\left(\frac{2(x-3)}{n} - 1\right)}{\sqrt{n}T_{2\sqrt{n}\log(n)}\left(-1-\frac{2}{n}\right)},$$

which implies that

$$p_2^{'}(2) = \frac{4\log(n)U_{2\sqrt{n}\log(n)-1}\left(-1-\frac{2}{n}\right)}{\sqrt{n}T_{2\sqrt{n}\log(n)}\left(-1-\frac{2}{n}\right)} = \left[\frac{\partial}{\partial x}\frac{T_{2\sqrt{n}\log(n)}\left(2\frac{x-3}{n} - 1\right)}{T_{2\sqrt{n}\log(n)}\left(-1-\frac{2}{n}\right)}\right](2)\,.$$

Since $T_{2\sqrt{n}\log(n)}\left(2\frac{x-3}{n} - 1\right)$ for $x = 2.5$ takes at most half of the value for $x = 2$ and since $T_{2\sqrt{n}\log(n)}\left(2\frac{x-3}{n} - 1\right)$ is convex in the interval $[2,3]$, $p_2^{'}(x) \leqslant -1$. By Lemma 7,

$$\frac{T^2_{2\sqrt{n}\log(n)}\left(-\frac{2}{n} - 1\right)}{T^2_{2\sqrt{n}\log(n)}\left(-\frac{1}{n} - 1\right)} \geqslant \frac{1}{4}\left(\frac{-1-\sqrt{\frac{4}{n}}}{-1-\sqrt{\frac{3}{n}}}\right)^{2\sqrt{n}\log(n)}.$$

Since $\frac{1}{4}\left(\frac{-1-\sqrt{\frac{4}{n}}}{-1-\sqrt{\frac{3}{n}}}\right)^{2\sqrt{n}\log(n)} \geqslant 2$ for $n \geqslant 100$ and by monotonicity, Property (3) is satisfied.

By Property (2), Property (4) holds for $x = 2$. By Property (5), it holds for $x = 3$ and $n \geqslant 10$. Since $p_2(x)$ is convex for $x \in [2,3]$, the property holds for $x \in [2,3]$. ◀

Now we are ready to prove the main lemma of this section.

▶ **Lemma 32.** *It holds that*

$$f(x) := x - 2 - p_1(x)h_1(x) - p_2(x)h_2(x) \geqslant 0 \qquad \text{for } x \in [0,n]. \tag{6.3}$$

Proof. Note that $h_1(x),\ h_2(x) \leqslant 0$ for $x \in [0,1]$. For all $x \in [0,\frac{1}{2}]$, $p_1(x)$ is decreasing and $h_1(x)$ is increasing in x. Thus, by Property (1), for $x \in [0,\frac{1}{2}]$, $f(x) \geqslant x - 2 - p_1(x)h_1(x) \geqslant -2 - p_1(1/2)h_1(1/2) \geqslant -2 + 4\cdot\frac{1}{2} \geqslant 0$. For $x \in [\frac{1}{2},1]$, both $p_2(x)$ and $h_2(x)$ are decreasing. Thus, for $x \in [\frac{1}{2},1]$ and by Property (1), $f(x) \geqslant x - 2 - p_2(x)h_2(x) \geqslant -\frac{3}{2} - p_2(1)h_2(1/2) \geqslant -\frac{3}{2} + 4\cdot\frac{3}{4} \geqslant 0$. To prove the statement for $x \in [1,2]$, we show that for every $a \in [0,1]$, we have $f(2-a) \geqslant 0$. By construction, we have $f(2) = 0$. Thus, the property holds for $a = 0$. By Property (2), for polynomial p_1, we have $p_1(2-a) \leqslant (0.9a + 0.1)a^2$. By Properties (2) and (3), for polynomial p_2, we have $p_2(2-a) \geqslant 1/2 + a$. Thus, $f(2-a) \geqslant -a - (0.9a+0.1)a^2(1-a) + (1/2+a)a(2-a) = a^2((0.9a-1.8)a+1.4)$, which is nonnegative for $a \in [0,1]$. This proves the statement for $x \in [1,2]$.

To prove the statement for $x \in [2,3]$ we show that for every $a \in [0,1]$, it holds that $f(2+a) \geqslant 0$. By Property (3), for $x \in [2,3]$ and $n \geqslant 2$, we get $p_1(2+a) \leqslant \frac{1}{4}a^2$. By Property (4), we get that $p_2(2+a) \leqslant -0.45a + \frac{1}{2}$. Thus, $f(2+a) \geqslant a - \frac{1}{4}a^2(1+a) - \left(-0.45a + \frac{1}{2}\right)(2+a)a = (0.15 + 0.2a)a^2$, which is non-negative for $a \in [0,1]$. This proves the statement for $x \in [2,3]$. Finally, for $x \in [3,n]$, we have $f(x) \geqslant x - 2 - \frac{1}{2n^2}(x-2)^2(x-1) - \frac{1}{2n}x(x-2) \geqslant 0$. ◀

6.2 Proof of Theorem 3

By Lemma 32, $x - 2 - p_1(x)h_1(x) - p_2(x)h_2(x) \geqslant 0$ for $x \in [0,n]$ and the degree of the polynomial on the LHS is at most $O(\sqrt{n}\log(n))$. Thus, by Theorem 6, there exist SoS polynomials s_0, s_1 such that $x - 2 - p_1(x)h_1(x) - p_2(x)h_2(x) = s_0(x) + x(n-x)s_1(x)$. Thus, $x - 2 = s_0(x) + x(n-x)s_1(x) + p_1(x)h_1(x) + p_2(x)h_2(x)$. Remark 5 and the fact that $|\mathbf{x}|(n - |\mathbf{x}|)$ has a degree 1 SoS certificate over the Boolean hypercube imply the existence of a degree $O(\sqrt{n}\log(n))$ certificate over the Boolean hypercube for the polynomial $\sum_{i=1}^n x_i - 2$.

References

1. S. Arora, S. Rao, and U. V. Vazirani. Expander flows, geometric embeddings and graph partitioning. *J. ACM*, 56(2):5:1–5:37, 2009.
2. B. Barak, S. B. Hopkins, J. A. Kelner, P. Kothari, A. Moitra, and A. Potechin. A nearly tight sum-of-squares lower bound for the planted clique problem. In *IEEE 57th Annual Symposium on Foundations of Computer Science, FOCS 2016, 9-11 October 2016, Hyatt Regency, New Brunswick, New Jersey, USA*, pages 428–437, 2016.
3. B. Barak, J. A. Kelner, and D. Steurer. Dictionary learning and tensor decomposition via the sum-of-squares method. In *STOC 2015, Portland, OR, USA, June 14-17, 2015*, pages 143–151, 2015.
4. B. Barak and A. Moitra. Noisy tensor completion via the sum-of-squares hierarchy. In *COLT 2016, New York, USA, June 23-26, 2016*, pages 417–445, 2016.
5. B. Barak, P. Raghavendra, and D. Steurer. Rounding semidefinite programming hierarchies via global correlation. In *FOCS*, pages 472–481, 2011.
6. B. Barak and D. Steurer. Proofs, beliefs, and algorithms through the lens of sum-of-squares, 2016. URL: `https://www.sumofsquares.org`.
7. Aditya Bhaskara, Moses Charikar, Aravindan Vijayaraghavan, Venkatesan Guruswami, and Yuan Zhou. Polynomial integrality gaps for strong sdp relaxations of densest k-subgraph. In *SODA*, 2012.
8. D. Bienstock and M. Zuckerberg. Subset algebra lift operators for 0-1 integer programming (extended version), 2002. Extended version of: Subset algebra lift operators for 0-1 integer programming - SIAM Journal on Optimization, 1(15): 63-95, 2004. URL: `http://www.corc.ieor.columbia.edu/reports/techreports.html`.
9. Grigoriy Blekherman, Pablo A Parrilo, and Rekha R Thomas. *Semidefinite optimization and convex algebraic geometry*. SIAM, 2012.
10. K. K. H. Cheung. Computation of the Lasserre ranks of some polytopes. *Math. Oper. Res.*, 32(1):88–94, 2007.
11. W. Cook and S. Dash. On the matrix-cut rank of polyhedra. *Math. Oper. Res.*, 26(1):19–30, 2001.
12. Hamza Fawzi, James Saunderson, and Pablo A. Parrilo. Sparse sum-of-squares certificates on finite abelian groups. In *54th IEEE Conference on Decision and Control, CDC 2015, Osaka, Japan, December 15-18, 2015*, pages 5909–5914, 2015. `doi:10.1109/CDC.2015.7403148`.
13. Mrinalkanti Ghosh, Fernando Granha Jeronimo, Chris Jones, Aaron Potechin, and Goutham Rajendran. Sum-of-squares lower bounds for sherrington-kirkpatrick via planted affine planes. In *61st IEEE Annual Symposium on Foundations of Computer Science, FOCS*, pages 954–965, 2020.
14. M. X. Goemans and D. P. Williamson. Improved approximation algorithms for maximum cut and satisfiability problems using semidefinite programming. *J. Assoc. Comput. Mach.*, 42(6):1115–1145, 1995.
15. D. Grigoriev. Complexity of positivstellensatz proofs for the knapsack. *Comput. Complexity*, 10(2):139–154, 2001.
16. D. Grigoriev, E. A. Hirsch, and D. V. Pasechnik. Complexity of semi-algebraic proofs. In *STACS*, pages 419–430, 2002.
17. D. Grigoriev and N. Vorobjov. Complexity of null-and positivstellensatz proofs. *Ann. Pure App. Logic*, 113(1-3):153–160, 2001.
18. Dima Grigoriev. Linear lower bound on degrees of positivstellensatz calculus proofs for the parity. *Theoretical Computer Science*, 259(1-2):613–622, 2001.
19. V. Guruswami and A. K. Sinop. Lasserre hierarchy, higher eigenvalues, and approximation schemes for graph partitioning and quadratic integer programming with psd objectives. In *FOCS*, pages 482–491, 2011.

20 S. B. Hopkins, T. Schramm, J. Shi, and D. Steurer. Fast spectral algorithms from sum-of-squares proofs: tensor decomposition and planted sparse vectors. In *STOC 2016, Cambridge, MA, USA, June 18-21, 2016*, pages 178–191, 2016.

21 P. Kothari, J. Steinhardt, and D. Steurer. Robust moment estimation and improved clustering via sum of squares. In *STOC 2018*, 2018.

22 P. K. Kothari, R. Mori, R. O'Donnell, and D. Witmer. Sum of squares lower bounds for refuting any CSP. In *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2017, Montreal, QC, Canada, June 19-23, 2017*, pages 132–145, 2017.

23 Dmitriy Kunisky and Afonso S. Bandeira. A tight degree 4 sum-of-squares lower bound for the sherrington-kirkpatrick hamiltonian. *CoRR*, abs/1907.11686, 2019. `arXiv:1907.11686`.

24 A. Kurpisz, S. Leppänen, and M. Mastrolilli. Sum-of-squares hierarchy lower bounds for symmetric formulations. In *Integer Programming and Combinatorial Optimization - 18th International Conference, IPCO 2016, Liège, Belgium, June 1-3, 2016, Proceedings*, pages 362–374, 2016.

25 A. Kurpisz, S. Leppänen, and M. Mastrolilli. Tight sum-of-squares lower bounds for binary polynomial optimization problems. In *43rd International Colloquium on Automata, Languages, and Programming, ICALP 2016, July 11-15, 2016, Rome, Italy*, pages 78:1–78:14, 2016.

26 A. Kurpisz, S. Leppänen, and M. Mastrolilli. On the hardest problem formulations for the 0/1 lasserre hierarchy. *Math. Oper. Res.*, 42(1):135–143, 2017.

27 A. Kurpisz, S. Leppänen, and M. Mastrolilli. An unbounded sum-of-squares hierarchy integrality gap for a polynomially solvable problem. *Math. Program.*, 166(1-2):1–17, 2017.

28 Adam Kurpisz. Sum-of-squares bounds via boolean function analysis. In *ICALP July 9-12, 2019, Patras, Greece*, 2019.

29 J. B. Lasserre. An explicit exact SDP relaxation for nonlinear 0-1 programs. In *Integer Programming and Combinatorial Optimization, 8th International IPCO Conference, Utrecht, The Netherlands, June 13-15, 2001, Proceedings*, pages 293–303, 2001.

30 M. Laurent. A comparison of the Sherali-Adams, Lovász-Schrijver, and Lasserre relaxations for 0-1 programming. *Math. Oper. Res.*, 28(3):470–496, 2003.

31 M. Laurent. Lower bound for the number of iterations in semidefinite hierarchies for the cut polytope. *Math. Oper. Res.*, 28(4):871–883, 2003.

32 J. R. Lee, P. Raghavendra, and D. Steurer. Lower bounds on the size of semidefinite programming relaxations. In *STOC*, pages 567–576, 2015.

33 T. Lee, A. Prakash, R. Wolf, and H. Yuen. On the sum-of-squares degree of symmetric quadratic functions. In *31st Conference on Computational Complexity, CCC 2016, May 29 to June 1, 2016, Tokyo, Japan*, pages 17:1–17:31, 2016.

34 László Lovász. On the shannon capacity of a graph. *IEEE Transactions on Information Theory*, 25:1–7, 1979.

35 Raghu Meka, Aaron Potechin, and Avi Wigderson. Sum-of-squares lower bounds for planted clique. In *Proceedings of the 47th Annual ACM on Symposium on Theory of Computing, STOC 2015, Portland, OR, USA*, pages 87–96, 2015.

36 M. J. Moore. An n job, one machine sequencing algorithm for minimizing the number of late jobs. *Management Science*, 15:102–109, 1968.

37 Katta G. Murty and Santosh N. Kabadi. Some np-complete problems in quadratic and nonlinear programming. *Mathematical Programming*, 39(2):117–129, 1987. `doi:10.1007/BF02592948`.

38 Y. Nesterov. *Global quadratic optimization via conic relaxation*, pages 363–384. Kluwer Academic Publishers, 2000.

39 P. Parrilo. *Structured Semidefinite Programs and Semialgebraic Geometry Methods in Robustness and Optimization*. PhD thesis, California Institute of Technology, 2000.

40 R. Paturi. On the degree of polynomials that approximate symmetric boolean functions (preliminary version). In *Proceedings of the 24th Annual ACM Symposium on Theory of Computing, May 4-6, 1992, Victoria, British Columbia, Canada*, pages 468–474, 1992.

41 A. Potechin and D. Steurer. Exact tensor completion with sum-of-squares. In *COLT 2017, Amsterdam, The Netherlands, 7-10 July 2017*, pages 1619–1673, 2017.

42 Aaron Potechin. Sum of squares bounds for the ordering principle. In *Proceedings of the 35th Computational Complexity Conference*, pages 1–37, 2020.

43 T. Rivlin. The chebyshev polynomials. *SERBIULA (sistema Librum 2.0)*, February 1974.

44 S. Sakaue, A. Takeda, S. Kim, and N. Ito. Exact semidefinite programming relaxations with truncated moment matrix for binary polynomial optimization problems. *SIAM Journal on Optimization*, 27(1):565–582, 2017.

45 T. Schramm and D. Steurer. Fast and robust tensor decomposition with applications to dictionary learning. In *COLT 2017, Amsterdam, The Netherlands, 7-10 July 2017*, pages 1760–1793, 2017.

46 N. Shor. Class of global minimum bounds of polynomial functions. *Cybernetics*, 23(6):731–734, 1987.

47 Lucas Slot and Monique Laurent. Improved convergence analysis of lasserre's measure-based upper bounds for polynomial minimization on compact sets. *CoRR*, 2019. doi:10.1007/s10107-020-01468-3.

48 J. Thapper and S. Zivny. The power of sherali-adams relaxations for general-valued csps. *SIAM J. Comput.*, 46(4):1241–1279, 2017.

49 Madhur Tulsiani. Csp gaps and reductions in the lasserre hierarchy. In *STOC*, pages 303–312, 2009.

50 R. Wolf. A note on quantum algorithms and the minimal degree of ϵ-error polynomials for symmetric functions. *Quantum Information & Computation*, 8(10):943–950, 2010.

On Counting (Quantum-)Graph Homomorphisms in Finite Fields of Prime Order

J. A. Gregor Lagodzinski
Hasso Plattner Institute, University of Potsdam, Germany

Andreas Göbel
Hasso Plattner Institute, University of Potsdam, Germany

Katrin Casel
Hasso Plattner Institute, University of Potsdam, Germany

Tobias Friedrich
Hasso Plattner Institute, University of Potsdam, Germany

Abstract

We study the problem of counting the number of homomorphisms from an input graph G to a fixed (quantum) graph $\bar{H}$ in any finite field of prime order $\mathbb{Z}_p$. The subproblem with graph H was introduced by Faben and Jerrum [ToC'15] and its complexity is still uncharacterised despite active research, e.g. the very recent work of Focke, Goldberg, Roth, and Zivný [SODA'21]. Our contribution is threefold.

First, we introduce the study of quantum graphs to the study of modular counting homomorphisms. We show that the complexity for a quantum graph $\bar{H}$ collapses to the complexity criteria found at dimension 1: graphs. Second, in order to prove cases of intractability we establish a further reduction to the study of bipartite graphs. Lastly, we establish a dichotomy for all bipartite ($K_{3,3}\backslash\{e\}$, *domino*)-free graphs by a thorough structural study incorporating both local and global arguments. This result subsumes all results on bipartite graphs known for all prime moduli and extends them significantly. Even for the subproblem with $p = 2$ this establishes new results.

2012 ACM Subject Classification Theory of computation → Problems, reductions and completeness; Mathematics of computing → Discrete mathematics

Keywords and phrases Algorithms, Theory, Quantum Graphs, Bipartite Graphs, Graph Homomorphisms, Modular Counting, Complexity Dichotomy

Digital Object Identifier 10.4230/LIPIcs.ICALP.2021.91

Category Track A: Algorithms, Complexity and Games

Related Version *Full Version*: `https://arxiv.org/abs/2011.04827`

Acknowledgements The authors would like to thank Holger Dell for bringing [12] to their attention. Our gratitude also goes to Jacob Focke and Marc Roth for their valuable insights on partially surjective homomorphisms and for pointing out a mistake in a previous version.

1 Introduction

The study of graph homomorphisms represents one of the classic bodies of work in both discrete mathematics and computer science but remains a very active research area. These homomorphisms play a crucial role in the study of graph limits and networks [4, 19, 20, 47], in the study of databases [11, 33, 39, 40], and in the study of spin-systems in statistical physics [2, 5]. Formally, a graph-homomorphism from G to H is a map from $V(G)$ to $V(H)$ that preserves edges. Many classic problems studied in computer science can be expressed with graph homomorphisms. Examples range from the *decision* problem of determining the chromatic number of a graph, through the problem of *counting* the number of independent

sets, to the problem of *counting* the number of k-colourings using all k colours. The latter can be expressed by a *linear combination* of the number of graph homomorphisms to a set of non-isomorphic graphs.

Graph homomorphisms are a prime example of a very general class of problems that frequently yields complexity dichotomies with structural characterisations, where the properties of a graph implying (in)tractibility are easily computable. However, the dichotomy itself is hard to establish and by Ladner [37] not obvious to exist. Hell and Nešetřil studied the *decision* problem HomsToH with fixed image graph H, that asks whether there exists a homomorphism from an input graph G to H. In [35] they showed that the problem HomsToH can be solved in polynomial time if H contains a loop or is bipartite; otherwise it is NP-complete. Dyer and Greenhill introduced the *counting* problem #HomsToH with fixed image graph H, that asks for the number of homomorphisms from an input graph G to H. In their seminal work [16] they showed that #HomsToH can be solved in polynomial time if the connected components of H are complete bipartite graphs or reflexive complete graphs; otherwise it is # P-complete.

Lovász [38] observed that there are many graph parameters that can only be expressed by a linear combination of computational problems #HomsToH for a set of at least two graphs $H \in \mathcal{H}$. Examples are the class of *vertex surjective homomorphisms* and *compactions* studied in this context by Focke, Goldberg, and Zivný [24]. Lovász [38] introduced the notion of a *quantum graph* for a linear combination of finitely many graphs called its *constituents*. We refer by the dimension of a quantum graph to its number of constituents and find the set of graphs at dimension 1. With every increase of dimension, the set of graph parameters expressible by #HomsToH increases as well. For a quantum graph $\bar{H}$ the counting problem #HomsTo$\bar{H}$ denotes then *linear combination* of problems #HomsToH for all constituents H of $\bar{H}$. Chen, Curticapean and Dell [12] studied the complexity of #HomsTo$\bar{H}$ and showed that the complexity is inherited from the complexity of #HomsToH for all constituents H of $\bar{H}$, which is given by the criterion of Dyer and Greenhill. Motivated by this strong connection, Chen et al. raised the question of whether techniques based on quantum graphs can advance the state of the art of open problems regarding modular counting homomorphisms.

We study the complexity of the problem $\#_p$HomsTo$\bar{H}$ for any prime p and answer the question of Chen et al. in the affirmative, where the problem $\#_p$HomsTo$\bar{H}$ asks for the value of #HomsTo$\bar{H}$ in the finite field $\mathbb{Z}_p$. Our contribution is threefold. First, we obtain results for the whole class of quantum graphs by showing that the complexity of $\#_p$HomsTo$\bar{H}$ is inherited from the complexity $\#_p$HomsToH. Second, we reduce the study of $\#_p$HomsToH to a study of bipartite graphs by establishing a reduction to a restricted homomorphism problem. Finally, we employ a structural analysis on the set of ($K_{3,3}\backslash\{e\}$, *domino*)-free graphs and establish a dichotomy for these.

The line of research on modular counting homomorphisms was initiated with the study of the parity of #HomsToH by Faben and Jerrum [21]. Despite the clear picture on the non-modular version #HomsToH the modulus implies additional cases of tractibility as structures in H implying intractibility for #HomsToH get "cancelled" when counting in a finite field $\mathbb{Z}_p$. Faben and Jerrum [21] showed that automorphisms of order p capture a subset of these "cancellations" and reduced the study to a structural analysis of parameter graphs H that do not enjoy such automorphisms. These graphs are called *order p reduced*. In particular, for $p = 2$ they conjectured that automorphisms of order 2 capture all cancellations and that $\#_2$HomsToH for an order 2 reduced graph enjoys the same complexity criterion as the non-modular version #HomsToH given by Dyer and Greenhill. Despite a growing line of research by Göbel, Goldberg, and Richerby [26, 27] and the very recent work of Focke,

Table 1 History of the study of $\#_p$HomsToH on **bipartite** graphs H. (Note that the complexity study can be restricted to bipartite graphs by the bipartization result of this paper.) Crosses denote that the result incorporates the dichotomy for the graphclass, a p denotes that the result holds for all primes. Parenthesis denote that the result is not intrinsic but given by additional argumentation.

	Mod	Trees	Cactus	Square-free	K_4-minor-free	$(K_{3,3}\backslash\{e\}, domino)$-free
Faben and Jerrum [21]	2	×				
Göbel et al. [26]	2	×	×			
Göbel et al. [27]	2	×		×		
Focke et al. [23]	2	×	×	(×)	×	
Göbel et al. [28]	p	×				
Kazeminia and Bulatov [36]	p	×		×		
This paper	p	×	×	×		×

Goldberg, Roth, and Zivný [23] the conjecture remains open. The body of work is dominated by a study of structures as the modulus commands incorporating not only the local but also global properties of the graph H.

The research on $\#_p$HomsToH for arbitrary primes p was already suggested by Faben and Jerrum [21] as they showed that their results concerning automorphisms of order p apply for any prime p. However, Valiant [48] showed the existence of computational counting problems that enjoy a change of complexity with respect to different moduli. Therefore, a uniform complexity criterion for $\#_p$HomsToH would emphasize the special role of graph homomorphisms even more. The study of $\#_p$HomsToH was finally initiated by Göbel, Lagodzinski, and Seidel [28] and followed by Kazeminia and Bulatov [36]. In light of the richer structure due to the higher moduli far less is known about the complexity of $\#_p$HomsToH compared to $\#_2$HomsToH. Even though Faben and Jerrum [21] as well as Göbel et al. [28] considered an extension of the conjecture to all prime moduli and the results so far suggest it, no one has gone that far yet. We illustrate the individual contributions and the state of the art in Table 1.

1.1 Contribution

We establish a plethora of technical results, which we believe to be a major asset to future works on the complexity of $\#_p$HomsToH and may be of independent interest to different lines of research. The main contributions are given in the following and discussed in more depth in the subsequent subsection.

Quantum Homomorphisms

We introduce the study of quantum graphs to the study of $\#_p$HomsToH. For any quantum graph $\bar{H}$ we find that $\#_p$HomsTo$\bar{H}$ is equivalent to $\#_p$HomsTo$\bar{H}'$, where $\bar{H}'$ is a quantum graph whose constituents are order p reduced with coefficients in $\mathbb{Z}_p^* = \mathbb{Z}_p\backslash\{0\}$. We call these constituents the *p-constituents* of $\bar{H}'$. Focusing on these "reduced" quantum graphs we obtain the following inheritance theorem.

▶ **Theorem 1.1.** *Let p be a prime and $\bar{H} = \sum_{H\in\mathcal{H}} \alpha_H H$ be a quantum graph with p-constituents $\mathcal{H} = \{H_1, \dots, H_r\}$ that are order p reduced pairwise non-isomorphic graphs and a set of associated constants $\{\alpha_H\}_{H\in\mathcal{H}}$ that are in $\mathbb{Z}_p^*$. Then,*

- *if there exists a graph $H \in \mathcal{H}$ such that $\#_p$HomsToH is $\#_p$P-hard, then $\#_p$HomsTo$\bar{H}$ is $\#_p$P-hard;*
- *if, for all $H \in \mathcal{H}$, $\#_p$HomsToH is solvable in polynomial time, then $\#_p$HomsTo$\bar{H}$ is also solvable in polynomial time.*

This shows that the complexity of $\#_p\text{HOMSTO}\bar{H}$ collapses to the complexity of $\#_p\text{HOMSTO}H$. Even though the set of graph parameters expressible by $\#\text{HOMSTO}\bar{H}$ is arbitrarily larger compared to the parameters expressible by $\#\text{HOMSTO}H$, the complexity behaviour is captured at dimension $r = 1$, i.e. graphs.

In the same spirit, we show that the reduction technique applied to show Theorem 1.1 yields a universal technique that can be applied to obtain so-called *pinning* in classes of graph-homomorphisms closed under the composition. This technique is helpful for our study as we also obtain pinning for the restricted class of homomorphisms introduced in the following.

Bipartization

We restrict the study of $\#_p\text{HOMSTO}H$ to the study of bipartite graphs by a restricted class of homomorphisms. For two bipartite graphs $G = (L_G, R_G, E_G)$ and $H = (L_H, R_H, E_H)$ with fixed bipartition we say that a homomorphism from G to H *preserves the order of the bipartition* if the homomorphism maps L_G to L_H and R_G to R_H. The problem $\#_p\text{BIPHOMSTO}H$ with fixed bipartite graph H then asks for the number of these homomorphisms to H. It allows us to restrict the study of $\#_p\text{HOMSTO}H$ to the study of bipartite graphs by the following theorem.

▶ **Theorem 1.2.** *For any prime p and any graph H, there exists a bipartite graph H' such that*

- *if $\#_p\text{BIPHOMSTO}H'$ is $\#_p\,\mathsf{P}$-hard then $\#_p\text{HOMSTO}H$ is $\#_p\,\mathsf{P}$-hard;*
- *if $\#_p\text{BIPHOMSTO}H'$ is solvable in polynomial time then $\#_p\text{HOMSTO}H$ is solvable in polynomial time.*

This implies that a dichotomy for $\#_p\text{BIPHOMSTO}H'$ yields a dichotomy for $\#_p\text{HOMSTO}H$. As we will later show, the graph H' is a collection of complete bipartite graphs if and only if H satisfies the Dyer and Greenhill criterion. An additional feature of Theorem 1.2 is that it allows for the graph H to contain loops whereas the bipartite graph H' is always loop-less by definition. So far no study of $\#_p\text{HOMSTO}H$ allowed for loops. The structural implications of a bipartite graph H are also heavily exploited in the following analysis.

Hardness in Bipartite $(K_{3,3}\backslash\{e\}$, *domino*$)$-Free Graphs

In the longest and most technically involved part of the paper we study bipartite graphs H not satisfying the Dyer and Greenhill criterion with the goal of finding enough structural information to establish hardness of $\#_p\text{BIPHOMSTO}H$. We find that it suffices to study bipartite graphs in the class denoted $\mathcal{G}^{*p}_{\text{bip}}$ consisting of bipartite graphs without automorphisms of order p, that preserve the order of the bipartition. To this end, we conduct a rigorous structural analysis of the class of bipartite graphs that contain no induced subgraph isomorphic to $K_{3,3}\backslash\{e\}$ or *domino* (see Figure 1 for an illustration). Our insights on the structure of bipartite graphs allow us to establish the following theorem.

▶ **Theorem 1.3.** *Let p be a prime and $H \in \mathcal{G}^{*p}_{\text{bip}}$ be a $(K_{3,3}\backslash\{e\}$, *domino*$)$-free graph. If there exists a connected component of H that is not a complete bipartite graph, then $\#_p\text{BIPHOMSTO}H$ is $\#_p\,\mathsf{P}$-hard.*

In many cases, a *domino* as induced subgraph yields a pair of vertices x, y where x *dominates* y. The class of bipartite domination-free $K_{3,3}\backslash\{e\}$-free graphs is one of the focal points of the seminal work by Feder and Vardi [22]. They showed that the class of *graph*

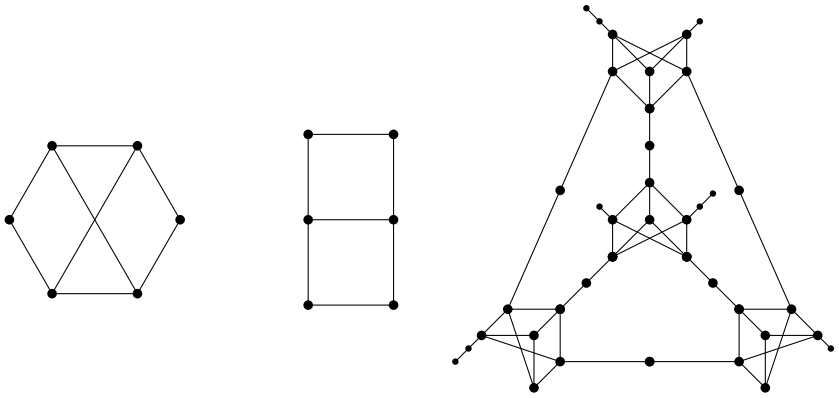


Figure 1 From left to right: $K_{3,3}\backslash\{e\}$; *domino*; Example of a bipartite ($K_{3,3}\backslash\{e\}$, *domino*)-free and asymmetric graph containing locally and globally K_4 as a minor.

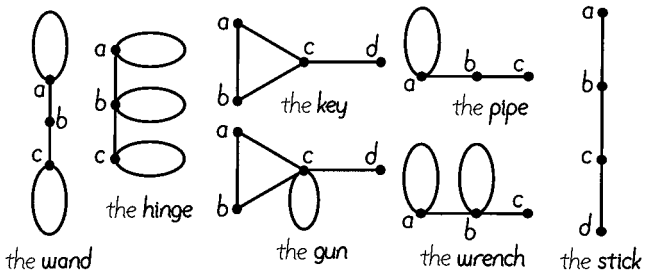


Figure 2 Depiction of the seven minimal fertile graphs as given in Brightwell and Winkler [5, Fig. 6].

retract problems – a notion equivalent to a partially labelled graph homomorphism – to the class of bipartite domination-free $K_{3,3}\backslash\{e\}$-free graphs contains as much computational power as the whole class of *constraint satisfaction problem*s (CSP's), i.e. every CSP is polynomially equivalent to a partially labelled graph homomorphism problem, where the image is a bipartite domination-free $K_{3,3}\backslash\{e\}$-free graph.

Consider the graphs studied in the work of Brightwell and Winkler [5] shown in Figure 2. The set of graph homomorphisms to these graphs played a key role in their study of spin systems in statistical physics. Prior results incorporate only two out of the seven minimal fertile graphs: "the stick" and "the key". Following the line of argumentation, our results incorporate the previous and three additional minimal fertile graphs. The only missing ones are "the hinge" and "the gun" as the construction used for bipartization yields graphs that are not *domino*-free. In fact, the class of bipartite ($K_{3,3}\backslash\{e\}$, *domino*)-free graphs captures all the classes of bipartite graphs studied in previous works on $\#_2\text{HomsTo}H$ and $\#_p\text{HomsTo}H$ except for the recent work by Focke et al. [23] on K_4-minor-free graphs. Clearly, every biclique with at least 3 vertices in each part contains a K_4 as minor, as is the case with $K_{3,3}\backslash\{e\}$. A *domino* is K_4-minor-free. Hence, our result given by a local property is orthogonal to the result of Focke et al. [23] given by a global property. An example is depicted in Figure 1.

1.2 Technical Overview

We are going to explain our results and argumentative routes in more detail. Due to their length, details are omitted but can be found in the full version.

In this work, hardness for modular counting problems is indicated by reducing from problems that are $\#_p\,\mathsf{P}$-hard. The class $\#_p\,\mathsf{P}$ contains functions of the form "$f \mod p$", where $f \in \#\,\mathsf{P}$. Notably, for the case $p = 2$ the whole polynomial hierarchy reduces to problems in $\#_2\,\mathsf{P}$ by Toda [46].

We briefly discuss the insights by Faben and Jerrum [21]. For a pair of graphs G, H we denote by $\mathrm{Hom}(G \to H)$ the set of homomorphisms from G to H. By $\hom(G \to H)$ we denote the cardinality $|\mathrm{Hom}(G \to H)|$ and, for a modulus p, by $\hom_p(G \to H)$ we denote $\hom(G \to H) \pmod p$. The computational problem $\#\textsc{HomsTo}H$ with parameter H then asks to compute $\hom(G \to H)$ for an input G. Similarly, $\#_p\textsc{HomsTo}H$ asks to compute $\hom_p(G \to H)$. A central point in the study of $\#_p\textsc{HomsTo}H$ is the (non)-existence of automorphisms of order p, where for $p = 2$ these automorphisms are called *involutions*. Given an automorphism ϱ of order p acting as a derangement on the subset $V' \subseteq V(H)$, Faben and Jerrum [21] showed that the number of homomorphisms σ from any input graph G to H is equivalent to 0 in $\mathbb{Z}_p$ if the image of σ intersects V'. They deduced that, for the subgraph H^ϱ of H induced by the fixpoints $V(H)\backslash V'$, there exists a parsimonious reduction from $\#_p\textsc{HomsTo}H^\varrho$ to $\#_p\textsc{HomsTo}H$. Iteratively applying this reduction, one ends up with a subgraph H^{*p} of H that admits no automorphism of order p called the *order p reduced* form of H. This subgraph is also unique up to ismomorphism and thus well defined by [21, Theorem 3.7]. The study of $\#_p\textsc{HomsTo}H$ focusses on graphs H that do not admit automorphisms of order p, which are called *order p reduced*.

1.2.1 Quantum Homomorphisms

It has been observed by Borgs, Chayes, Kahn, and Lovász [3] that the study of linear combinations of homomorphisms provides great insights especially on the comparability of pairs of graphs, for instance if one is a subgraph of the other. Lovász [38] introduced the term *quantum graph*, denoted $\bar{H}$, for a linear combination of finitely many graphs and calls the set of pairwise non-isomorphic graphs $\mathcal{H}$ with coefficient $\alpha_H \neq 0$ in $\bar{H}$ its *constituents*:

$$\bar{H} = \sum_{H \in \mathcal{H}} \alpha_H\, H.$$

A computational problem on $\bar{H}$ translates into the linear combination of computational problems on entities H in $\mathcal{H}$ with coefficient α_H. By Lovász [38] every graph parameter has – if any – a unique expression by a linear combination of finitely many graph homomorphisms up to isomorphisms.

We establish a polynomial time reduction from $\#_p\textsc{HomsTo}H$ to $\#_p\textsc{HomsTo}\bar{H}$, for any quantum graph $\bar{H}$ and any constituent H in $\bar{H}$. Such a polynomial time reduction is commonly referred to as a *pinning*-reduction because it enables us to consider the subproblem where a partially mapping is already fixed. One of the main problems of reduction algorithms on modular counting problems is the loss of control of summations in a finite field because we cannot infer from a number of non-zero summands that the sum is non-zero. For instance, let $p = 2$ and $\bar{H}$ be the quantum graph consisting of the two graphs H_1 and H_2 with coefficients $\alpha_{H_1} = \alpha_{H_2} = 1$, where H_1 is an asymmetric tree and H_2 is the disjoint union of a copy of H_1 and an isolated vertex. Let G be a connected graph and input for $\#\textsc{HomsTo}\bar{H}$, then we obtain $\hom(G \to H_2) = \hom(G \to H_1) + \hom(G \to K_1)$. Consequently, when computing $\hom(G \to H_1) + \hom(G \to H_2)$ in $\mathbb{Z}_2$ the term referring to H_1 vanishes and this amounts to computing $\hom(G \to K_1)$, which is polynomial time solvable. However, Theorem 1.1 yields that $\#_2\textsc{HomsTo}\bar{H}$ is $\#_2\,\mathsf{P}$-hard. The reason is that the split into $\hom(G \to H_1) + \hom(G \to K_1)$ only works if G is connected and by utilizing disconnected graphs the additional vertex in H_2 yields enough information to distinguish between H_1 and H_2. Therefore, we can extract $\hom_p(G \to H_1)$ from $\hom_p(G \to \bar{H})$.

In finite fields, reduction algorithms usually rely heavily on multiplication. We find that the beautiful insight on specific matrices defined on families $\mathcal{F}$ of simple graphs provided by Borgs, Chayes, Kahn, and Lovász [3, Lemma 4.2] is able to lift us above this hurdle. In order to adapt this result we first extend it to allow for graphs that contain loops. Then, we translate the result to counting in a finite field of prime order. A straightforward application of the modulo operator is not sufficient as the graphs in $\mathcal{F}$ might contain a number of automorphisms that is a multiple of p. We restrict to order p reduced graphs and argue why this allows for an application of the modulo operator. In this way, we show the following.

▶ **Corollary 1.4.** *Let $k \geqslant 1$ and let $\mathcal{F} = \{F_1, \ldots, F_k\}$ be a finite family of non-isomorphic order p reduced graphs closed under surjective homomorphic image, that contain no multi-edge. Then the matrix*

$$M_{hom} = \left[\hom_p(F_i \to F_j)\right]_{i,j=1}^{k}$$

is nonsingular.

The strength of this result for our purposes is twofold. First, it allows us to show Theorem 1.1 in a concise manner. Given a quantum graph $\bar{F}$ with set of p-constituents $\mathcal{F} = (F_1, \ldots, F_r)$ closed under surjective homomorphic image, we obtain by Corollary 1.4 that any system of linear equations of the form $\bar{x}\, M_{hom} = \bar{v}$ has a unique solution in the field $\mathbb{Z}_p$. Therefore, for any vector $\bar{v}$ with entries $(v_i)_{i\in[k]}$ there exists a unique linear combination of entities in $\mathcal{F}$ with coefficients α_F that yield the vector $\bar{v}$. In fact, we observe that this corresponds to a quantum graph $\bar{F}'$ with $\hom_p(\bar{F}' \to F_i) = v_i$ that *implements* the vector $\bar{v}$. In particular, there exists a quantum graph $\bar{F}'$ implementing the i-th standard vector allowing us to "pick" the i-th entry of $\mathcal{F}$, i.e. $\hom_p(\bar{F}' \to F_j) = 1$ if $j = i$ and $\hom_p(\bar{F}' \to F_j) = 0$ otherwise. Given an input graph G for $\hom_p(G \to \bar{F})$, we then construct a quantum graph $\bar{F}^*$ from G and $\bar{F}'$ such that $\hom_p(\bar{F}^* \to \bar{F}) = \hom_p(G \to F_i)$. The main problem for this application is then that the set $\mathcal{F}$ of p-constituents might not be closed under surjective homomorphic image. Given any quantum graph $\bar{H}$ with set of p-constituents $\mathcal{H}$, we need to define a suitable family $\mathcal{F}$ that contains all the image graphs needed. We find that the subgraphs of the maximal closure are sufficient for this purpose and obtain Theorem 1.1.

The second strength is the adaptability to subproblems of homomorphisms. The main property needed is that the subset of homomorphisms has to be closed under composition, i.e. the subset is actually a *subgroup* of the group of homomorphisms. Examples are *vertex surjective homomorphisms* and *compactions* as studied by Focke et al. [24]. A homomorphism $\sigma \in \mathrm{Hom}(G \to H)$ is *vertex surjective* if the image-set of σ is the whole set $V(H)$. The homomorphism σ is a *compaction* if it is vertex surjective and every non-loop edge e is in the image of σ. A closely related example is the problem of counting *partially labelled* homomorphisms $\#\textsc{PartLabHomsTo}H$, that are homomorphisms from an input graph G to H that have to respect a given mapping from a subset $V_G \subset V(G)$ to a subset $V_H \subset V(G)$ and are also referred to as *retractions* (see e.g. Focke et al. [24]). The reduction from $\#_p\textsc{PartLabHomsTo}H$ to $\#_p\textsc{HomsTo}H$ is a building stone of every paper in the study of $\#_p\textsc{HomsTo}H$ and can be obtained in a swift manner due to the strength of Corollary 1.4. A third example will be discussed in the next subsection.

1.2.2 Bipartization

Chen et al. [12] employed the tensor product $H \otimes K_2 = H'$ to reduce to $\#\textsc{HomsTo}H$ from $\#\textsc{HomsTo}H'$, where H' is bipartite. The main problem when adapting this construction to modular counting $\#_p\textsc{HomsTo}H$ is that for every graph G the number of homomorphisms

$\hom_2(G \to K_2)$ is 0 and thus the tensor product with K_2 annihilates seemingly any structure that might imply hardness. Instead of branching the study of $\#_p\text{HomsTo}H$ into one studying the modulus 2 and one studying the modulus of odd primes, we solve this issue in a uniform way for all prime moduli.

The key insight towards this is that for an involution-free graph H the tensor product $H' = H \otimes K_2$ only yields involutions on H' that exchange the parts of the bipartition. A very important example is the graph H consisting of a single edge with one loop, for which it is known that $\#\text{HomsTo}H$ is equivalent to counting the number of independent sets. The graph $H' = H \otimes K_2$ is then the path with 4 vertices (see Figure 2), that admits only the reflection across the middle edge. It is known that $\#\text{HomsTo}H'$ is equivalent to counting the number of bipartite independent sets #BIS and also that $\#_4\text{HomsTo}H'$ is $\#_2$ P-hard (see [28]) whereas $\#_2\text{HomsTo}H'$ is polynomial time solvable. In order to evade the artificial involutions yielded by the tensor product with K_2 we introduce the study on the problem of counting homomorphisms between bipartite graphs that preserve the order of the bipartition denoted $\#\text{BipHomsTo}H$. For example, if H is the path with 4 vertices then $\#_2\text{BipHomsTo}H$ is equivalent to $\#_4\text{HomsTo}H$.

▶ **Lemma 1.5.** *Let p be a prime, let H be a graph, and let $H' = H \otimes K_2$. Then, $\#_p\text{BipHomsTo}H'$ reduces to $\#_p\text{HomsTo}H$ under parsimonious reduction.*

We note that the graph $H' = H \otimes K_2$ is a collection of complete bipartite graphs if and only if H satisfies the Dyer and Greenhill criterion, for these graphs $\#_p\text{BipHomsTo}H'$ is solvable in polynomial time.

The reduction from $\#_p\text{BipHomsTo}H$ has the downside that the machinery developed over the course of multiple papers on $\#_p\text{HomsTo}H$ is not stated for the subgroup of homomorphisms counted by $\#_p\text{BipHomsTo}H$. We remedy this. First, by the strong adaptability of Corollary 1.4 and the subsequent reduction algorithm we obtain pinning for the problem $\#_p\text{BipHomsTo}H$. Second, using automorphisms of order p that preserve the order of the bipartition we reduce the bipartite graph H to a *part-wise order p reduced* bipartite graph $(H)^{*p}$. We deduce that the goal towards a dichotomy for $\#\text{HomsTo}H$ is captured by Theorem 1.2. The chain of reductions is displayed below.

$$\#_p\text{BipPartLabHomsTo}H^* =_P \#_p\text{BipHomsTo}H^* \leq_{pars}^{(H^* = (H')^{*p})} \#_p\text{BipHomsTo}H' \leq_{pars}^{(H' = H \otimes K_2)} \#_p\text{HomsTo}H$$

We employ a gadgetry that establishes a reduction to $\#_p\text{BipHomsTo}H$ from a variant of #BIS with weights on both types of vertices. Such a gadgetry yielding hardness is called a *p-hardness gadget*. By an adaptation of the dichotomy for #BIS with weights on the vertices *in* the independent set given in [28] this reduction establishes hardness when counting in $\mathbb{Z}_p$ if and only if none of the weights is equivalent to 0 in $\mathbb{Z}_p$. The problem of #BIS with weights on the vertices in the independent set is established as terminal problem yielding hardness in the study of $\#_p\text{HomsTo}H$ [28, 36] as the bigger modulus implies a richer structure compared to the study of $\#_2\text{HomsTo}H$ that traditionally focusses on counting the number of independent sets.

1.2.3 Hardness in Bipartite $(K_{3,3}\backslash\{e\}, \textit{domino})$-Free Graphs

A central argument in the work of Chen et al. [12] is that for a bipartite graph H and the problem $\#\text{HomsTo}H$ there exists a simple reduction from $\#\text{HomsTo}B$, where B is the ball of radius 2 around a vertex v in H denoted $B_2(v)$, i.e. vertices of distance at most 2 from v.

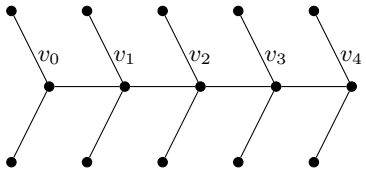


Figure 3 For $p = 3$ the tree H contains no ball of radius 2 around any vertex with enough structure to yield hardness even though $\#_3\text{HomsTo}H$ is $\#_3$ P-hard.

By an iterative application of this argument they establish a reduction from $\#\text{HomsTo}P$, where P is a generalization of the path with 4 vertices. Even though the reduction argument can be made valid for $\#_p\text{HomsTo}H$ and $\#_p\text{BipHomsTo}H$ the restriction to a substructure might destroy the properties that yield hardness already for trees H, a class of graphs for which the dichotomy is proved (see [21, 28]). An example is depicted in Figure 3.

In a nutshell, the induced subgraphs of radius at most 2 can admit too many automorphisms of order p. In Figure 3 we observe that the problem originates from too many instances of complete bipartite graphs $K_{1,b}$, where $b \equiv 0 \pmod 3$ or $b \equiv 1 \pmod 3$. The way to overcome this is to also consider the global structure. In the case displayed in Figure 3 the number of walks of length 4 from v_4 to a vertex v in the neighbourhood of v_1 is 0 (mod 3) if $v = v_2$ and 1 else. The goal is then to construct a reduction restricting the study to $B_2(v_0)\backslash\{v_2\}$, a graph that yields hardness. We do this in a general form by a second type of gadgetry called (B, p)-*gadget* that reduces $\#_p\text{BipHomsTo}H$ from $\#_p\text{BipHomsTo}B$.

As we have argued, one of the main obstacles for the study on $\#_p\text{HomsTo}H$ are complete bipartite graphs. The graph $K_{3,3}\backslash\{e\}$ denotes the graph obtained from $K_{3,3}$ by deleting an edge, and the graph *domino* denotes the graph obtained from $K_{3,3}$ by deleting two edges without introducing a cut-vertex (see Figure 1). By the restriction to ($K_{3,3}\backslash\{e\}$, *domino*)-free bipartite graphs we study exactly the case of a great many of complete bipartite induced subgraphs. To this end, we observe that for every vertex $v \in H$ the induced subgraph $B_2(v)$ splits into connected components obtained from deleting v. The *split* of $B_2(v)$ at v corresponds to the set of these connected components, where every component contains a copy of v. By the absence of induced subgraphs isomorphic to $K_{3,3}\backslash\{e\}$ or *domino* we deduce that the blocks containing v in these components have to be complete bipartite.

The overall line of argumentations towards Theorem 1.3 is then the following. First, we establish the dichotomy for all ($K_{3,3}\backslash\{e\}$, *domino*)-free bipartite graphs H of radius at most 2 by a combination of (B, p)- and p-hardness gadgets. This is done by a careful structural study of the split of H at a central vertex v. An important first result is then that any bipartite graph H in $\mathcal{G}_{\text{bip}}^{*p}$ that contains a vertex v where $B_2(v)$ falls into the hard cases of the dichotomy, is itself such that $\#_p\text{BipHomsTo}H$ is $\#_p$ P-hard. Second, we study graphs H of radius larger than 2 in order to establish enough structural information of H allowing us to construct either a p-hardness gadget for H or a (B, p)-gadget such that $\#_p\text{BipHomsTo}B$ is $\#_p$ P-hard. This second step is very long and technically involved because the class of ($K_{3,3}\backslash\{e\}$, *domino*)-free graphs allows for many cases commanding us to explore the global structure of H. Before we shed more light on how we proceed towards the second step, we display the chain of reduction arguments below, where the intermediate steps for H_i refer to H itself or an induced subgraph obtained by a (H_i, p)-gadget.

$$\#_p\text{BIS}^{\kappa_\ell,\kappa_r}_{\lambda_\ell,\lambda_r} \leq_P \#_p\text{BipHomsTo}H_k \leq_P \cdots \leq_P \#_p\text{BipHomsTo}H_1 \leq_P \#_p\text{BipHomsTo}H$$

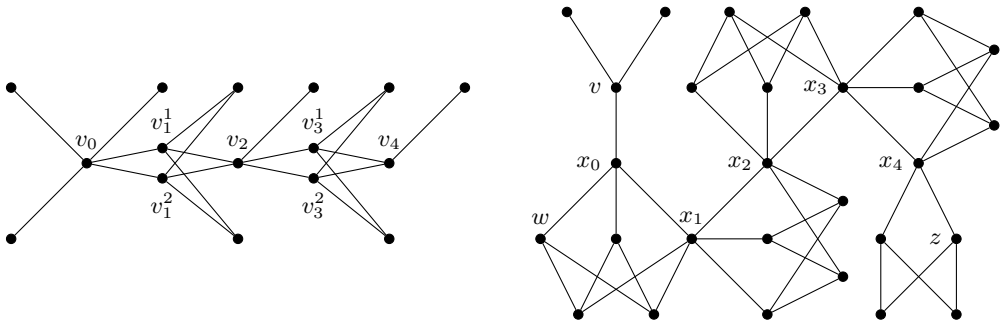


Figure 4 For $p = 3$ the left figure depicts an example of a generalized hardness path and the right figure depicts an example of a p-mosaic path.

Informally, we split the study of bipartite $(K_{3,3}\backslash\{e\}, \textit{domino})$-free graphs of radius larger than 2 into two broad cases: graphs with no pair of vertices that have a multiple of p common neighbours, and graphs with such a pair of vertices. Focusing on the first case, if the graph H contains a cycle of length at least 6 we argue that the cycle provides enough structure to show hardness. Otherwise, the more restricted structure of H renders the graph "tree-like". The "leafs" of H are vertices v, such that the split of $B_2(v)$ at v contains at most one component adjacent to a vertex not in $B_2(v)$. We call such a vertex a *dead end* and traverse the graph H along a path P starting at a dead end v. The path P is constructed such that it allows us to establish hardness depending only on the local properties of its endvertices, we call such a path P a *generalized hardness path*; an example is depicted in Figure 4. The first broad case is then established by showing that H contains a generalized hardness path whose endvertices are such that P yields hardness.

Turning towards the second broad case, we traverse the graph H again along a path P. Contrary to the first case, we can encounter a pair of vertices with a multiple of p common neighbours. For our purposes, it is important to evade such pairs of vertices. We argue that by the property of H being in $\mathcal{G}_{\text{bip}}^{*p}$ we are always able to do so. This leads to a structure we call *p-mosaic path* that, similar to a generalized hardness path, provides enough structural information towards establishing hardness depending only on the local properties of its endvertices. An example is shown in Figure 4. We find that if a p-mosaic path is a cycle then this cycle satisfies the same properties we found to be sufficient to yield hardness in the first broad case. Interestingly, we show that if the p-mosaic path does not yield hardness the only remaining option for an endvertex of a p-mosaic path is to also be the endvertex of a generalized hardness path. We conclude that H has to contain a concatenation of generalized hardness paths and p-mosaic paths. Arguing by the finiteness of H we show that this concatenation has to yield hardness.

1.2.4 Beyond $(K_{3,3}\backslash\{e\}, \textit{domino})$-Free Graphs

In light of our findings we conjecture that for a bipartite graph H in $\mathcal{G}_{\text{bip}}^{*p}$ the problem $\#_p\textsc{BipHomsTo}H$ is $\#_p\,\mathsf{P}$-hard if H is not a collection of complete bipartite graphs. This conjecture then extends towards a conjecture on $\#_p\textsc{HomsTo}H$ and also incorporates the conjecture of Faben and Jerrum.

▶ **Conjecture 1.6.** *Let p be a prime and H a graph with order p reduced form H^{*p}. Then, $\#_p\textsc{HomsTo}H$ is solvable in polynomial time if the connected components of H^{*p} are complete bipartite or reflexive complete. Otherwise, $\#_p\textsc{HomsTo}H$ is $\#_p\,\mathsf{P}$-complete.*

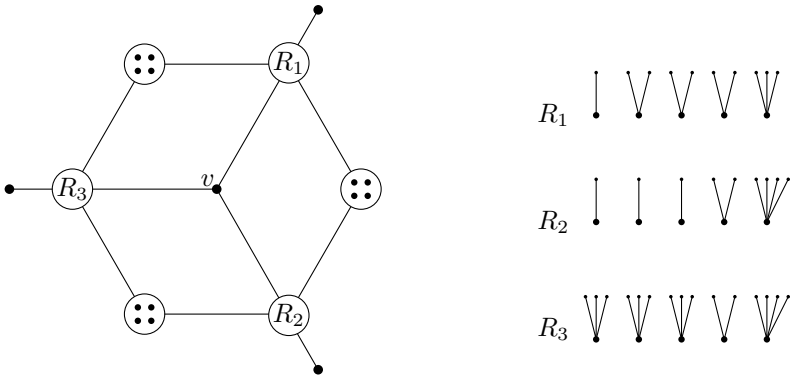


Figure 5 Illustration of a 5-Catherine wheel. Edges to encircled sets illustrate edges to every vertex in the set. The smaller substructure in the sets $R_1, \ldots, R_5$ is illustrated to the right, where the vertices in the sets R_i are the more prominent ones at the bottom of the row.

We emphasize this conjecture by a study of the set of *partially surjective homomorphisms* from a graph G to a graph H denoted $\mathrm{PartSurj}(G \to H)$. Partially surjective homomorphisms have to be surjective on a set of distinguished vertices $V^{\mathrm{dist}} \subseteq V(H)$ and a set of distinguished edges $E^{\mathrm{dist}} \subseteq E(H)$. We deduce that it suffices to study graphs H without automorphisms of order p acting bijectively on V^{dist} and E^{dist}. However, this reduction does not capture all cancellations because the graph H might still admit too many general automorphisms.

By our results on quantum graphs and an application of the inclusion-exclusion principle we find that the dichotomy presented in Conjecture 1.6 extends to a dichotomy on the whole class of partially surjective homomorphisms. Contrary to the non-modular version established in [12], this dichotomy does not state a clear structural characterisation of the hard instances due to the mentioned possibility of additional cancellations. For the special cases in which the parameter graph H is order p reduced we amplify the dichotomy such that it states clear structural characterisations. Two examples for this case are the problems $\#_p\textsc{VertSurjHomsTo}H$ and $\#_p\textsc{CompTo}H$ of counting in $\mathbb{Z}_p$ the number of vertex surjective homomorphisms and compactions, respectively. We obtain the following criteria analogous to the criteria in the non-modular setting given by Focke et al. [24].

▶ **Corollary 1.7.** *Let p be a prime and H be a graph. The problem $\#_p\textsc{VertSurjHomsTo}H$ is solvable in polynomial time if either H admits an automorphism of order p, or every connected component of H is a complete bipartite graph or a reflexive complete graph.*

The problem $\#_p\textsc{CompTo}H$ is solvable in polynomial time if either H admits an automorphism of order p, or every connected component of H is an irreflexive star or a reflexive complete graph of size at most two.

Assuming Conjecture 1.6 both problems are $\#_p\,\mathsf{P}$-hard in every other case.

In order to prove Conjecture 1.6, we need to study bipartite graphs that contain $K_{3,3}\backslash\{e\}$ or *domino* as an induced subgraph. The strong restrictions on the structure of the graphs under study are a double-edged sword. On one hand, it is more plausible to find enough structure that yields hardness. On the other hand, it is more difficult to pin the structural analysis down to a handful of cases. Furthermore, the higher moduli imply even more complexity of the structural analysis. We illustrate an especially difficult example in Figure 5.

We call such a graph as illustrated in Figure 5 a *p-Catherine wheel*. Even though these graphs are 2-connected and of radius 2 their highly symmetric global structure together with the lack of small structure in the sets R_i makes it difficult to identify sources for hardness.

Here, the case displayed in Figure 5 where the sets R_i only yield a collection of trees and the sum of degrees of the vertices in the sets is $0 \pmod p$ is especially difficult. We note that such a graph cannot be part-wise order p reduced for $p = 2, 3$, which highlights the gain of complexity due to higher moduli.

1.3 Related Literature

Before we conclude the introduction, we mention related bodies of work. The study of homomorphisms under the point of view of *parameterized algorithms* has been long established (see Diaz, Serna, and Thilikos [18]) but enriched by the work of Amini, Fomin, and Saurabh [1] and by Curticapean, Dell, and Marx [13], who also introduced linear combinations of graph homomorphisms to the study and motivated subsequent works, for instance Roth and Wellnitz [41].

The study of homomorphisms from the point of view of *extremal combinatorics* incorporates important conjectures like *Sidorenko's conjecture* [43, 44], which states a universal lower bound on the number of homomorphisms from a bipartite graph and, in a weaker version, can be found in the work of Simonovits [45]. Until today the conjecture remains open but still enjoys new contributions like the recent article by Shams, Ruozzi, and Csikvári [42].

This leads to the body of work studying *approximation algorithms* including the work of Goldberg and Jerrum [30] on tree homomorphisms and the work of Galanis, Goldberg, and Jerrum [25], who showed that approximating the number of homomorphisms to a fixed graph H is #BIS-hard, a notorious complexity class in this body of work. These findings yield an interesting connection to ours in the form of the reduction from (versions of) #BIS.

The body of studies concerning different versions of homomorphism problems is vast. It contains dichotomies for the affiliated problem, where the pre-image is from a fixed class of graphs, given by Dalmau and Jonsson [14] and Grohe [32]. Turning towards versions of the problem with fixed image, Focke, Goldberg, and Zivný [24] gave a dichotomy for surjective homomorphisms and compactions, and Dyer, Goldberg, and Paterson [15] gave a dichotomy for directed homomorphisms if the target is acyclic. The line of research towards the dichotomy for the generalization of #HomsToH allowing weights by Cai, Chen, and Lu [9] incorporates works by Bulatov and Grohe [7] and Goldberg, Grohe, Jerrum, and Thurley [29]. Recently, Govorov, Cai, and Dyer [31] extended this research body.

The connection of homomorphisms and CSP's was already shown by Feder and Vardi [22]. Bulatov [6] showed that the problem of counting satisfying assignments to a CSP enjoys a dichotomy theorem, a result on which Dyer and Richerby [17] shed more light. Furthermore, a complete dichotomy for directed homomorphism can be found in the dichotomy on counting weighted versions of CSP's by Cai and Chen [8]. Guo, Huang, Lu, and Xia [34] gave a dichotomy for the associated modular problem.

Finally, the recent work by Cai and Govorov [10] studied the power of expression of the class of homomorphisms. By studying algebras of quantum graphs they provide a general technique and showed, for instance, that the problem of counting perfect matchings cannot be expressed by counting homomorphisms to a fixed graph H regardless of possible weights.

References

1 Omid Amini, Fedor V. Fomin, and Saket Saurabh. Counting subgraphs via homomorphisms. *SIAM J. Discret. Math.*, 26(2):695–717, 2012. doi:10.1137/100789403.

2 Christian Borgs, Jennifer Chayes, László Lovász, Vera T. Sós, and Katalin Vesztergombi. Counting graph homomorphisms. In *Topics in Discrete Mathematics*, pages 315–371, 2006. doi:10.1007/3-540-33700-8_18.

3 Christian Borgs, Jennifer T. Chayes, Jeff Kahn, and László Lovász. Left and right convergence of graphs with bounded degree. *Random Structures and Algorithms*, 42(1):1–28, 2013. doi:10.1002/rsa.20414.

4 Christian Borgs, Jennifer T. Chayes, László Lovász, Vera T. Sós, Balázs Szegedy, and Katalin Vesztergombi. Graph limits and parameter testing. In *Proceedings of STOC 2006*, pages 261–270, 2006. doi:10.1145/1132516.1132556.

5 Graham R. Brightwell and Peter Winkler. Graph homomorphisms and phase transitions. *J. Comb. Theory, Ser. B*, 77(2):221–262, 1999. doi:10.1006/jctb.1999.1899.

6 Andrei A. Bulatov. The complexity of the counting constraint satisfaction problem. *J. ACM*, 60(5):34:1–34:41, 2013. doi:10.1145/2528400.

7 Andrei A. Bulatov and Martin Grohe. The complexity of partition functions. *Theor. Comput. Sci.*, 348(2-3):148–186, 2005. doi:10.1016/j.tcs.2005.09.011.

8 Jin-Yi Cai and Xi Chen. Complexity of counting CSP with complex weights. *J. ACM*, 64(3):19:1–19:39, 2017. doi:10.1145/2822891.

9 Jin-Yi Cai, Xi Chen, and Pinyan Lu. Graph homomorphisms with complex values: A dichotomy theorem. *SIAM J. Comput.*, 42(3):924–1029, 2013. doi:10.1137/110840194.

10 Jin-Yi Cai and Artem Govorov. Perfect matchings, rank of connection tensors and graph homomorphisms. In *Proceedings of SODA 2019*, pages 476–495, 2019. doi:10.1137/1.9781611975482.30.

11 Ashok K. Chandra and Philip M. Merlin. Optimal implementation of conjunctive queries in relational data bases. In *Proceedings of STOC 1977*, pages 77–90, 1977. doi:10.1145/800105.803397.

12 Hubie Chen, Radu Curticapean, and Holger Dell. The exponential-time complexity of counting (quantum) graph homomorphisms. In *Proceedings of WG 2019*, pages 364–378, 2019. doi:10.1007/978-3-030-30786-8_28.

13 Radu Curticapean, Holger Dell, and Dániel Marx. Homomorphisms are a good basis for counting small subgraphs. In *Proceedings of STOC 2017*, page 210–223, 2017. doi:10.1145/3055399.3055502.

14 Víctor Dalmau and Peter Jonsson. The complexity of counting homomorphisms seen from the other side. *Theor. Comput. Sci.*, 329(1-3):315–323, 2004. doi:10.1016/j.tcs.2004.08.008.

15 Martin E. Dyer, Leslie Ann Goldberg, and Mike Paterson. On counting homomorphisms to directed acyclic graphs. *J. ACM*, 54(6):27, 2007. doi:10.1145/1314690.1314691.

16 Martin E. Dyer and Catherine Greenhill. The complexity of counting graph homomorphisms. *Random Structures and Algorithms*, 17(3-4):260–289, 2000. doi:10.1002/1098-2418(200010/12)17:3/4<260::AID-RSA5>3.0.CO;2-W.

17 Martin E. Dyer and David Richerby. An effective dichotomy for the counting constraint satisfaction problem. *SIAM J. Comput.*, 42(3):1245–1274, 2013. doi:10.1137/100811258.

18 Josep Díaz, Maria Serna, and Dimitrios M. Thilikos. Counting H-colorings of partial k-trees. *Theor. Comput. Sci.*, 281(1):291–309, 2002. Selected Papers in honour of Maurice Nivat. doi:10.1016/S0304-3975(02)00017-8.

19 Ethan R. Elenberg, Karthikeyan Shanmugam, Michael Borokhovich, and Alexandros G. Dimakis. Beyond triangles: A distributed framework for estimating 3-profiles of large graphs. In *Proceedings of KDD 2015*, pages 229–238, 2015. doi:10.1145/2783258.2783413.

20 Ethan R. Elenberg, Karthikeyan Shanmugam, Michael Borokhovich, and Alexandros G. Dimakis. Distributed estimation of graph 4-profiles. In *Proceedings of WWW 2016*, pages 483–493, 2016. doi:10.1145/2872427.2883082.

21 John Faben and Mark Jerrum. The complexity of parity graph homomorphism: An initial investigation. *Theory of Computing*, 11:35–57, 2015. doi:10.4086/toc.2015.v011a002.

22 Tomás Feder and Moshe Y. Vardi. The computational structure of monotone monadic snp and constraint satisfaction: A study through datalog and group theory. *SIAM J. Comput.*, 28(1):57–104, 1998. doi:10.1137/S0097539794266766.

23 Jacob Focke, Leslie Ann Goldberg, Marc Roth, and Stanislav Zivný. Counting homomorphisms to K_4-minor-free graphs, modulo 2. In *Proceedings of SODA 2021*, pages 2303–2314, 2021. doi:10.1137/1.9781611976465.137.

24 Jacob Focke, Leslie Ann Goldberg, and Stanislav Zivný. The complexity of counting surjective homomorphisms and compactions. *SIAM J. Discret. Math.*, 33(2):1006–1043, 2019. doi:10.1137/17M1153182.

25 Andreas Galanis, Leslie Ann Goldberg, and Mark Jerrum. Approximately counting H-colorings is #BIS-hard. *SIAM J. Comput.*, 45(3):680–711, 2016. doi:10.1137/15M1020551.

26 Andreas Göbel, Leslie Ann Goldberg, and David Richerby. The complexity of counting homomorphisms to cactus graphs modulo 2. *ACM Trans. Comput. Theory*, 6(4):17:1–17:29, 2014. doi:10.1145/2635825.

27 Andreas Göbel, Leslie Ann Goldberg, and David Richerby. Counting homomorphisms to square-free graphs, modulo 2. *ACM Trans. Comput. Theory*, 8(3):12:1–12:29, 2016. doi:10.1145/2898441.

28 Andreas Göbel, J. A. Gregor Lagodzinski, and Karen Seidel. Counting homomorphisms to trees modulo a prime. In *Proceedings of MFCS 2018*, pages 49:1–49:13, 2018. doi:10.4230/LIPIcs.MFCS.2018.49.

29 Leslie Ann Goldberg, Martin Grohe, Mark Jerrum, and Marc Thurley. A complexity dichotomy for partition functions with mixed signs. *SIAM J. Comput.*, 39(7):3336–3402, 2010. doi:10.1137/090757496.

30 Leslie Ann Goldberg and Mark Jerrum. The complexity of approximately counting tree homomorphisms. *ACM Trans. Comput. Theory*, 6(2):8:1–8:31, 2014. doi:10.1145/2600917.

31 Artem Govorov, Jin-Yi Cai, and Martin E. Dyer. A dichotomy for bounded degree graph homomorphisms with nonnegative weights. In *Proceedings of ICALP 2020*, pages 66:1–66:18, 2020. doi:10.4230/LIPIcs.ICALP.2020.66.

32 Martin Grohe. The complexity of homomorphism and constraint satisfaction problems seen from the other side. *J. ACM*, 54(1):1:1–1:24, 2007. doi:10.1145/1206035.1206036.

33 Martin Grohe, Thomas Schwentick, and Luc Segoufin. When is the evaluation of conjunctive queries tractable? In *Proceedings of STOC 2001*, pages 657–666, 2001. doi:10.1145/380752.380867.

34 Heng Guo, Sangxia Huang, Pinyan Lu, and Mingji Xia. The complexity of weighted boolean #csp modulo k. In *Proceedings of STACS 2011*, pages 249–260, 2011. doi:10.4230/LIPIcs.STACS.2011.249.

35 Pavol Hell and Jaroslav Nešetřil. On the complexity of H-coloring. *J. Comb. Theory, Ser. B*, 48(1):92–110, 1990. doi:10.1016/0095-8956(90)90132-J.

36 Amirhossein Kazeminia and Andrei A. Bulatov. Counting homomorphisms modulo a prime number. In *Proceedings of MFCS 2019*, pages 59:1–59:13, 2019. doi:10.4230/LIPIcs.MFCS.2019.59.

37 Richard E. Ladner. On the structure of polynomial time reducibility. *J. ACM*, 22(1):155–171, 1975. doi:10.1145/321864.321877.

38 László Lovász. *Large Networks and Graph Limits*, volume 60 of *Colloquium Publications*. AMS, 2012. URL: http://www.ams.org/bookstore-getitem/item=COLL-60.

39 Benjamin Rossman. Homomorphism preservation theorems. *J. ACM*, 55(3):15:1–15:53, 2008. doi:10.1145/1379759.1379763.

40 Benjamin Rossman. An improved homomorphism preservation theorem from lower bounds in circuit complexity. In *Proceedings of ITCS 2017*, pages 27:1–27:17, 2017. doi:10.4230/LIPIcs.ITCS.2017.27.

41 Marc Roth and Philip Wellnitz. Counting and finding homomorphisms is universal for parameterized complexity theory. In *Proceedings of SODA 2020*, pages 2161–2180, 2020. doi:10.1137/1.9781611975994.133.

42 Shahab Shams, Nicholas Ruozzi, and Peter Csikvári. Counting homomorphisms in bipartite graphs. In *Proceedings of ISIT 2019*, pages 1487–1491, 2019. doi:10.1109/ISIT.2019.8849389.

43 Alexander F. Sidorenko. Inequalities for functionals generated by bipartite graphs. *(Russian) Diskret. Mat*, 3(3):50–65, 1991.

44 Alexander F. Sidorenko. A correlation inequality for bipartite graphs. *Graphs and Combin.*, 9(2-4):201–204, 1993. doi:10.1007/BF02988307.

45 Miklós Simonovits. Extremal graph problems, degenerate extremal problems,and supersaturated graphs. *Progress in Graph Theory*, page 419–437, 1984.

46 Seinosuke Toda. PP is as hard as the polynomial-time hierarchy. *SIAM J. Comput.*, 20(5):865–877, 1991. doi:10.1137/0220053.

47 Johan Ugander, Lars Backstrom, and Jon M. Kleinberg. Subgraph frequencies: mapping the empirical and extremal geography of large graph collections. In *Proceedings of WWW 2013*, pages 1307–1318, 2013. doi:10.1145/2488388.2488502.

48 Leslie G. Valiant. Accidental algorithms. In *Proceedings of FOCS 2006*, pages 509–517, 2006. doi:10.1109/FOCS.2006.7.

Minimum Stable Cut and Treewidth

Michael Lampis
Université Paris-Dauphine, PSL University, CNRS, LAMSADE, 75016, Paris, France

Abstract

A stable or locally-optimal cut of a graph is a cut whose weight cannot be increased by changing the side of a single vertex. Equivalently, a cut is stable if all vertices have the (weighted) majority of their neighbors on the other side. Finding a stable cut is a prototypical PLS-complete problem that has been studied in the context of local search and of algorithmic game theory.

In this paper we study Min Stable Cut, the problem of finding a stable cut of minimum weight, which is closely related to the Price of Anarchy of the Max Cut game. Since this problem is NP-hard, we study its complexity on graphs of low treewidth, low degree, or both. We begin by showing that the problem remains weakly NP-hard on severely restricted trees, so bounding treewidth alone cannot make it tractable. We match this hardness with a pseudo-polynomial DP algorithm solving the problem in time $(\Delta \cdot W)^{O(\mathrm{tw})} n^{O(1)}$, where tw is the treewidth, Δ the maximum degree, and W the maximum weight. On the other hand, bounding Δ is also not enough, as the problem is NP-hard for unweighted graphs of bounded degree. We therefore parameterize Min Stable Cut by both tw and Δ and obtain an FPT algorithm running in time $2^{O(\Delta \mathrm{tw})}(n + \log W)^{O(1)}$. Our main result for the weighted problem is to provide a reduction showing that both aforementioned algorithms are essentially optimal, even if we replace treewidth by pathwidth: if there exists an algorithm running in $(nW)^{o(\mathrm{pw})}$ or $2^{o(\Delta \mathrm{pw})}(n + \log W)^{O(1)}$, then the ETH is false. Complementing this, we show that we can, however, obtain an FPT *approximation scheme* parameterized by treewidth, if we consider almost-stable solutions, that is, solutions where no single vertex can unilaterally increase the weight of its incident cut edges by more than a factor of $(1 + \varepsilon)$.

Motivated by these mostly negative results, we consider Unweighted Min Stable Cut. Here our results already imply a much faster exact algorithm running in time $\Delta^{O(\mathrm{tw})} n^{O(1)}$. We show that this is also probably essentially optimal: an algorithm running in $n^{o(\mathrm{pw})}$ would contradict the ETH.

2012 ACM Subject Classification Mathematics of computing → Graph algorithms; Theory of computation → Parameterized complexity and exact algorithms

Keywords and phrases Treewidth, Local Max-Cut, Nash Stability

Digital Object Identifier 10.4230/LIPIcs.ICALP.2021.92

Category Track A: Algorithms, Complexity and Games

Related Version *Full Version*: https://arxiv.org/abs/2104.13097

Funding The author is supported by ANR JCJC project "ASSK" (ANR-18-CE40-0025-01).

Acknowledgements Part of this work was conducted while the author was on sabbatical at IRIF, UMR 8243, Université de Paris.

1 Introduction

In this paper we study problems related to *stable cuts* in graphs. A *stable* cut of an edge-weighted graph $G = (V, E)$ is a partition of V into two sets V_0, V_1 that satisfies the following property: for each $i \in \{0, 1\}$ and $v \in V_i$, the total weight of edges incident on v whose other endpoint is in V_{1-i} is at least half the total weight of all edges incident on v. In other words, a cut is stable if all vertices have the (weighted) majority of their incident edges cut.

The notion of stable cuts has been very widely studied from two different points of view. First, in the context of local search, a stable cut is a locally optimal cut: switching the side of any single vertex cannot increase the total weight of the cut. Hence, stable cuts have

been studied with the aim to further our understanding of the basic local search heuristic for Max Cut. Second, in the context of algorithmic game theory a Max Cut game has often been considered, where each vertex is an agent whose utility is the total weight of edges connecting it to the other side. In this game, a stable cut corresponds exactly to the notion of a Nash equilibrium, that is, a state where no agent has an incentive to change her choice. The complexity of producing a Nash stable or locally optimal cut of a given edge-weighted graph has been heavily studied under the name Local Max Cut. The problem is known to be PLS-complete, under various restrictions (we give detailed references below).

In this paper we focus on a different but closely related optimization problem: given an edge-weighted graph we would like to produce a stable cut *of minumum total weight*. We call this problem Min Stable Cut. In addition to being a fairly natural problem on its own, we believe that Min Stable Cut is interesting from the perspective of both local search and algorithmic game theory. In the context of local search, Min Stable Cut is the problem of bounding the performance of the local search heuristic on a particular instance. It is folklore (and easy to see) that in general there exist graphs where the smallest stable cut has size half the maximum cut (e.g. consider a C_4) and this is tight since any stable cut must cut at least half the total edge weight. However, for most graphs this bound is far from tight. Min Stable Cut therefore essentially asks to estimate the ratio between the largest and smallest stable cut for a given specific instance. Similarly, in the context of algorithmic game theory, solving Min Stable Cut is essentially equivalent to calculating the Price of Anarchy of the Max Cut game on the given instance, that is, the ratio between the smallest stable cut and the maximum cut. Since we will mostly focus on cases where Max Cut is tractable, Min Stable Cut can, therefore, be seen as the problem of computing either the approximation ratio of local search or the Price of Anarchy of the Max Cut game on a given graph.

Our results. It appears that little is currently known about the complexity of Min Stable Cut. However, since finding a (not necessarily minimum) stable cut is PLS-complete, finding the minimum such cut would be expected to be hard. Our focus is therefore to study the parameterized complexity of Min Stable Cut using structural parameters such as treewidth and the maximum degree of the input graph[1]. Our results are the following.

- First, we show that bounding only one of the two mentioned parameters is not sufficient to render the problem tractable. This is not suprising for the maximum degree Δ, where a reduction from Max Cut allows us to show the problem is NP-hard for $\Delta \leq 6$ even in the unweighted case (Theorem 4). It is, however, somewhat more disappointing that bounded treewidth also does not help, as the problem remains weakly NP-hard on trees of diameter 4 (Theorem 1) and bipartite graphs of vertex cover 2 (Theorem 3).
- These hardness results point to two directions for obtaining algorithms for Min Stable Cut: first, since the problem is "only" weakly NP-hard for bounded treewidth one could hope to obtain a pseudo-polynomial time algorithm in this case. We show that this is indeed possible and the problem is solvable in time $(\Delta \cdot W)^{O(\mathrm{tw})} n^{O(1)}$, where W is the maximum edge weight (Theorem 5). Second, one may hope to obtain an FPT algorithm when both tw and Δ are parameters. We show that this is also possible and obtain an algorithm with complexity $2^{O(\Delta \mathrm{tw})}(n + \log W)^{O(1)}$ (Theorem 6).
- These two algorithms lead to two further questions. First, can the $(\Delta \cdot W)^{O(\mathrm{tw})} n^{O(1)}$ algorithm be improved to an FPT dependence on tw, that is, to running time $f(\mathrm{tw})(nW)^{O(1)}$? And second, can the $2^{\Delta \mathrm{tw}}$ parameter dependence of the FPT algorithm be improved,

[1] We assume familiarity with the basics of parameterized complexity as given in standard textbooks [21].

for example to $2^{O(\Delta+\mathrm{tw})}$ or even $\Delta^{O(\mathrm{tw})}$? We show that the answer to both questions is negative, even if we replace treewidth with pathwidth: under the ETH there is no algorithm running in $(nW)^{o(\mathrm{pw})}$ or $2^{o(\Delta \mathrm{tw})}(n + \log W)^{O(1)}$ (Theorem 8).

- Complementing the above, we show that the problem does become FPT by treewidth alone if we allow the notion of approximation to be used in the concept of stability: there exists an algorithm which, for any $\varepsilon > 0$, runs in time $(\mathrm{tw}/\varepsilon)^{O(\mathrm{tw})}(n + \log W)^{O(1)}$ and produces a cut with the following properties: all vertices are $(1 + \epsilon)$-stable, that is, no vertex can unilaterally increase its incident cut weight by more than a factor of $(1 + \varepsilon)$; the cut has weight at most equal to that of the minimum stable cut.
- Finally, motivated by the above mostly negative results, we also consider Unweighted Min Stable Cut, the restriction of the problem where all edge weights are uniform. Our previous results give a much faster algorithm with parameter dependence $\Delta^{O(\mathrm{tw})}$, rather than $2^{\Delta \mathrm{tw}}$ (Corollary 12). However, this poses the natural question if in this case the problem finally becomes FPT by treewidth alone. Our main result in this part is to answer this question in the negative and show that, under the ETH, Unweighted Min Stable Cut cannot be solved in time $n^{o(\mathrm{pw})}$ (Theorem 13).

Taken together, our results paint a detailed picture of the complexity of Min Stable Cut parameterized by tw and Δ. All our exact algorithms (Theorems 5, 6) are obtained using standard dynamic programming on tree decompositions, the only minor complication being that for Theorem 6 we edit the decomposition to make sure that for each vertex some bag contains all of its neighborhood (this helps us verify that a cut is stable). The main technical challenge is in proving our complexity lower bounds. It is therefore perhaps somewhat surprising that the lower bounds turn out to be essentially tight, as this indicates that for Min Stable Cut and Unweighted Min Stable Cut, the straightforward DP algorithms are essentially optimal, if one wants to solve the problem exactly.

For the approximation algorithm, we rely on two rounding techniques: one is a rounding step similar to the one that gives an FPTAS for Knapsack by truncating weights so that the maximum weight is polynomially bounded. However, Min Stable Cut is more complicated than Knapsack, as an edge which is light for one of its endpoints may be heavy for the other. We therefore define a more general version of the problem, allowing us to decouple the contribution each edge makes to the stability of each endpoint. This helps us bound the largest stability-weight by a polynomial, but is still not sufficient to obtain an FPT algorithm, as the lower bound of Theorem 8 applies to polynomially bounded weights. We then go on to apply a technique introduced in [46] (see also [2, 10, 43, 44]) which allows us to obtain FPT approximation algorithms for problems which are W-hard by treewidth by applying a different notion of rounding to the dynamic program. This allows us to produce a solution that is simultaneously of optimal weight (compared to the best stable solution) and almost-stable, using essentially the same algorithm as in Theorem 5. However, it is worth noting that in general there is no obvious way to transform almost-stable solutions to stable solutions [11, 17], so our algorithm is not immediately sufficient to obtain an FPT approximation for Min Stable Cut if we insist on obtaining a cut which is exactly stable.

Related work. From the point of view of local search algorithms, there is an extensive literature on the Local Max Cut problem, which asks us to find a stable cut (of any size). The problem has long been known to be PLS-complete [42, 52]. It remains PLS-complete for graphs of maximum degree 5 [27], but becomes polynomial-time solvable for graphs of maximum degree 3 [48, 51]. The problem remains PLS-complete if weights are assigned to vertices, instead of edges, and the weight of an edge is defined simply as the product of the

weights of its endpoints [31]. Even though the problem is PLS-complete, it has long been observed that local search quickly finds a stable solution in most practical instances. One theoretical explanation for this phenomenon was given in a recent line of work which showed that Local Max Cut has quasi-polynomial time smoothed complexity [3, 12, 18, 29]. Local Max Cut is of course polynomial time solvable if all weights are polynomially bounded in n, as local improvements always increase the size of the cut.

In algorithmic game theory much work has been done on the complexity of computing Nash equilibria for the cut game and the closely related *party affiliation game*, in which players, represented by vertices, have to pick one of two parties and edge weights indicate how much two players gain if they are in the same party [6, 7, 19, 30, 35]. Note that for general graphical games finding an equilibrium is PPAD-hard on trees of constant pathwidth [25]. Because computing a stable solution is generally intractable, approximate equilibria have also been considered [11, 17]. Note that the notion of approximate equilibrium corresponds exactly to the approximation guarantee given by Theorem 11, but unlike the cited works, Theorem 11 produces a solution that is both approximately stable and as good as the optimal.

The problem we consider in this paper is more closely related to the problem of computing the *worst* (or best) Nash equilibrium, which in turn is closely linked to the notion of Price of Anarchy. For most problems in algorithmic game theory this type of question is usually NP-hard [13, 20, 26, 32, 34, 37, 53] and hard to approximate [5, 16, 22, 39, 49]. Even though these results show that finding a Nash equilibrium that maximizes an objective function is NP-hard under various restrictions (e.g. graphical games of bounded degree), to the best of our knowledge the complexity of finding the worst equilibrium of the Max Cut game (which corresponds to the Min Stable Cut problem of this paper) has not been considered.

Finally, another topic that has recently attracted attention in the literature is that of MinMax and MaxMin versions of standard optimization problems, where we search the worst solution which cannot be improved using a simple local search heuristic. The motivation behind this line of research is to provide bounds and a refined analysis of such basic heuristics. Problems that have been considered under this lens are Max Min Dominating Set [8, 24], Max Min Vertex Cover [15, 54],Max Min Separator [38], Max Min Cut [28], Min Max Knapsack [4, 33, 36], Max Min Edge Cover [45], Max Min Feedback Vertex Set [23]. Some problems in this area also arise naturally in other forms and have been extensively studied, such as Min Max Matching (also known as Edge Dominating Set [41]) and Grundy Coloring, which can be seen as a Max Min version of Coloring [1, 9].

2 Definitions – Preliminaries

We generally use standard graph-theoretic notation and consider edge-weighted graphs, that is, graphs $G = (V, E)$ supplied with a weight function $w : E \to \mathbb{N}$. For a vertex $v \in V$, The weighted degree of a vertex $v \in V$ is $d_w(v) = \sum_{uv \in E} w(uv)$. A cut of a graph is a partition of V into V_0, V_1. A cut is *stable* for vertex $v \in V_i$ if $\sum_{vu \in E \wedge u \in V_{1-i}} w(vu) \geq \frac{d_w(v)}{2}$, that is, if the total weight of edges incident on v crossing the cut is at least half the weighted degree of v. In the Min Stable Cut problem we are given an edge-weighted graph and are looking for a cut that is stable for all vertices that minimizes the sum of weights of cut edges (that is, edges with endpoints on both sides of the cut). In Unweighted Min Stable Cut we restrict the problem so that the w function returns 1 for all edges. When describing stable cuts we will sometimes say that we "assign" value 0 (or 1) to a vertex; by this we mean that we place this vertex in V_0 (or V_1 respectively).

For the definitions of treewidth, pathwidth, and the related (nice) decompositions we refer to [21]. We will use as a complexity assumption the Exponential Time Hypothesis (ETH) [40] which states that there exists a constant $c > 1$ such that 3-SAT with n variables and m clauses cannot be solved in time c^{n+m}. In fact, we will use the slightly weaker and simpler form of the ETH which states that 3-SAT cannot be solved in time $2^{o(n+m)}$.

3 Weighted Min Stable Cut

In this section we present our results on exact algorithms for (weighted) Min Stable Cut. We begin with some basic NP-hardness reductions in Section 3.1, which establish that the problem remains (weakly) NP-hard when either the treewidth or the maximum degree are bounded. These set the stage for two algorithms, given in Section 3.2, solving the problem in pseudo-polynomial time for constant treewidth; and in FPT time parameterized by $\mathrm{tw} + \Delta$. In Section 3.3 we present a more fine-grained hardness argument, based on the ETH, which shows that the dependence on tw and Δ of our two algorithms is essentially optimal.

3.1 Basic Hardness Proofs

▶ **Theorem 1.** *Min Stable Cut is weakly NP-hard on trees of diameter* 4.

Proof. We describe a reduction from Partition. Recall that in this problem we are given n positive integers $x_1, \ldots, x_n$ such that $\sum_{i=1}^{n} x_i = 2B$ and are asked if there exists $S \subseteq [n]$ such that $\sum_{i \in S} x_i = B$. We construct a star with n leaves and subdivide every edge once. For each $i \in [n]$ we select a distinct leaf of the tree and set the weight of both edges in the path from the center to this leaf to x_i. We claim that the graph has a stable cut of weight $3B$ if and only if there is a partition of $x_1, \ldots, x_n$ into two sets with the same sum.

For the first direction, suppose $S \subseteq [n]$ is such that $\sum_{i \in S} x_i = B$. For each $i \in S$ we select a degree two vertex of the tree whose incident edges have weight x_i and assign it value 1. We assign all other degree two vertices value 0 and assign to all leaves the opposite of the value of their neighbor. We give the center value 0. This partition is stable as the center has edge weight exactly B towards each side, and all degree two vertices have a leaf attached that is placed on the other side and contributes half their total incident weight. The total weight cut is $2B$ from edges incident on leaves, plus B from half the weight incident on the center.

For the converse direction, observe that in any stable solution all edges incident on leaves are cut, contributing a weight of $2B$. As a result, in a stable cut of size $3B$, the weight of cut edges incident on the center is at most B. However, this weight is also at least B, since the edge weight incident on the center is $2B$. We conclude that the neighborhood of the center must be perfectly balanced. From this we can infer a solution to the Partition instance. ◀

▶ Remark 2. Theorem 1 is tight, because Min Stable Cut is trivial on trees of diameter at most 3.

▶ **Theorem 3.** *Min Stable Cut is weakly NP-hard on bipartite graphs with vertex cover* 2.

▶ **Theorem 4.** *Unweighted Min Stable Cut is strongly NP-hard and APX-hard on bipartite graphs of maximum degree* 6.

3.2 Algorithms

▶ **Theorem 5.** *There is an algorithm which, given an instance of Min Stable Cut with n vertices, maximum weight W, and a tree decomposition of width* tw*, finds an optimal solution in time $(\Delta \cdot W)^{O(\mathrm{tw})} n^{O(1)}$.*

▶ **Theorem 6.** *There is an algorithm which, given an instance of* Min Stable Cut *with n vertices, maximum weight W, maximum degree Δ and a tree decomposition of width* tw*, finds an optimal solution in time $2^{O(\Delta \mathrm{tw})}(n + \log W)^{O(1)}$.*

Proof. We describe an algorithm which works in a way similar to the standard algorithm for Max Cut parameterized by treewidth, except that we work in a tree decomposition that is essentially a decomposition of the square of G. More precisely, before we begin, we do the following: for each $v \in V$ we add to every bag of the decomposition that contains v all the vertices of $N(v)$. It is not hard to see that we now have a decomposition of width at most $(\Delta + 1)(\mathrm{tw} + 1)$ and also that the new decomposition is still a valid tree decomposition. Crucially, we now also have the following property: for each $v \in V$ there exists at least one bag of the decomposition that contains all of $N[v]$.

The algorithm now performs dynamic programming by storing for each bag the value of the best solution for each partition of B_t. As a result, the size of the DP table is $2^{O(\Delta \mathrm{tw})}$. The only difference with the standard Max Cut algorithm (beyond the fact that we are looking for a cut of minimum weight) is that when we consider a bag that contains all of $N[v]$, for some $v \in V$, we discard all partitions which are unstable for v. Since the bag contains all of $N[v]$, this can be checked in time polynomial in n and $\log W$ (assuming weights are given in binary). ◀

3.3 Tight ETH-based Hardness

We first give a reduction from 3-Set Splitting to Min Stable Cut whose main properties are laid out in Lemma 7. This reduction gives the lower bound of Theorem 8.

▶ **Lemma 7.** *There is a polynomial-time algorithm which, given a 3-Set Splitting instance $H = (V, E)$ with n elements, produces a Min Stable Cut instance G with the following properties: (i) G is a Yes instance if and only if H is a Yes instance; (ii) if Δ is the maximum degree of G and* pw *its pathwidth, then $\Delta = O(\log n)$ and $\mathrm{pw} = O(n/\log n)$; (iii) the maximum weight of G is $W = O(2^{\Delta})$.*

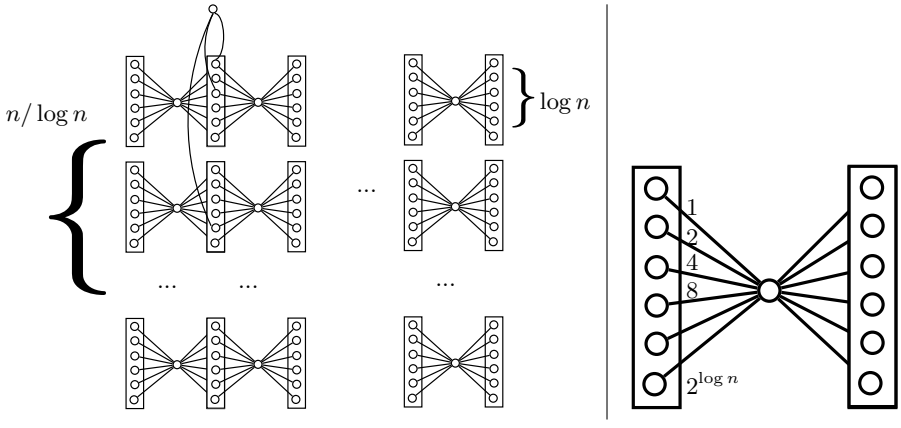


Figure 1 Sketch of the construction of Lemma 7. On the left, the general architecture: m columns, each with n vertices, partitioned into groups of size $\log n$. On each column we add a checker vertex (on top). Between the same groups of consecutive columns we add propagator vertices. On the right, more details about the exponentially increasing weights of edges incident on propagators.

Proof. Let $H = (V, E)$ be the given 3-Set Splitting instance, $V = \{v_0, \ldots, v_{n-1}\}$ and suppose that E contains e_2 sets of size 2 and e_3 sets of size 3, where $|E| = e_2 + e_3$ will be denoted by m. Assume without loss of generality that n is a power of 2 (otherwise add some dummy elements to V). Let $\delta = \log n$. We construct a graph by first making m copies of V, call them $V_j, j \in [m]$ and label their vertices as $V_j = \{v_{(i,j)} \mid i \in \{0, \ldots, n-1\}\}$. Intuitively, the vertices $\{v_{(i,j)} \mid j \in [m]\}$ are all meant to represent the element v_i of H. We now add to the graph the following:

1. Checkers: Suppose that the j-th set of E contains elements $v_{i_1}, v_{i_2}, v_{i_3}$. Then we construct a vertex c_j and connect it to $v_{(i_1,j)}, v_{(i_2,j)}, v_{(i_3,j)}$ with edges of weight 1. If the j-th set has size two, we do the same (ignoring v_{i_3}).
2. Propagators: For each $j \in [m-1]$ we construct $\rho = \lceil n/\delta \rceil$ vertices labeled $p_{(i,j)}, i \in \{0, \ldots, \rho - 1\}$. Each $p_{(i,j)}$ is connected to (at most) δ vertices of V_j and δ vertices of V_{j+1} with edges of exponentially increasing weight. Specifically, for $i \in \{0, \ldots, \rho-1\}, \ell \in \{0, \ldots, \delta - 1\}$, we connect $p_{(i,j)}$ to $v_{(i\delta+\ell,j)}$ and to $v_{(i\delta+\ell,j+1)}$ (if they exist) with an edge of weight 2^ℓ.
3. Stabilizers: For each $j \in [m], i \in \{0, \ldots, n-1\}$ we attach to $v_{(i,j)}$ a leaf. The edge connecting this leaf to $v_{(i,j)}$ has weight $3 \cdot 2^{(i \bmod \delta)}$.

This completes the construction of the graph. Let L_w be the total weight of edges incident on leaves and P be the total weight of edges incident on Propagator vertices $p_{(i,j)}$. We set $B = L_w + \frac{P}{2} + e_2 + 2e_3$ and claim that the new instance has a stable cut of weight B if and only if H can be split.

For the forward direction, suppose that H can be split by the partition of V into $L, R = V \setminus L$. We assign the following values for our new instance: for each $j \in [m]$ odd, we set $v_{(i,j)}$ to value 0 if and only if $v_i \in L$; for each $j \in [m]$ even, we set $v_{(i,j)}$ to value 0 if and only if $v_i \in R$. In other words, we use the same partition for all copies of V, but flip the roles of $0, 1$ between consecutive copies. We place leaves on the opposite side from their neighbors and greedily assign values to all other vertices of the graph to obtain a stable partition. Observe that all vertices $v_{(i,j)}$ are stable with the values we assigned, since the edge connecting each such vertex to a leaf has weight at least half its total incident weight.

In the partition we have, we observe that (i) all edges incident on leaves are cut (total weight L_w) (ii) all Propagator vertices have balanced neighborhoods, so exactly half of their incident weight is cut (total weight $P/2$) (iii) since L, R splits all sets of E, each checker vertex will have exactly one neighbor on the same side (total weight $e_2 + 2e_3$). So, the total weight of the cut is B.

For the converse direction, suppose we have a stable cut of size B in the constructed instance. Because of the stability condition, this solution must cut all edges incident on leaves (total weight L_w); at least half of the total weight of edges incident on Propagators (total weight $P/2$); and for each checker vertex all its incident edges except at most one (total weight at least $e_2 + 2e_3$). We conclude that, in order to achieve weight B, the cut must properly balance the neighborhood of all Propagators and make sure that each Checker vertex has one neighbor on its own side.

We now argue that because the neighborhood of each Propagator is balanced we have for all $i \in \{0, \ldots, n-1\}, j \in [m-1]$ that $v_{(i,j)}, v_{(i,j+1)}$ are on different sides of the partition. To see this, suppose for contradiction that for two such vertices this is not the case and to ease notation consider the vertices $v_{(i\delta+\ell,j)}, v_{(i\delta+\ell,j+1)}$, where $0 \leq \ell \leq \delta - 1$. Among all such pairs select one that maximizes ℓ. Both vertices are connected to the Propagator $p_{(i,j)}$ with edges of weight 2^ℓ. But now $p_{(i,j)}$ has strictly larger edge weight connecting it to the side of the

partition that contains $v_{(i\delta+\ell,j)}$ and $v_{(i\delta+\ell,j+1)}$ than to the other side because (i) for neighbors of $p_{(i,j)}$ connected to it with edges of higher weight, the neighborhood of $p_{(i,j)}$ is balanced by the maximality of ℓ (ii) the total weight of all other edges is $2 \cdot (2^{\ell-1} + 2^{\ell-2} + \ldots + 1) < 2 \cdot 2^{\ell}$.

We thus have that for all i, j, $v_{(i,j)}, v_{(i,j+1)}$ must be on different sides, and therefore all V_j are partitioned in the same way (except some have the roles of 0 and 1 reversed). From this, we obtain a partition of V. To conclude this direction, we argue that this partition of V must split all sets. Indeed, if not, there will be a checker vertex such that all its neighbors are on the same side, which, as we argued, means that the cut must have weight strictly more than B.

Finally, let us show that the constructed instance has the claimed properties. The maximum degree is $\Delta = 2\delta = O(\log n)$ in the Propagators vertices (all other vertices have degree at most 4); the maximum weight is $O(2^{\delta}) = O(2^{\Delta})$. Let us also consider the pathwidth of the constructed graph. Let G_j be the subgraph induced by V_j and its attached leaves, the Checker c_j, and all Propagators adjacent to V_j. We claim that we can build a path decomposition of G_j that contains all Propagators adjacent to V_j in all bags and has width $O(n/\log n)$. Indeed, if we place all the (at most $\lceil 2n/\delta \rceil$) Propagators and c_j in all bags, we can delete them from G_j, and all that is left is a union of isolated edges, which has pathwidth 1. Now, since the union of all G_j covers all vertices and edges, we can construct a path decomposition of the whole graph of width $O(n/\log n)$ by gluing together the decompositions of each G_j, that is, by connecting the last bag of the decomposition of G_j to the first bag of the decomposition of G_{j+1}. ◀

▶ **Theorem 8.** *If the ETH is true then (i) there is no algorithm solving* Min Stable Cut *in time* $(nW)^{o(\mathrm{pw})}$ *(ii) there is no algorithm solving* Min Stable Cut *in time* $2^{o(\Delta \mathrm{pw})}(n + \log W)^{O(1)}$. *These statements apply even if we restrict the input to instances where weights are written in unary and the maximum degree is* $O(\log n)$.

4 Approximately Stable Cuts

In this section we present an algorithm which runs in FPT time parameterized by treewidth and produces a solution that is $(1+\varepsilon)$-stable and has weight upper bounded by the weight of the optimal stable cut. Before we proceed, we will need to define a more general version of our problem. In Extended Min Stable Cut we are given as input: a graph $G = (V, E)$; a cut-weight function $w : E \to \mathbb{N}$; and a stability-weight function $s : E \times V \to \mathbb{N}$. For $v \in V$ we denote $d_s(v) = \sum_{vu \in E} s(vu, v)$, which we call the stability degree of v. If we are also given an error parameter $\rho > 1$, we will then be looking for a partition of V into V_0, V_1 which satisfies the following: (i) each vertex is ρ-stable, that is, for each $i \in \{0, 1\}$ and $v \in V_i$ we have $\sum_{vu \in E \wedge u \in V_{1-i}} s(vu, v) \geq \frac{d_s(v)}{2\rho}$ (ii) the total cut weight $\sum_{u \in V_0, v \in V_1, uv \in E} w(uv)$ is minimum. Observe that this extended version of the problem contains Min Stable Cut as a special case if $\rho = 1$ and for all $uv \in E$ we have $s(uv, v) = s(uv, u) = w(uv)$.

The generalization of Min Stable Cut is motivated by three considerations. First, the algorithm of Theorem 5 is inefficient because it has to store exact weight values to satisfy the stability constraints; however, it can efficiently store the total weight of the cut. We therefore decouple the contribution of an edge to the size of the cut (given by w) from a contribution of an edge to the stability of its endpoints (given by s). Second, our strategy will be to truncate the values of s so that the DP of the algorithm of Theorem 5 can be run more efficiently. To do this we will first simply divide all stability-weights by an appropriate value. However, a problem we run into if we do this is that the edge uv could simultaneously be one of the heavier edges incident on u and one of the lighter edges incident on v, so it

is not clear how we can adjust its weight in a way that minimizes the distortion for both endpoints. As a result it is simpler if we allow edges to contribute different amounts to the stability of their endpoints. In this sense, $s(uv, u)$ is the amount that the edge uv contributes to the stability of u if the edge is cut. Observe that with the new definition, if we set a new stability-weight function for a specific vertex u as $s'(uv, v) = c \cdot s(uv, v)$ for all $v \in N(u)$, that is, if we multiply the stability-weight of all edges incident on u by a constant c and leave all other values unchanged, we obtain an equivalent instance, and this does not affect the stability of other vertices. Finally, the parameter ρ allows us to consider solutions where a vertex is stable if its cut incident edges are at least a $(\frac{1}{2\rho})$-fraction of its stability degree.

Armed with this intuition we can now explain our approach to obtaining our FPT approximation algorithm. Given an instance of the extended problem, we first adjust the s function so that its maximum value is bounded by a polynomial in n. We achieve this by dividing $s(uv, u)$ by a value that depends only on $d_s(u)$ and n. This allows us to guarantee that near-stable solutions are preserved. Then, given an instance where the maximum value of s is polynomially bounded, we apply the technique of [46], using the algorithm of Theorem 5 as a base, to obtain our approximation. We give these separate steps in the Lemmas below.

▶ **Lemma 9.** *There is an algorithm which, given a graph $G = (V, E)$ on n vertices and a stability-weight function $s : E \times V \to \mathbb{N}$ with maximum value S, runs in time polynomial in $n + \log S$ and produces a stability-weight function $s' : E \times V \to \mathbb{N}$ with the following properties: (i) the maximum value of s' is $O(n^2)$ (ii) for all partitions V into V_0, V_1, $i \in \{0, 1\}$, $v \in V_i$ we have*

$$\left(\frac{\sum_{vu \in E, u \in V_{1-i}} s(vu, v)}{d_s(v)}\right) / \left(\frac{\sum_{vu \in E, u \in V_{1-i}} s'(vu, v)}{d_{s'}(v)}\right) \in [1 - 1/n, 1 + 1/n]$$

Using Lemma 9 we can assume that all stability-weights are bounded by $O(n^2)$. The most important part is that Lemma 9 guarantees us that almost-optimal solutions are preserved in both directions, as for any cut and for each vertex the ratio of stability weight going to the other side over the total stability-degree of the vertex does not change by more than a factor $(1 + \frac{1}{n})$. Let us now see the second ingredient of our algorithm.

▶ **Lemma 10.** *There is an algorithm which takes as input a graph $G = (V, E)$, a cut-weight function $w : E \to \mathbb{N}$ with maximum W, a stability-weight function $s : E \times V \to \mathbb{N}$ with maximum S, a tree decomposition of G of width* tw*, and an error parameter $\varepsilon > 0$ and returns a $(1+2\varepsilon)$-stable solution that has cut-weight at most equal to that of the minimum $(1+\epsilon)$-stable solution. If $S = O(n^2)$, then the algorithm runs in time $(\mathrm{tw}/\varepsilon)^{O(\mathrm{tw})}(n + \log W)^{O(1)}$.*

Proof. We use the methodology of [46]. Before we proceed, let us explain that we are actually aiming for an algorithm with running time roughly $(\log n/\varepsilon)^{O(\mathrm{tw})}$. This type of running time implies the time stated in the lemma using a standard Win/Win argument: if $\mathrm{tw} \leq \sqrt{\log n}$ then $(\log n)^{O(\mathrm{tw})}$ is $n^{o(1)}$, so the $(\log n)^{O(\mathrm{tw})}$ factor is absorbed in the $n^{O(1)}$ factor; while if $\log n \leq \mathrm{tw}^2$, then an algorithm running in $(\log n)^{\mathrm{tw}}$ actually runs in time $(\mathrm{tw})^{O(\mathrm{tw})}$.

To be more precise, if the given tree decomposition has height H, then we will formulate an algorithm with running time $(H \log S/\varepsilon)^{O(\mathrm{tw})}(n + \log W)^{O(1)}$. This running time achieves parameter dependence $(\log n/\varepsilon)^{O(\mathrm{tw})}$ if we use the fact that $S = O(n^2)$ and a theorem due to [14] which proves that any tree decomposition can be edited (in polynomial time) so that its height becomes $O(\log n)$, without increasing its width by more than a constant factor.

The basis of our algorithm will be the algorithm of Theorem 5, appropriately adjusted to the extended version of the problem. Let us first sketch the modifications to the algorithm of Theorem 5 that we would need to do to solve this more general problem, since the details

are straightforward. First, we observe that in solution signatures we would now take into account stability-weights, and signatures would have values going up to S. Second, in Forget nodes, since we are happy with a $(1+\varepsilon)$-stable solution, we would only discard solutions which violate this constraint. With these modifications, we can run this exact algorithm to return the minimum $(1+\varepsilon)$-stable solution in time $(2S)^{O(\mathrm{tw})}(n+\log W+\log(1/\varepsilon))^{O(1)}$.

The idea is to modify this algorithm so that the DP tables go from size $(2S)^{\mathrm{tw}}$ to roughly $(H\log S)^{\mathrm{tw}}$. To do this, we define a parameter $\delta=\frac{\varepsilon}{5H}$. We intend to replace every value x that would be stored in the signature of a solution in the DP table, with the next larger integer power of $(1+\delta)$, that is, to construct a DP table where x is replaced by $(1+\delta)^{\lceil \log_{(1+\delta)} x\rceil}$.

More precisely, the invariant we maintain is the following. Consider a node t of the decomposition at height h, where $h=0$ corresponds to leaves. We maintain a collection of solution signatures such that: (i) each signature contains a partition of B_t and for each $v\in B_t$ an integer that is upper-bounded by $\lceil \log_{(1+\delta)} d_s(v)\rceil$; (ii) Soundness: for each stored signature there exists a partition of $B_t^{\downarrow}$ which approximately corresponds to it. Specifically, the partition and the signature agree exactly on the assignment of B_t and the total cut-weight; the partition is $(1+2\varepsilon)$-stable for all vertices of $B_t^{\downarrow}\setminus B_t$; and for each $v\in B_t$, if the signature stores the value $x(v)$ for v, that is, it states that v has approximate stability-weight $(1+\delta)^{x(v)}$ towards its own side in $B_t^{\downarrow}\setminus B_t$, then in the actual partition the stability-weight of v to its own side of $B_t^{\downarrow}\setminus B_t$ is at most $(1+\delta)^h(1+\delta)^{x(v)}$. (iii) Completeness: conversely, for each partition of $B_t^{\downarrow}$ that is $(1+\varepsilon)$-stable for all vertices of $B_t^{\downarrow}\setminus B_t$ there exists a signature that approximately corresponds to it. Specifically, the partition and signature agree on the assignment of B_t and the total cut-weight; and for each $v\in B_t$, if the stability-weight of v towards its side of the partition of $B_t^{\downarrow}\setminus B_t$ is $y(v)$, and the signature stores the value $x(v)$, then $(1+\delta)^{x(v)}\le(1+\delta)^h y(v)$.

In more simple terms, the signatures in our DP table store values $x(v)$ so that we estimate that in the corresponding solution v has approximately $(1+\delta)^{x(v)}$ weight towards its own side in $B_t^{\downarrow}$, that is, we estimate that the DP of the exact algorithm would store approximately the value $(1+\delta)^{x(v)}$ for this solution. Of course, it is hard to maintain this relation exactly, so we are happy if for a node at height h the "true" value which we are approximating is at most a factor of $(1+\delta)^h$ off from our approximation.

Now, the crucial observation is that the approximate DP tables can be maintained because our invariant allows the error to increase with the height. For example, suppose that t is a Forget node at height h and let $u\in B_t$ be a neighbor of the vertex v we forget. The exact algorithm would construct the signature of a solution in t by looking at the signature of a solution in its child node, and then adding to the value stored for u the weight $s(vu,u)$ (if u,v are on the same side). Our algorithm will take an approximate signature from the child node, which may have a value at most $(1+\delta)^{h-1}$ the correct value, add to it $s(vu,u)$ and then, perhaps, round-up the value to an integer power of $(1+\delta)$. The new approximation will be at most $(1+\delta)^h$ larger than the value that the exact algorithm would have calculated. Similar argumentation holds for Join nodes. Furthermore, in Forget nodes we will only discard a solution if according to our approximation it is not $(1+2\varepsilon)$-stable. We may be over-estimating the stability-weight a vertex has to its own side of the cut by a factor of at most $(1+\delta)^h\le(1+\frac{\varepsilon}{5H})^H\le 1+\frac{\varepsilon}{2}$ so if for a signature our approximation says that the solution is not $(1+2\varepsilon)$-stable, the solution cannot be $(1+\varepsilon)$-stable, because $(1+\varepsilon)(1+\frac{\varepsilon}{2})<1+2\varepsilon$ (for sufficiently small ε).

Finally, to estimate the running time, the maximum value we have to store for each vertex in a bag is $\log_{(1+\delta)} S=\frac{\log S}{\log(1+\delta)}\le O(\frac{\log n}{\delta})=O(\frac{H\log n}{\varepsilon})$. Using the fact that $H=O(\log n)$ we get that the size of the DP table is $(\log n/\varepsilon)^{O(\mathrm{tw})}$. ◀

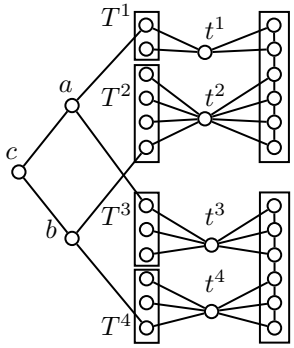


Figure 2 Checker gadget for Theorem 13. On the right two Selector gadgets. This Checker verifies that we have not taken an edge which has endpoints $(2, 3)$, hence t^1, t^3 are connected to the first 2 and 3 vertices of the Selectors.

▶ **Theorem 11.** *There is an algorithm which, given an instance of* Min Stable Cut *$G = (V, E)$ with n vertices, maximum weight W, a tree decomposition of width* tw*, and a desired error $\varepsilon > 0$, runs in time $(\mathrm{tw}/\varepsilon)^{O(\mathrm{tw})}(n + \log W)^{O(1)}$ and returns a cut with the following properties: (i) for all $v \in V$, the total weight of edges incident on v crossing the cut is at least $(1 - \varepsilon)\frac{d_w(v)}{2}$ (ii) the cut has total weight at most equal to the weight of the minimum stable cut.*

5 Unweighted Min Stable Cut

In this section we consider Unweighted Min Stable Cut. We first observe that applying Theorem 5 gives a parameter dependence of $\Delta^{O(\mathrm{tw})}$, since $W = 1$. We then show that this algorithm is essentially optimal, as the problem cannot be solved in $n^{o(\mathrm{pw})}$ under the ETH.

▶ **Corollary 12.** *There is an algorithm which, given an instance of* Unweighted Min Stable Cut *with n vertices, maximum degree Δ, and a tree decomposition of width* tw*, returns an optimal solution in time $\Delta^{O(\mathrm{tw})}n^{O(1)}$.*

We now first state our hardness result, then describe the construction of our reduction, and finally go through a series of lemmas that establish its correctness.

▶ **Theorem 13.** *If the ETH is true then no algorithm can solve* Unweighted Min Stable Cut *on graphs with n vertices in time $n^{o(\mathrm{pw})}$. Furthermore,* Unweighted Min Stable Cut *is W[1]-hard parameterized by pathwidth.*

To prove Theorem 13 we will describe a reduction from k-Multi-Colored Independent Set, a well-known W[1]-hard problem that cannot be solved in $n^{o(k)}$ time under the ETH [21]. In this problem we are given a graph $G = (V, E)$ with V partitioned into k color classes $V_1, \ldots, V_k$, each of size n, and we are asked to find an independent set of size k which selects one vertex from each V_i. In the remainder we use m to denote the number of edges of E and assume that vertices of V are labeled $v_{(i,j)}, i \in [k], j \in [n]$, where $V_i = \{v_{(i,j)} \mid j \in [n]\}$.

Before we proceed, let us give some intuition. Our reduction will rely on a $k \times m$ grid-like construction, where each row represents the selection of a vertex in the corresponding color class of G and each column represents an edge of G. The main ingredients will be a Selector gadget, which will represent a choice of an index in $[n]$; a Propagator gadget which will make sure that the choice we make in each row stays consistent throughout; and a Checker gadget which will verify that we did not select the two endpoints of any edge. Each Selector gadget

will contain a path on (roughly) n vertices such that any reasonable stable cut will have to cut exactly one edge of the path. The choice of where to cut this path will represent an index in $[n]$ encoding a vertex of G.

In our construction we will also make use of a simple but important gadget which we will call a "heavy" edge. Let $A = n^5$. When we say that we connect u, v with a heavy edge we will mean that we construct A new vertices and connect them to both u and v. The intuitive idea behind this gadget is that the large number of degree two vertices will force u and v to be on different sides of the partition (otherwise too many edges will be cut). We will also sometimes attach leaves on some vertices with the intention of making it easier for this vertex to achieve stability (as its attached leaves will always be on the other side of the partition).

Let us now describe our construction step-by-step.

1. Construct two "palette" vertices p_0, p_1 and a heavy edge connecting them. Note that all heavy edges we will add will be incident on at least one palette vertex.
2. For each $i \in [k], j \in [m]$ construct the following Selector gadget:
   a. Construct a path on $n+1$ vertices $P_{(i,j)}$ and label its vertices $P^1_{(i,j)}, \ldots, P^{n+1}_{(i,j)}$.
   b. If j is odd, then add a heavy edge from $P^1_{(i,j)}$ to p_1 and a heavy edge from $P^{n+1}_{(i,j)}$ to p_0. If j is even, then add a heavy edge from $P^1_{(i,j)}$ to p_0 and a heavy edge from $P^{n+1}_{(i,j)}$ to p_1.
   c. Attach 5 leaves to each $P^\ell_{(i,j)}$ for $\ell \in \{2, \ldots, n\}$. Attach $A+5$ leaves to $P^1_{(i,j)}$ and $P^{n+1}_{(i,j)}$.
3. For each $i \in [k], j \in [m-1]$ construct a new vertex connected to all vertices of the paths $P_{(i,j)}$ and $P_{(i,j+1)}$. This vertex is the Propagator gadget.
4. For each $j \in [m]$ consider the j-th edge of the original instance and suppose it connects $v_{(i_1,j_1)}$ to $v_{(i_2,j_2)}$. We construct the following Checker gadget (see Figure 2)
   a. We construct four vertices $t^1_j, t^2_j, t^3_j, t^4_j$. These are connected to existing vertices as follows: t^1_j is connected to $\{P^1_{(i_1,j)}, \ldots, P^{j_1}_{(i_1,j)}\}$ (that is, the first j_1 vertices of the path $P_{(i_1,j)}$); t^2_j is connected to $\{P^{j_1+1}_{(i_1,j)}, \ldots, P^{n+1}_{(i_1,j)}\}$ (that is, the remaining $n+1-j_1$ vertices of $P_{i_1,j}$); similarly, t^3_j is connected to $\{P^1_{(i_2,j)}, \ldots, P^{j_2}_{(i_2,j)}\}$; and finally t^4_j is connected to $\{P^{j_2+1}_{(i_2,j)}, \ldots, P^{n+1}_{(i_2,j)}\}$.
   b. We construct four independent sets $T^1_j, T^2_j, T^3_j, T^4_j$ with respective sizes $j_1, n+1-j_1, j_2, n+1-j_2$. We connect t^1_j to all vertices of T^1_j, t^2_j to T^2_j, t^3_j to T^3_j, and t^4_j to T^4_j. We attach two leaves to each vertex of $T^1_j \cup T^2_j \cup T^3_j \cup T^4_j$.
   c. We construct three vertices a_j, b_j, c_j. We connect c_j to both a_j and b_j. We connect a_j to an arbitrary vertex of T^1_j and an arbitrary vertex of T^3_j. We connect b_j to an arbitrary vertex of T^2_j and an arbitrary vertex of T^4_j.

Let L_1 be the number of leaves of the construction we described above and L_2 be the number of degree two vertices which are part of heavy edges. We set $B = L_1 + L_2 + km + k(m-1)(n+1) + m(2n+6)$.

▶ **Lemma 14.** *If G has a multi-colored independent set of size k, then the constructed instance has a stable cut of size at most B.*

▶ **Lemma 15.** *If the constructed instance has a stable cut of size at most B, then G has a multi-colored independent set of size k.*

▶ **Lemma 16.** *The constructed graph has pathwidth $O(k)$.*

6 Conclusions

Our results paint a clear picture of the complexity of Min Stable Cut with respect to tw and Δ. As directions for further work one could consider stronger notions of stability such as demanding that switching sets of k vertices cannot increase the cut, for constant k. We conjecture that, since the structure of this problem has the form $\exists\forall_k$, its complexity with respect to treewidth will turn out to be double-exponential in k [47]. Another direction is to consider *hedonic games* where vertices self-partition into an unbounded number of groups. The complexity of finding a stable solution in such games parameterized by $\text{tw} + \Delta$ has already been considered by Peters [50], whose algorithm runs in time exponential in $\Delta^5\text{tw}$. Can we bridge the gap between this complexity and the $2^{O(\Delta\text{tw})}$ complexity of Min Stable Cut?

References

1 Pierre Aboulker, Édouard Bonnet, Eun Jung Kim, and Florian Sikora. Grundy coloring & friends, half-graphs, bicliques. In *STACS*, volume 154 of *LIPIcs*, pages 58:1–58:18. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020.

2 Eric Angel, Evripidis Bampis, Bruno Escoffier, and Michael Lampis. Parameterized power vertex cover. *Discret. Math. Theor. Comput. Sci.*, 20(2), 2018. URL: `http://dmtcs.episciences.org/4873`.

3 Omer Angel, Sébastien Bubeck, Yuval Peres, and Fan Wei. Local max-cut in smoothed polynomial time. In Hamed Hatami, Pierre McKenzie, and Valerie King, editors, *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2017, Montreal, QC, Canada, June 19-23, 2017*, pages 429–437. ACM, 2017. `doi:10.1145/3055399.3055402`.

4 Esther M. Arkin, Michael A. Bender, Joseph S. B. Mitchell, and Steven Skiena. The lazy bureaucrat scheduling problem. *Inf. Comput.*, 184(1):129–146, 2003.

5 Per Austrin, Mark Braverman, and Eden Chlamtac. Inapproximability of NP-complete variants of Nash equilibrium. *Theory Comput.*, 9:117–142, 2013. `doi:10.4086/toc.2013.v009a003`.

6 Baruch Awerbuch, Yossi Azar, Amir Epstein, Vahab S. Mirrokni, and Alexander Skopalik. Fast convergence to nearly optimal solutions in potential games. In Lance Fortnow, John Riedl, and Tuomas Sandholm, editors, *Proceedings 9th ACM Conference on Electronic Commerce (EC-2008), Chicago, IL, USA, June 8-12, 2008*, pages 264–273. ACM, 2008. `doi:10.1145/1386790.1386832`.

7 Maria-Florina Balcan, Avrim Blum, and Yishay Mansour. Improved equilibria via public service advertising. In Claire Mathieu, editor, *Proceedings of the Twentieth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2009, New York, NY, USA, January 4-6, 2009*, pages 728–737. SIAM, 2009. URL: `http://dl.acm.org/citation.cfm?id=1496770.1496850`.

8 C. Bazgan, L. Brankovic, K. Casel, H. Fernau, K. Jansen, K.-M. Klein, M. Lampis, M. Liedloff, J. Monnot, and V. T. Paschos. The many facets of upper domination. *Theoretical Computer Science*, 717:2–25, 2018.

9 Rémy Belmonte, Eun Jung Kim, Michael Lampis, Valia Mitsou, and Yota Otachi. Grundy distinguishes treewidth from pathwidth. In Fabrizio Grandoni, Grzegorz Herman, and Peter Sanders, editors, *28th Annual European Symposium on Algorithms, ESA 2020, September 7-9, 2020, Pisa, Italy (Virtual Conference)*, volume 173 of *LIPIcs*, pages 14:1–14:19. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020. `doi:10.4230/LIPIcs.ESA.2020.14`.

10 Rémy Belmonte, Michael Lampis, and Valia Mitsou. Parameterized (approximate) defective coloring. *SIAM J. Discret. Math.*, 34(2):1084–1106, 2020. `doi:10.1137/18M1223666`.

11 Anand Bhalgat, Tanmoy Chakraborty, and Sanjeev Khanna. Approximating pure Nash equilibrium in cut, party affiliation, and satisfiability games. In David C. Parkes, Chrysanthos Dellarocas, and Moshe Tennenholtz, editors, *Proceedings 11th ACM Conference on Electronic Commerce (EC-2010), Cambridge, Massachusetts, USA, June 7-11, 2010*, pages 73–82. ACM, 2010. `doi:10.1145/1807342.1807353`.

12 Ali Bibak, Charles Carlson, and Karthekeyan Chandrasekaran. Improving the smoothed complexity of FLIP for max cut problems. In Timothy M. Chan, editor, *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2019, San Diego, California, USA, January 6-9, 2019*, pages 897–916. SIAM, 2019. doi:10.1137/1.9781611975482.55.

13 Vittorio Bilò and Marios Mavronicolas. The complexity of computational problems about Nash equilibria in symmetric win-lose games. *CoRR*, abs/1907.10468, 2019. arXiv:1907.10468.

14 Hans L. Bodlaender and Torben Hagerup. Parallel algorithms with optimal speedup for bounded treewidth. *SIAM J. Comput.*, 27(6):1725–1746, 1998.

15 É. Bonnet, M. Lampis, and V. T. Paschos. Time-approximation trade-offs for inapproximable problems. *Journal of Computer and System Sciences*, 92:171–180, 2018.

16 Mark Braverman, Young Kun-Ko, and Omri Weinstein. Approximating the best Nash equilibrium in $n^{o(\log n)}$-time breaks the exponential time hypothesis. In Piotr Indyk, editor, *Proceedings of the Twenty-Sixth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2015, San Diego, CA, USA, January 4-6, 2015*, pages 970–982. SIAM, 2015. doi:10.1137/1.9781611973730.66.

17 Ioannis Caragiannis, Angelo Fanelli, Nick Gravin, and Alexander Skopalik. Approximate pure Nash equilibria in weighted congestion games: Existence, efficient computation, and structure. *ACM Trans. Economics and Comput.*, 3(1):2:1–2:32, 2015. doi:10.1145/2614687.

18 Xi Chen, Chenghao Guo, Emmanouil-Vasileios Vlatakis-Gkaragkounis, Mihalis Yannakakis, and Xinzhi Zhang. Smoothed complexity of local max-cut and binary max-CSP. In Konstantin Makarychev, Yury Makarychev, Madhur Tulsiani, Gautam Kamath, and Julia Chuzhoy, editors, *Proccedings of the 52nd Annual ACM SIGACT Symposium on Theory of Computing, STOC 2020, Chicago, IL, USA, June 22-26, 2020*, pages 1052–1065. ACM, 2020. doi:10.1145/3357713.3384325.

19 George Christodoulou, Vahab S. Mirrokni, and Anastasios Sidiropoulos. Convergence and approximation in potential games. *Theor. Comput. Sci.*, 438:13–27, 2012. doi:10.1016/j.tcs.2012.02.033.

20 Vincent Conitzer and Tuomas Sandholm. New complexity results about Nash equilibria. *Games Econ. Behav.*, 63(2):621–641, 2008. doi:10.1016/j.geb.2008.02.015.

21 Marek Cygan, Fedor V. Fomin, Lukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michal Pilipczuk, and Saket Saurabh. *Parameterized Algorithms*. Springer, 2015. doi:10.1007/978-3-319-21275-3.

22 Argyrios Deligkas, John Fearnley, and Rahul Savani. Inapproximability results for constrained approximate Nash equilibria. *Inf. Comput.*, 262(Part):40–56, 2018. doi:10.1016/j.ic.2018.06.001.

23 Louis Dublois, Tesshu Hanaka, Mehdi Khosravian Ghadikolaei, Michael Lampis, and Nikolaos Melissinos. (in)approximability of maximum minimal FVS. In *ISAAC*, volume 181 of *LIPIcs*, pages 3:1–3:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020.

24 Louis Dublois, Michael Lampis, and Vangelis Th. Paschos. Upper dominating set: Tight algorithms for pathwidth and sub-exponential approximation. *CoRR*, abs/2101.07550, 2021. arXiv:2101.07550.

25 Edith Elkind, Leslie Ann Goldberg, and Paul W. Goldberg. Nash equilibria in graphical games on trees revisited. In Joan Feigenbaum, John C.-I. Chuang, and David M. Pennock, editors, *Proceedings 7th ACM Conference on Electronic Commerce (EC-2006), Ann Arbor, Michigan, USA, June 11-15, 2006*, pages 100–109. ACM, 2006. doi:10.1145/1134707.1134719.

26 Edith Elkind, Leslie Ann Goldberg, and Paul W. Goldberg. Computing good nash equilibria in graphical games. In Jeffrey K. MacKie-Mason, David C. Parkes, and Paul Resnick, editors, *Proceedings 8th ACM Conference on Electronic Commerce (EC-2007), San Diego, California, USA, June 11-15, 2007*, pages 162–171. ACM, 2007. doi:10.1145/1250910.1250935.

27 Robert Elsässer and Tobias Tscheuschner. Settling the complexity of local max-cut (almost) completely. In Luca Aceto, Monika Henzinger, and Jirí Sgall, editors, *Automata, Languages and Programming - 38th International Colloquium, ICALP 2011, Zurich, Switzerland, July 4-8, 2011, Proceedings, Part I*, volume 6755 of *Lecture Notes in Computer Science*, pages 171–182. Springer, 2011. doi:10.1007/978-3-642-22006-7_15.

28 Hiroshi Eto, Tesshu Hanaka, Yasuaki Kobayashi, and Yusuke Kobayashi. Parameterized Algorithms for Maximum Cut with Connectivity Constraints. In *IPEC 2019*, pages 13:1–13:15, 2019.

29 Michael Etscheid and Heiko Röglin. Smoothed analysis of local search for the maximum-cut problem. *ACM Trans. Algorithms*, 13(2):25:1–25:12, 2017. doi:10.1145/3011870.

30 Alex Fabrikant, Christos H. Papadimitriou, and Kunal Talwar. The complexity of pure Nash equilibria. In László Babai, editor, *Proceedings of the 36th Annual ACM Symposium on Theory of Computing, Chicago, IL, USA, June 13-16, 2004*, pages 604–612. ACM, 2004. doi:10.1145/1007352.1007445.

31 Dimitris Fotakis, Vardis Kandiros, Thanasis Lianeas, Nikos Mouzakis, Panagiotis Patsilinakos, and Stratis Skoulakis. Node-max-cut and the complexity of equilibrium in linear weighted congestion games. In Artur Czumaj, Anuj Dawar, and Emanuela Merelli, editors, *47th International Colloquium on Automata, Languages, and Programming, ICALP 2020, July 8-11, 2020, Saarbrücken, Germany (Virtual Conference)*, volume 168 of *LIPIcs*, pages 50:1–50:19. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPIcs.ICALP.2020.50.

32 Dimitris Fotakis, Spyros C. Kontogiannis, Elias Koutsoupias, Marios Mavronicolas, and Paul G. Spirakis. The structure and complexity of Nash equilibria for a selfish routing game. *Theor. Comput. Sci.*, 410(36):3305–3326, 2009. doi:10.1016/j.tcs.2008.01.004.

33 F. Furini, I. Ljubić, and M. Sinnl. An effective dynamic programming algorithm for the minimum-cost maximal knapsack packing problem. *European Journal of Operational Research*, 262(2):438–448, 2017.

34 Itzhak Gilboa and Eitan Zemel. Nash and correlated equilibria: Some complexity considerations. *Games and Economic Behavior*, 1(1):80–93, 1989.

35 Laurent Gourvès and Jérôme Monnot. On strong equilibria in the max cut game. In Stefano Leonardi, editor, *Internet and Network Economics, 5th International Workshop, WINE 2009, Rome, Italy, December 14-18, 2009. Proceedings*, volume 5929 of *Lecture Notes in Computer Science*, pages 608–615. Springer, 2009. doi:10.1007/978-3-642-10841-9_62.

36 Laurent Gourvès, Jérôme Monnot, and Aris Pagourtzis. The lazy bureaucrat problem with common arrivals and deadlines: Approximation and mechanism design. In *FCT*, volume 8070 of *Lecture Notes in Computer Science*, pages 171–182. Springer, 2013.

37 Gianluigi Greco and Francesco Scarcello. On the complexity of constrained Nash equilibria in graphical games. *Theor. Comput. Sci.*, 410(38-40):3901–3924, 2009. doi:10.1016/j.tcs.2009.05.030.

38 Tesshu Hanaka, Hans L. Bodlaender, Tom C. van der Zanden, and Hirotaka Ono. On the maximum weight minimal separator. *Theoretical Computer Science*, 796:294–308, 2019.

39 Elad Hazan and Robert Krauthgamer. How hard is it to approximate the best Nash equilibrium? *SIAM J. Comput.*, 40(1):79–91, 2011. doi:10.1137/090766991.

40 Russell Impagliazzo, Ramamohan Paturi, and Francis Zane. Which problems have strongly exponential complexity? *J. Comput. Syst. Sci.*, 63(4):512–530, 2001. doi:10.1006/jcss.2001.1774.

41 Ken Iwaide and Hiroshi Nagamochi. An improved algorithm for parameterized edge dominating set problem. *J. Graph Algorithms Appl.*, 20(1):23–58, 2016.

42 David S. Johnson, Christos H. Papadimitriou, and Mihalis Yannakakis. How easy is local search? *J. Comput. Syst. Sci.*, 37(1):79–100, 1988. doi:10.1016/0022-0000(88)90046-3.

43 Ioannis Katsikarelis, Michael Lampis, and Vangelis Th. Paschos. Structural parameters, tight bounds, and approximation for (k, r)-center. *Discret. Appl. Math.*, 264:90–117, 2019. doi:10.1016/j.dam.2018.11.002.

44 Ioannis Katsikarelis, Michael Lampis, and Vangelis Th. Paschos. Structurally parameterized d-scattered set. *Discrete Applied Mathematics*, 2020. doi:10.1016/j.dam.2020.03.052.

45 Kaveh Khoshkhah, Mehdi Khosravian Ghadikolaei, Jérôme Monnot, and Florian Sikora. Weighted upper edge cover: Complexity and approximability. *J. Graph Algorithms Appl.*, 24(2):65–88, 2020.

46 Michael Lampis. Parameterized approximation schemes using graph widths. In Javier Esparza, Pierre Fraigniaud, Thore Husfeldt, and Elias Koutsoupias, editors, *Automata, Languages, and Programming - 41st International Colloquium, ICALP 2014, Copenhagen, Denmark, July 8-11, 2014, Proceedings, Part I*, volume 8572 of *Lecture Notes in Computer Science*, pages 775–786. Springer, 2014. doi:10.1007/978-3-662-43948-7_64.

47 Michael Lampis and Valia Mitsou. Treewidth with a quantifier alternation revisited. In Daniel Lokshtanov and Naomi Nishimura, editors, *12th International Symposium on Parameterized and Exact Computation, IPEC 2017, September 6-8, 2017, Vienna, Austria*, volume 89 of *LIPIcs*, pages 26:1–26:12. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2017. doi:10.4230/LIPIcs.IPEC.2017.26.

48 Martin Loebl. Efficient maximal cubic graph cuts (extended abstract). In Javier Leach Albert, Burkhard Monien, and Mario Rodríguez-Artalejo, editors, *Automata, Languages and Programming, 18th International Colloquium, ICALP91, Madrid, Spain, July 8-12, 1991, Proceedings*, volume 510 of *Lecture Notes in Computer Science*, pages 351–362. Springer, 1991. doi:10.1007/3-540-54233-7_147.

49 Lorenz Minder and Dan Vilenchik. Small clique detection and approximate Nash equilibria. In Irit Dinur, Klaus Jansen, Joseph Naor, and José D. P. Rolim, editors, *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques, 12th International Workshop, APPROX 2009, and 13th International Workshop, RANDOM 2009, Berkeley, CA, USA, August 21-23, 2009. Proceedings*, volume 5687 of *Lecture Notes in Computer Science*, pages 673–685. Springer, 2009. doi:10.1007/978-3-642-03685-9_50.

50 Dominik Peters. Graphical hedonic games of bounded treewidth. In Dale Schuurmans and Michael P. Wellman, editors, *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence, February 12-17, 2016, Phoenix, Arizona, USA*, pages 586–593. AAAI Press, 2016. URL: http://www.aaai.org/ocs/index.php/AAAI/AAAI16/paper/view/12400.

51 Svatopluk Poljak. Integer linear programs and local search for max-cut. *SIAM J. Comput.*, 24(4):822–839, 1995. doi:10.1137/S0097539793245350.

52 Alejandro A. Schäffer and Mihalis Yannakakis. Simple local search problems that are hard to solve. *SIAM J. Comput.*, 20(1):56–87, 1991. doi:10.1137/0220004.

53 Grant Schoenebeck and Salil P. Vadhan. The computational complexity of Nash equilibria in concisely represented games. *ACM Trans. Comput. Theory*, 4(2):4:1–4:50, 2012. doi:10.1145/2189778.2189779.

54 M. Zehavi. Maximum minimal vertex cover parameterized by vertex cover. *SIAM Journal on Discrete Mathematics*, 31(4):2440–2456, 2017.

Testing Triangle Freeness in the General Model in Graphs with Arboricity $O(\sqrt{n})$

Reut Levi
Efi Arazi School of Computer Science, The Interdisciplinary Center Herzliya, Israel

Abstract

We study the problem of testing triangle freeness in the general graph model. This problem was first studied in the general graph model by Alon et al. (SIAM J. Discret. Math. 2008) who provided both lower bounds and upper bounds that depend on the number of vertices and the average degree of the graph. Their bounds are tight only when $d_{\max} = O(d)$ and $\bar{d} \leq \sqrt{n}$ or when $\bar{d} = \Theta(1)$, where $d_{\max}$ denotes the maximum degree and $\bar{d}$ denotes the average degree of the graph. In this paper we provide bounds that depend on the arboricity of the graph and the average degree. As in Alon et al., the parameters of our tester is the number of vertices, n, the number of edges, m, and the proximity parameter ϵ (the arboricity of the graph is not a parameter of the algorithm). The query complexity of our tester is $\tilde{O}(\Gamma/\bar{d} + \Gamma) \cdot \text{poly}(1/\epsilon)$ on expectation, where Γ denotes the arboricity of the input graph (we use $\tilde{O}(\cdot)$ to suppress $O(\log \log n)$ factors). We show that for graphs with arboricity $O(\sqrt{n})$ this upper bound is tight in the following sense. For any $\Gamma \in [s]$ where $s = \Theta(\sqrt{n})$ there exists a family of graphs with arboricity Γ and average degree $\bar{d}$ such that $\Omega(\Gamma/\bar{d} + \Gamma)$ queries are required for testing triangle freeness on this family of graphs. Moreover, this lower bound holds for any such Γ and for a large range of feasible average degrees [1].

2012 ACM Subject Classification Theory of computation → Streaming, sublinear and near linear time algorithms

Keywords and phrases Property Testing, Triangle-Freeness

Digital Object Identifier 10.4230/LIPIcs.ICALP.2021.93

Category Track A: Algorithms, Complexity and Games

Funding *Reut Levi*: This research was supported by the Israel Science Foundation grant No. 1867/20.

1 Introduction

Testing triangle-freeness is one of the most basic decision problems on graphs. The existence of triangles in a graph is often a crucial property for various applications. In the realm of property testing, decision problems are relaxed so that a tester for a property $\mathcal{P}$ is only required to distinguish between graphs that have the property $\mathcal{P}$ from graphs which are "far" according to some predetermined distance measure, from having the property $\mathcal{P}$, which in our case are graphs which are far from being triangle free.

Testing triangle freeness is known to be possible with query complexity which only depends on the proximity parameter, ϵ, in graphs which are either dense or sparse. More specifically, Alon, Fischer, Krivelevich and Szegedy [2] showed that in the dense-graphs model [8] it is possible to test triangle-freeness with query complexity which is independent

[1] For a graph, G, whose arboricity is Γ, the number of edges is at most $n \cdot \Gamma$ and at least Γ^2. Thus, the average degree of G is at least Γ^2/n and at most Γ.

of the size of the graph but has a tower-type dependence in $1/\epsilon$. In the other extreme, Goldreich and Ron [9] observed that in the bounded-degree model [9] it is possible to test triangle-freeness with query complexity $O(1/\epsilon)$ given that the maximum degree of the input graph is constant.

Alon, Kaufman, Krivelevich, Ron [3] were the first to study this problem in the general-graphs model [12, 11]. This model is more stringent in the sense that we do not assume anything on the density of the graph and the distance is measured with respect to the actual number of edges in the graph (instead of the maximum possible number of edges). They provided several upper bounds which apply for almost the entire range of average degrees. They also provided lower bounds that show that their upper bounds are at most quadratic in the optimal bounds. Shortly after, Rast [13] and Gugelmann [10] improved their upper bounds and lower bounds, respectively, for some ranges of the parameters.

Although there is a fairly significant gap between the known upper bounds and lower bounds for the vast range of parameters, there has been no progress on this question since then. In this paper we provide an upper bound and several lower bounds which are tight for a large range of parameters. Surprisingly, our bounds depend on the arboricity of the graph although it is not a parameter of our algorithm.

1.1 Results

We provide an upper bound whose running time complexity is $\tilde{O}(\Gamma/\bar{d} + \Gamma) \cdot \mathrm{poly}(1/\epsilon)$ on expectation. Therefore, for $m \leq n$ our upper bound is $\tilde{O}(\Gamma/\bar{d})$ and when $m > n$ our upper bound is $\tilde{O}(\Gamma)$ (ignoring polynomial dependencies in $1/\epsilon$).

We provide three lower bounds, each suitable for a different range of parameters.

1. For any Γ and any feasible $m \geq 1$, we provide a lower bound of $\Omega((\Gamma n)/m) = \Omega(\Gamma/\bar{d})$ queries. Therefore our upper bound is tight when $m \leq n$ (up to polynomial dependencies in $1/\epsilon$ and $O(\log \log n)$ factors).
2. For any Γ and any feasible $m \geq \Gamma^3$ we provide a lower bound of $\Omega(\Gamma)$ queries. Therefore, our upper bound is also essentially tight as long as $m \geq \Gamma^3$ (notice that since $m \leq n \cdot \Gamma$, it is implied that this lower bound applies only for graphs in which $\Gamma = O(n^{1/3})$). Since we may assume that $m \geq n$ (otherwise we already have essentially tight lower bound), one implication of this lower bound is that our upper bound is tight in the strong sense for graphs with arboricity $O(n^{1/3})$ (namely it is tight for any feasible m) as it is always the case that $m \geq \Gamma^3$ for $\Gamma = O(n^{1/3})$.
3. For any $\Gamma \leq (n/2)^{1/2}$ and any feasible $n \leq m \leq \Gamma^3$ we provide a lower bound of $\Omega(m^{1/3})$ queries. Since it is always the case that $m \geq \Gamma^2$, a lower bound of $\Omega(\Gamma^{2/3})$ queries is also implied.

To summarize, for graphs of arboricity $\Gamma = O(n^{1/2})$ we obtain that our upper bound is tight for a large range of average degrees. Additionally, for the range of average degrees in which we do not provide tight bounds, our upper bound is essentially $O(\Gamma)$ while our lower bound is $\Omega(\Gamma^{2/3})$ in the worst case.

1.2 Related Work

1.2.1 Property testing of triangle freeness

As mentioned above, testing triangle freeness, in the context of property testing, was first studied by Alon et al. [2] in the dense graphs model. They showed that triangle freeness can be tested in time which is independent of the size of the graph. However, their upper bound has tower-type dependence in $1/\epsilon$. Alon [1] showed that the query complexity of this problem in the dense-graphs model is indeed super-polynomial in $1/\epsilon$.

In the bounded degree model Goldreich and Ron [9] observed that it is possible to test triangle freeness with query complexity $O(1/\epsilon)$ in graphs of maximum degree bounded by some constant.

The problem of testing triangle freeness in the general graph model was first studied by Alon, Kaufman, Krivelevich, Ron [3]. The query complexity of their algorithms dependent on n and $\bar{d}$, the number of vertices in the graph and the average degree, respectively. They provided sublinear upper bounds for almost the entire range of parameters. Moreover, their upper bounds are at most quadratic in their lower bounds. Specifically, their upper bound, which is combined from several upper bounds is $\tilde{O}(\min\{(n\bar{d})^{1/2}/\epsilon^{3/2}, (n^{4/3}/\bar{d}^{2/3})/\epsilon^2\})$. Their lower bound, which is also combined from several lower bounds, is $\Omega(\max\{(n/\bar{d})^{1/2}, \min\{\bar{d}, n/\bar{d}\}, \min\{\bar{d}^{1/2}, n^{2/3}/\bar{d}^{1/3}\} \cdot n^{-o(1)}\})$.

Rast [13] improved the upper bound of [3] for graphs with average degree in the range $[c_1 n^{1/5}, c_2 n^{1/2}]$ where c_1 and c_2 are some constants. The upper bound in [13] is $O(\max\{(n\bar{d})^{4/9}, n^{2/3}/\bar{d}^{1/3}\})$.

Gugelmann [10] provided a lower bound which improves the lower bound in [3] for graphs with average degree in the range $[c_1 n^{2/5}, c_2 n^{4/5}]$ where c_1 and c_2 are some constants. The lower bound in [10] is $\Omega(\min\{(n\bar{d})^{1/3}, n/\bar{d}\})$.

1.2.2 Sublinear algorithms that receive the arboricity of the graph as a parameter

Eden, Ron and Rosenbaum [5] designed an algorithm that given n, the number of edges of the input graph and an upper bound on the arboricity of the input graph, Γ, the algorithm makes $O(\Gamma/\bar{d} + \log^3 n/\epsilon)$ queries on expectation and samples an edge of the graph almost uniformly. More specifically, each edge in the graph is sampled with probability in the range $[(1-\epsilon)m, (1+\epsilon)m]$.

Eden, Ron and Seshadhri [6] estimate the degree distribution moments of an undirected graph. In particular, for estimating the average degree of a graph, their algorithm has query complexity of $\tilde{O}(\Gamma/\bar{d})$. As they show in their paper, if Γ is not given as an input to the algorithm then estimating the average degree is not possible in general with this query complexity.

In another paper, Eden, Ron and Seshadhri [7] give a $(1 \pm \epsilon)$-approximation for the number of k-cliques in a graph given a bound on the arboricity of the graph Γ. In particular for triangles they provide an upper bound with expected running time, in terms of n, Γ and the number of triangles in the graph, n_3, of $\min\{n\Gamma^2/n_3, n/n_3^{1/3} + (m\Gamma)/n_3\} \cdot \text{poly}(\log n, 1/\epsilon)$.

1.2.3 Testing graphs for bounded arboricity

Eden, Levi and Ron [4] provided an algorithm for testing whether a graph has bounded arboricity. Specifically, they provide a tolerant tester that distinguished graphs that are ϵ-close to having arboricity Γ from which are $c \cdot \epsilon$-far from having arboricity 3Γ, where c is an absolute constant. The query complexity and the running time of their algorithm is in terms of n, m and Γ is $\tilde{O}(n/\sqrt{m} + n\Gamma/m)$ and is quasi-polynomial in $1/\epsilon$.

1.3 Comparison between our upper bound and upper bounds in previous work

As mentioned before, Alon et al. [3] provide tight bounds only in two cases. The first case is when $d_{\max} = O(\bar{d})$ and $\bar{d} \leq \sqrt{n}$, where $d_{\max}$ denotes the maximum degree and $\bar{d}$ denotes the average degree of the graph. In this case, it follows that $\Gamma = \Theta(d_{\max}) = \Theta(\bar{d})$ an so our upper bounds essentially match. Additionally we note that a bound on the arboricty of the graph does not imply a bound on the maximum degree of the graph. In fact, the maximum degree could be $\Theta(n)$ while the arboricity is $\Theta(1)$ (as it is the case in the star graph). Consequently, the tightness of our upper bound is not restricted for graphs which have bounded maximum degree.

The second case is when $\bar{d} = \Theta(1)$, for these graphs the running time complexity of their algorithm is $\tilde{\Theta}(n^{1/2})$. For this case, the running time complexity of our upper bound is $\tilde{O}(\Gamma)$. We note that in graphs in which $\bar{d} = \Theta(1)$, Γ could range between $\Theta(1)$ and $\Theta(n^{1/2})$. Therefore when $\bar{d} = \Theta(1)$ the complexity of our upper bound is not worse than the complexity of the upper bound in [3] but could be much better, depending on Γ.

For average degree in the range between $\Omega(1)$ and $O(n^{2/5})$ and in the range between $\Omega(n^{1/2})$ and $O(n^{2/3})$ the upper bound of $O(m^{1/2})$ queries of Alon et al. achieves the best running time, in terms of n and m. For these ranges, the running time of our algorithm is $\tilde{O}(\Gamma)$. Since $m^{1/2} \geq \Gamma$, we obtain that for this ranges as well the performances of our algorithm are at least as good (up to $O(\log \log n)$ and $\text{poly}(1/\epsilon)$ factors) but could be significantly better.

For average degree in the range between $\Omega(n^{2/5})$ and $O(n^{1/2})$ the upper bound of $O(\max\{(n\bar{d})^{4/9}, n^{2/3}/\bar{d}^{1/3}\})$ queries of Rast [13] achieves the best running time. In this range our upper bound is always better than the upper bound of [13] for graphs of arboricity $O(n^{12/21})$.

1.4 High-level of Our Algorithm

It is well known that a graph which is ϵ-far from being triangle free has $\Omega(\epsilon m)$ edge-disjoint triangles (see Claim 1). Therefore if we were able to sample edges uniformly from the graph then after sampling $O(1/\epsilon)$ edges we would sample an edge $\{u, v\}$ which belongs to a triangle. Thus, if we revealed the entire neighborhood of u and the entire neighborhood of v then we would find a triangle in the graph. Our algorithm is based on this simple approach. There are only two problems that need to be addressed. The first problem is that sampling edges uniformly in a graph in which the degrees have high variability is too costly. The second problem, which also stems from the variability of the degrees in the graph, is that revealing the entire neighborhood of a vertex can be too costly, depending on its degree.

This is where the arboricity of the graph comes into play. For a graph of arboricity Γ, as was shown in [4], the fraction of edges in the subgraph induced on *heavy* vertices, that is, vertices with degree greater than $c\Gamma/\epsilon$ where c is some absolute constant, is at most $\epsilon/2$.

Therefore, if Γ was given to us as a parameter then we could, in some sense, ignore the subgraph induced on vertices of degree greater than $\Theta(\Gamma/\epsilon)$ since a graph which is ϵ-far from being triangle free still have $\Omega(\epsilon m)$ violating edges (and $\Omega(\epsilon m)$ edge-disjoint triangles) even after we remove this subgraph entirely. Ignoring this subgraph allows us on one hand to sample edges almost uniformly from the resulting graph while making only $\Omega(\Gamma/\bar{d})$ queries, and also guarantees that there are $\Omega(\epsilon m)$ violating edges for which both endpoints are not heavy. This solves the two problems we had with taking the simple approach.

However a bound on the arboricty of the graph is not given to the algorithm as a parameter. Since approximating the arboricity of a graph up to a constant factor is not possible in sublinear time (to see this consider a graph with a hidden clique), we estimate a different parameter which we informally refer to as the *effective arboricity* of the graph. We show that this parameter suffices for our needs. In fact, this parameter could be much smaller than Γ, in which case the complexity of our algorithm is better than $O(\Gamma/\bar{d}+\Gamma)$. We reduce the problem of approximating the effective arboricity of the graph to the problem of estimating the number of edges in the graph in which we remove the subgraph induced on heavy vertices, where heavy vertices are defined with respect to increasing thresholds. We stop increasing our threshold once the estimation of the number of edges is sufficiently large. As we prove, with high constant probability, our approximation to the effective arboricity is bounded by $O(\Gamma)$ which leads to a tester with running time $O(\Gamma/\bar{d}+\Gamma)$, as claimed.

1.5 Lower Bounds

Our first lower bound of $\Omega(\Gamma/\bar{d})$ for graphs in which $\bar{d} \leq 1$ is based on a simple hitting argument. Specifically, construct a graph which is 1/3-far from being triangle free in which $\Omega(\Gamma/\bar{d}+\Gamma) = \Omega(\Gamma/\bar{d})$ queries are required in order to sample a vertex which is not isolated with probability that is at least 1/3.

Our other two lower bounds are simple adaptations of the lower bound of $\Omega(\min\{\bar{d}, n/\bar{d}\})$ queries presented in Alon et al. [3].

2 Preliminaries

Let $G = (V, E)$ be an undirected graph and let $\bar{d} = 2m/n$ denote the average degree of G where $n = |V|$ and $m = |E|$. For each vertex $v \in V$, let $\deg(v)$ denote the number of neighbors of v. For a subset of vertices $S \subseteq V$ we denote by $G([S])$ the subgraph induced on S. For a directed graph D we denote by $\bar{d}_{out}(D)$ the average out-degree of D.

A graph G is triangle free if for every three vertices, u, v, w in G at least one pair in $\{\{u, v\}, \{v, w\}, \{w, u\}\}$ is not an edge of G. A graph G is ϵ-far from being triangle free if more than ϵm edges need to be removed in order to make G triangle free.

In the general graph model the tester accesses the graph via the following oracle queries.

1. Degree queries: on query v the oracle returns $\deg(v)$.
2. Neighbor queries: on query (v, i) where $i \in [\deg(v)]$, the oracle returns the i-th neighbor of v.
3. Vertex-pair queries: on query $\{v, u\}$ the oracle returns whether there is an edge between u and v.

An algorithm is a tester for the property of triangle freeness if given a proximity parameter ϵ and access to an input graph G, it accepts G with probability at least 2/3 if G is triangle free and rejects G with probability at least 2/3 if G is ϵ-far from being triangle free. If the tester always accepts graphs which are triangle free we say it has one-side error. Otherwise we say it has two-sided error.

▷ Claim 1. A graph $G = (V, E)$ which is ϵ-far from being triangle free has at least $\epsilon m/3$ edge-disjoint triangles.

Proof. Consider a procedure that given a graph G, as long as there is triangle, t in G it deletes all the edges of t and proceeds in this manner until there are no triangles in the graph. The number of edges which are deleted by this process is at least ϵm. Therefore the number of edge disjoint triangles that are deleted is at least $\epsilon m/3$. The claim follows. ◁

The *arboricity* of an undirected graph G, denoted by $\Gamma(G)$, is the minimum number of forests into which its edges can be partitioned. Equivalently it is the minimum number of spanning forests needed to cover all the edges of the graph.

3 The Algorithm

3.1 First Step: computing the threshold for defining heavy vertices

As described above, for an input graph G, the number of edges in the subgraph induced on the heavy vertices w.r.t. the threshold $4\Gamma(G)/\epsilon$ is at most $(\epsilon/2)|E(G)|$ (see Claim 4). Therefore, when testing triangle freeness, we may, roughly speaking, ignore this subgraph with the hope of obtaining better complexity. Since $\Gamma(G)$ is not given to the algorithm as a parameter, we compute, in Algorithm 1, a different parameter of the graph, denoted by Γ^*, which is, roughly speaking, an approximation of the *effective* arboricity of the graph. In order to specify the guarantees on Γ^* we shall need a couple of definitions.

▶ **Definition 2.** *For a graph $G = (V, E)$ and a threshold t we define the set of* heavy vertices with respect to t *as $H_t(G) = \{v \in V : d(v) > t\}$ and the set of* light vertices with respect to t *as $L_t(G) = V \setminus H_t(G)$.*

When G is clear from the context we may simply use H_t and L_t. Using the definition of $H_t(G)$, we next define the graph $H(G, t)$ which is defined w.r.t. G and a threshold t.

▶ **Definition 3** (The undirected graph $H(G, t)$)**.** *For a graph $G = (V, E)$ and a threshold t, the graph $H(G, t)$ is an undirected graph defined as follows. The set of vertices of $H(G, t)$ is V and the set of edges of $H(G, t)$ is $E(G) \setminus \{\{u, v\} : u \in H_t(G)$ and $v \in H_t(G)\}$. Namely, $H(G, t)$ is the graph G after removing the edges for which both endpoints are heavy with respect to t.*

▷ Claim 4. For a graph G, $\Gamma' \geq \Gamma(G)$ and $\eta \in (0, 1]$ it holds that $|E(H(G, \Gamma'/\eta))| \geq (1 - 2\eta)|E(G)|$.

Proof. We shall prove the claim about $\Gamma' = \Gamma$. The general claim will follow from the fact that $|E(H(G, x)|$ is monotonically non-decreasing in x. Let $G([H_t])$ denote the sub-graph induced on $H_t(G)$ where $t = \Gamma/\eta$ and let k denote the number of edges of this graph. Our goal is to show that $k < 2\eta m$ where $m = |E(G)|$. The sum of the degrees of vertices in $H_t(G)$ is greater than $t \cdot |H_t(G)|$, therefore $m > t \cdot |H_t(G)|/2$. On the other hand, since the arboricity of $G([H_t])$ is also bounded by Γ it follows that $k \leq |H_t(G)| \cdot \Gamma$. Therefore $k < 2m/t \cdot \Gamma = 2\eta m$, as desired. ◁

The guarantees on Γ^*, which is the return value of Algorithm 1 (that will be described next), are as described in the following claim.

▷ Claim 5. With probability at least 5/6, Γ^* returned by Algorithm 1 is such that:

1. $|E(H(G,t))| \geq (1-(\epsilon/6))m$ where $t = \Gamma^*/(48\epsilon)$,
2. $\Gamma^* \leq 2\Gamma(G)$.

Algorithm 1 proceeds in iterations where in each iteration it multiplies Γ^* by a factor of 2, where initially Γ^* is set to 1. It stops when the estimated number of edges in $E(H(G,t))$, for t which is $\Theta(\Gamma^*/\epsilon)$, is at least $|E(G)|(1-\Theta(\epsilon))$. In order to estimate the number of edges in $E(H(G,t))$, Algorithm 1 calls Algorithm 2.

In turn, Algorithm 2 uses the directed graph $D(G,t)$, which we defined momentarily, that is constructed from G and can be accessed by making a constant number of queries to G.

▶ **Definition 6** (The directed graph $D(G,t)$)**.** *The graph $D(G,t)$ is a directed version of the graph $H(G,t)$ in which we orient the edges as follows. For every edge $\{u,v\}$ of $H(G,t)$, we orient the edge from u to v if: (a) $u \in L_t$ and $v \in H_t$ or (b) both $u \in L_t$ and $v \in L_t$ and $id(u) < id(v)$. Otherwise, we orient it from v to u.*

Algorithm 1 Compute Γ^*.

Input: Access to a graph G and parameters n, m and $\epsilon \in (0,1]$
Output: Γ^* as described in Lemma 5

Set $\Gamma_1 = 1$
for $i = 1$ ***to*** $\log n$ **do**
  Run Algorithm 2 on G with parameters n, m, $\epsilon/24$, t_i and $\delta = \Theta(\log\log n)$ where $t_i \stackrel{\text{def}}{=} \Gamma_i/(24\epsilon)$. Let Z_i denote the returned value.
  If $Z_i \leq (1-\epsilon/12)m$ then set $\Gamma_{i+1} = 2\Gamma_i$, otherwise, return Γ_i

Algorithm 2 Estimate the number of edges of $H(G,t)$.

Input: Access to an undirected graph G and parameters parameters n, m, ϵ, t and δ, where $n = |V(G)|$ and $m = |E(G)|$
Output: Estimation to the number of edges of $H(G,t)$

Sample $r = \Theta(\delta^{-1}\epsilon^{-2}t/\bar{d})$, where $\bar{d} = m/n$, vertices $v_1,\ldots,v_r$, uniformly at random from $V(G)$.
For each $i \in [r]$:
  1. Sample a random neighbor of v_i, u. If the edge between u and v_i is oriented from v_i to u in D then set $Y_i = 1$, otherwise set $Y_i = 0$
  2. Set $X_i = \frac{(d_{\text{out}}(v)+d_{\text{in}}(v))\cdot Y_i}{t}$

Return $X = \frac{t\cdot|V(G)|}{r} \cdot \sum_{i\in[r]} X_i$

The following claim specifies the guarantees of Algorithm 2.

▷ Claim 7. Given a query access to a graph G and parameters n, m, ϵ, t and δ where $n = |V(G)|$ and $m = |E(G)|$, Algorithm 2 outputs X such that w.p. at least $1-2^{1/\delta}$, $(1-\epsilon)m' \leq X \leq (1+\epsilon)m'$ if $m' \geq (1-2\epsilon)m$, and $X < (1-\epsilon)m$, otherwise, where $m' = |E(H(G,t))|$.

Proof. First observe that Y_i is an indicator variable to the event that the edge selected in the i-th iteration of Algorithm 2, $\{v_i,u\}$ is an out-edge of v_i in $D(G,t)$. Since $V(G) = V(D(G,t))$ we obtain the following.

$$\mathrm{E}(Y_i) = \frac{1}{|V(D(G,t))|} \cdot \Sigma_{v \in V(D(G,t))} \frac{d_{\mathrm{out}}(v)}{d_{\mathrm{out}}(v) + d_{\mathrm{in}}(v)} \ . \tag{1}$$

Similarly,

$$\begin{aligned} \mathrm{E}(X_i) &= \frac{1}{|V(D(G,t))|} \cdot \Sigma_{v \in V(D(G,t))} \frac{d_{\mathrm{out}}(v) + d_{\mathrm{in}}(v)}{t} \cdot \frac{d_{\mathrm{out}}(v)}{d_{\mathrm{out}}(v) + d_{\mathrm{in}}(v)} \\ &= \frac{1}{|V(D(G,t))|} \cdot \Sigma_{v \in V(D(G,t))} \frac{d_{\mathrm{out}}(v)}{t} = \frac{\bar{d}_{\mathrm{out}}(D(G,t))}{t} \ . \end{aligned} \tag{2}$$

Observe that if $v_i \in H_t(G)$ then $d_{\mathrm{out}}(v) = 0$ and so $Y_i = X_i = 0$. On the other hand, if $v_i \in L_t(G)$ then $d_{\mathrm{out}}(v) + d_{\mathrm{in}}(v) \leq t$. Therefore, in both cases $X_i \in [0,1]$. Thus, for r which is $\Theta(1/(\delta\epsilon^2 E(X_1)))$ it follows by Multiplicative Chernoff's bound that with probability at least $1 - 2^{1/\delta}$,

$$(1-\epsilon)E(X_1) \leq \sum_{i \in [r]} X_i/r \leq (1+\epsilon)E(X_1) \ .$$

And so

$$t \cdot |V(G)| \cdot (1-\epsilon)E(X_1) \leq X \leq t \cdot |V(G)| \cdot (1+\epsilon)E(X_1) \ .$$

Since $t \cdot |V(G)| \cdot E(X_1) = |E(H(G,t))|$ we obtain that

$$(1-\epsilon) \cdot |E(H(G,t))| \leq X \leq (1+\epsilon) \cdot |E(H(G,t))| \ .$$

Hence, if $|E(H(G,t))| \geq (1-2\epsilon)m$ then $\bar{d}_{\mathrm{out}}(D(G,t)) = \Theta(\bar{d}(G))$ and so $\mathrm{E}(X_1) = \Theta(\bar{d}(G)/t)$, implying that $r = \Theta(1/(\delta\epsilon^2 E(X_1)))$, as desired. On the other hand, if $|E(H(G,t))| < (1-2\epsilon)m$, then it is not hard to see that the claim follows by a straightforward coupling argument. More specifically, first assume that $|E(H(G,t))| = (1-2\epsilon)m$ and so by the above, with probability at least $1 - 2^{1/\delta}$,

$$X \leq (1+\epsilon)|E(H(G,t))| = (1+\epsilon)(1-2\epsilon)m < (1-\epsilon)m.$$

Therefore, it follows by a coupling argument that with probability at least $1 - 2^{1/\delta}$, $X < (1-\epsilon)m$ also in the case that $|E(H(G,t))| < (1-2\epsilon)m$. ◁

▷ Claim 8. For an input graph G and parameters n, m, ϵ, t and δ, where $n = |V(G)|$ and $m = |E(G)|$, the time complexity and query complexity of Algorithm 2 is $O(\delta^{-1}\epsilon^{-2}t/\bar{d}(G))$.

Proof. The claim follows from the fact that in order to implement Steps 2.1 and 2.2 of Algorithm 2 the algorithm makes a constant number of queries to G. Specifically, for each v_i the algorithm either performs a single degree query (in case $v_i \in H_t(G)$ then $Y_i = X_i = 0$) or a single adjacency-list query and 2 degree queries in case $v_i \in L_t(G)$ (the orientation of the edge $\{v_i, u\}$ can be determined by the degrees and ids of v_i and u). The implementation of Step 2.2 does not require additional queries as $d_{\mathrm{out}}(v_i) + d_{\mathrm{in}}(v_i) = d_G(v_i)$. ◁

We are now ready to prove Claim 5.

Proof of Claim 5. For the sake of analysis assume that Algorithm 1 performs all $\log n$ iterations of the for-loop. Let E_i denote the event that Z_i is as claimed in Claim 7. By Claim 7, for a fixed i, the probability that E_i occurs is at least $1/(6\log n)$ for an appropriate setting of δ. Therefore, by the union bound the probability that E_i occurs for all $i \in [\log n]$ is at least $5/6$. From this point on we condition on the event that indeed E_i occurs for all $i \in [\log n]$.

Let $m_i = |E(H(G,t_i))|$ for every $i \in [\log n]$. Let j denote the iteration in which Algorithm 1 returns a value. By Step 4 of Algorithm 1, $Z_j > (1-\epsilon/12)m$. By Claim 7, it follows that $m_j \geq (1-\epsilon/6)m$, as desired (to see this note that if $m_j < (1-\epsilon/6)m$ then By Claim 7, $Z_j < (1-\epsilon/12)m$).

To prove the claim about Γ^* we consider the minimum $j' \geq 1$, for which $2^{j'-1} \geq \Gamma$. If $j < j'$ then clearly $\Gamma^* < \Gamma$, as desired. Otherwise, we claim that $j = j'$ (namely, that $\Gamma^* = 2^{j'-1}$) which implies that $\Gamma^* \leq 2\Gamma$, as desired. To see this, first note that by Claim 4, for any $\Gamma' \geq \Gamma$ and $\eta \in (0,1]$, $|E(H(G,\Gamma'/\eta))| \geq (1-2\eta)m$. Therefore $m'_j \geq (1-\epsilon/24)m$. Therefore, by Claim 7, $Z_{j'} \geq (1-\epsilon/24)m'_j \geq (1-\epsilon/24)^2 m > (1-\epsilon/12)m$, which implies that the algorithms stops at the j'-th iteration. ◁

3.2 Second Step: sampling edges almost uniformly given a threshold for heavy vertices

Given a threshold t, Algorithm 3 samples an edge from $H(G,t)$ almost uniformly as described in the next claim.

▷ Claim 9. Algorithm 3 samples an edge from $H(G,t)$ such that for each edge, e, of $H(G,t)$, the probability to sample e is in $\left[\frac{c_1}{m'}, \frac{c_2}{m'}\right]$, where c_1 and c_2 are absolute constants and $m' \stackrel{\text{def}}{=} |E(H(G,t))|$. If t is such that $m' \geq |E(G)|/2$ then the expected running time of the algorithm is $O(t/\bar{d}(G))$.

Proof. Consider a single iteration of the while loop of Algorithm 3. For an edge e in $H(G,t)$ let $p(e)$ denote the probability that e is returned in this iteration of Algorithm 3. If e is an edge such that both endpoints are in $L_t(G)$, then $p(e) = \frac{2}{n} \cdot \frac{1}{t}$. If e is an edge such that one endpoint in $L_t(G)$ and the other endpoint is in $H_t(G)$, then $p(e) = \frac{1}{n} \cdot \frac{1}{t}$. Therefore for any two edges e_1 and e_2 in $H(G,t)$ the probability that e_1 is picked by the algorithm is at most twice the probability that e_2 is picked by the algorithm. Since Algorithm 3 only returns edges in $H(G,t)$, the claim about the probability to sample an edge follows.

For a fixed iteration of the while loop, the probability that the algorithm returns one of the edges of $E(H(G,t))$ is at least $m' \cdot \frac{1}{n} \cdot \frac{1}{t}$ which is at least $\bar{d}(G)/(2t)$ in the case that $m' \geq |E(G)|/2$. Therefore in this case the expected number of iterations of the while loop is at most $(2t/\bar{d}(G))$. Hence the claim about the expected running time follows. ◁

Algorithm 3 Sample an edge from $H(G,t)$ almost uniformly.

Input: Access to a graph G and a parameter t.
1 **while do**
2 Pick u.a.r. a vertex v from $V(G)$
3 Pick u.a.r. $j \in [t]$
4 If $v \in L_t(G)$ and v has a j-th neighbor, u, then return $\{v,u\}$.

▶ Remark 10. We remark that Algorithm 3 is stated as a Las Vegas algorithm. Moreover, if $m < |E(G)|/2$ then we can not obtain from Claim 9 any bound on the expected running time of the algorithm. However, we note that since t and $\bar{d}(G)$ are known then we can set a timeout for the algorithm (specifically $ct/\bar{d}(G)$ for some constant c) and incorporate the event that we were forced to stop the algorithm in the failure probability of the tester.

3.3 Putting things together - the algorithm for testing triangle freeness

Using Algorithms 2 and 3 we are now ready to describe our tester (Algorithm 4).

Algorithm 4 Testing Triangle-Freeness.

Input: Access to a graph G and parameters n, m, and ϵ.
1 Execute Algorithm 2 with parameters n, m and ϵ and let Γ^* denote the returned value
2 Let $t = \Gamma^*/\epsilon$
3 **for** $i = 1$ ***to*** $s = \Theta(\epsilon^{-1})$ **do**
4 Execute Algorithm 3 on G with parameter t and let $\{u, v\}$ denote the edge returned by the algorithm.
5 If both $u \in L_t(G)$ and $v \in L_t(G)$ then return REJECT if $N(u) \cap N(v) \neq \emptyset$
6 Return ACCEPT

Since the tester has one-sided error (it rejects only if it finds a witness for violation, i.e., a triangle) its correctness follows from the following claim.

▷ Claim 11. If G is ϵ-far from being triangle free then **Test Triangle Freeness** finds a triangle with probability at least 2/3. The expected running time of the algorithm is $\tilde{O}\left(\Gamma/\bar{d}(G) + \Gamma\right) \cdot \text{poly}(\epsilon^{-1})$.

Proof. Let $G = (V, E)$ be an input graph which is ϵ-far from being triangle free. There are at least $\epsilon m/3$ edge-disjoint triangles in G (see Claim 1). Let E_1 denote the event that for Γ^* that is return by Algorithm 1 it holds that $|E(H(G,t))| \geq (1 - (\epsilon/6))m$ where $t = \Gamma^*/\epsilon$. By Claim 5 E_1 occurs with probability at least 5/6. Given that E_1 occurred, it follows that there are at least $(\epsilon/3)m - (\epsilon/6)m = (\epsilon/6)m$ edge-disjoint triangles in $H(G,t)$. Let $\{t_1, \ldots, t_k\}$ be an arbitrary subset of these edge-disjoint triangles where $k \stackrel{\text{def}}{=} (\epsilon/6)m$. By the definition of $H(G,t)$ it holds that for every edge $\{u, v\} \in E(H(G,t))$ either $u \in L_t(G)$ or $v \in L_t(G)$. Thus, for every $i \in [k]$ the triangle t_i includes an edge $\{x_i, y_i\}$ such that both x_i and y_i are in $L_t(G)$. Therefore, there are at least k edges in $H(G,t)$ such that if Algorithm 3 returns one of these edges in Step 4 of Algorithm 4 then Algorithm 4 rejects. For every $i \in [s]$, let $E_{2,i}$ denote the event that Algorithm 3 returns one of these edges in the i-th iteration of Algorithm 4.

By Claim 9, given that E_1 occurred, the probability that $E_{2,i}$ occurs (given that the algorithm did not return REJECT before the i-th iteration) is at least $c\epsilon$ for some absolute constant c. Therefore the probability that both E_1 and $E_{2,i}$ occur for some $i \in [s]$ is at least 2/3 for an appropriate setting of s. Thus the algorithm rejects with probability at least 2/3 as desired. ◁

4 Lower Bounds

4.1 Lower bound of $\Omega((\Gamma n)/m)$ for any Γ and any feasible $m \geq 1$

▶ **Theorem 12.** *For any $\Gamma \leq n - 1$ and any $m \geq 1$ which is feasible w.r.t. Γ, any algorithm for testing triangle-freeness must perform $\Omega((\Gamma n)/m)$ queries where n, m and Γ denote the number of vertices, the number of edges and the arboricity of the input graph, receptively. This lower bound holds even if the algorithm is allowed two-sided error.*

Proof. We consider the following graph G over n vertices, m edges and of arboricity Γ. V_1 and V_2 are subsets of $2m/(3\Gamma)$ vertices each and V_3 is a subset of $\Gamma/2$ vertices. V_1, V_2 and V_3 are pairwise disjoint. The edges of the graph are as follows. The sub-graph induced on V_1 and V_3 is a complete bipartite graph with V_1 on one side and V_3 on the other side. Similarly the subgraph induced on V_2 and V_3 is also a complete bipartite graph. Between V_1 and V_2 we take $\Gamma/2$ edge disjoint prefect matchings. Consequently the degree of every node in V_1 and V_2, as well as the arboricity of the graph, is exactly Γ. The number of edge disjoint triangles in the graph is at least $m/3$. To see this consider the following correspondence between an edge $\{u, v\}$ such that $u \in V_1$ and $v \in V_2$ and a triangle in the graph. Let $i \in [\Gamma/2]$ denote the matching for which $\{u, v\}$ belongs to, then triangle that corresponds to $\{u, v\}$ is $\{u, v, w_i\}$.

Therefore the graph is $(1/3)$-far from being triangle free (if $t_1, \ldots, t_{m/3}$ are edge disjoint triangles of G then we need to delete at least one edge per triangle in order to make G triangle free). The number of queries we need to make to hit either V_1 or V_2 is $\Omega((\Gamma n)/m)$. The number of queries we need to make to hit V_3 is $\Omega(n/\Gamma)$. Since $m = \Omega(\Gamma^2)$, we obtain a lower bound of $\Omega((\Gamma n)/m)$ queries in order to hit a vertex from $V_1 \cup V_2 \cup V_3$. Therefore unless the tester makes $\Omega((\Gamma n)/m)$ queries, it can not distinguish between G and the empty graph. The theorem follows. ◀

4.2 Lower bound of $\Omega(\Gamma)$ for any Γ and any feasible $m \geq \Gamma^3$

We adapt the following lower bound of Alon et al. [3].

▶ **Theorem 13.** *Any algorithm for testing triangle-freeness must perform* $\Omega(\min\{\bar{d}, n/\bar{d}\})$ *queries. This lower bound holds even if the algorithm is allowed two-sided error and even for* $d_{\max} = O(\bar{d})$.

Using Theorem 13, we shall prove the following claims.

▷ Claim 14. For any Γ and any feasible m w.r.t. Γ such that $m \geq \Gamma^3$, any algorithm for testing triangle-freeness must perform $\Omega(\Gamma)$ queries, where m and Γ denote the number of edges and the arboricity of the input graph, respectively. This lower bound holds even if the algorithm is allowed two-sided error.

Proof. Assume towards contradiction that there exists an algorithm $\mathcal{A}$ for testing triangle-freeness that is allowed two-sided error and performs $o(\Gamma)$ queries even for input graphs for which $m \geq \Gamma^3$, where m and Γ denote the number of edges and the arboricity of the input graph of $\mathcal{A}$, respectively. We will show that there exists an algorithm $\mathcal{B}$ for testing triangle-freeness (with two-sided error) for graphs in which $M/N = \Gamma$, $N = m/\Gamma$ and the maximum degree is Γ, whose query complexity is $o(\Gamma)$, where M and N denote the number of edges and the number of vertices of the input graph of $\mathcal{B}$, respectively. This will contradict the lower bound in Theorem 13 as $\min\{M/N, N^2/M\} = \min\{\Gamma, m/\Gamma^2\} = \Gamma$, where the last inequality follows from the fact that $m \geq \Gamma^3$.

Let m, n and Γ be such that m is feasible w.r.t. Γ and $m \geq \Gamma^3$. Therefore $\Gamma^3 \leq m \leq \Gamma n$. Let G be a graph over N vertices and M edges for which $M/N = \Gamma$, $N = m/\Gamma$ and the maximum degree is Γ. Given such an input graph G, the algorithm $\mathcal{B}$ simulates $\mathcal{A}$ on another graph, G', that will be described momentarily, and returns the output of $\mathcal{A}$ on G'. The graph G' is constructed from G and has the following properties:

1. G' has arboricity Γ.
2. Any query on G' can be answered by performing at most a single query to G
3. If G is triangle free then G' is triangle free as well.
4. If G is ϵ-far from being triangle free then G' is ϵ-far from being triangle free as well.

G' is simply the graph G with $n - N$ isolated vertices (observe that $n \geq N$ since $m \leq \Gamma n$). Since $M = m$, Item 4 follows. Items 1 and 3 follow from construction and the bound on the maximum degree of G. Item 2 follows from the fact that any query to the graph G' is can be answered either by performing a single query the graph G' or to without performing any query to G (in case the algorithm queries the subgraph induced on the additional $n - N$ isolated vertices of G').

The completeness and soundness of algorithm $\mathcal{B}$ follows from the correctness of algorithm $\mathcal{A}$ and Items 3 and 4, respectively. The claim about the query complexity of algorithm $\mathcal{B}$ follows from Item 2 and the assumption on the query complexity of algorithm $\mathcal{A}$. This completes the proof of the claim. ◁

4.3 Lower bound of $\Omega(m^{1/3})$ for any $\Gamma \leq (n/2)^{1/2}$ and any feasible $n \leq m \leq \Gamma^3$

▷ Claim 15. For any $\Gamma \leq (n/2)^{1/2}$ and any feasible m w.r.t. Γ such that $n \leq m \leq \Gamma^3$, any algorithm for testing triangle-freeness must perform $\Omega(m^{1/3})$ queries, where m, n and Γ denote the number of edges, number of vertices and the arboricity of the input graph, respectively. This lower bound holds even if the algorithm is allowed two-sided error.

Proof. The proof of this claim follows the same lines as the proof of Claim 14. Assume towards contradiction that there exists an algorithm $\mathcal{A}$ for testing triangle-freeness that is allowed two-sided error and performs $o(m^{1/3})$ queries even for input graphs for which $n \leq m \leq \Gamma^3$, where m, n and Γ denote the number of edges, number of vertices and the arboricity of the input graph, respectively. We will show that there exists an algorithm $\mathcal{B}$ for testing triangle-freeness (with two-sided error) for graphs in which $M = N^{3/2} = \Theta(m)$ and the maximum degree is M/N, whose query complexity is $o(M/N)$, where M and N denote the number of edges and the number of vertices of the input graph of $\mathcal{B}$, respectively. This will contradict the lower bound in Theorem 13 as $\min\{M/N, N^2/M\} = M/N$, where the last inequality follows from the fact that $M = N^{3/2}$.

Let m, n and Γ be such that m is feasible w.r.t. Γ and $n \leq m \leq \Gamma^3$. Let G be a graph over N vertices and M edges for which $M = N^{3/2} = m/2$ and for which the maximum degree is M/N. Given such an input graph G, the algorithm $\mathcal{B}$ simulates $\mathcal{A}$ on another graph, G', that will be described momentarily, and returns the output of $\mathcal{A}$ on G'. The graph G' is constructed from G and has the following properties:

1. G' has arboricity Γ.
2. Any query on G' can be answered by performing at most a single query to G
3. If G is triangle free then G' is triangle free as well.
4. If G is ϵ-far from being triangle free then G' is at least $\epsilon/2$-far from being triangle free as well.

G' is composed of the graph G, a complete bipartite graph A over 2Γ vertices and a graph I of $n - N - 2\Gamma$ isolated vertices. Observe that $n - N \geq 2\Gamma$ since $\Gamma^2 \leq n/2$ and $N = M^{2/3} \leq \Gamma^2/2^{2/3} \leq n/2$. The graph A has Γ vertices on each side, A_1 and A_2 and Γ^2 edges. Item 4 follows from the fact that $M = m - \Gamma^2 \geq m - n/2 \geq m/2$. Observe that maximum degree of G is at most Γ since $M/N \leq m^{1/3} \leq \Gamma$. Therefore, Items 1 and 3 follow from the fact that the arboricity of A is Γ and the bound on the maximum degree of G. Item 2 follows from the fact that any query to the graph G' is can be answered either by performing a single query the graph G' or to without performing any query to G (in case the algorithm queries the subgraphs A or I).

The completeness and soundness of algorithm $\mathcal{B}$ follows from the correctness of algorithm $\mathcal{A}$ and Items 3 and 4, respectively. The claim about the query complexity of algorithm $\mathcal{B}$ follows from Item 2 and the assumption on the query complexity of algorithm $\mathcal{A}$. This completes the proof of the claim. ◁

References

1. Noga Alon. Testing subgraphs in large graphs. *Random Struct. Algorithms*, 21(3-4):359–370, 2002. doi:10.1002/rsa.10056.
2. Noga Alon, Eldar Fischer, Michael Krivelevich, and Mario Szegedy. Efficient testing of large graphs. *Combinatorica*, 20(4):451–476, 2000. doi:10.1007/s004930070001.
3. Noga Alon, Tali Kaufman, Michael Krivelevich, and Dana Ron. Testing triangle-freeness in general graphs. *SIAM J. Discret. Math.*, 22(2):786–819, 2008. doi:10.1137/07067917X.
4. Talya Eden, Reut Levi, and Dana Ron. Testing bounded arboricity. *ACM Trans. Algorithms*, 16(2):18:1–18:22, 2020. doi:10.1145/3381418.
5. Talya Eden, Dana Ron, and Will Rosenbaum. The arboricity captures the complexity of sampling edges. In Christel Baier, Ioannis Chatzigiannakis, Paola Flocchini, and Stefano Leonardi, editors, *46th International Colloquium on Automata, Languages, and Programming, ICALP 2019, July 9-12, 2019, Patras, Greece*, volume 132 of *LIPIcs*, pages 52:1–52:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019. doi:10.4230/LIPIcs.ICALP.2019.52.
6. Talya Eden, Dana Ron, and C. Seshadhri. Sublinear time estimation of degree distribution moments: The arboricity connection. *SIAM J. Discret. Math.*, 33(4):2267–2285, 2019. doi:10.1137/17M1159014.
7. Talya Eden, Dana Ron, and C. Seshadhri. Faster sublinear approximation of the number of k-cliques in low-arboricity graphs. In Shuchi Chawla, editor, *Proceedings of the 2020 ACM-SIAM Symposium on Discrete Algorithms, SODA 2020, Salt Lake City, UT, USA, January 5-8, 2020*, pages 1467–1478. SIAM, 2020. doi:10.1137/1.9781611975994.89.
8. Oded Goldreich, Shafi Goldwasser, and Dana Ron. Property testing and its connection to learning and approximation. *Journal of the ACM (JACM)*, 45(4):653–750, 1998.
9. Oded Goldreich and Dana Ron. Property testing in bounded degree graphs. *Algorithmica*, 32(2):302–343, 2002. doi:10.1007/s00453-001-0078-7.
10. L. Gugelmann. Testing triangle-freeness in general graphs: Lower bounds. Bachelor thesis, Dept. of Mathematics, ETH, Zurich, 2006.
11. Tali Kaufman, Michael Krivelevich, and Dana Ron. Tight bounds for testing bipartiteness in general graphs. *SIAM Journal on Computing*, 33(6):1441–1483, 2004. doi:10.1137/S0097539703436424.
12. Michal Parnas and Dana Ron. Testing the diameter of graphs. *Random Struct. Algorithms*, 20(2):165–183, 2002. doi:10.1002/rsa.10013.
13. T. Rast. Testing triangle-freeness in general graphs: Upper bounds. Bachelor thesis, Dept. of Mathematics, ETH, Zurich, 2006.

An Efficient Coding Theorem via Probabilistic Representations and Its Applications

Zhenjian Lu
University of Warwick, Coventry, UK

Igor C. Oliveira
University of Warwick, Coventry, UK

Abstract

A *probabilistic representation* of a string $x \in \{0,1\}^n$ is given by the code of a randomized algorithm that outputs x with high probability (Oliveira, ICALP 2019, [30]). We employ probabilistic representations to establish the first unconditional Coding Theorem in *time-bounded* Kolmogorov complexity. More precisely, we show that if a distribution ensemble $\mathcal{D}_m$ can be uniformly sampled in time $T(m)$ and generates a string $x \in \{0,1\}^*$ with probability at least δ, then x admits a time-bounded probabilistic representation of complexity $O(\log(1/\delta) + \log(T) + \log(m))$. Under mild assumptions, a representation of this form can be computed from x and the code of the sampler in time polynomial in $n = |x|$.

We derive consequences of this result relevant to the study of data compression, pseudodeterministic algorithms, time hierarchies for sampling distributions, and complexity lower bounds. In particular, we describe an *instance-based* search-to-decision reduction for Levin's Kt complexity (Levin, Information and Control 1984, [23]) and its probabilistic analogue rKt [30]. As a consequence, if a string x admits a succinct time-bounded representation, then a near-optimal representation can be generated from x with high probability in polynomial time. This partially addresses in a time-bounded setting a question from [23] on the efficiency of computing an optimal encoding of a string.

2012 ACM Subject Classification Theory of computation

Keywords and phrases computational complexity, randomized algorithms, Kolmogorov complexity

Digital Object Identifier 10.4230/LIPIcs.ICALP.2021.94

Category Track A: Algorithms, Complexity and Games

Related Version *Full Version*: `https://eccc.weizmann.ac.il/report/2021/041/` [28]

Acknowledgements We are grateful to Lijie Chen for a suggestion that allowed us to improve the list size in Corollary 3. We also thank Rahul Santhanam and Rahul Ilango for discussions on search-to-decision reductions, and James Maynard for an email exchange on number-theoretic techniques and the problem of generating primes. Igor would like to thank Michal Koucký for asking a question about the probability threshold in the definition of rKt during the Dagstuhl Seminar "Computational Complexity of Discrete Problems" (2019). This work received support from the Royal Society University Research Fellowship URF\R1\191059.

1 Introduction

Shannon's information theory provides a foundation for the study of data transmission and data compression. However, it inherently considers *probability distributions* and *random variables*, and for this reason it does not apply to an *individual* object. The theory of Kolmogorov complexity on the other hand captures the information or computational content of an individual string or message. Despite significant conceptual differences between the two theories, results obtained in one setting sometimes admit an analogue in the other (see e.g. the textbooks [12, 24, 34]).

A fundamental result connecting the distributional framework of Shannon and the information of an individual object x is the *Coding Theorem* in Kolmogorov complexity [22].[1] This theorem states that if a randomized machine A generates a string x with probability δ, then its Kolmogorov complexity $\mathsf{K}(x)$ is at most $\log(1/\delta) + O_A(1)$. The result is part of a deep and beautiful theory that we will not be able to survey here. For a complexity-theoretic perspective that is closer to our work, we refer to [21], where the coding theorem is referred to as one of the four pillars of Kolmogorov complexity.

An issue with this result and with Kolmogorov complexity more broadly is that many aspects of the theory are *nonconstructive*. For instance, computing or even estimating $\mathsf{K}(x)$ of an input string x is known to be undecidable. This limits the applicability of Kolmogorov complexity and of the aforementioned coding theorem in algorithms and complexity theory.

In order to import methods of Kolmogorov complexity from computability to complexity theory, a number of works have introduced *time-bounded* variants of Kolmogorov complexity (cf. [1, 2, 13, 3] for a survey of results). In other words, one considers the minimum description length of a string x with respect to machines that operate *under a time constraint*. Among many applications, this idea has led Sipser [35] to a proof that BPP is contained in the polynomial hierarchy, and Levin [23, Section 1.3] to further develop universal search and optimal search algorithms.

Naturally, many authors have investigated time-bounded variants of the main results in Kolmogorov complexity.[2] Unfortunately, under standard hardness assumptions some of its most powerful theorems do not survive in time-bounded settings. Two notable examples are the language compression theorem (see [8] and references therein) and the principle of symmetry of information (see [26, 27]).

Our focus in this work is on the coding theorem and its applications. Interestingly, there is no barrier to proving certain versions of the coding theorem in a time-bounded setting. For instance, Fortnow and Antunes [5] have implicitly established a form of this result, *under a strong computational assumption*. While their results are conditional and currently beyond reach without an assumption, they indicate that a useful time-bounded version of the coding theorem might be true.

Before proceeding with our discussion, we recall Levin's time-bounded Kolmogorov complexity. Let M be a deterministic machine and $|M|$ be its description length. For a string $x \in \{0,1\}^*$,

$$\mathsf{Kt}(x) \stackrel{\text{def}}{=} \min_{\text{TM } M,\, t \geq 1} \{|M| + \log t \mid M(\varepsilon) \text{ outputs } x \text{ in } t \text{ steps}\},$$

where $M(\varepsilon)$ denotes the computation of M over the empty string.[3] An important advantage of this definition over Kolmogorov complexity $\mathsf{K}(x)$ is that an encoding of minimal Kt complexity can be computed in exponential time via exhaustive search. Moreover, given an encoding w of x of Kt complexity at most k, we can recover x from w in time at most 2^k. Beyond its established applications in topics such as optimal search algorithms and pseudorandomness, Levin's Kt complexity plays an important role in a few other areas. Recent examples include hardness magnification (e.g. [31, 10]) and the investigation of non-disjoint promise problems [18].

---

[1] Some authors also refer to it as the Source Compression Theorem or Source Coding Theorem.

[2] Troy Lee's PhD thesis [21] contains a particularly nice exposition of the contrast between Kolmogorov complexity and its time-bounded variants, including pointers to relevant references.

[3] To obtain precise results about the encoding lengths, it is necessary to fix a universal machine and to consider short inputs for this machine that produce x, as done by Levin. Since our techniques are not sensitive to encoding choices and our results incur a constant factor overhead in the encoding length, this is not relevant for our discussion.

A *probabilistic version* of Levin's Kt complexity has been recently investigated in [30]. In this definition, we minimize over *randomized* machines that output x with probability at least $2/3$ after computing for at most t steps. In other words,

$$\mathsf{rKt}(x) \stackrel{\text{def}}{=} \min_{\text{RTM } M,\, t\geq 1}\{|M| + \log t \mid M(\varepsilon) \text{ outputs } x \text{ in } t \text{ steps with probability} \geq 2/3\}.$$

Consequently, an rKt description w of a string x provides a *probabilistic representation* of x, in the sense that x can be recovered with high probability from w. (Note that the description itself is a deterministic object.)

Under a mild derandomization assumption, $\mathsf{Kt}(x) = \Theta(\mathsf{rKt}(x))$ for every string x [30, Theorem 5]. If so, this would show that any object that can be succinctly described in a probabilistic way can also be succinctly described in a deterministic way, and that results about rKt can be transferred to Kt. However, we appear to be far from establishing this relation, and it is consistent with our knowledge that there is an infinite sequence $\{x_n\}_{n\geq 1}$ where each x_n is an n-bit string satisfying $\mathsf{rKt}(x) = O(\log n)$ and $\mathsf{Kt}(x) = \Omega(n)$.

We do know without assumptions that various natural objects such as certain n-bit prime numbers can have rKt complexity $n^{o(1)}$ [30, 32], while establishing even an upper bound of $o(n)$ on the Kt complexity of a sequence of n-bit primes would lead to a breakthrough in the deterministic generation of primes [37]. It seems therefore that probabilistic representations are useful to represent data, given our current knowledge of algorithms and complexity.

Another interesting aspect of rKt is that, despite its conjectured equivalence to Kt, we are able to settle basic questions about rKt that remain longstanding conjectures in the case of Kt. For instance, a natural computational problem is to approximate, given a string x, the value $\mathsf{rKt}(x)$. In other words, we would like to decide if a string has a short probabilistic representation. [30] proved *unconditionally* that this problem cannot be solved in probabilistic polynomial time. In contrast, showing that computing Kt cannot be done in deterministic polynomial time (i.e., proving $\mathsf{MKtP} \notin \mathsf{P}$) is an important problem in time-bounded Kolmogorov complexity (cf. [4]).

We are motivated by these intriguing recent results and by the possibility of studying algorithmic information theory from the vantage point of probabilistic descriptions. The following questions are particularly relevant in this context.

1. Is it possible to employ rKt to establish new results in time-bounded Kolmogorov complexity? Specifically, can we establish an *unconditional* version of the coding theorem discussed above?
2. We have natural examples where probabilistic representations might be helpful, but no positive algorithmic results about *finding* such descriptions. Is it possible to compute a probabilistic representation of a string in some non-trivial way?
3. Can we further develop the theory of probabilistic representations and show stronger results for natural objects, such as prime numbers?

1.1 Contributions

We make progress in the context of these questions, obtaining results that suggest new research directions connected to data compression, time-bounded Kolmogorov complexity, search-to-decision reductions, and related areas. In particular, our results highlight the usefulness of *probabilistic representations* in algorithms and complexity. We describe these contributions in detail next.

1.1.1 Main results

For an algorithm or machine A, recall that we use $|A|$ to denote its description length under a fixed encoding scheme. One of our main contributions in this work is to show that an important result in Kolmogorov complexity survives in the time-bounded setting.

▶ **Theorem 1** (Efficient Coding Theorem for rKt). *Suppose there is a randomized algorithm A for sampling strings such that $A(1^m)$ runs in time $T(m)$ and outputs a string $x \in \{0,1\}^*$ of length $n \leq T(m)$ with probability at least $\delta > 0$. Then*

$$\mathsf{rKt}(x) \;=\; O\big(\log(1/\delta) + \log(T(m)) + \log(m)\big),$$

where the constant behind the $O(\cdot)$ depends on $|A|$ and is independent of the remaining parameters. Moreover, given x, m, the code of A, and δ, it is possible to compute with probability ≥ 0.99 some rKt *encoding of x of at most this complexity in time* $\mathsf{poly}(|A|, \log(m), |x|, \log(1/\delta))$.

A few comments are in order. It is not hard to prove an effective coding theorem for a distribution whose *cumulative probability function* is computable in polynomial time (cf. [24]). Other works have established incomparable results under the assumption that deciding if a string is in the support of the distribution is easy (see e.g. [39]). Crucially, the theorem above only assumes that the distribution is *samplable*, which makes it more broadly applicable. To our knowledge, Theorem 1 is the first result that provides a general approach to constructing a probabilistic encoding of a string.

It follows from Shannon's Coding Theorem that the expected encoding length in Theorem 1 is essentially optimal up to a constant factor. An interesting feature of the result is that the algorithm computing the rKt encoding runs in polynomial time regardless of the time complexity of the sampler $A(1^m)$. Furthermore, producing the encoding of a string x only requires knowledge of a lower bound on its probability weight, in contrast to encoding algorithms such as Huffman coding where knowledge of the probabilities of all elements in the support of the distribution is necessary.

Using an argument of Levin (see [16] and [21, Section 5.3] or [28, Section A.3]), under the existence of one-way functions there is a polynomial-time samplable source $\mathcal{D}_n$ supported over $\{0,1\}^n$ such that every $x \in \mathsf{Support}(\mathcal{D}_n)$ has probability weight $\mathcal{D}_n(x) \geq 2^{-n^\varepsilon}$, but $\mathcal{D}_n$ does not admit a pair $(\mathsf{Enc}_n, \mathsf{Dec}_n)$ of efficient (probabilistic) encoding and decoding algorithms such that each $x \in \mathsf{Support}(\mathcal{D}_n)$ is assigned a description of length $\leq n - 3$. In contrast, Theorem 1 is able to sidestep this limitation by using an efficient encoding algorithm whose associated decoding procedure (provided by the rKt representation itself) is not necessarily efficient. There are applications where this trade-off might be acceptable, i.e., where it is crucial to achieve high compression rates and fast (or low energy) encoding, but for which decoding is allowed to take more resources.[4]

Similarly to the coding theorem in Kolmogorov complexity, it is possible to derive a variety of consequences from Theorem 1. We cover in more detail one of its most unexpected implications, deferring the discussion of a couple of other results to Section 1.1.2 below.

A *search-to-decision reduction* is an efficient procedure that allows one to find solutions to a problem from the mere ability to decide when a solution exists. These reductions are particularly important in algorithms and complexity. On the one hand, theories of computational complexity are often easier to develop in the context of *decision problems*.

[4] As a speculative scenario, one can imagine data transmission for deep space exploration.

On the other hand, in practice solving a *search problem* tends to be the relevant task. A search-to-decision reduction for a given problem shows that the complexities of its search and decision versions are similar.

It is well known that any NP-complete problem admits a search-to-decision reduction. However, there are problems of interest for which such reductions are still unknown. A notable example in the realm of time-bounded Kolmogorov complexity is whether MCSP, the Minimum Circuit Size Problem, admits a search-to-decision reduction (cf. [19] for a discussion and a recent result).

We establish the existence of the following search-to-decision reductions for rKt and Kt complexities.

▶ **Theorem 2** (Instance-based search-to-decision reductions). *The following results hold:*

(i) *There is a randomized polynomial-time algorithm that, when given an input string $x \in \{0,1\}^n$ and a value $k \geq \mathsf{rKt}(x)$, outputs with probability ≥ 0.99 a valid* rKt *representation of x of complexity $O(k)$.*

(ii) *Similarly, there is a randomized polynomial-time algorithm that, when given an input string $x \in \{0,1\}^n$ and a value $k \geq \mathsf{Kt}(x)$, outputs with probability ≥ 0.99 a valid* Kt *representation of x of complexity $O(k)$.*

We note that [4] established that any language in deterministic exponential time E can be non-uniformly reduced via polynomial-size circuits to the problem of deciding Kt complexity (or even of just approximating Kt on a large fraction of inputs). Since the problem of finding a minimal Kt representation of an input string can be encoded as a language in E, it follows from their work that there is a polynomial-size search-to-decision reduction for Kt. Observe that the reduction given by [4] finds a description of minimum length, while Theorem 2 only provides a description that is within a constant factor of the optimal description length.

There are now a number of search-to-decision reductions in the context of time-bounded Kolmogorov complexity with respect to a variety of string complexity measures (e.g. [9, 17, 19, 20, 25]). We are not aware, however, of a previous *instance-based* search-to-decision reduction in the sense of Theorem 2. In other words, Theorem 2 shows that it is possible to produce a near-optimal Kt representation of x from a decision oracle for Kt *that is only queried on x*.[5] In contrast, the aforementioned reductions require an oracle to the decision problem that is correct on all or at least on a large fraction of inputs. More broadly, search-to-decision reductions for problems in other domains such as circuit satisfiability and graph theory tend to query inputs that modify the original input x.

As a result of the property described above, we are able to derive consequences from Theorem 2 that do not follow from other reductions. Unsurprisingly, our approach employs significantly different techniques compared to the papers cited above. In particular, it departs from the PRG-based approach of [4] and of a few other subsequent works.

We elaborate next on some aspects of Theorem 2 that make it particularly interesting, at least to the authors. Note that Kt (similarly for rKt and probabilistic algorithms) is a *universal* complexity measure for data compression, in the following precise sense. If a string x can be encoded/decoded by *some* uniform compression scheme in time $\leq T$ and using a description of length $\leq \log T$, then its Kt complexity is at most $O(\log T)$. As a consequence, compressing a string x to $O(\mathsf{Kt}(x))$ bits is not far from optimal *in a strong sense*.

[5] A binary search is sufficient to compute $\mathsf{rKt}(x) \in \mathbb{N}$ from a decision oracle that checks if $\mathsf{rKt}(x) \leq \gamma$ for a given threshold γ.

Now suppose we have a string x of length n, and we estimate its Kt complexity to be at most, say, $O(\sqrt{n})$. Then finding a valid Kt representation of this complexity via *exhaustive search* would take time $2^{O(\sqrt{n})}$. Assuming the widely believed hardness of estimating the Kt complexity of a string (which is provably true for rKt by [30]), one is tempted to conjecture that nothing better can be done. Still, Theorem 2 tells us that there is an algorithm that can produce a valid Kt representation of x of complexity $O(\sqrt{n})$ in time just $\mathsf{poly}(n)$.[6] This result addresses to some extent in the time-bounded setting a question from [23, Page 5] on the efficiency of generating programs of length close to the optimal description length.

It is unclear to us whether there are *deterministic* search-to-decision reductions for Items (i) and (ii) of Theorem 2. Although this requires more investigation, it is possible that these reductions provide natural examples of randomized algorithms that cannot be replaced by deterministic ones with only a polynomial overhead.[7]

1.1.2 Further applications and open problems

Efficient construction of time-bounded programs and complexity lower bounds for estimating rKt. By executing the efficient search-to-decision reduction from Theorem 2 with all values of $k = O(n)$ that are powers of 2, the following corollary is immediate.

▶ **Corollary 3** (Effective short lists with short programs in short time)**.** *Given an arbitrary string x of length n, it is possible to compute with high probability and in polynomial time a collection of at most $d = \log(n) + O(1)$ strings $w_1, \ldots, w_d$ such that at least one of these strings is a valid rKt encoding of x of complexity $O(\mathsf{rKt}(x))$.*

The same result can be obtained for Kt complexity. Corollary 26 should be contrasted with the results from [7, 6] in the context of (time-unbounded) Kolmogorov complexity. They achieve optimal compression rates, by mapping a string of Kolmogorov complexity k to a collection of strings that contains a valid representation of length $k + O(1)$. While we are not able to achieve this level of compression, our setting is more stringent, since we need to construct a short representation that can be decompressed under a time constraint. It would be interesting to explore connections between our techniques and those employed in Kolmogorov complexity to see if our parameters can be further improved.

In light of Corollary 3 and the complexity lower bound for estimating the rKt complexity of a string ([30]; see Theorem 14), it follows that the intractability of estimating rKt does not lie in the exhaustive search required to *find* a succinct representation. Instead, the hardness is a consequence of the intractability of *checking* if a given rKt representation is valid for a string x.

It is unlikely that circuit minimization of a given truth-table (MCSP) admits a search-to-decision reduction of the form given by Theorem 2. This would allow one to construct in time polynomial in the size of the input truth-table a collection of circuits that contains a circuit of near-optimal size for the truth-table. As opposed to rKt, checking if a circuit correctly encodes a string can be done in polynomial time, which implies that such a search-to-decision reduction provides a natural property in the sense of [33]. Consequently, under cryptographic assumptions, minimizing circuit size and minimizing Kt/rKt behave differently with respect to instance-based search-to-decision reductions.

---

[6] This does not contradict the intractability result from [30]. Indeed, it implies that *checking* if a given representation generates a particular string is the problem that requires super-polynomial time.

[7] Note that this is not inconsistent with $\mathsf{P} = \mathsf{BPP}$ because these are not decision problems.

Hardness of approximately sampling distributions. It is not hard to show that if Promise-BPP $\subseteq$ Promise-P then BPTIME[$\cdot$] admits a time hierarchy theorem. This easily follows from the deterministic time hierarchy theorem for decision problems. As a consequence of our results, we observe that a similar derandomization hypothesis for *decision problems* implies a strong time hierarchy theorem for *sampling distributions*.

To our knowledge, a uniform time hierarchy theorem for sampling distributions was first established in [41]. While the results of his work are *unconditional*, [41] left open the problems of proving an *almost-everywhere* lower bound and of achieving a *larger statistical gap* [41, Section 5]. Our next result provides a conditional solution to these questions.

▶ **Theorem 4** (A strong time hierarchy theorem for sampling distributions)**.** *Under the assumption that* Promise-BPE $\subseteq$ Promise-E*, there is a constant $\zeta > 0$ for which the following holds. Let $n^{1/\zeta} \leq T(n) \leq 2^n$ be any constructive time bound. There is an ensemble $\{\mathcal{D}_n\}_{n\geq 1}$ of distributions $\mathcal{D}_n$ such that:*

- **(i)** *Each distribution $\mathcal{D}_n$ is supported over a single string $z_n \in \{0,1\}^n$.*
- **(ii)** *There is a deterministic algorithm $A(1^n)$ that samples $\mathcal{D}_n$ and runs in time $O(T(n))$.*
- **(iii)** *For each randomized algorithm $B(1^n)$ that runs in time $O(T(n)^\zeta)$ and for every large enough n, the statistical distance of $\mathcal{D}_n$ and $B(1^n)$ is at least $1 - 1/T(n)^\zeta$.*

We are not aware of a previous time hierarchy theorem for sampling distributions able to produce hard distributions of support size one, even under computational assumptions. (Interestingly, the unconditional results of [41] hold for support size $k \geq 2$.) Note that Theorem 4 exploits *uniformity* in a crucial way: each distribution $\mathcal{D}_n$ can be sampled by a (non-uniform) linear-size circuit C_n that ignores its random input and outputs the unique string z_n in the support of $\mathcal{D}_n$.

Computational patterns in prime numbers. We now step back and revisit one of the most basic questions associated with the power of probabilistic representations. The problem stated below is concerned with the computational (in)compressibility of n-bit prime numbers.

▶ **Problem 5** (Primes with short descriptions)**.** *Is there an infinite sequence $\{p_n\}_{n\geq 1}$ of prime numbers $p_n \in [2^{n-1}, 2^n - 1]$ such that $\mathsf{rKt}(p_n) = o(n)$?*

This would show that there are primes of *every* large length that admit effective short encodings, i.e., such primes possess computational patterns that translate into effective representations (e.g. Mersenne primes). A partial result appears in [32], where this is proved for *infinitely* many n. Consequently, some primes can have short descriptions. *Is this a rare phenomenon, or does it happen for primes of all lengths?* This is the question captured by Problem 5.

We note that obtaining a solution to Problem 5 is necessary before showing the existence of a deterministic algorithm that generates n-bit primes in time $2^{o(n)}$. We refer to [37] for more background on this problem.

Theorem 1 offers a path to solving this problem. In other words, it shows that it is enough to sample n-bit numbers in time $2^{o(n)}$ in a way that assigns enough weight to some (possibly unknown) prime. Given that many advances to our understanding of prime numbers employ *probabilistic ideas* (see e.g. [38, 29] and references therein), this perspective could be fruitful.

Our last result shows that the existence of a sampler of this form is in fact *equivalent* to a positive solution to Problem 5.

▶ **Theorem 6** (Equivalence between faster samplability and improved time-bounded descriptions)**.** *The following statements are equivalent:*

- **(i)** Sampling Algorithm. *For every $\varepsilon > 0$, there is a randomized algorithm $A(1^n)$ sampling strings in $\{0,1\}^*$ that runs in time $T(n) = O(2^{\varepsilon n})$ and for which the following holds. For every large n, there is an n-bit prime q_n such that $\Pr[A(1^n)$ outputs $q_n] \geq 2^{-\varepsilon n}$.*
- **(ii)** Short Descriptions. *Let $\delta > 0$ be an arbitrary constant. For every large n, there is an n-bit prime p_n such that $\mathsf{rKt}(p_n) \leq \delta n$.*

We stress that the equivalence in Theorem 6 is not particular to prime numbers and to exponential time bounds. It can be seen as the analogue for rKt of Levin's fundamental insight that a sequence of objects (such as n-bit primes or solutions to search problems) can be deterministically generated in time $T(n)$ if and only if they have Kt complexity of order $\log(T(n))$.

Finally, complementing the applications and open problems mentioned above, it would be interesting to understand when a mathematical method employed to show the existence of certain combinatorial objects also implies the existence of objects of bounded rKt complexity. This is particularly interesting in settings where the desired objects are "rare" and the probability of producing them is small (e.g. Lovász local lemma and techniques from discrepancy theory). Extracting rKt upper bounds from existential proofs offers an alternate way of designing non-trivial algorithms for generating the corresponding objects, since it is sufficient to exhaustively search for objects of bounded description length.[8]

1.2 Techniques

In this section, we describe the main conceptual ideas behind Theorems 1 and 2. Since our goal is to establish a coding theorem in a time-bounded setting and our applications require an algorithm that produces a valid encoding in polynomial time, it is not clear if arguments employed in the context of (unbounded) Kolmogorov complexity can be adapted to our setting. For instance, it is not hard to construct an encoding given all strings in the support of the distribution and their corresponding probabilities, but we cannot assume in the time-bounded setting that this is available. Similarly, under additional assumptions on the distribution, such as the ability to compute its cumulative probability function, producing an encoding is easier. However, we are aiming for a result that applies to the larger class of samplable distributions.

Sketch of the proof of Theorem 1. We are given a randomized algorithm $A(1^m)$ that runs in time T and samples from a distribution $\mathcal{D}_m$. Fix a string $x \in \mathsf{Support}(\mathcal{D}_m)$, and assume that its probability weight $\mathcal{D}_m(x) \geq \delta$. Our goal is to (efficiently) produce a succinct probabilistic representation of x in the sense of rKt complexity. The first thing to notice is that there are at most $1/\delta$ strings $y \in \mathsf{Support}(\mathcal{D}_m)$ such that $\mathcal{D}_m(y) \geq \delta$. Let $S_\delta \stackrel{\text{def}}{=} \{y \in \{0,1\}^* \mid \mathcal{D}_m(y) \geq \delta\}$ be the set of such strings, which includes our target string x. We assume for simplicity of the exposition that $S_\delta \subseteq \{0,1\}^n$, where $n = |x|$ is the length of x.

Let's pretend for now that we know the set S_δ. Note that this is not really a realistic assumption, since we are aiming to output a valid rKt description of x in a number of steps that might not even allow us to sample a single string from $\mathcal{D}_m$. We will revisit this assumption later on, and argue that explicit knowledge of S_δ is not needed to produce a probabilistic representation of x.

[8] In some applications, one might need to consider *conditional* rKt complexity. The techniques employed in this work also extend in this direction.

Our current goal is to be able to identify x among the elements of S_δ, ideally with an advice string of length close to $O(\log |S_\delta|) = O(\log(1/\delta))$. A natural way to try to achieve this goal is by producing a "fingerprint" or "hash value" from x that uniquely specifies this string. In other words, we would like to have a function $h\colon S_\delta \to \{0,1\}^*$ such that $h(x) \neq h(y)$ for every $y \in S_\delta \backslash \{x\}$. Then we can identify x in S_δ using h and the value $z = h(x)$. Assuming that we know S_δ, the total description length of x would be upper bounded by roughly $|h| + |z|$, where $|h|$ is the description length of h and $|z|$ is the length of z.

This is a basic algorithmic problem, and one way to achieve this goal with a reasonable upper bound on the description length is as follows. Given an explicit polynomial-time computable error-correcting code $E\colon \{0,1\}^n \to \{0,1\}^{O(n)}$, let $T_\delta \stackrel{\text{def}}{=} E(S_\delta)$, i.e., each string $y' \in T_\delta$ is obtained by applying E to a string $y \in S_\delta$. Consequently, for every distinct pair $y', y'' \in T_\delta$, their relative hamming distance $d(y', y'') = \Omega(1)$. For this reason, it follows by a simple probabilistic analysis that if we *randomly project* about $O(\log(1/\delta))$ coordinates of the strings in T_δ, we are likely to produce a "fingerprint" that uniquely specifies each string. Thus to specify x in S_δ it is enough to compute $E(x)$ and to store $O(\log(1/\delta))$ random pairs (i, b_i), where $i \sim [O(n)]$ and $b_i \stackrel{\text{def}}{=} E(x)_i$, the i-th bit of the string $E(x)$. Following our notation from above, one can think of h as being given by the sequence of coordinates, and z by the bits obtained from the projection of $E(x)$.[9]

There are two potential issues with this approach:

(a) In order to store x's fingerprint information (h and z), we still need $O(|h| + |z|) = O(\log(1/\delta) \cdot \log(n) + \log(1/\delta))$ bits, instead of just $O(\log(1/\delta))$.

(b) We have assumed explicit knowledge of S_δ to recover x from h and z.

The first issue is of a quantitative nature, and it can be handled via standard techniques. By projecting coordinates of $E(x)$ according to a random walk on an explicit constant-degree expander graph on $O(n)$ vertices, the total description length can be reduced to $O(\log(n) + \log(1/\delta))$.

Regarding the more challenging issue (b), first notice that the argument we have described so far uses randomness only to produce h. In particular, it gives a *randomized* algorithm that with high probability produces a *deterministic* representation of x from h, $z = h(x)$, and S_δ. Therefore, we have not yet exploited the power of *probabilistic representations*.

A simple but crucial idea is that we can use the *code* of the sampler A and m to efficiently *compute a valid* rKt *representation* of x, without ever running $A(1^m)$. More precisely, the rKt *instructions* to output x include running $A(1^m)$ about $O((1/\delta) \cdot \log(1/\delta))$ times to collect (with high probability) a superset $W_\delta \supseteq S_\delta$. Let's assume for simplicity that $W_\delta = S_\delta$ (it is possible to take care of extra elements by a more delicate argument). Given the set W_δ, $n = |x|$, and an independently generated h obtained before we compute the rKt representation, h and the hash value $z = h(x)$ are likely to isolate x among the elements in W_δ. A careful implementation of these ideas allows us in randomized polynomial-time to generate an rKt representation of x whose description length is $O(\log(m) + \log(1/\delta) + \log(n))$ and whose logarithm of the running time is $O(\log(T(m)) + \log(1/\delta))$. Since generating an n-bit string x takes time $T \geq n$, the overall rKt complexity (description length + log of the running time) is $O(\log(1/\delta) + \log(T(m)) + \log(m))$. ◀

[9] Notice that an *explicit* error-correcting code together with a bounded number of random coordinates of the string $E(x)$ were used to minimize the description length of x's fingerprint. We can also generate a fingerprint by collecting the value of random XORs χ_S applied to x, but storing the relevant sets S would have been expensive.

Sketch of the proof of Theorem 2. First, we discuss a reduction for rKt. Given a string x and a parameter $k \geq \mathsf{rKt}(x)$, we need to output a valid rKt representation of x of complexity $O(k)$. This is not a lot of information, but we have a general tool at our disposal: the Coding Theorem for rKt (Theorem 1). In particular, if we could sample x in time $2^{O(k)}$ and with probability $2^{-\Omega(k)}$ using an *explicit* algorithm A, we would be done by the moreover part of this result. The only challenge is to uniformly construct an explicit sampler of this form.

Fortunately, there is a *universal* sampler U that works for all strings of rKt complexity at most k. In more detail, the sampler randomly selects the code of a randomized machine M of length at most k, simulates M with its internal randomness for at most 2^k steps, and outputs whatever is left on the output tape of M after this simulation. Using that $\mathsf{rKt}(x) \leq k$, which implies that some randomized machine of length at most k outputs x within 2^k steps with probability at least $2/3$, it is easy to see that x has probability weight at least $2^{-\Omega(k)}$ under U. Given that the code of U is explicit and we know x and k, the desired representation of x can be generated with high probability in polynomial time via Theorem 1.

We now consider a search-to-decision reduction for Kt. Recall that, under a derandomization assumption, $\mathsf{Kt}(x) = \Theta(\mathsf{rKt}(x))$ [30, Theorem 5]. Moreover, it is not hard to adapt the proof to give an efficient deterministic algorithm that converts a probabilistic representation in the sense of rKt into a deterministic one in the sense of Kt. For this reason, it is possible to show, under a plausible computational assumption, that there is an instance-based search-to-decision reduction for Kt.

The most interesting aspect of Item (ii) of Theorem 2 is that it is possible to *unconditionally* establish the result. This is obtained by a more careful investigation of the elements employed in the proofs of Theorem 1 and Theorem 2 Item (i), which reveals that the derandomization assumption is not really needed. We refer the reader to the main body of the paper for the details. ◀

Organization. The proof of Theorem 1 appears in Section 3, while the remaining results are established in Section 4. Due to lack of space, the proof of the strong time hierarchy theorem for sampling distributions (Theorem 4) appears in the full version of the paper [28].

2 Preliminaries

2.1 Basic notions

We use $|x|$ to denote the length of a binary string $x \in \{0,1\}^*$. We abuse notation and use $|M|$ denote the length of the binary encoding of a machine M with respect to a fixed universal machine. Although this will not be essential, we assume a prefix-free encoding of machines, and remark that our statements are robust with respect to encoding choices.

Probabilistic machines have an extra tape with random bits. We use $\boldsymbol{M_{\leq t}}$ to denote a random variable that represents the content of the output tape of M when it computes for t steps over the empty string ε (or its final content if the machine halts before that).

▶ **Definition 7** (rKt_δ Complexity). *For $\delta \in [0,1]$ and a string $x \in \{0,1\}^*$, we let*

$$\mathsf{rKt}_\delta(x) = \min_{M,t} \left\{ |M| + \lceil \log t \rceil \;\middle|\; \Pr[\boldsymbol{M_{\leq t}} = x] \geq \delta \right\}.$$

The randomized time-bounded Kolmogorov complexity of x is given by $\mathsf{rKt}(x) \stackrel{\text{def}}{=} \mathsf{rKt}_{2/3}(x)$.

Levin's complexity $\mathsf{Kt}(x)$ can be defined similarly, either by taking $\delta = 1$ in the definition above or by restricting the minimization over M and t to deterministic machines. For more information about rKt, see [30].

We will assume from our encoding that for every string x of length n, $\mathsf{rKt}(x) \leq \gamma \cdot n$, where $\gamma \in \mathbb{N}$ is a universal constant. This is achieved by a trivial machine that simply stores x and prints it when given an empty string. Since printing a string x takes time at least $|x|$, we have $\mathsf{Kt}(x) \geq \mathsf{rKt}(x) \geq \log |x|$. We might implicitly use this fact to omit certain $\log n$ additive factors in our upper bounds.

These definitions and conventions are sufficient for the formalization of our results, given that the proof of Theorem 1 incurs a constant factor overhead in the resulting rKt upper bound. For the interested reader, we present a careful discussion of the parameters of Theorem 1 in the full version [28, Section A]. For more background in time-bounded Kolmogorov complexity, see [24].

For a discrete probability distribution $\mathcal{D}$, we use $\mathsf{Support}(\mathcal{D})$ to denote its support. For $x \in \mathsf{Support}(\mathcal{D})$, we let $\mathcal{D}(x)$ denote its probability weight under $\mathcal{D}$. We extend this definition to a set T in the natural way, i.e., $\mathcal{D}(T) = \sum_{x \in T} \mathcal{D}(x)$. If $\mathcal{D}$ and $\mathcal{D}'$ are distributions with support contained in a set S, their statistical distance $|\mathcal{D} - \mathcal{D}'|$ is defined as $\max_{T \subseteq [S]} |\mathcal{D}(T) - \mathcal{D}'(T)|$.

▶ **Definition 8** (Entropy). *The entropy $H(\mathcal{D})$ of a discrete probability distribution $\mathcal{D}$ is defined as*

$$H(\mathcal{D}) \;=\; \sum_{x \in \mathsf{Support}(\mathcal{D})} \mathcal{D}(x) \cdot \log \frac{1}{\mathcal{D}(x)},$$

where log *is the binary logarithm function.*

We will also require a standard application of expander graphs in order to minimize the amount of randomness in one of our constructions.

▶ **Definition 9** (Expander Graph). *An m-vertex undirected graph G is an (m, d, λ)-expander if G is d-regular and $\lambda(G) \leq \lambda$, where $\lambda(G)$ denotes the second largest eigenvalue (in absolute value) of the normalized adjacency matrix of G (i.e., the adjacency matrix of G divided by d).*

2.2 Technical tools

The version of the concentration bound appearing below can be found for instance in [40].

▶ **Theorem 10** (Chernoff Bound). *Let $X = \sum_{i=1}^{n} X_i$, where each X_i is an independent $0/1$-valued random variable. The following inequalities hold.*

- **(i)** *For every $\gamma > 0$, if $\mu \geq \mathbf{E}[X]$ then $\mathbf{Pr}[X \geq (1+\gamma)\mu] \leq \left(\frac{e^{\gamma}}{(1+\gamma)^{(1+\gamma)}}\right)^{\mu}$.*
- **(ii)** *For every $0 < \gamma \leq 1$, if $\mu \leq \mathbf{E}[X]$ then $\mathbf{Pr}[X \leq (1-\gamma)\mu] \leq \left(\frac{e^{-\gamma}}{(1-\gamma)^{(1-\gamma)}}\right)^{\mu}$.*

▶ **Theorem 11** (Explicit ECCs; see e.g. [36]). *There is a constant $C \in \mathbb{N}$ and a polynomial-time computable function $E_n \colon \{0,1\}^n \to \{0,1\}^{Cn}$ such that for each $n \geq 1$ and for any distinct strings $a, b \in \{0,1\}^n$, the relative hamming distance between $E_n(a)$ and $E_n(b)$ is at least $1/10$.*

We will rely on the following explicit construction of expander graphs.

▶ **Theorem 12** ((Strongly) Explicit Expander Graphs [14]). *There are constants $d \in \mathbb{N}$ and $0 < \lambda < 1$ for which the following holds. There is an (m, d, λ)-expander family $\{G_m\}$ of m-vertex graphs and a deterministic algorithm A that on inputs $m \in \mathbb{N}$, $v \in [m]$, and $i \in [d]$ outputs the i-th neighbor of v in G_m in time polynomial in $\log(m)$.*

We will also need the following well-known property of a random walk on an expander graph.

▶ **Theorem 13** (Expander Chernoff Bound [15]). *Let $G_m = (V, E)$ be an (m, d, λ)-expander, $f\colon V \to \{0,1\}$ be an arbitrary function, and $\mu \stackrel{\text{def}}{=} \mathbf{E}_{v\sim V}[f(v)]$. Let $v_1 \sim V$ be a uniformly chosen vertex and $v_1, \ldots, v_t$ be a random walk on G of length t. Then, for any $\alpha > 0$,*

$$\Pr_{v_1,\ldots,v_t}\left[\frac{1}{t}\sum_{i=1}^{t} f(v_i) < \mu - \alpha\right] \;\le\; e^{-(1-\lambda)\alpha^2 t/4}.$$

It is known that estimating the rKt complexity of an input string is intractable.

▶ **Theorem 14** (Hardness of estimating rKt [30]). *Let $\varepsilon \in (0,1)$ and $C \in \mathbb{N}$. There is no randomized algorithm A running in time $T(n) = O(n^{(\log n)^C})$ such that for every large enough n:*

- *For every $x \in \{0,1\}^n$ such that $\mathsf{rKt}(x) \le n^{\varepsilon}$, $\Pr_A[A(x) = 1] \ge 2/3$.*
- *For every $x \in \{0,1\}^n$ such that $\mathsf{rKt}(x) \ge n - 10$, $\Pr_A[A(x) = 0] \ge 2/3$.*

3 A Coding Theorem via Probabilistic Representations

3.1 An efficient String Isolation Lemma

The next lemma allows us to efficiently isolate a string x from a collection W of strings using a short advice string v whose length depends on the logarithm of the size of W. We follow a construction described in the proof of [11, Lemma 6.1].

▶ **Lemma 15** (String Isolation Lemma). *There is a deterministic algorithm M for which the following holds. For any set $W \subseteq \{0,1\}^n$ of size ℓ, there exists a string $u \in \{0,1\}^{O(\log(n\cdot\ell))}$ such that $M(1^n, u)$ runs in $\mathsf{poly}(n)$ time and outputs a Boolean circuit that computes a function $H\colon \{0,1\}^n \to \{0,1\}^{O(\log \ell)}$ with the following property:*

$$H(w) \neq H(w') \text{ for every distinct pair } w, w' \in W.$$

Moreover, the same guarantee is achieved by a random string u of the same length with probability at least 0.99.

Proof. Let $E_n\colon \{0,1\}^n \to \{0,1\}^{Cn}$ be the error-correcting code from Theorem 11. Moreover, let $t = c_1 \log \ell$, where c_1 is a large enough universal constant. Finally, let G_{Cn} be the expander graph from Theorem 12, where $m = Cn$.

Given a walk $\gamma = (v_1, \ldots, v_t)$ in G_{Cn}, we define the function $H_\gamma\colon \{0,1\}^n \to \{0,1\}^t$ as follows. On a string z, let $z' = E_n(z)$, and set $H_\gamma(z)_i = z'_{v_i}$, where $v_i \in [Cn]$ is given by γ and z'_j denotes the j-th bit of z'.

For distinct strings $z^1, z^2 \in \{0,1\}^n$, $E_n(z^1)$ and $E_n(z^2)$ have relative distance at least $1/10$. As a consequence, if $\boldsymbol{\gamma}$ is a length-t *random walk* on G_{Cn}, it follows from the Expander Chernoff Bound (Theorem 13) that

$$\Pr_{\boldsymbol{\gamma}}\left[H_{\boldsymbol{\gamma}}(z^1) = H_{\boldsymbol{\gamma}}(z^2)\right] \;\le\; \Pr_{\boldsymbol{\gamma}}\left[\frac{1}{t}\cdot\sum_{i=1}^{t} 1_{[H_{\boldsymbol{\gamma}}(z^1)_i \neq H_{\boldsymbol{\gamma}}(z^2)_i]} < \frac{1}{20}\right] \;\le\; e^{-\Omega(t)} \;<\; \frac{1}{100\ell^2},$$

where we have used that c_1 is a large enough constant in the definition of t. By a union bound over all distinct pairs of strings in W, there is some length-t random walk such that the corresponding function H satisfies the condition of the lemma.

Note that any length-t walk $\gamma = (v_1, \ldots, v_t)$ can be described by a string of length $\log(Cn) + O(t) = O(\log(n \cdot \ell))$, given that G_{Cn} is an m-vertex d-regular graph with $d = O(1)$ and $m = Cn$.

Finally, given 1^n and a description u of a length-t walk γ, it is possible to produce a circuit that computes as the function H_γ in time polynomial in n. This is because E_n, G_{Cn} and the walk encoded by u can be computed in time $\mathsf{poly}(n,t) = \mathsf{poly}(n, \log \ell) = \mathsf{poly}(n)$, where the last step uses that $\ell \leq 2^n$. This completes the proof of the lemma. ◀

The power of Lemma 15 comes from the fact that we don't need to know the set W, i.e., an upper bound on its size is sufficient. We remark that it is possible to achieve better parameters in Lemma 15 if we do not consider the efficiency of M. However, we need an efficient M for our applications in time-bounded Kolmogorov complexity. We also note that in our proofs we will only need that for a particular string $x \in W$ fixed in advance the value $H(x)$ is not contained in the image of H on $W \backslash \{x\}$.

3.2 An efficient Coding Theorem for rKt

We prove Theorem 1 in this section, restated below for convenience of the reader.

▶ **Theorem 16** (Efficient Coding Theorem for rKt)**.** *Suppose there is a randomized algorithm A for sampling strings such that $A(1^m)$ runs in time $T(m)$ and outputs a string $x \in \{0,1\}^*$ of length $n \leq T(m)$ with probability at least $\delta > 0$. Then*

$$\mathsf{rKt}(x) = O(\log(1/\delta) + \log(T(m)) + \log(m)),$$

where the constant behind the $O(\cdot)$ depends on $|A|$ and is independent of the remaining parameters. Moreover, given x, m, the code of A, and δ, it is possible to compute with probability ≥ 0.99 some rKt *encoding of x of at most this complexity in randomized time* $\mathsf{poly}(|A|, \log(m), |x|, \log(1/\delta))$.

Proof. Let x be a string generated with probability at least δ by a sampler $A(1^m)$. Since we only need a *lower bound* on the probability and our rKt upper bound is stated asymptotically, for representation purposes we assume without loss of generality that δ is of the form $2^{-\ell}$ for some $\ell \in \mathbb{N}$. We now show how to obtain an rKt description of x.

With the binary descriptions of m and $1/\delta$, we first run the sampler $A(1^m)$ for $t = 64 \cdot (1/\delta) \cdot \log(1/\delta)$ times to obtain a multi-set of strings S_0 of size t. Let V be the set

$$V \stackrel{\text{def}}{=} \{y \mid A(1^m) \text{ outputs } y \text{ with probability at least } \delta\}.$$

Note that $x \in V$ and that $|V| \leq 1/\delta$. Also, without loss of generality, we assume $\delta < 2/3$; otherwise the sampler $A(1^m)$ yields a desired rKt description of x.

▷ Claim 17. With probability at least $5/6$ *(over S_0)*, every $y \in V$ appears in S_0 at least $\alpha = 32 \cdot \log(1/\delta)$ times.

Proof of Claim 17. Fix a $y \in V$. Note that the expected number of times that y appears in S_0 is at least

$$\mu \stackrel{\text{def}}{=} t \cdot \delta \;=\; 64 \cdot \log(1/\delta).$$

By the Chernoff bound (Item (*ii*) of Theorem 10), we have

$$\Pr_{S_0}[y \text{ appears in } S_0 \text{ less than } \alpha \text{ times}] \;\leq\; \left(\frac{e^{-1/2}}{(1/2)^{(1/2)}}\right)^{64 \cdot \log(1/\delta)} \;<\; \delta^6.$$

By a union bound over the strings in V, where $|V| \leq 1/\delta$, we have that the probability that there exists a $y \in V$ such that y appears in S_0 less than α times is less than $\delta^5 < 1/6$. ◁

▷ Claim 18. With probability at most 1/6 *(*over S_0*)*, there exists a string z in S_0 such that z appears at least $\alpha = 32 \cdot \log(1/\delta)$ times in S_0 and that $A(1^m)$ outputs z with probability less than $\delta/32$.

Proof of Claim 18. Let's view S_0 as an ordered multi-set $(z_1, z_2, \ldots, z_t)$. For $z_i \in S_0$, we say "z_i is bad" if that $A(1^m)$ outputs z_i with probability less than $\delta/32$ and that it appears in S_0 at least α times. Let $\mathcal{E}$ be the event that there exists some $z_i \in S_0$ such that z_i is bad. Then we have

$$\mathbf{Pr}[\mathcal{E}] \;\leq\; \sum_{i=1}^{t} \mathbf{Pr}[z_i \text{ is bad}]. \tag{1}$$

Fix an $i \in [t]$, we have

$$\begin{aligned}
\mathop{\mathbf{Pr}}_{z_1,z_2,\ldots,z_t \sim A}[z_i \text{ is bad}] \;&=\; \sum_{\substack{z \,:\, A \text{ outputs } z \\ \text{w.p. less than } \delta/32}} \mathbf{Pr}[z_i = z \text{ AND } z_i \text{ appears in } S_0 \text{ at least } \alpha \text{ times}] \\
&\leq \sum_{z \text{ as above}} \mathbf{Pr}[z_i \text{ appears in } S_0 \text{ at least } \alpha \text{ times} \mid z_i = z] \cdot \mathbf{Pr}[z_i = z] \\
&\leq \max_{z \text{ as above}} \mathbf{Pr}[z_i \text{ appears in } S_0 \text{ at least } \alpha \text{ times} \mid z_i = z] \\
&\leq \max_{z \text{ as above}} \mathbf{Pr}[z \text{ appears } S_0 \backslash \{z_i\} \text{ at least } \alpha - 1 \text{ times}]. \qquad (2)
\end{aligned}$$

Note that if for a string z, A outputs z with probability less than $\delta/32$, then the expected number of times that z appears in the multi-set $S_0 \backslash \{z_i\}$ is less than

$$\mu \stackrel{\text{def}}{=} (t-1) \cdot \delta/32 \;=\; 2 \cdot \log(1/\delta) - \delta/32,$$

and hence $\alpha - 1 > 11 \cdot \mu$. Then by the Chernoff bound (Item (i) of Theorem 10), we have

$$\textit{Equation (2)} \;\leq\; \left(\frac{e^9}{10^{10}}\right)^{\log(1/\delta)} \;<\; \delta^{18}.$$

Therefore, we have

$$\textit{Equation (1)} \;\leq\; t \cdot \delta^{18} \;<\; 64 \cdot (1/\delta)^2 \cdot \delta^{18} \;<\; 1/6,$$

as desired. ◁

Next, from S_0, we build a set S by removing every string in S_0 that appears less than $\alpha = 32 \cdot \log(1/\delta)$ times in S_0, and keeping only one copy for each of the remaining strings. Let W be the set

$$W \stackrel{\text{def}}{=} \{w \mid A(1^m) \text{ outputs } w \text{ with probability at least } \delta/32\}.$$

Note that $|W| \leq 32/\delta$. From Claim 17 and Claim 18, we get that with probability at least 2/3 over the choices of S_0, we obtain a "good" set S in the sense that $V \subseteq S$ (hence $x \in S$) and $S \subseteq W$.

Consider the algorithm in the String Isolation Lemma (Lemma 15), and let $n = |x| \leq T(m)$. Let u be a string of length $O(\log(n \cdot |W|)) = O(\log(T(m)) + \log(1/\delta))$ such that the algorithm in Lemma 15 running on u produces a good hash function $H\colon \{0,1\}^n \to \{0,1\}^\tau$ for all length-n strings in W, where $\tau = O(\log(|W|)) = O(\log(1/\delta))$. That is, for every distinct

$w, w' \in W \cap \{0,1\}^n$, we have $H(w) \neq H(w')$. Then for every "good" set S, H maps the strings in S of length n into different buckets. Therefore, given u, n, and the hash value for x, $H(x)$, we can recover x from a good set S.

The whole algorithm runs in randomized time $\mathrm{poly}(m, T(m), 1/\delta)$ and requires an advice of length $O(\log(1/\delta) + \log(m) + \log(n))$, which gives the desired upper bound for $\mathsf{rKt}(x)$.

For the moreover part of the theorem, first observe that to output a description of x it is not necessary to run the sampling algorithm $A(1^m)$ nor to know an upper bound on its running time. We need instead the following information: the code of A, the input parameter m, the probability lower bound δ, the length $n = |x|$, the advice string u given by Lemma 15, and the hash value $H(x)$. Crucially, the proof of Lemma 15 shows that a *random* string u (encoding a random walk) satisfies the conditions of the lemma with probability ≥ 0.99. Furthermore, the same proof shows that, given u and x, we can compute $H(x)$ in time $\mathsf{poly}(n) = \mathsf{poly}(|x|)$. Therefore, by sampling a random string u of an appropriate length that depends on n and δ, it is possible to compute a correct description of x with probability at least 0.99. The necessary information can be computed from x, the code of A, n, and δ in time $\mathsf{poly}(|A|, \log(m), |x|, \log(1/\delta))$. ◀

4 Applications

4.1 Equivalence between samplability and succinct probabilistic descriptions

It will be useful in the proof of some results to introduce the following sampler.

Definition of $U(1^m)$.

1. Given 1^m, U samples an integer $\ell \sim [m]$ uniformly at random. It then samples a uniformly random string z of length ℓ, and an independent uniformly random string r of length 2^m.
2. U interprets z as the code of a randomized machine M_z, simulates M_z on the empty input string with randomness r for 2^m steps, and outputs the string y that is left on the output tape of M_z after this simulation.

This sampler satisfies the following properties.

▷ Claim 19. On every input 1^m and for every choice of its randomness, $U(1^m)$ runs in time at most $2^{O(m)}$.

Proof. This is immediate from the definition. ◁

▷ Claim 20. Suppose $x \in \{0,1\}^*$ is a string such that $\mathsf{rKt}(x) \leq k$. Then the probability of x under $U(1^k)$ is at least $(1/k) \cdot 2^{-k} \cdot (2/3)$.

Proof. Since $\mathsf{rKt}(x) \leq k$, there is a randomized machine M running in time at most t such that

$$\Pr[\boldsymbol{M}_{\leq \boldsymbol{t}} = x] \geq 2/3 \quad \text{and} \quad |M| + \log t \leq k.$$

Let $\ell = |M| \leq k$, and note that $t \leq 2^k$. It is clear that x is output by $U(1^k)$ with probability at least $(1/\ell) \cdot 2^{-\ell} \cdot (2/3) \geq (1/k) \cdot 2^{-k} \cdot (2/3)$. ◁

We state next an immediate consequence of the coding theorem for rKt and the existence of universal time-bounded samplers. For concreteness, we focus on the generation of prime numbers in the context of Problem 5.

▶ **Theorem 21** (Equivalence between fast samplability and short time-bounded descriptions)**.** *The following statements are equivalent:*

- **(i)** Sampling Algorithm. *For every $\varepsilon > 0$, there is a randomized algorithm $A(1^n)$ sampling strings in $\{0,1\}^*$ that runs in time $T(n) = O(2^{\varepsilon n})$ and for which the following holds. For every large n, there is an n-bit prime q_n such that $\Pr[A(1^n) \text{ outputs } q_n] \geq 2^{-\varepsilon n}$.*
- **(ii)** Short Descriptions. *Let $\delta > 0$ be an arbitrary constant. For every large n, there is an n-bit prime p_n such that $\mathsf{rKt}(p_n) \leq \delta n$.*

Proof. That samplability as in the first item leads to short descriptions follows immediately from Theorem 16 by taking $\varepsilon > 0$ small enough as a function of the given $\delta > 0$. For the other direction, for a given $\varepsilon > 0$, we consider the sampler $A(1^n) \stackrel{\text{def}}{=} U(1^m)$ with $m = \varepsilon' n$, where $\varepsilon' = \varepsilon/C$ for a large constant C. By Claim 19, $A(1^n)$ runs in time $O(2^{\varepsilon n})$. Under the assumption that (ii) holds, Claim 20 guarantees that for large n there is an n-bit prime q_n that is output by $A(1^n)$ with probability at least $2^{-\varepsilon n}$. ◀

This should be contrasted with the equivalence between the existence of primes of bounded Kt complexity and fast deterministic generation of primes. As mentioned in Section 1.1.2, the equivalence in Theorem 21 also holds in a broader sense.

4.2 Instance-based search-to-decision reduction for rKt and its consequences

In this section, we show that there is a *uniform* polynomial-time "approximate" search-to-decision reduction for rKt. More precisely, we show that given a *linear* approximation of the rKt complexity of the input string, it is possible to output a valid rKt representation that is optimal up to a *constant factor*.

We will need the following definition.

▶ **Definition 22.** *Let $\gamma\colon \mathbb{N} \to \mathbb{R}$. We say that a function $\mathcal{O}\colon \{0,1\}^* \to \mathbb{N}$ γ-approximates rKt complexity if for every string $x \in \{0,1\}^*$,*

$$\frac{\mathsf{rKt}(x)}{\gamma(|x|)} \;\leq\; \mathcal{O}(x) \;\leq\; \gamma(|x|) \cdot \mathsf{rKt}(x).$$

If this holds for a constant $\gamma \in \mathbb{N}$, we say that $\mathcal{O}$ linearly approximates rKt.

▶ **Theorem 23** (An instance-based search-to-decision reduction for rKt)**.** *Let $\mathcal{O}$ be a function that linearly approximates rKt complexity. There is a randomized polynomial-time algorithm with access to $\mathcal{O}$ that, when given an input string x, outputs with probability ≥ 0.99 a valid rKt representation of x of complexity $O(\mathsf{rKt}(x))$. Furthermore, this algorithm makes a single query q to $\mathcal{O}$, where $q = x$.*

Proof. Given an input $x \in \{0,1\}^*$, let $\widetilde{k} \stackrel{\text{def}}{=} \mathcal{O}(x)$ be the estimate provided by the oracle, and recall that $\mathsf{rKt}(x)/C \leq \widetilde{k} \leq C \cdot \mathsf{rKt}(x)$ for a universal constant C. By Claim 20, we get that the universal sampler $U(1^{C\cdot\widetilde{k}})$ outputs x with probability at least $\delta \stackrel{\text{def}}{=} (1/(C\cdot\widetilde{k}))\cdot 2^{-C\cdot\widetilde{k}}\cdot(2/3)$. In addition, by Claim 19, the running time of this sampler is bounded by $2^{O(C\cdot\widetilde{k})}$. It then follows from the "moreover" part of Theorem 16 that it is possible to efficiently compute with probability at least 0.99 a valid rKt representation of x of complexity $O(C \cdot \widetilde{k}) = O(C^2 \cdot \mathsf{rKt}(x)) = O(\mathsf{rKt}(x))$. ◀

As explained in Section 1, an interesting aspect of this reduction is that it works on an *input by input* basis. In other words, if we are able to linearly approximate the rKt complexity of the input string, then we can compute a near-optimal representation for the same string.

We note that a reduction that works on an input by input basis is easy to come up with in the setting of (time-unbounded) Kolmogorov complexity: from an upper bound on $K(x)$, it is possible to compute a string that represents x of complexity at most $K(x)$ simply by running in parallel all machines of at most this description length. Theorem 23 can be interpreted as an analogue result in the more challenging setting of time-bounded Kolmogorov complexity.

Using ideas from the proof of the coding theorem for rKt, we can also show an *unconditional* instance-based search-to-decision reduction for Kt.

▶ **Theorem 24** (An instance-based search-to-decision reduction for Kt)**.** *Let $\mathcal{O}$ be a function that linearly approximates Kt complexity. There is a randomized polynomial-time algorithm with access to $\mathcal{O}$ that, when given an input string x, outputs with probability ≥ 0.99 a valid Kt representation of x of complexity $O(\mathsf{Kt}(x))$. Furthermore, this algorithm makes a single query q to $\mathcal{O}$, where $q = x$.*

Proof. We are given an input $x \in \{0,1\}^*$ where $|x| = n$, and $\mathsf{Kt}(x)/C \leq \widetilde{k} \leq C \cdot \mathsf{Kt}(x)$ for a universal constant C, where $\widetilde{k}$ is obtained via a single query to $\mathcal{O}$ on x. Our encoding algorithm is as follows:

1. Pick $u \in \{0,1\}^d$ uniformly at random, where $d = O\Big(\log\Big(n \cdot 2 \cdot 2^{C \cdot \widetilde{k}}\Big)\Big) = O(\widetilde{k} + \log(n))$.
2. Invoke the deterministic algorithm $M(1^n, u)$ from Lemma 15 to obtain a hash function $H\colon \{0,1\}^n \to \{0,1\}^{O(\widetilde{k})}$, and compute $z = H(x)$.
3. Output $(n, \widetilde{k}, u, z)$.

It is easy to verify that the output of the above procedure has length

$$O(\log(\widetilde{k}) + \log(n)) = O(\mathsf{Kt}(x) + \log(n)) = O(\mathsf{Kt}(x)).$$

Next, we claim that with probability at least 0.99 over the choice of u, the tuple generated in Step 3 can be easily converted into a valid Kt representation of x. (Note that the randomness is only over the encoding algorithm that generates the Kt representation, which provides a deterministic description.) To decode x from this information, we first run $M(1^n, u)$ from Lemma 15 to recover the hash function H. We then enumerate each deterministic machine of description length at most $C \cdot \widetilde{k}$ and simulate it for at most $2^{C \cdot \widetilde{k}}$ steps. For each output y of these machines that is in $\{0,1\}^n$, we compute $H(y)$ and output the first y such that $H(y) = z$. Note that there are at most $\ell = 2 \cdot 2^{C \cdot \widetilde{k}}$ distinct machines of length at most $C \cdot \widetilde{k}$, and each machine produces at most one output string. By the fact that $\mathsf{Kt}(x) \leq C \cdot \widetilde{k}$, at least one of them will output x. Also, by Lemma 15, for at least a fraction of .99 of the u's, the hash function H obtained from u isolates every n-bit string in the set of outputs of these ℓ machines, and x is the only string such that $H(x) = z$. Therefore, for any such "good" u, the string x can be decoded correctly from n, $\widetilde{k}$, u, and z. It is easy to see that the running time of the decoding algorithm is $\mathrm{poly}(n) \cdot 2^{O(\widetilde{k})}$, so the Kt complexity of the resulting representation for x is $O(\mathsf{Kt}(x))$. ◀

Note that Theorem 23 (search-to-decision reduction for rKt) and Theorem 24 (search-to-decision reduction for Kt) complete the proof of Theorem 2 from Section 1.

The next result should be contrasted with the unconditional lower bound of [30] for estimating the rKt complexity of an input string x.

▶ **Theorem 25** (Efficient generation of rKt descriptions). *There is a randomized polynomial-time algorithm E such that on any input string $x \in \{0,1\}^n$, given as advice a string $\alpha = \alpha(x)$ of length $\leq \log\log n + O(1)$, $E(x,\alpha)$ outputs with probability ≥ 0.99 an* rKt *description of x of complexity $O(\mathsf{rKt}(x))$.*

Proof. Algorithm E computes on $x \in \{0,1\}^n$ as follows. It expects $\alpha \in \{0,1\}^{\log\log n + O(1)}$ to encode a tight approximation $\tilde{k}$ to the value $k \stackrel{\text{def}}{=} \mathsf{rKt}(x) \in [\gamma \cdot n]$, where $\gamma \in \mathbb{N}$ is a universal constant, and $\gamma \cdot n$ is an upper bound on $\max\{\mathsf{rKt}(x) \mid x \in \{0,1\}^n\}$. Since we are only aiming for a constant factor approximation of $\mathsf{rKt}(x)$, we use α to encode the smallest integer β such that $2^\beta \geq \mathsf{rKt}(x)$. Since $\beta \leq \log n + O_\gamma(1)$, α can be encoded with just $\log\log n + O(1)$ bits, and a value $\tilde{k}$ such that $k \leq \tilde{k} \leq 2k$ can be obtained from α. $E(x,\alpha)$ considers the universal sampler $U(1^{\tilde{k}})$ as the algorithm $A(1^{\tilde{k}})$ in Theorem 16. Furthermore, it sets $\delta \stackrel{\text{def}}{=} (1/\tilde{k}) \cdot 2^{-\tilde{k}} \cdot (2/3)$. By Claims 19 and 20, $A(1^{\tilde{k}}) \equiv U(1^{\tilde{k}})$ outputs x with probability at least δ and in time at most $T = 2^{O(\tilde{k})}$. As a consequence, from the "moreover" part of Theorem 16 it follows that $E(x,\alpha)$ can compute with probability at least 0.99 a valid rKt representation of x of complexity $O(\log(\tilde{k}) + \log(T) + \log(1/\delta)) = O(\tilde{k}) = O(\mathsf{rKt}(x))$ in time $\mathsf{poly}(|U|, \log(\tilde{k}), n, \log(1/\delta)) = \mathsf{poly}(n)$. ◀

The next corollary is immediate from Theorem 23.

▶ **Corollary 26** (Time-bounded short lists with short programs in short time). *Given an arbitrary string x of length n, it is possible to compute with high probability and in polynomial time a collection of at most $d = \log(n) + O(1)$ strings $w_1, \ldots, w_d$ such that at least one of these strings is a valid* rKt *encoding of x of complexity $O(\mathsf{rKt}(x))$.*

Proof. We can enumerate all values of $k \in [O(n)]$ that are a power of 2, and run the instance-based search-to-decision reduction from Theorem 23 on x using k as the query answer. One of such k will be a linear approximation of $\mathsf{rKt}(x)$, and the output of the search-to-decision reduction for this k will be a rKt description of x with complexity $O(\mathsf{rKt}(x))$. ◀

References

1. Eric Allender. Applications of time-bounded Kolmogorov complexity in complexity theory. In *Kolmogorov complexity and computational complexity*, pages 4–22. Springer, 1992.
2. Eric Allender. When worlds collide: Derandomization, lower bounds, and Kolmogorov complexity. In *International Conference on Foundations of Software Technology and Theoretical Computer Science* (FSTTCS), pages 1–15. Springer, 2001.
3. Eric Allender. The complexity of complexity. In *Computability and Complexity*, pages 79–94. Springer, 2017.
4. Eric Allender, Harry Buhrman, Michal Koucký, Dieter van Melkebeek, and Detlef Ronneburger. Power from random strings. *SIAM J. Comput.*, 35(6):1467–1493, 2006.
5. Luis Filipe Coelho Antunes and Lance Fortnow. Worst-case running times for average-case algorithms. In *Conference on Computational Complexity* (CCC), pages 298–303, 2009.
6. Bruno Bauwens, Anton Makhlin, Nikolai K. Vereshchagin, and Marius Zimand. Short lists with short programs in short time. *Comput. Complex.*, 27(1):31–61, 2018.
7. Bruno Bauwens and Marius Zimand. Linear list-approximation for short programs (or the power of a few random bits). In *Conference on Computational Complexity* (CCC), pages 241–247, 2014.
8. Harry Buhrman, Troy Lee, and Dieter van Melkebeek. Language compression and pseudorandom generators. *Comput. Complex.*, 14(3):228–255, 2005.

9 Marco L. Carmosino, Russell Impagliazzo, Valentine Kabanets, and Antonina Kolokolova. Learning algorithms from natural proofs. In *Conference on Computational Complexity* (CCC), pages 10:1–10:24, 2016.

10 Lijie Chen, Ce Jin, and R. Ryan Williams. Hardness magnification for all sparse NP languages. In *Symposium on Foundations of Computer Science* (FOCS), pages 1240–1255, 2019.

11 Lijie Chen, Ce Jin, and R. Ryan Williams. Sharp threshold results for computational complexity. In *Symposium on Theory of Computing* (STOC), 2020.

12 Thomas M. Cover and Joy A. Thomas. *Elements of Information Theory*. Wiley, 2006.

13 Lance Fortnow. Kolmogorov complexity and computational complexity. *Complexity of Computations and Proofs. Quaderni di Matematica*, 13, 2004.

14 Ofer Gabber and Zvi Galil. Explicit constructions of linear-sized superconcentrators. *J. Comput. Syst. Sci.*, 22(3):407–420, 1981.

15 David Gillman. A Chernoff bound for random walks on expander graphs. *SIAM J. Comput.*, 27(4):1203–1220, 1998.

16 Andrew V. Goldberg and Michael Sipser. Compression and ranking. *SIAM J. Comput.*, 20(3):524–536, 1991.

17 Shuichi Hirahara. Non-black-box worst-case to average-case reductions within NP. In *Symposium on Foundations of Computer Science* (FOCS), pages 247–258, 2018.

18 Shuichi Hirahara. Non-disjoint promise problems from meta-computational view of pseudorandom generator constructions. In *Conference on Computational Complexity* (CCC), 2020.

19 Rahul Ilango. Connecting Perebor conjectures: Towards a search to decision reduction for minimizing formulas. In *Computational Complexity Conference* (CCC), 2020.

20 Rahul Ilango, Bruno Loff, and Igor C. Oliveira. NP-hardness of circuit minimization for multi-output functions. In *Computational Complexity Conference* (CCC), 2020.

21 Troy Lee. *Kolmogorov complexity and formula lower bounds*. PhD thesis, University of Amsterdam, 2006.

22 Leonid A. Levin. Laws of information conservation (nongrowth) and aspects of the foundation of probability theory. *Problemy Peredachi Informatsii*, 10(3):30–35, 1974.

23 Leonid A. Levin. Randomness conservation inequalities; information and independence in mathematical theories. *Information and Control*, 61(1):15–37, 1984.

24 Ming Li and Paul M. B. Vitányi. *An Introduction to Kolmogorov Complexity and Its Applications*. Texts in Computer Science. Springer, 2008.

25 Yanyi Liu and Rafael Pass. On one-way functions and Kolmogorov complexity. *IACR Cryptol. ePrint Arch.*, 2020:423, 2020.

26 Luc Longpré and Sarah Mocas. Symmetry of information and one-way functions. *Inf. Process. Lett.*, 46(2):95–100, 1993.

27 Luc Longpré and Osamu Watanabe. On symmetry of information and polynomial time invertibility. *Inf. Comput.*, 121(1):14–22, 1995.

28 Zhenjian Lu and Igor Carboni Oliveira. An efficient coding theorem via probabilistic representations and its applications. *Electron. Colloquium Comput. Complex.*, 28:41, 2021.

29 James Maynard. The twin prime conjecture. *Japanese Journal of Mathematics*, 14:175–206, 2019.

30 Igor C. Oliveira. Randomness and intractability in Kolmogorov complexity. In *International Colloquium on Automata, Languages, and Programming* (ICALP), pages 32:1–32:14, 2019.

31 Igor C. Oliveira, Ján Pich, and Rahul Santhanam. Hardness magnification near state-of-the-art lower bounds. In *Computational Complexity Conference* (CCC), pages 27:1–27:29, 2019.

32 Igor C. Oliveira and Rahul Santhanam. Pseudodeterministic constructions in subexponential time. In *Symposium on Theory of Computing* (STOC), pages 665–677, 2017.

33 Alexander A. Razborov and Steven Rudich. Natural proofs. *J. Comput. Syst. Sci.*, 55(1):24–35, 1997.

34 Alexander Shen, Vladimir A. Uspensky, and Nikolay Vereshchagin. *Kolmogorov complexity and algorithmic randomness*. American Mathematical Society, 2017.

35 Michael Sipser. A complexity theoretic approach to randomness. In *Symposium on Theory of Computing* (STOC), pages 330–335, 1983.

36 Daniel A. Spielman. Linear-time encodable and decodable error-correcting codes. *IEEE Trans. Inf. Theory*, 42(6):1723–1731, 1996.

37 Terence Tao, Ernest Croot, III, and Harald Helfgott. Deterministic methods to find primes. *Math. Comp.*, 81(278):1233–1246, 2012.

38 Gérald Tenenbaum. *Introduction to analytic and probabilistic number theory*, volume 163. American Mathematical Society, 2015.

39 Luca Trevisan, Salil P. Vadhan, and David Zuckerman. Compression of samplable sources. *Comput. Complex.*, 14(3):186–227, 2005.

40 Gregory Valiant. Lecture Notes for CS265/CME309: Randomized Algorithms and Probabilistic Analysis (Lecture 6), 2019.

41 Thomas Watson. Time hierarchies for sampling distributions. *SIAM J. Comput.*, 43(5):1709–1727, 2014.

Degrees and Gaps: Tight Complexity Results of General Factor Problems Parameterized by Treewidth and Cutwidth

Dániel Marx
CISPA Helmholtz Center for Information Security,
Saarland Informatics Campus, Saarbrücken, Germany

Govind S. Sankar
Indian Institute of Technology Madras, Chennai, India

Philipp Schepper
CISPA Helmholtz Center for Information Security,
Saarland Informatics Campus, Saarbrücken, Germany
Saarbrücken Graduate School of Computer Science,
Saarland Informatics Campus, Germany

Abstract

In the General Factor problem, we are given an undirected graph G and for each vertex $v \in V(G)$ a finite set B_v of non-negative integers. The task is to decide if there is a subset $S \subseteq E(G)$ such that $\deg_S(v) \in B_v$ for all vertices v of G. Define the max-gap of a finite integer set B to be the largest $d \geq 0$ such that there is an $a \geq 0$ with $[a, a+d+1] \cap B = \{a, a+d+1\}$. Cornuéjols showed in 1988 that if the max-gap of all sets B_v is at most 1, then the decision version of General Factor is polynomial-time solvable. This result was extended 2018 by Dudycz and Paluch for the optimization (i.e. minimization and maximization) versions. We present a general algorithm counting the number of solutions of a certain size in time $(M+1)^{\mathrm{tw}} n^{\mathcal{O}(1)}$, given a tree decomposition of width tw, where M is the maximum integer over all B_v. By using convolution techniques from van Rooij (2020), we improve upon the previous $(M+1)^{3\mathrm{tw}} n^{\mathcal{O}(1)}$ time algorithm by Arulselvan et al. from 2018.

We prove that this algorithm is essentially optimal for all cases that are not trivial or polynomial time solvable for the decision, minimization or maximization versions. Our lower bounds show that such an improvement is not even possible for B-Factor, which is General Factor on graphs where all sets B_v agree with the fixed set B. We show that for every fixed B where the problem is NP-hard, our $(\max B + 1)^{\mathrm{tw}} n^{\mathcal{O}(1)}$ algorithm cannot be significantly improved: assuming the Strong Exponential Time Hypothesis (SETH), no algorithm can solve B-Factor in time $(\max B + 1 - \epsilon)^{\mathrm{tw}} n^{\mathcal{O}(1)}$ for any $\epsilon > 0$. We extend this bound to the counting version of B-Factor for arbitrary, non-trivial sets B, assuming #SETH.

We also investigate the parameterization of the problem by cutwidth. Unlike for treewidth, having a larger set B does not appear to make the problem harder: we give a $2^{\mathrm{cutw}} n^{\mathcal{O}(1)}$ algorithm for any B and provide a matching lower bound that this is optimal for the NP-hard cases.

2012 ACM Subject Classification Theory of computation → Parameterized complexity and exact algorithms

Keywords and phrases General Factor, General Matching, Treewidth, Cutwidth

Digital Object Identifier 10.4230/LIPIcs.ICALP.2021.95

Category Track A: Algorithms, Complexity and Games

Related Version *Full Version*: https://arxiv.org/abs/2105.08980

Funding Research supported by the European Research Council (ERC) consolidator grant No. 725978 SYSTEMATICGRAPH.

1 Introduction

Matching problems for graphs are widely studied in computer science [1, 7, 11, 18, 19, 24, 27, 31, 32, 34]. The most prominent ones are Perfect Matching (PerfMatch) and Maximum-Weight Matching. Both problems have long known polynomial-time algorithms [19, 32] and various generalizations were investigated in the graph-theory literature. These range from simple extensions such as the k-factor problem for a positive integer k (every vertex has to be incident to exactly k edges) [2, 27], to more complex ones, where the vertices are assigned intervals [34]. These problems are generally solved by a reduction to PerfMatch by replacing the vertices of the original instance with suitable gadgets. Lovász introduced a general version of these problems which we call General Factor [31]:

▶ **Definition 1.1** (General Factor (GenFac)). *Let $G = (V, E)$ be an undirected node labeled graph where the label of a vertex v is a set $B_v \subseteq \mathbb{N}$. We say $S \subseteq E$ is a* solution *if $\deg_S(v) \in B_v$ for all $v \in V$. GenFac is the problem of deciding whether G has a solution.*

The minimization and maximization versions of GenFac are the problems of finding the size of the solution with smallest and largest cardinality, respectively.

Polynomial-Time Solvable Cases. For several cases (e.g. k-factor, sets are intervals) reductions to PerfMatch are known, leading directly to polynomial-time algorithms. Cornuéjols analyzed the complexity of the general problem to identify properties of the sets that make the problem easier to solve [7]. For this he introduced the *gap* of a set: A gap is a finite sequence of consecutive integers not contained in the set but whose boundaries are contained in the set (cf. Definition 2.2). For example, the set $\{1, 5, 6, 8\}$ has gaps of size 3 and 1. For a set S, max-gap S denotes the size of its largest gap. Cornuéjols showed that if the max-gaps of all sets are at most 1, then the problem is polynomial-time solvable. Later this result was extended to the maximization and minimization (optimization) versions of GenFac.

▶ **Theorem 1.2** ([7, 18]). *The decision, maximization, and minimization version of GenFac can be solved in polynomial time on arbitrary graphs if for all nodes v,* max-gap $B_v \leq 1$.

On the other side Cornuéjols proved GenFac to be NP-complete if there are nodes with a gap of size two, namely $\{1\}$ and $\{0, 3\}$, by a reduction from *exact* 3*-cover*. More generally, it can be deduced from the work of Feder [22] that GenFac becomes NP-complete whenever every set B_v is restricted to be the same fixed set B having gap size at least two.

Treewidth. This paper is part of a long sequence of works studying problems parameterized by treewidth and related metrics like cutwidth or cliquewidth. Treewidth received significant attention as many NP-hard problems like Colouring, Independent Set, or Dominating Set (see [4] for a survey) are polynomial-time solvable on bounded-treewidth graphs. Courcelle's Theorem [8, 9] shows that a large class of graph problems can be solved in linear time on graphs of bounded treewidth. Recent developments on the algorithmic side include various techniques such as Cut & Count [15, 33], rank-based dynamic programming [3, 14, 23] and fast subset convolution [37, 38]. On the negative side, there have been a large number of results showing lower bounds based on complexity assumptions such as the Exponential Time Hypothesis (ETH) and the Strong Exponential Time Hypothesis (SETH) [5, 12, 29, 30]. For many such problems, their optimal algorithms utilize some form of dynamic programming, where a "state" is stored for every node in the tree decomposition. The number of such states determines the running time of the algorithms, seemingly suggesting

that this number is a natural barrier to the running time of any algorithm. Typically, the conditional lower bounds confirm this intuition by showing that no algorithm can break this barrier.

New Faster Algorithms. One of the first algorithmic results for GenFac parameterized by treewidth was given by Arulselvan et al. [1]. They present an algorithm for a restricted version of the problem where the sets contain zero and an interval of integers. This algorithm can be easily extended to handle arbitrary instances while preserving the running time of $(M+1)^{3\mathrm{tw}}n^{\mathcal{O}(1)}$ where M is the maximum over all sets assigned to the vertices. Their algorithm is based on the standard dynamic programming approach when parameterizing by treewidth, i.e. it considers all possible states for each node of the tree decomposition. The number of states in the dynamic programming is about $(M+1)^{\mathrm{tw}+1}$: one needs to keep track of the degree of the partial solution at each of the at most $\mathrm{tw}+1$ vertices of a bag of the tree decomposition, and this degree can be between 0 and M. Therefore, a natural question is whether the algorithm can be improved to obtain an $(M+1)^{\mathrm{tw}}n^{\mathcal{O}(1)}$ running time, matching the number of states. Such improvements are known for other problems, for example for Dominating Set and #PerfMatch in [38]. We base our algorithm on the same dynamic programming idea, but instead of processing all combination of states at join nodes, we make use of the technique of van Rooij [37] to compute fast convolutions, avoiding this bottle-neck of the computation. The algorithm can be easily generalized to the optimization and counting versions as well; to unify the results, we present the algorithm in a way that counts all solutions of a certain given size.

▶ **Theorem 1.3.** *Given a GenFac instance G and a tree decomposition of width* tw*. Let $M = \max_{v\in V(G)} \max B_v$. Then for all s, we can count the solutions of size exactly s in time $(M+1)^{\mathrm{tw}}n^{\mathcal{O}(1)}$.*

As we shall see, this algorithm is essentially optimal for every fixed B where B-Factor is NP-hard. Note that in order to obtain this optimal running time, we have to use a well-known, but non-trivial technique; beyond that, our algorithm does not provide new insights into the problem. Due to space constraints, we omit the algorithm here and refer the reader instead to the full version.

Tight Lower Bounds for the Decision Version. To investigate how the properties of the sets B_v influence the complexity of the problem, we give conditional lower bounds based on SETH for the restrictive B-Factor problem, where all sets have to be the same fixed set B. By a careful design our lower bounds also hold for a parameterization by *pathwidth*. Note that if the set is not fixed, Arulselvan et al. showed that GenFac is W[1]-hard when parameterizing only by treewidth [1]. Thus, it is reasonable to focus only on the cases with fixed sets to prove tight lower bounds.

▶ **Theorem 1.4** (Lower Bound for Decision Version)**.** *Let $B \subseteq \mathbb{N}$ be a fixed, finite set with $0 \notin B$ and* max-gap $B > 1$*. If, given a path decomposition of width* pw*, B-Factor can be solved in time $(\max B + 1 - \epsilon)^{\mathrm{pw}}n^{\mathcal{O}(1)}$ for some $\epsilon > 0$ even on graphs with degree at most $2\max B$, then SETH is false.*

The same result immediately follows for treewidth as for all graphs the pathwidth forms an upper-bound for the treewidth [13]. Hence, our algorithm is optimal not only for GenFac but also for B-Factor parameterized by treewidth and does not allow major improvements.

Tight Lower Bounds for the Optimization Version. It suffices to consider the maximization version with max-gap $B > 1$ and $0 \in B$ for the optimization version. The other cases are either polynomial-time solvable (max-gap $B \leq 1$ or $0 \in B$ for Min-B-Factor) [18] or the hardness directly follows from the lower bound for the decision version. Observe that the assumption $0 \in B$ does not make the problem trivially solvable. For these cases, we give essentially the same lower bound as for the decision version. Again the bound rules out that we can improve the given algorithm substantially; the running time is essentially optimal.

▶ **Theorem 1.5** (Lower Bound for Maximization Version). *Let $B \subseteq \mathbb{N}$ be a fixed, finite set with* max-gap $B > 1$. *If, given a path decomposition of width* pw, Max-B-Factor *can be solved in time* $(\max B + 1 - \epsilon)^{\mathrm{pw}} n^{\mathcal{O}(1)}$ *for some $\epsilon > 0$ even on graphs with degree at most* $2 \max B$, *then SETH is false.*

Counting. It is well known that PerfMatch can be solved in polynomial time [19]. Surprisingly, Valiant showed in [35] that *counting* the number of perfect matchings of a graph is as hard as counting satisfying assignment of a boolean formula. This is curious as (presumably) no polynomial-time algorithm for the decision version of the latter problem exists. The observation then led to the definition of the complexity class #P containing the counting problems whose corresponding decision version lies in NP. Indeed, this feature that some structures are easy to find but hard to count appears in our work as well. Apart from #PerfMatch, which itself is #$\{1\}$-Factor, our results imply that #B-Factor is #P-hard for any finite, fixed B. This contrasts with the decision version, where the problem is easy when max-gap $B \leq 1$. Over and above showing #P-hardness, we show a tight lower bound for #B-Factor, assuming #SETH, the counting version of SETH. There have been several results [10, 11, 17] based on #SETH and #ETH. Some of our constructions were inspired by one such work by Curticapean and Marx [11], where they show a lower bound of $(2 - \epsilon)^{\mathrm{pw}} n^{\mathcal{O}(1)}$ for #PerfMatch on graphs assuming #SETH. We prove a wide generalization of this result by providing a tight lower bound for every #B-Factor problem. As for the optimization and decision version, our algorithm shows the tightness of this lower bound.

▶ **Theorem 1.6** (Lower Bound for Counting Version). *Let $B \subseteq \mathbb{N}$ be a nonempty, fixed, and finite set such that $B \neq \{0\}$. If, given a path decomposition of width* pw, #B-Factor *can be solved in time* $(\max B + 1 - \epsilon)^{\mathrm{pw}} n^{\mathcal{O}(1)}$ *for some $\epsilon > 0$ even on graphs with degree at most* $2 \max B + 6$, *then #SETH is false.*

We also investigate #Max-B-Factor, the problem of counting maximum-sized solutions. The following argument shows that #$\{\max B\}$-Factor can be reduced to #Max-B-Factor without increasing pathwidth, hence Theorem 1.6 gives a lower bound of $(\max B + 1 - \epsilon)^{\mathrm{pw}} n^{\mathcal{O}(1)}$. Consider an instance of #$\{\max B\}$-Factor on a graph G of pathwidth pw. In polynomial time, check if G has some $\{\max B\}$-Factor [7]. If it does not, then output 0. If it does, then solve #Max-B-Factor on G. As now every maximum-sized B-factor is actually a $\{\max B\}$-factor, this indeed solves the #$\{\max B\}$-Factor problem.

▶ **Corollary 1.7.** *Let $B \subseteq \mathbb{N}$ be a fixed, finite set such that $B \neq \{0\}$. If, given a path decomposition of width* pw, #Max-B-Factor *can be solved in time* $(\max B + 1 - \epsilon)^{\mathrm{pw}} n^{\mathcal{O}(1)}$ *for some $\epsilon > 0$ even on graphs with degree at most* $2 \max B + 6$, *then SETH is false.*

We leave open the question of a tight lower bound for the minimization version.

Parameterizing by Cutwidth. As previously mentioned, pathwidth and treewidth are not the only parameters used in parameterized complexity. Cutwidth, cliquewidth, genus, and crossing number are only a few more examples of a vast class of possible parameters. For cutwidth, we consider linear layouts of graphs with n vertices, which are just enumerations $v_1, \ldots, v_n$ of all graph vertices. Then the cut after vertex v_i consists of all edges in G with one end in $\{v_1, \ldots, v_i\}$ and the other end in $\{v_{i+1}, \ldots, v_n\}$. The *cutwidth* of a linear layout is the maximum over the size of the cut after every v_i. The cutwidth cutw of a graph is the minimum over the cutwidths of all possible linear layouts. As $\mathrm{tw} \leq \mathrm{pw} \leq \mathrm{cutw}$, it is not completely surprising that we get different upper bounds for cutwidth. But now a simple dynamic program suffices to prove the upper bound for this case. Further, the running time of the algorithm is independent from the maximum of the set B.

▶ **Theorem 1.8.** *Given a linear layout of a* GenFac *instance G with width* cutw*, for all s we can count the number of solutions of size exactly s in time $2^{\mathrm{cutw}} n^{\mathcal{O}(1)}$,.*

This again matches the number of states for each cut of the linear layout. Like before, we omit the algorithm here and refer the reader to the full version. Note that the running times appearing in Theorems 1.3 and 1.8 cannot be directly compared: the base is lower when parameterized by cutwidth, but cutwidth can be larger than treewidth.

By a modified high-level construction, we show matching lower bounds based on SETH for the decision and optimization versions, and, based on #SETH, for the counting version.

▶ **Theorem 1.9** (Lower Bounds for Cutwidth)**.** *Let $B \subseteq \mathbb{N}$ be a fixed, nonempty set of finite size. If, given a linear layout of width* cutw*, the following problems can be solved in time $(2-\epsilon)^{\mathrm{cutw}} n^{\mathcal{O}(1)}$ for any $\epsilon > 0$ even on graphs with degree at most $2 \max B + 6$, then SETH (resp. #SETH) fails: (1) B-Factor and Min-B-Factor if $0 \notin B$ and $\text{max-gap}\, B > 1$, (2) Max-B-Factor if $\text{max-gap}\, B > 1$, and (3) #B-Factor if $B \neq \{0\}$.*

2 Preliminaries

We introduce homogeneous graphs to formally define B-Factor.

▶ **Definition 2.1** (Homogeneous Graphs and B-Factor)**.** *Let $B \subseteq \mathbb{N}$ be some fixed, finite set. We say a node-labeled graph is* B-homogeneous *if for each node $v \in V$ it holds that $B_v = B$. Then B-Factor is the restriction of* GenFac *to B-homogeneous graphs.*

This definition directly transfers to the optimization and counting version. We now formally introduce the max-gap of integer sets along with some other properties.

▶ **Definition 2.2.** *Let $B \subseteq \mathbb{N}$ be finite. We define* $\text{max-gap}\, B$ *as the largest non-negative integer d such that there is an $a \in B$ with $[a, a+d+1] \cap B = \{a, a+d+1\}$.*

In this paper we regularly insert graphs into other graphs. To make this operation formal, we make use of *dangling edges*: these are edges that have only one endpoint. We denote a dangling edge with endpoint v by $(?, v)$. For the sake of completeness we now formally define this procedure of replacing the relations, i.e. the insertion of a graph into another graph.

▶ **Definition 2.3** (Insertion)**.** *Let $G = (V, E)$ be a graph and $v \in V$ be of degree k, with incident edges $e_1 = (v_1, v), \ldots, e_k = (v_k, v)$ that are ordered in some fixed way. Let $H = (W, F)$ be a graph with dangling edges $d_1 = (?, u_1), \ldots, d_k = (?, u_k)$ where the u_i are not necessarily pairwise distinct.* Inserting *H in G at v gives us a new graph $G' = (V', E')$ where:*

$$V' = (V \cup W) \setminus \{v\} \quad \text{and} \quad E' = (E \cup F) \setminus \{e_1, \ldots, e_k, d_1, \ldots, d_k\} \cup \{(v_1, u_1), \ldots, (v_k, u_k)\}$$

All lower bounds we prove in this paper are based on the Strong Exponential Time Hypothesis. But instead of using the original statement we use a formulation which is more useful to work with.

▶ **Conjecture 2.4** (Strong Exponential Time Hypothesis (SETH) [6, 25]). *For all $\delta > 0$, there is a $k \geq 3$ such that satisfiability of k-CNF formulas on n variables requires more than $(2-\delta)^n$ time.*

About Relations. A relation $R : \{0,1\}^k \to \{0,1\}$ can also be seen as a set $R' \subseteq \{0,1\}^k$ such that $x \in R'$ iff $R(x) = 1$. We can also identify R with a set $R'' \subseteq 2^{[k]}$, where each element of R'' contains the positions of the 1s of an accepted input. Precisely, $x' = \{i \mid x[i] = 1\} \in R''$ iff $R(x) = 1$. We switch between these definitions depending on the context. Recall, a relation is *symmetric* if its output only depends on the Hamming weight of its input.

To simplify notation, we introduce the following generic classes of symmetric relations.

▶ **Definition 2.5.** *For a vector $x \in \{0,1\}^k$, we define $\mathrm{hw}(x)$ as the number of 1s in x, i.e. the Hamming weight of x. We define the following for $S \subseteq \mathbb{N}$, and $i, j \in \mathbb{N}$:*

$$\mathtt{HW}^{(j)}_{\in S} \coloneqq \{x \in \{0,1\}^j \mid \mathrm{hw}(x) \in S\} \qquad \mathtt{EQ}_j \coloneqq \mathtt{HW}^{(j)}_{\in \{0,j\}} \qquad \mathtt{HW}^{(j)}_{=i} \coloneqq \mathtt{HW}^{(j)}_{\in \{i\}}$$

$\mathtt{EQ}_j$ is the equality *relation on j inputs. We use $\mathtt{HW}_{\in S}$ to denote $\mathtt{HW}^{(j)}_{\in S}$ when the arity j of the relation is implicit. We also use this as the set of the relations $\mathtt{HW}^{(j)}_{\in S}$ for all $j \in \mathbb{N}$. We transfer this abuse of notation to $\mathtt{HW}_{=i}$.*

Note that assigning the relations $\mathtt{HW}_{\in B}$ to a vertex corresponds to assigning the set B to the vertex. Which notation is used depends on the context we are in.

3 Lower Bound when Parameterizing by Pathwidth

We show the lower bounds in two steps. The first step is a reduction from CNF-SAT to the intermediate problem B-Factor with Relations. In the second step, we reduce to the actual version of B-Factor for which we want to show the lower bound. As the lower bounds are for pathwidth they immediately hold for treewidth as it can be upper-bounded by pathwidth for all graphs.

▶ **Definition 3.1** (B-Factor with Relations (B-Factor$^{\mathcal{R}}$)). *Let $B \subseteq \mathbb{N}$ be fixed of finite size. $G = (V_S \dot\cup V_C, E)$ is an instance of B-Factor with Relations if all nodes in V_S are labeled with set B and all nodes $v \in V_C$ are labeled with a relation R_v that is given as a truth table such that the following holds:*

1. *Let $I(v)$ be the set of edges incident to v in G. Then $R_v \subseteq 2^{I(v)}$.*
2. *There is an even $c_v > 0$ such that for all $x \in R_v$ we have $\mathrm{hw}(x) = c_v$.*

A set $\widehat{E} \subseteq E$ is a solution *for G if (1) for $v \in V_S$: $\deg_{\widehat{E}}(v) \in B$ and (2) for $v \in V_C$: $I(v) \cap \widehat{E} \in R_v$. B-Factor$^{\mathcal{R}}$ is the problem of deciding if such an instance has a solution. We call V_S the set of* simple *nodes and V_C the set of* complex *nodes.*

Using this intermediate problem, we can formally state the first part of the reduction. The lower bound needs a careful formulation: when we reduce B-Factor$^{\mathcal{R}}$ to B-Factor by inserting gadgets realizing the relations at the complex nodes, the size and pathwidth of the graph can increase significantly. Therefore, we state a stronger lower bound that can tolerate additional terms to take care of such increases. The key point is that this increase is mainly influenced by the total degree of the complex nodes in a bag of the path decomposition.

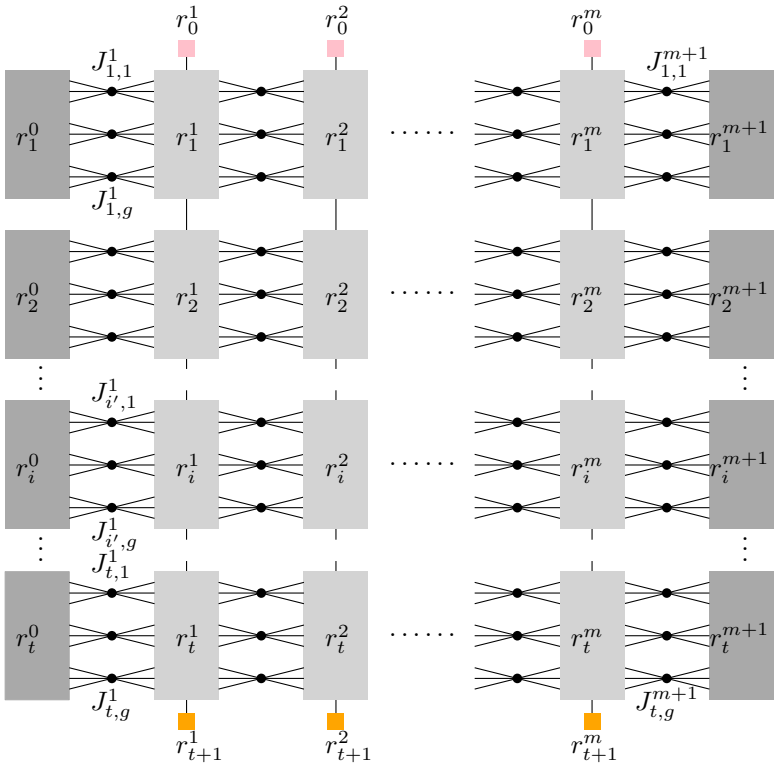


Figure 1 An example illustrating the construction from the proof of Theorem 3.2. The simple nodes are represented by circles while the complex nodes are represented by boxes.

▶ **Theorem 3.2.** *Let $B \subseteq \mathbb{N}$ be a fixed set of finite size with $B \neq \{0\}$. Given a B-Factor$^{\mathcal{R}}$ instance along with a path decomposition of width* pw *such that $\Delta^* = \max_{bag\ X} \sum_{v \in X \cap V_C} \deg(v)$. Assume B-Factor$^{\mathcal{R}}$ can be solved in $(\max B + 1 - \epsilon)^{\mathrm{pw}+f_B(\Delta^*)} n^{\mathcal{O}(1)}$ time on graphs with n vertices for some $\epsilon > 0$ and some function $f_B : \mathbb{N} \to \mathbb{R}^+$ that may depend on the set B. Then SETH fails.*

High Level Idea. We follow the ideas of previous lower bound reductions from [30] and combine them with the concept of using relations from [11]. From now on let $M := \max B$. Let ϕ be the given CNF formula with n variables and clauses $C_1, \ldots, C_m$. Instead of encoding each variable separately, we group q variables together and encode (partial) assignments to these groups. For each partial assignment, we define a vector in $[0, M]^g$, where g is chosen such that $2^q \leq (M+1)^g$. For each group we define a *layer* with g parallel *rows*, where each row corresponds to one dimension of the vector. The layers consist of an alternation of g parallel simple nodes and a complex node that is related to a clause. All simple nodes are connected to their neighboring complex nodes by M parallel edges. The vector from above then corresponds to the number of selected edges from a simple node to the following shared complex node. The complex nodes check whether the assignment represented by the selected edges of a layer satisfies the related clause. For each clause we connect the related complex nodes by a path. This path is used to propagate the information whether the clause is already satisfied by some partial assignment or whether it still needs to be satisfied. We ensure that each clause is initially not satisfied and eventually all clauses must be satisfied.

Constructing the B-FACTOR$^{\mathcal{R}}$ Instance. See Figure 1 for an example of the following construction. Split the variables of ϕ into $t \coloneqq \lceil n/q \rceil$ groups $F_1, \ldots, F_t$ of size at most q, where q is chosen later. For each of the t groups we encode the 2^q partial assignments by vectors from $[0, M]^g$ for some g chosen later. Instead of using all $(M+1)^g$ possible encodings we only use those vectors where the total weight of the coordinates is equal to $gM/2$ (we will choose g as a multiple of 4, hence $gM/2$ is an even number). It can easily be shown that there are more than $(M+1)^g/(gM+1)$ vectors with exactly this weight. Thus, after setting $q = \lfloor \log((M+1)^g) - \log(gM+1) \rfloor$, we can map each of the 2^q assignments of a group F_i to a distinct vector $[0, M]^g$ with weight exactly $gM/2$. We say that an partial assignment τ to a group F_i satisfies a clause C_j if at least one literal in the clause is satisfied under the assignment τ. Note, that a group F_i does not have to cover all variables of C_j to satisfy the clause.

We define the graph now as follows:

1. For all $i \in [t]$, $\ell \in [g]$, and $j \in [m]$: create a simple node $J^j_{i,\ell}$.
2. For all $i \in [t]$ and $j \in [m]$: create complex nodes r^j_i with relation R^j_i to be defined later.
3. For all $i \in [t]$: create complex nodes r^0_i and r^{m+1}_i with relation R^0.
4. For all $j \in [m]$: create complex nodes r^j_0 (resp. r^j_{t+1}) with relation $\texttt{HW}_{=0}$ (resp. $\texttt{HW}_{=1}$).
5. For all $i \in [t]$, $\ell \in [g]$, and $j \in [m]$: make $J^j_{i,\ell}$ adjacent to r^j_{i-1} and r^j_i by M parallel edges each. We call these edges backwards and forwards edges, respectively.
6. For all $i \in [t]$ and $j \in [m]$, make r^j_i additionally adjacent to r^j_{i-1} and r^j_{i+1} by one edge each. The degree of the nodes is now $2gM + 2$.

We call the set of nodes $\{r^j_i, J^j_{i,\ell}\}_{j,\ell}$ the ith *layer*. The set $\{J^j_{i,\ell}\}_j$ forms the ℓth *row* of the ith layer. For a fixed $j \in [m]$, the set of nodes $\{r^j_i\}_i$ is called the jth *column*.

The idea is now the following: For each partial assignment τ to a group F_i, we define a vector $v_\tau \in [0, M]^g$ of weight $gM/2$ as its *encoding*.[1] Then $v_\tau[\ell]$ corresponds to the number of selected forward edges of the simple nodes in the ℓth row of the ith layer. The vertical edges encode whether a clause was already satisfied. That is, if the edge between r^j_i and r^j_{i+1} is selected, then there is some group F_k with $k \leq i$ where the corresponding assignment satisfies the clause C_j. By the relation of the nodes r^j_0 every clause is initially not satisfied. But the relation of the r^j_{t+1} nodes ensures that every clause is eventually satisfied.

Defining the Relations. R^0 accepts exactly those inputs of Hamming weight exactly $gM/2$, an even number by assumption, where the selected edges for each row must precede the unselected edges, i.e. the first k edges are selected, the next $M - k$ are not selected.

The relation $R^j_i \subseteq \{0,1\}^{2Mg+2}$ of node r^j_i is defined as follows:

- For $\ell \in [g]$, let x_ℓ (resp. y_ℓ) be the number of selected incident edges to $J^j_{i,\ell}$ (resp. $J^{j+1}_{i,\ell}$).
- $\sum_{\ell \in [g]} x_\ell = gM/2 = \sum_{\ell \in [g]} y_\ell$.
- $\langle x_1, \ldots, x_g \rangle$ describes a valid encoding, i.e. it corresponds to a partial assignment for F_i.
- $x_\ell + y_\ell = M$. Further, the x_ℓ (resp. y_ℓ) selected edges precede the $M - x_\ell$ (resp. $M - y_\ell$) unselected edges of the M parallel edges going to a simple node.
- If the ingoing top edge is selected, then the outgoing bottom edge is also selected.
- If the ingoing top edge is not selected:
  - If C_j does not contain a variable of F_i, then the outgoing bottom edge is not selected.
  - If C_j contains at least one variable of F_i, then the outgoing bottom edge is selected if and only if the selected edges correspond to a valid partial assignment satisfying C_j.

[1] Note that for different groups the encoding of the same partial assignment do not need to be the same.

Final Modifications. Due to parity issues, we can only realize relations where the Hamming weight of the accepted inputs is an *even* constant for each relation. Thus, we slightly have to modify this construction. We leave the exact details to the full version.

▶ **Lemma 3.3.** *ϕ is satisfiable if and only if there is a solution to the B-Factor$^{\mathcal{R}}$ instance.*

To obtain a tight lower bound, we need to analyze the pathwidth of our construction and have to bound the degree of the complex nodes.

▶ **Lemma 3.4.** *The graph has $\mathcal{O}(tgm)$ simple and $\mathcal{O}(tm)$ complex nodes. The degree of the complex nodes is bounded by $2gM + 4$. The degree of the simple nodes is bounded by $2M$. We can efficiently construct a path decomposition of width $tg + \mathcal{O}(1)$ where at most three complex nodes are simultaneously in one bag.*

Now we have everything ready to prove the lower bound for the intermediate problem B-Factor$^{\mathcal{R}}$ based on the previous construction. Recall, we defined Δ^* as the maximum total degree of the complex nodes appearing in one bag, that is $\Delta^* = \max_{\text{bag } X} \sum_{v \in X \cap V_C} \deg(v)$.

Proof of Theorem 3.2 (Sketch). As $(M+1)^g \approx 2^q$ and $t \approx n/q$, we intuitively get

$$(M+1-\epsilon)^{tg} n^{\mathcal{O}(1)} = ((M+1-\epsilon)^g)^{\frac{n}{q}} n^{\mathcal{O}(1)} \ll ((M+1)^g)^{\frac{n}{q}} n^{\mathcal{O}(1)} = (2^q)^{\frac{n}{q}} n^{\mathcal{O}(1)} = 2^n n^{\mathcal{O}(1)},$$

showing that the reduction and the assumed algorithm would solve the SAT instance too fast. Note that due to rounding and other issues the calculation has to be done way more carefully and is thus deferred to the full version. ◀

4 Decision Version

In this section we prove the lower bound for the decision version of B-Factor by a reduction from the intermediate B-Factor$^{\mathcal{R}}$ problem. For this we formally define the concept of realizations and show that we can realize all relations of a B-Factor$^{\mathcal{R}}$ instance. Replacing the nodes and their relations by these realizations yields the final lower bound.

▶ **Definition 4.1** (Realization). *Let $R \subseteq \{0,1\}^k$ be a relation. Let G be a node-labeled graph with dangling edges $D = \{d_1, \ldots, d_k\} \subseteq E(G)$. We say that graph G* realizes *R if for all $D' \subseteq D$: $D' \in R$ if and only if there is a solution $S \subseteq E(G)$ with $S \cap D = D'$. We say that G B-*realizes *R if G is B-homogeneous. The endpoints of the dangling edges are called* portals.

The crucial part of the reduction is the proof of the following theorem. We postpone its proof and first show the lower bound.

▶ **Theorem 4.2.** *Let $B \subseteq \mathbb{N}$ be a fixed set of finite size with* max-gap $B > 1$ *and $0 \notin B$. There is a $f : \mathbb{N} \to \mathbb{N}$ such that the following holds. Let $R \subseteq \{0,1\}^e$ be an even relation (i.e.* $\mathrm{hw}(x)$ *is even for all $x \in R$). Then we can B-realize R by a simple graph with $f(e)$ vertices of degree at most* $\max B + 2$*, the portal nodes are pairwise distinct.*

Now we can prove the lower bound under SETH. We assume that $B \subseteq \mathbb{N}$ is a fixed, finite set such that $0 \notin B$ and max-gap $B > 1$.

Proof of Theorem 1.4 (Sketch). Replace every complex node v and its relation R_v in the B-Factor$^{\mathcal{R}}$ instance H by its B-realization of size at most $f(\deg(v))$ according to Theorem 4.2. This increases the size of the graph at most by a factor of $f(\Delta^*)$. As we can bound the pathwidth of the inserted graphs by their size, we modify each bag of the path decomposition

of H by replacing all complex nodes with the nodes of their realization. Thus, the pathwidth of the new graph is bounded by $\mathrm{pw}_H + \Delta^* f(\Delta^*)$. Assuming the faster algorithm, this already contradicts SETH by Theorem 3.2. ◀

From now on let $B \subseteq \mathbb{N}$ be our fixed, finite set with $\min B \geq 1$ and max-gap $B = d > 1$ such that $[a, a+d+1] \cap B = \{a, a+d+1\}$ for some $a \geq 1$. We first realize three quite basic relations which we use later to realize the more complex relations.

▶ **Lemma 4.3.** *We can B-realize each of the relations $\mathtt{HW}^{(2)}_{=2}$, $\mathtt{EQ}_{d+1}$, and $\mathtt{EQ}_2$ by a simple graph with $\mathcal{O}(\mathrm{poly}(\max B))$ vertices of degree at most $\max B$.*

Proof.

1. Define a $\min B + 1$-clique with new vertices. Split an arbitrary edge (u, v) into two dangling edges $(?, u)$ and $(?, v)$. The construction of the clique and the fact that we chose $\min B$ as degree forces the two dangling edges to be selected in any solution.
2. We start with two new vertices u, v and connect each to a many common $\mathtt{HW}^{(2)}_{=2}$ nodes. We add $d+1$ dangling edges to u and zero to v. Finally the nodes are replaced by their realization. Observe that u has a forced edges and $d+1$ dangling edges. Thus we must select none or all of the dangling edges since $[a, a+d+1] \cap B = \{a, a+d+1\}$.
3. Define a $d+2$-clique with $\mathtt{EQ}_{d+1}$ nodes. Split an arbitrary edge (u, v) into two dangling edges $(?, u)$ and $(?, v)$. Replace the nodes by their realization.
   Either both dangling edges are selected in which case all nodes have $d+1$ incident edges in the solution, or neither is selected in which case every node has zero incident edges in the solution. ◀

The following lemma helps us to keep the later constructions simple. Instead of constructing the relations for arbitrary degree, only the very low degree cases are necessary.

▶ **Lemma 4.4.** *If we can realize $\mathtt{HW}^{(a)}_{=1}$ for $a \in \{1, 2, 3\}$ by a simple graph with at most N vertices of degree at most D, then we can realize $\mathtt{HW}^{(k)}_{=1}$ for all $k \geq 1$ by a simple graph using $\mathcal{O}(kN)$ nodes of degree at most D.*

Proof. We construct the graph for the realization inductively starting with the basis for $k = 1, 2, 3$. See Figure 2 for an example.

For the inductive step from k to $k+1$ we start with a node u with relation $\mathtt{HW}^{(k)}_{=1}$. Connect one dangling edge of u to a new node v with $\mathtt{HW}^{(2)}_{=1}$. Connect the other dangling edge of v to a node w with relation $\mathtt{HW}^{(3)}_{=1}$. Observe that the final graph has $k+1$ dangling edges.

Assume one dangling edge of u is selected, then the edge between u and v is not selected but the edge from v to w is. Hence, no dangling edge of w can be selected. The analogue holds if one of the dangling edges of w is selected. It cannot be the case that more than one or zero dangling edges are selected, as then the relation of one of the three nodes u, v, or w would not be satisfied. ◀

Due to parity issues, the construction of the realizations depends on the set B. Each of the possible cases (B contains only even numbers, only odd number, or even and odd numbers) is treated separately. We focus on the all even case and refer the reader to the full version for the other cases.

▶ **Lemma 4.5.** *If B contains only even numbers, we can B-realize the following relations by simple graphs:*

1. $\mathtt{EQ}_k$ *for even $k \geq 2$ using $\mathcal{O}(k \,\mathrm{poly}(\max B))$ vertices of degree at most $\max B$.*
2. $\mathtt{HW}^{(k)}_{=1}$ *together with $\mathtt{HW}^{(\ell)}_{=1}$ for all $k, \ell \geq 1$ using $\mathcal{O}((k+\ell)\,\mathrm{poly}(\max B))$ vertices of degree at most $\max B + 2$.*

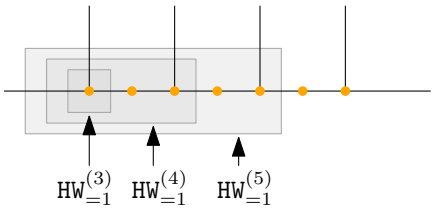


Figure 2 Example of the inductive construction from Lemma 4.4 for $\mathtt{HW}_{=1}^{(6)}$ using $\mathtt{HW}_{=1}^{(3)}$ and $\mathtt{HW}_{=1}^{(2)}$.

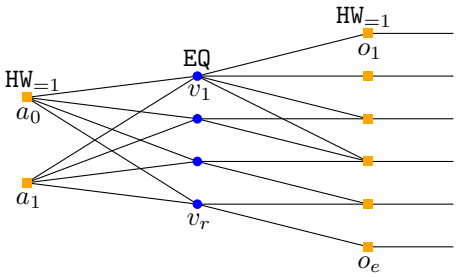


Figure 3 An example illustrating the construction from the proof of Theorem 4.2 for the relation R with $R(000011) = R(110011) = R(111001) = R(111100) = 1$ and zero otherwise.

Proof.

1. For $k = 2$ we can use the construction of Lemma 4.3. For the other case we first realize $\mathtt{EQ}_4$. Then we use a chain of these relations to realize $\mathtt{EQ}_k$ for even $k \geq 6$.
   Start with a $\mathtt{EQ}_{d+1}$ node u and make it adjacent to $\frac{d+1-4}{2}$ many $\mathtt{EQ}_2$ nodes (note that an even B can have only gaps of odd size, hence d is odd). Then we add four dangling edges to u. hence the construction actually works. The graph is simple as the dangling edges in the realization of $\mathtt{EQ}_2$ are different.
2. To use Lemma 4.4 for the general construction, observe that the number of $\mathtt{HW}_{=1}$ nodes used in the construction is odd. Hence, we will always realize two nodes. For this we show how to realize $\mathtt{HW}_{=1}^{(k)}$ together with $\mathtt{HW}_{=1}^{(\ell)}$ for all $k, \ell \in \{1, 2, 3\}$.
   Start with two vertices u, v. Make u and v adjacent to $\max B - 1$ common $\mathtt{HW}_{=2}^{(2)}$ nodes. We add k dangling edges to u and ℓ dangling edges to v. As B does not contain $\max B - 1$, the correctness follows. ◀

Now we have everything ready to prove that even relations can be realized.

Proof of Theorem 4.2. See Figure 3 for an example of the following construction. We use essentially the construction from Lemma 3.3 in [11]. Let $R = \{x_1, \ldots, x_r\} \subseteq \{0,1\}^e$ be the even relation for some r. Let further $P = \{(1 + e \mod 2), 0\}$.

1. Create nodes $o_1, \ldots, o_e$ with relation $\mathtt{HW}_{=1}$.
2. Create vertices a_j for all $j \in P$ with relation $\mathtt{HW}_{=1}$.
3. For all $i \in [r]$:
   a. Let $O_i = \{n_1^{(i)}, \ldots, n_{h_i}^{(i)}\} = \{k \in [e] \mid x_i[k] = 0\}$ for $h_i = e - \mathrm{hw}(x_i)$.
   b. Create the node v_i with relation $\mathtt{EQ}$ and connect it to $o_{n_j^{(i)}}$ for all $j \in [h_i]$.
   c. Connect v_i to all a_j.
4. Replace all nodes by their realization.

There are $|P| + e$ many $\mathtt{HW}_{=1}$ nodes. Since $|P| = 1 + (1 + e \mod 2)$, we can replace pairs of these nodes by their realization. Every v_i is connected to $|P| + |O_i|$ nodes, where $|O_i| = e - \mathrm{hw}(x_i)$. Thus, v_i has even degree as the relation R is even, i.e. $\mathrm{hw}(x)$ is even. Hence, we can replace these nodes by their realization according to the previous lemmas.

To show that the construction actually realizes the relation, assume the selected dangling edges corresponds to some element $x \in R$, let it w.l.o.g. be x_1. Then we can select all edges incident to x_1, the dangling edges, and the extension of this to all nodes as a solution. As x_1 is adjacent to all a_j they are in a valid state. Further x_1 is adjacent to those o_k where $x_1[k] = 0$ and hence every o_k is incident to exactly one edge in the solution.

Now assume we are given a solution. As the nodes a_j have exactly one incident edge in the solution, there is exactly one node v_i where all incident edges are in the solution. Let O be the set of nodes o_k to which v_i is adjacent. By construction v_i corresponds to some $x \in R$ with $x[k] = 0$ iff $k \in O$. As all selected dangling edges must be in the solution, let O' be the set of nodes incident to the selected dangling edges. But as we are given a solution we get $O \dot\cup O' = \{o_1, \dots, o_e\}$. Hence, the dangling edges correspond to x. ◀

5 Optimization Version

In the previous section we have seen the realization of the relations for the decision version. As we are interested in the largest solution for Max-B-Factor, we also allow $0 \in B$ since this does not make the problem trivially solvable. This makes it necessary to change the definition of a realization, as the pure existence of a solution is not sufficient anymore. We change it such that if the relation is satisfied (i.e. the dangling edges are selected in a good way), then there is a *large* solution. Otherwise, there must be a gap by which any solution is *smaller* compared to the solutions in the good cases. We call this gap the penalty (of the realization).

▶ **Definition 5.1** (Realization). *Let $R \subseteq \{0,1\}^k$ be a relation. Let G be a node labeled graph, with dangling edges $D = \{d_1, \dots, d_k\}$. We say that graph G* realizes *R with* penalty *β if we can efficiently construct/find a target value $\alpha > 0$ such that for all $D' \subseteq D$:*

- *If $D' \in R$, then there is a solution $S \subseteq E(G)$ with $S \cap D = D'$ and $|S| = \alpha$.*
- *If $D' \notin R$, then for all solutions $S \subseteq E(G)$ with $S \cap D = D'$ we have $|S| \leq \alpha - \beta$.*

We say that G B-realizes R if G is additionally B-homogeneous. We call the endpoints of the dangling edges portal nodes.

In the main part of this section we show how to realize the relations of B-Factor$^{\mathcal{R}}$. The following theorem corresponds to Theorem 4.2 for the decision version.

▶ **Theorem 5.2** (Realization of Relations). *Let $B \subseteq \mathbb{N}$ be a fixed, finite set with* max-gap $B > 1$ *and $0 \in B$. There is a $f : \mathbb{N}^2 \to \mathbb{N}$ such that the following holds. Let $R \subseteq \{0,1\}^e$ be a relation with a constant $c_R \in 2\mathbb{N}$ such that for all $x \in R$ we have* $\mathrm{hw}(x) = c_R$.

We can B-realize the relation R with arbitrary penalty $\beta > 0$ by a simple graph with $f(e, \beta)$ vertices of degree at most $\max B + 2$.

It remains to compute the target value by which we decide if the B-Factor$^{\mathcal{R}}$ instance has a solution or not.

▶ **Lemma 5.3.** *Let G be a B-Factor$^{\mathcal{R}}$ instance from Section 3. Let G' be a B-Factor instance resulting from G by replacing every complex nodes with degree δ by its realization with penalty 2δ. Then, there is an efficiently computable constant α such that G has a solution if and only if the largest solution for G' has size α.*

Proof (Sketch). The target value α is essentially the sum of the target values α_v for the realizations of the complex nodes $v \in V_C$. But we have to take care that the edges between complex nodes are not counted twice. ◀

Now we are ready to prove the conditional lower bound for Max-B-Factor when $0 \in B$.

Proof of Theorem 1.5. Use Lemma 5.3 to construct the final graph and the target value. Then the proof goes analogous to the proof for the decision version (cf. Theorem 1.4). ◀

High Girth Graphs. We know that there is a gap of size at least two between a and $a+d+1$ in B. This allows us to define relatively simple conditions of the form "if one incident edge of a vertex with degree $a+d+1$ is not selected, then another edge is also not selected". In other words, this propagates the penalty to a neighboring vertex. We combine this with high girth graphs to introduce an arbitrary large penalty for not selecting an edge.

The construction of r regular graphs with girth g is a long studied problem in graph theory. Erdős and Sachs proved the existence of such graphs for all combinations of r and g.

▶ **Lemma 5.4** (Theorem 1 in [20]). *For all $r \geq 2$ and $g \geq 3$, there is a r-regular graph $G_{r,g}$ of girth g with at most $4gr^g$ vertices.*

Finding the smallest graph for each r, g is a non-trivial task and known as the (r, g)-cage problem. For several cases (e.g. r is a prime power) constructions are known reducing the number of vertices in the graph. See [16, 21, 26, 28] for more results.

Realizing Relations. From now on let $d \coloneqq \text{max-gap}\, B > 1$ such that $[a, a+d+1] \cap B = \{a, a+d+1\}$ for some $a \geq 0$. As we allow $0 \in B$, we can always find a trivial solution. Thus, we cannot force edges as we did for the decision version. Instead we construct a gadget where we can select many edges when the "forced" edges are selected. Otherwise we ensure that the solution is small. We use the graphs with high girth for this.

▶ **Lemma 5.5.** *There is a $f : \mathbb{N} \to \mathbb{N}$ such that the following holds. We can B-realize $\mathtt{HW}^{(2)}_{=2}$ (with distinct portal vertices) with arbitrary penalty β by simple graphs using at most $f(\beta)$ vertices of degree at most* $\max B$.

Proof. We use Lemma 5.4 to get an $a+d+1$-regular graph $G_{a+d+1,\beta}$ of girth at least β. Split an arbitrary edge (u, v) into two dangling edges for u and v each and assign the set B to every vertex.

The graph has the claimed properties: If both dangling edges are selected, then we can use the set of all edges in the graph as a solution since $a+d+1 \in B$.

It remains to check the case when at least one dangling edge is not selected, let it w.l.o.g. be the one incident to u. Assume S is the optimal solution. We show that this solution does not contain more than $|E_{a+d+1,\beta}| - \beta$ edges.

By assumption $\deg_S(u) \leq a$. Hence, there must be at least one other incident edge to u that is not in the solution, because $a+d-1 \geq a+1 \notin B$. Then we can apply this argument always to the next vertex. Observe that this sequence can only stop if we reach another vertex w we have already visited because for this vertex we already know that two incident edges were not selected in the solution. The length of this path, i.e. the number of not selected edges from w to w, is at least the girth of the graph. Hence the number of edges that are not selected in the solution is at least the girth of the graph which is at least β. ◀

The remaining part follows mainly the constructions from the decision version. However, as we care about the size of the solution a more careful construction and analysis is needed. The detailed construction of the realization is given in the full paper.

6 Counting Version

From a certain perspective the optimization version can be seen as a relaxation of the decision version: The assumption $\min B > 0$ is dropped while still assuming max-gap $B > 1$. For the counting version we now even drop this last assumption such that there might be no gap at all in B. Thus the only polynomial-time solvable cases for the counting version are $B = \{0\}$ and $B = \emptyset$ with one and zero solutions, respectively. This implies that we additionally must realize equality relations. Surprisingly this also reduces to realizing $\mathtt{HW}^{(1)}_{=1}$ nodes in the end, i.e. forcing edges.

We use the Holant framework and lemmas and definitions analogous to those from [11]. A signature graph Ω is a graph with weights w_e for all edges e and all vertices are labeled by *signatures* $f_v : \{0,1\}^{I(v)} \to \mathbb{Q}$, which are rational functions on the incidence vector $I(v)$ of the edges incident to v. We define Holant(Ω) to be the quantity

$$\sum_{x \in \{0,1\}^{E(\Omega)}} \prod_{e \in x} w_e \prod_{v \in V(\Omega)} f_v(x|_{I(v)}).$$

The Holant framework can be seen as a natural generalization of GenFac. If each signature f_v is a symmetric Boolean function and each edge weight is 1, then it is exactly #GenFac. If additionally each vertex has signature $\mathtt{HW}_{\in B}$, this corresponds to #B-Factor.

▶ **Definition 6.1** (Holant(F)). *If F is a set of rational functions, we say that* Holant(F) *is the set of all Holant problems where the signature graph has signatures only from F.*

▶ **Definition 6.2** (Gate). *A gate is a signature graph Γ, possibly containing a set $D \subseteq E(\Gamma)$ of dangling edges, all of which have edge weight 1. The signature realized by Γ is the function* $\mathtt{SIG}(\Gamma) : \{0,1\}^D \to \mathbb{Q}$ *that maps an assignment of dangling edges $x \in \{0,1\}^D$ to*

$$\mathtt{SIG}(\Gamma, x) = \sum_{y \in \{0,1\}^{E(\Gamma) \setminus D}} \left(\prod_{e \in E(\Gamma)} w(e) \prod_{v \in V(\Gamma)} f_v\Big((x \cup y)|_{I(v)}\Big) \right)$$

Note that unless mentioned otherwise, we restrict ourselves to signature graphs with unit edge weights and hence they are usually omitted.

In essence, gates in the Holant framework play the role of realizations in the previous sections. Given these definitions, we are now ready to state our main theorem, which can then be used to prove Theorem 1.6. Observe for this that the reduction in Theorem 3.2 is parsimonious.

▶ **Theorem 6.3.** *For all fixed, finite $B \subseteq \mathbb{N}$ with $B \neq \{0\}$ there is a $f : \mathbb{N} \to \mathbb{N}$ such that the following holds. Let $G = (V_S \dot\cup V_C, E)$ be an instance of #B-Factor$^{\mathcal{R}}$ with a path decomposition of width* pw *such that $\Delta^* = \max_{bag\ X} \sum_{v \in X \cap V_C} \deg(v)$. Then there is a $f(\Delta^*)n^{\mathcal{O}(1)}$ time Turing reduction from #B-Factor$^{\mathcal{R}}$ to #B-Factor such that for every constructed instance of #B-Factor pathwidth and cutwidth increase at most by $f(\Delta^*)$.*

We can think of #B-Factor$^{\mathcal{R}}$ as a Holant problem where the allowed signatures are either $\mathtt{HW}_{\in B}$ or restricted even relations. We first use a lemma from [11] to realize these relations through nodes with signature $\mathtt{HW}_{=1}$. Since their constructions are in the perfect matching setting, they can equivalently be seen as gates that use vertices with signature $\mathtt{HW}_{=1}$. After using this lemma to reduce from #B-Factor$^{\mathcal{R}}$ to a Holant problem, we give a chain of reductions (see Figure 4) that ends at #B-Factor and preserves the pathwidth up to an additive constant.

▶ **Lemma 6.4** (Informal, Lemma 3.3 from [11]). *Every even relation can be realized through a graph whose vertices have signature $\mathtt{HW}_{=1}$ and whose edges have weights in $\{-1, \frac{1}{2}, 1\}$.*

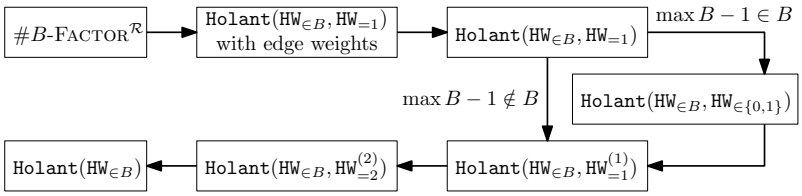


Figure 4 The chain of reductions that starts with B-Factor$^{\mathcal{R}}$ and ends at $\#B$-Factor (i.e. $\texttt{Holant}(\texttt{HW}_{\in B})$). Arrows show the direction of Turing or many-one reductions.

Main Ideas. The next step is to remove the edge weights. We do this through polynomial interpolation, which was first used by Valiant [36]. The idea is that we can recover a polynomial $P(\cdot)$ if we know the value of $P(x)$ for sufficiently many x. We represent the solution of one problem as the value of a polynomial $P(\cdot)$ and the second problem as a function $f(P)$ of the polynomial itself. Then, we recover the value of the second problem by using an oracle of the first problem, giving a Turing reduction from the second problem to the first.

For the removal of the edge weight it suffices by the polynomial interpolation to consider edge weights that are a power of two. Assume for simplicity, we just have edge weight 2. Replace such edges by two parallel edges of unit weight. This leaves the output unchanged, as we duplicated the number of solutions, which compensates for the unit edge weight.

The main difficulty is to realize $\texttt{HW}_{=1}$ nodes using $\texttt{HW}_{\in B}$ nodes. To replace the $\texttt{HW}_{=1}$ nodes, we distinguish between the case where $\max B - 1$ is in B or not. In the latter case, the construction from the decision version works. But in the former case we use an argument similar to the procedure for the final step. The last step replaces $\texttt{HW}^{(2)}_{=2}$ nodes by $\texttt{HW}_{\in B}$ nodes. For this we "separate" the case where the forced edges are selected and where they are not. We define a pathlike gadget with many solutions if the dangling edges are not selected and significantly fewer otherwise. Each vertex on this long path is connected to many fresh vertices. We choose their number to be higher than the maximum element of B. Then, if an incident edge of the path is already selected, there are fewer solutions as if the edge is not selected. Combining this with the interpolation we arrive at a point, where all nodes have relation $\texttt{HW}_{\in B}$.

We now describe one case in the final step in the chain of reductions, where we realize $\texttt{HW}^{(2)}_{=2}$ nodes by $\texttt{HW}_{\in B}$ nodes. For the remaining cases and steps, we refer the reader to the full version of this paper.

For the interpolation we make use of the following result which is proven in the full version.

▶ **Proposition 6.5.** *Suppose we have two non-zero sequences* $\{A_n\}_{n\in\mathbb{N}}, \{B_n\}_{n\in\mathbb{N}}$ *that are related as*

$$\begin{bmatrix} A_n \\ B_n \end{bmatrix} = M \begin{bmatrix} A_{n-1} \\ B_{n-1} \end{bmatrix} = M^n U \qquad \text{, where } U = \begin{bmatrix} A_0 \\ B_0 \end{bmatrix}$$

and M *is a symmetric and invertible* 2×2 *matrix such that* U *is not an eigenvector of* M*. Then* $\{\frac{B_n}{A_n}\}_{n\in\mathbb{N}}$ *is a sequence which does not contain any repetitions.*

▶ **Lemma 6.6.** *Let* $B \subseteq \mathbb{N}$ *be a fixed finite set. There is a polynomial-time Turing reduction from* $\texttt{Holant}(\texttt{HW}_{\in B}, \texttt{HW}^{(2)}_{=2})$ *to* $\texttt{Holant}(\texttt{HW}_{\in B})$ *increasing pathwidth and cutwidth only by a fixed constant and leaving the max degree unaffected, or increasing it to* $2 \max B + 6$*.*

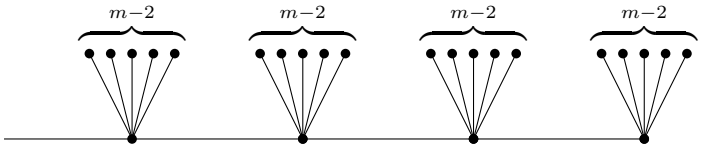


Figure 5 The gadget for case 1: Black nodes are $\mathtt{HW}_{\in B}$ nodes.

Proof. If $0 \notin B$, we can use the construction from Lemma 4.3 to get a $\mathtt{HW}^{(2)}_{=2}$ node. For the case when $0 \in B$, we do a case-by-case analysis depending on B. In either case, we attach a subgraph with a constant pathwidth and cutwidth to vertices. This does not affect either of them by more than $2 \max B + 6$, a fixed constant.

Case 1: B contains 1. Define $m \geq 2$ to be the smallest integer not in B. Consider the gadget in Figure 5. Suppose there are d such vertices with $m-2$ pendant nodes each. Let all of them have the relation $\mathtt{HW}_{\in B}$. Let $P_1(d)$ be the number of solutions of the gadget where the dangling edge is selected in the solution. Similarly define $P_0(d)$ when the dangling edge is not selected. We claim that the gadget described can be effectively used to force edges, i.e. a $\mathtt{HW}^{(1)}_{=1}$ node. Two such gadgets will give us a $\mathtt{HW}^{(2)}_{=2}$ node. Suppose any graph G contains t such gadgets. We have

$$\text{Holant}(G) = \sum_{i=0}^{t} A_i (P_0(d))^{t-i} (P_1(d))^i = (P_0(d))^t \sum_{i=0}^{t} A_i \left(\frac{P_1(d)}{P_0(d)}\right)^i$$

where A_i is the number of ways of extending the solution in G when i of the gadgets choose to match their dangling edge. Through standard interpolation techniques, we can recover the A_is, and thus A_t will give us the solution where each gadget behaves like a $\mathtt{HW}^{(1)}_{=1}$. Now, we can replace $\mathtt{HW}^{(2)}_{=2}$ nodes in the $\mathtt{Holant}(\mathtt{HW}_{\in B}, \mathtt{HW}^{(2)}_{=2})$ instance with pairs of $\mathtt{HW}^{(1)}_{=1}$ nodes.

To argue that we can do the interpolation, we need to show that $\frac{P_1(d)}{P_0(d)}$ will take at least t unique values, and that these are computable in polynomial time. Since we can define such a gadget for any integer d we have

$$P_0(d) = kP_0(d-1) + kP_1(d-1) \qquad \text{and} \qquad P_1(d) = kP_0(d-1) + (k-1)P_1(d-1)$$

for $k = 2^{m-2}$. We now apply Proposition 6.5 with $M = \left[\begin{smallmatrix} k & k \\ k & k-1 \end{smallmatrix}\right]$ and $U = \left[\begin{smallmatrix} k \\ k \end{smallmatrix}\right]$.

This completes the realization for the case when B contains 1. The remaining cases when B does not contain 1 but some odd number and when B only consists of even numbers can be found in the full version. ◀

Proof of Theorem 6.3 (Sketch). Given any instance of $\#B$-Factor$^{\mathcal{R}}$, we can sequentially apply the reductions from Figure 4 to get a polynomial number of instances of $\#B$-Factor such that the pathwidth is affected only by some function of Δ^*. ◀

7 Lower Bound when Parameterizing by Cutwidth

The algorithmic result from Theorem 1.8 shows that the pathwidth lower bound breaks when parameterizing by cutwidth. Nevertheless, we can show that this "improved" running time is the best we can hope for assuming SETH and #SETH. For this we use the same high level ideas Curticapean and Marx presented in Figure 6 of [11] where they reduce from #SAT to computing the Holant and then reduce to counting perfect matchings. But the construction

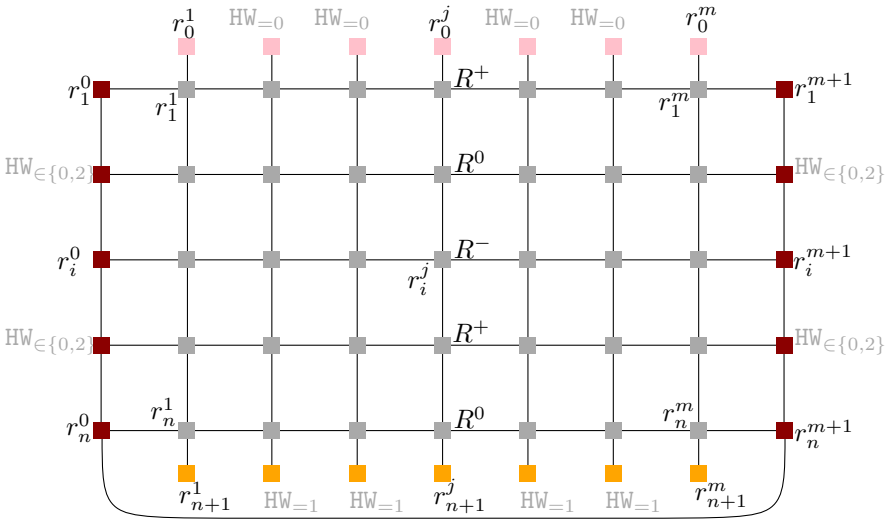


Figure 6 The example graph for a formula containing the clause $(x_1 \vee \bar{x}_3 \vee x_4)$.

can also be seen as a modification of our reduction for the pathwidth lower bound. We again first reduce to the intermediate problem B-Factor$^{\mathcal{R}}$ and then to B-Factor. By this we can reuse the results of realizing relations that we have seen in the previous sections.

▶ **Theorem 7.1.** *Let $B \subseteq \mathbb{N}$ be a fixed set of finite size. Given a CNF-formula ϕ with n variables and m clauses. We can construct a (simple) B-Factor$^{\mathcal{R}}$ instance G with $\mathcal{O}(nm)$ vertices, bounded degree and a linear layout of width* cutw $\leq n + \mathcal{O}(1)$ *in time linear in the output size. Further, the number of solutions for ϕ is equal to the number of solutions for G.*

Recall, that for the pathwidth lower bound we grouped variables together. This was needed to keep the pathwidth of the construction low. But this increased the cutwidth of the graph. Now, we do not group variables together but encode each variable on its own. See Figure 6 for an example of the construction we describe formally in the following.

Let $x_1, \ldots, x_n$ be the variables and $C_1, \ldots, C_m$ the clauses of ϕ. For each $i \in [n]$ and every $j \in [m]$ we create a vertex r_i^j. We assign the relation R^+ to r_i^j if x_i appears positively in C_j, R^- if it appears negatively, and otherwise R^0, where R^0, R^+, and R^- are defined later. Additionally add vertices r_i^0 and r_i^{m+1} with relation $\mathtt{HW}_{\in\{0,2\}}$ for all $i \in [n]$. We say that the vertices $r_i^0, \ldots, r_i^{m+1}$ form the ith *row*, i.e. the row of variable x_i. Create new nodes r_0^j and r_{n+1}^j and assign the relations $\mathtt{HW}_{=0}$ and $\mathtt{HW}_{=1}$ to them for all $j \in [m]$, respectively. We say the vertices $\{r_i^j\}_i$ form the jth *column*. We connect two nodes r_i^j and $r_{i'}^{j'}$ by an edge if $|i - i'| \leq 1$ and $|j - j'| \leq 1$ for all $i, i' \in [0, n+1]$ and $j, j' \in [0, m+1]$, i.e. if they are neighbors in the grid.

The idea is the same as for the pathwidth construction, except that selecting the edges of the ith row corresponds to setting the variable x_i to true. The edges between the nodes of a column represent if a clause is already satisfied. The relation $\mathtt{HW}_{=0}$ ensures that we start with an initially unsatisfied clause. At each node r_i^j we check whether the assignment to this variable x_i satisfies the clause C_j and then force the output edge (i.e. the bottom edge) to be selected. Otherwise we propagate the current state (i.e. the selection of edges). Eventually we reach r_{n+1}^j with relation $\mathtt{HW}_{=1}$ where the edge has to be selected and thus the clause must be satisfied.

The relations R^0, R^+, and R^- accept exactly those inputs that satisfy all of the following conditions:

1. The left edge is selected if and only if the right edge is selected.
2. If the top edge is selected, the bottom edge is selected.
3. Only for R^+: If the top edge is unselected and the left edge is selected, then the bottom edge is selected.
4. Only for R^-: If the top edge is unselected and the left edge is not selected, then the bottom edge is selected.

For the proofs of the lower bounds, i.e. Theorem 1.9, we follow the ideas from the pathwidth lower bounds in Section 3. Thus we also have to modify the graph a bit such that we obtain a B-Factor$^{\mathcal{R}}$ instance and can replace all relations by their realizations. The details can be found in the full version of the paper.

References

1 Ashwin Arulselvan, Ágnes Cseh, Martin Groß, David F. Manlove, and Jannik Matuschke. Matchings with lower quotas: Algorithms and complexity. *Algorithmica*, 80(1):185–208, 2018. doi:10.1007/s00453-016-0252-6.

2 Claude Berge. *Graphs and Hypergraphs*. North-Holland mathematical library, Amsterdam, 1973.

3 Hans L. Bodlaender, Marek Cygan, Stefan Kratsch, and Jesper Nederlof. Deterministic single exponential time algorithms for connectivity problems parameterized by treewidth. In Fedor V. Fomin, Rusins Freivalds, Marta Z. Kwiatkowska, and David Peleg, editors, *Automata, Languages, and Programming - 40th International Colloquium, ICALP 2013, Riga, Latvia, July 8-12, 2013, Proceedings, Part I*, volume 7965 of *Lecture Notes in Computer Science*, pages 196–207. Springer, 2013. doi:10.1007/978-3-642-39206-1_17.

4 Hans L. Bodlaender and Arie M. C. A. Koster. Combinatorial optimization on graphs of bounded treewidth. *Comput. J.*, 51(3):255–269, 2008. doi:10.1093/comjnl/bxm037.

5 Liming Cai and David W. Juedes. On the existence of subexponential parameterized algorithms. *J. Comput. Syst. Sci.*, 67(4):789–807, 2003. doi:10.1016/S0022-0000(03)00074-6.

6 Chris Calabro, Russell Impagliazzo, and Ramamohan Paturi. The complexity of satisfiability of small depth circuits. In Jianer Chen and Fedor V. Fomin, editors, *Parameterized and Exact Computation, 4th International Workshop, IWPEC 2009, Copenhagen, Denmark, September 10-11, 2009, Revised Selected Papers*, volume 5917 of *Lecture Notes in Computer Science*, pages 75–85. Springer, 2009. doi:10.1007/978-3-642-11269-0_6.

7 Gérard Cornuéjols. General factors of graphs. *J. Comb. Theory, Ser. B*, 45(2):185–198, 1988. doi:10.1016/0095-8956(88)90068-8.

8 Bruno Courcelle. The monadic second-order logic of graphs. I. Recognizable sets of finite graphs. *Information and Computation*, 85(1):12–75, 1990. doi:10.1016/0890-5401(90)90043-H.

9 Bruno Courcelle. The monadic second-order logic of graphs III: tree-decompositions, minor and complexity issues. *RAIRO Theor. Informatics Appl.*, 26:257–286, 1992. doi:10.1051/ita/1992260302571.

10 Radu Curticapean. Parity separation: A scientifically proven method for permanent weight loss. In Ioannis Chatzigiannakis, Michael Mitzenmacher, Yuval Rabani, and Davide Sangiorgi, editors, *43rd International Colloquium on Automata, Languages, and Programming, ICALP 2016, July 11-15, 2016, Rome, Italy*, volume 55 of *LIPIcs*, pages 47:1–47:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2016. doi:10.4230/LIPIcs.ICALP.2016.47.

11 Radu Curticapean and Dániel Marx. Tight conditional lower bounds for counting perfect matchings on graphs of bounded treewidth, cliquewidth, and genus. In Robert Krauthgamer, editor, *Proceedings of the Twenty-Seventh Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2016, Arlington, VA, USA, January 10-12, 2016*, pages 1650–1669. SIAM, 2016. doi:10.1137/1.9781611974331.ch113.

12 Marek Cygan, Holger Dell, Daniel Lokshtanov, Dániel Marx, Jesper Nederlof, Yoshio Okamoto, Ramamohan Paturi, Saket Saurabh, and Magnus Wahlström. On problems as hard as CNF-SAT. *ACM Trans. Algorithms*, 12(3):41:1–41:24, 2016. doi:10.1145/2925416.

13 Marek Cygan, Fedor V. Fomin, Lukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michal Pilipczuk, and Saket Saurabh. *Parameterized Algorithms*. Springer, 2015. doi:10.1007/978-3-319-21275-3.

14 Marek Cygan, Stefan Kratsch, and Jesper Nederlof. Fast hamiltonicity checking via bases of perfect matchings. *J. ACM*, 65(3):12:1–12:46, 2018. doi:10.1145/3148227.

15 Marek Cygan, Jesper Nederlof, Marcin Pilipczuk, Michal Pilipczuk, Johan M. M. van Rooij, and Jakub Onufry Wojtaszczyk. Solving connectivity problems parameterized by treewidth in single exponential time. In Rafail Ostrovsky, editor, *IEEE 52nd Annual Symposium on Foundations of Computer Science, FOCS 2011, Palm Springs, CA, USA, October 22-25, 2011*, pages 150–159. IEEE Computer Society, 2011. doi:10.1109/FOCS.2011.23.

16 Xavier Dahan. Regular graphs of large girth and arbitrary degree. *Comb.*, 34(4):407–426, 2014. doi:10.1007/s00493-014-2897-6.

17 Holger Dell, Thore Husfeldt, Dániel Marx, Nina Taslaman, and Martin Wahlen. Exponential time complexity of the permanent and the Tutte polynomial. *ACM Trans. Algorithms*, 10(4):21:1–21:32, 2014. doi:10.1145/2635812.

18 Szymon Dudycz and Katarzyna Paluch. Optimal general matchings. In Andreas Brandstädt, Ekkehard Köhler, and Klaus Meer, editors, *Graph-Theoretic Concepts in Computer Science - 44th International Workshop, WG 2018, Cottbus, Germany, June 27-29, 2018, Proceedings*, volume 11159 of *Lecture Notes in Computer Science*, pages 176–189. Springer, 2018. Full version: arXiv:1706.07418. doi:10.1007/978-3-030-00256-5_15.

19 Jack Edmonds. Paths, trees, and flowers. *Canadian Journal of Mathematics*, 17:449–467, 1965. doi:10.4153/CJM-1965-045-4.

20 Paul Erdős and Horst Sachs. Reguläre Graphen gegebener Taillenweite mit minimaler Knotenzahl. *Wiss. Z. Martin-Luther-Univ. Halle-Wittenberg Math.-Natur. Reihe*, 12(251-257):22, 1963.

21 Geoffrey Exoo and Robert Jajcay. Recursive constructions of small regular graphs of given degree and girth. *Discret. Math.*, 312(17):2612–2619, 2012. doi:10.1016/j.disc.2011.10.021.

22 Tomás Feder. Fanout limitations on constraint systems. *Theor. Comput. Sci.*, 255(1-2):281–293, 2001. doi:10.1016/S0304-3975(99)00288-1.

23 Fedor V. Fomin, Daniel Lokshtanov, and Saket Saurabh. Efficient computation of representative sets with applications in parameterized and exact algorithms. In Chandra Chekuri, editor, *Proceedings of the Twenty-Fifth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2014, Portland, Oregon, USA, January 5-7, 2014*, pages 142–151. SIAM, 2014. doi:10.1137/1.9781611973402.10.

24 Arne Hoffmann and Lutz Volkmann. On unique k-factors and unique $[1, k]$-factors in graphs. *Discret. Math.*, 278(1-3):127–138, 2004. doi:10.1016/S0012-365X(03)00248-6.

25 Russell Impagliazzo and Ramamohan Paturi. On the complexity of k-SAT. *J. Comput. Syst. Sci.*, 62(2):367–375, 2001. doi:10.1006/jcss.2000.1727.

26 Wilfried Imrich. Explicit construction of regular graphs without small cycles. *Comb.*, 4(1):53–59, 1984. doi:10.1007/BF02579157.

27 Sanjana Kolisetty, Linh Le, Ilya Volkovich, and Mihalis Yannakakis. The complexity of finding S-factors in regular graphs. *Electron. Colloquium Comput. Complex.*, 26:40, 2019. URL: https://eccc.weizmann.ac.il/report/2019/040.

28 Felix Lazebnik, Vasiliy A Ustimenko, and Andrew J Woldar. A new series of dense graphs of high girth. *Bulletin of the American mathematical society*, 32(1):73–79, 1995.

29 Daniel Lokshtanov, Dániel Marx, and Saket Saurabh. Lower bounds based on the exponential time hypothesis. *Bull. EATCS*, 105:41–72, 2011. URL: http://eatcs.org/beatcs/index.php/beatcs/article/view/92.

30 Daniel Lokshtanov, Dániel Marx, and Saket Saurabh. Known algorithms on graphs of bounded treewidth are probably optimal. *ACM Trans. Algorithms*, 14(2):13:1–13:30, 2018. doi:10.1145/3170442.

31 László Lovász. The factorization of graphs. II. *Acta Mathematica Hungarica*, 23(1-2):223–246, 1972.

32 Silvio Micali and Vijay V. Vazirani. An O(sqrt(|v|) |E|) algorithm for finding maximum matching in general graphs. In *21st Annual Symposium on Foundations of Computer Science, Syracuse, New York, USA, 13-15 October 1980*, pages 17–27. IEEE Computer Society, 1980. doi:10.1109/SFCS.1980.12.

33 Michal Pilipczuk. Problems parameterized by treewidth tractable in single exponential time: A logical approach. In Filip Murlak and Piotr Sankowski, editors, *Mathematical Foundations of Computer Science 2011 - 36th International Symposium, MFCS 2011, Warsaw, Poland, August 22-26, 2011. Proceedings*, volume 6907 of *Lecture Notes in Computer Science*, pages 520–531. Springer, 2011. doi:10.1007/978-3-642-22993-0_47.

34 Yossi Shiloach. Another look at the degree constrained subgraph problem. *Inf. Process. Lett.*, 12(2):89–92, 1981. doi:10.1016/0020-0190(81)90009-0.

35 Leslie G. Valiant. The complexity of computing the permanent. *Theor. Comput. Sci.*, 8:189–201, 1979. doi:10.1016/0304-3975(79)90044-6.

36 Leslie G. Valiant. The complexity of enumeration and reliability problems. *SIAM J. Comput.*, 8(3):410–421, 1979. doi:10.1137/0208032.

37 Johan M. M. van Rooij. Fast algorithms for join operations on tree decompositions. In Fedor V. Fomin, Stefan Kratsch, and Erik Jan van Leeuwen, editors, *Treewidth, Kernels, and Algorithms - Essays Dedicated to Hans L. Bodlaender on the Occasion of His 60th Birthday*, volume 12160 of *Lecture Notes in Computer Science*, pages 262–297. Springer, 2020. doi:10.1007/978-3-030-42071-0_18.

38 Johan M. M. van Rooij, Hans L. Bodlaender, and Peter Rossmanith. Dynamic programming on tree decompositions using generalised fast subset convolution. In Amos Fiat and Peter Sanders, editors, *Algorithms - ESA 2009, 17th Annual European Symposium, Copenhagen, Denmark, September 7-9, 2009. Proceedings*, volume 5757 of *Lecture Notes in Computer Science*, pages 566–577. Springer, 2009. doi:10.1007/978-3-642-04128-0_51.

High-Girth Near-Ramanujan Graphs with Lossy Vertex Expansion

Theo McKenzie
Department of Mathematics, University of California, Berkeley, CA, USA

Sidhanth Mohanty
Department of Computer Science, University of California, Berkeley, CA, USA

Abstract

Kahale proved that linear sized sets in d-regular Ramanujan graphs have vertex expansion at least $\frac{d}{2}$ and complemented this with construction of near-Ramanujan graphs with vertex expansion no better than $\frac{d}{2}$. However, the construction of Kahale encounters highly local obstructions to better vertex expansion. In particular, the poorly expanding sets are associated with short cycles in the graph. Thus, it is natural to ask whether the vertex expansion of *high-girth* Ramanujan graphs breaks past the $\frac{d}{2}$ bound. Our results are two-fold:

1. For every $d = p + 1$ for prime $p \geq 3$ and infinitely many n, we exhibit an n-vertex d-regular graph with girth $\Omega(\log_{d-1} n)$ and vertex expansion of sublinear sized sets bounded by $\frac{d+1}{2}$ whose nontrivial eigenvalues are bounded in magnitude by $2\sqrt{d-1} + O\left(\frac{1}{\log_{d-1} n}\right)$.
2. In any Ramanujan graph with girth $C \log_{d-1} n$, all sets of size bounded by $n^{0.99C/4}$ have near-lossless vertex expansion $(1 - o_d(1))d$.

The tools in analyzing our construction include the nonbacktracking operator of an infinite graph, the Ihara–Bass formula, a trace moment method inspired by Bordenave's proof of Friedman's theorem [8], and a method of Kahale [16] to study dispersion of eigenvalues of perturbed graphs.

2012 ACM Subject Classification Mathematics of computing → Spectra of graphs; Theory of computation → Expander graphs and randomness extractors

Keywords and phrases expander graphs, Ramanujan graphs, vertex expansion, girth

Digital Object Identifier 10.4230/LIPIcs.ICALP.2021.96

Category Track A: Algorithms, Complexity and Games

Funding *Theo McKenzie*: This material is based upon work supported by the National Science Foundation Graduate Research Fellowship Program under Grant No. DGE-1752814.
Sidhanth Mohanty: Supported by NSF grant CCF-1718695.

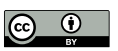
Acknowledgements We would like to thank Shirshendu Ganguly and Nikhil Srivastava for their highly valuable insights, intuition, and comments. We would also like to thank Amitay Kamber for helpful comments on the initial version of the preprint.

1 Introduction

This paper is concerned with expander graphs, which are ubiquitous in theoretical computer science. A natural and highly well-studied quantity associated with a d-regular graph is its *edge expansion* defined as

$$\min_{|S| \leq \epsilon n} E(S, \overline{S})/|S|,$$

for some constant ϵ. Namely it is the minimum ratio of edges leaving a set S to the size of S for all S of appropriately bounded size. While edge expansion is known to be intractable to compute, there are explicit constructions of good edge expanders, and it is closely related to the second largest magnitude eigenvalue of its adjacency matrix, also known as *spectral*



expansion of a graph, via the expander mixing lemma and Cheeger's inequality [1]. Spectral expansion is easily computable. In particular, an application of the expander mixing lemma proves that small enough sets in graphs with spectral expansion $o(d)$ have near-optimal edge expansion of $(1 - o_d(1))d$.

A natural analog to edge expansion is *vertex expansion*, defined as

$$\min_{|S| \leq \epsilon n} |\Gamma(S)|/|S|$$

for some constant ϵ, where $\Gamma(S)$ is the neighborhood of the set S (potentially containing vertices of S). However, as difficult as edge expansion is to ascertain, vertex expansion has proven far more challenging.

As witnessed by the neighborhood of a vertex, we cannot hope for vertex expansion greater than $d-1$. Therefore we call a graph a *lossless vertex expander* if for every δ, there exists an ϵ such that there is vertex expansion $d - 1 - \delta$ for sets of size ϵn. Lossless vertex expanders exist since a random d-regular graph is one with high probability (see [15, Theorem 4.16] for a proof). However no deterministic construction of such graphs is known. In an effort to understand lossless vertex expansion better and give explicit constructions, a natural question to ask is: *what properties of random graphs leads to lossless vertex expansion*?

Since a random d-regular graph is near-Ramanujan with high probability [11], and since near-Ramanujan graphs have near-optimal edge expansion, it is natural to inquire if spectral expansion has any implications for vertex expansion as well. Kahale [16] showed that the spectral expansion gives a bound on the vertex expansion. Specifically, Ramanujan graphs (namely graphs with optimal spectral expansion) have vertex expansion at least $d/2$. While this is a nontrivial implication, it falls short of achieving the coveted *losslessness* property. Kahale also proved that the bound of $d/2$ is tight. In particular, he exhibited an infinite family of near-Ramanujan graphs with vertex expansion $d/2$, which means spectral expansion alone is not sufficient for lossless vertex expansion.

The occurrence of a copy of $K_{2,d}$[1] as a subgraph is the obstruction to lossless vertex expansion in Kahale's example. Kahale's example deviates from a random graph in that it is highly unlikely for a random graph to contain a copy of $K_{2,d}$ as a subgraph. More generally, random graphs have the property that with high probability any two "short" cycles are far apart, which Kahale's example doesn't satisfy. Thus, it is natural to ask if the "near-Ramanujan" property in conjunction with the "separatedness of cycles" property of random graphs break past the $d/2$ barrier of Kahale. The "separatedness of cycles" property is especially interesting to consider since it is a key property of random graphs exploited in proofs of Alon's conjecture [11, 8]. A concrete question we can ask is: *Do Ramanujan graphs with $\Omega(\log_{d-1} n)$ girth have lossless vertex expansion?*

An affirmative answer to the above question would prove that the Ramanujan graphs of Lubotzky, Phillips, and Sarnak [18] are lossless vertex expanders. Towards answering the above question, we prove the following negative result:

▶ **Theorem 1.** *For every $d = p + 1$ for prime $p \geq 3$, there is an infinite family of d-regular graphs G on n vertices of girth $\geq \left(\frac{2}{3} - o_n(1)\right) \log_{d-1} n$ where there is a set of vertices U such that $|\Gamma(U)| \leq (d+1)|U|/2$, $|U| \leq n^{1/3}$, and $\max\{\lambda_2(G), -\lambda_n(G)\} \leq 2\sqrt{d-1} + O(1/\log_{d-1} n)$.*

We also complement the above with a positive result which can be summarized as "small enough sets in Ramanujan graphs expand nearly losslessly":

[1] complete bipartite graph with 2 vertices on one side and d vertices on the other

▶ **Theorem 2.** *Let G be a d-regular Ramanujan graph with girth $C \log_{d-1} n$, then every set of S of size $\leq n^{\kappa}$ for $\kappa < \frac{C}{4}$ has vertex expansion $(1 - o_d(1))d$.*

After posting our preprint, Amitay Kamber informed us that a theorem in an alternative version of [16] gives the same bound by a different argument. Moreover, his theorem does not depend on the spectral expansion of the graph. However, our proof may be of interest as an alternate method.

1.1 Technical Overview

We give a brief description of how Theorem 1 and Theorem 2 are proved.

1.1.1 Overview of Proof of Theorem 1

Our proof is inspired by that of Kahale's. At a high level, Kahale embeds a copy of $K_{2,d}$ within a Ramanujan graph. We proceed similarly to Kahale, but instead of embedding a $K_{2,d}$, we embed a single subgraph H that is high girth but a lossy vertex expander and show that if H has size n^{α} for some $0 < \alpha \leq 1/3$, the overall graph is still near-Ramanujan.

Our proof involves two steps: the first step is in proving that the subgraph H being embedded has spectral radius bounded by $2\sqrt{d-1}$, and the second step is in proving that planting H within a Ramanujan graph results in a near-Ramanujan graph. For the first step, we describe an infinite graph containing H and bound its spectral radius via a trace moment method. The trace moment method involves bounding the number of closed walks satisfying certain properties within a graph, and is inspired by an encoding argument from Bordenave's proof of Friedman's theorem [8].

The second step is in proving that our method of embedding a copy of H within a Ramanujan graph does not perturb the eigenvalues by a large amount. Towards doing so, we use the fact that the spectral radius of H is bounded by $2\sqrt{d-1}$ in conjunction with Kahale's argument about dispersion of eigenvalues in high-girth graphs.

1.1.2 Overview of Proof of Theorem 2

We first prove that if a set S in a Ramanujan graph has "lossy" vertex expansion, then we can construct a graph H on vertex set S such that (i) the girth of H is at least half the girth of G, and (ii) the average degree of H is "high" (in particular, the worse the vertex expansion of S, the higher the average degree of H). We then employ the irregular Moore bound, which gives a quantitative tradeoff between the average degree of a graph and its girth. In particular, this would imply that a Ramanujan graph with "lossy" vertex expansion necessarily must have "low" girth.

1.2 Related Work

1.2.1 Applications of Vertex Expanders

There are many applications of expander graphs where having vertex expansion is particularly useful. For example, lossless expanders are particularly of interest in the field of error correcting codes [19, 26, 27]. Lossless vertex expanders give linear error correcting codes that are decodable in linear time [26]. Guruswami, Lee and Razborov [14] use bipartite vertex expanders to construct large subspaces of $\mathbb{R}^n$ where all vectors x in the subspace satisfy $(\log n)^{-O(\log\log\log n)}||x||_2 \leq ||x||_1 \leq \sqrt{n}||x||_2$.

1.2.2 Explicit Constructions

Constructions of Ramanujan graphs of [18, 20, 22] of all degrees that are of the form $p^r + 1$ for p prime, as well as the construction of near-Ramanujan graphs of every degree of [21] have vertex expansion $\sim \frac{d}{2}$ just by virtue of being Ramanujan via Kahale's result. In fact no deterministic construction has improved upon the $d/2$ bound obtained from solely spectral information. In a remarkable work, Capalbo et al. [10] exhibited an explicit construction of a bipartite graph where subsets of one side of the bipartite graph expand losslessly to the other, using a zig-zag product so the the losslessness of a small, random-like graph boosts the expansion from a large, potentially lossy vertex expanding graph.

1.2.3 Quantum Ergodicity

Quantum ergodicity is another area where both local and global properties of random-like graphs are used. In particular, Anantharaman and Le Masson [6] proved that graphs that have few short cycles (and are therefore close to high girth) and spectral expansion are quantum ergodic, which in this context means the eigenvectors are equidistributed across vertices. Anantharaman, as well as Brooks, Le Masson, and Lindenstrauss exhibited alternative proofs [5, 9]. The proof from [9] shows that quantum ergodicity is equivalent to the mixing of a certain graphical operator. They then use high girth to show that this is equivalent to showing mixing on the infinite tree, then expansion to show the nonbacktracking operator mixes on the tree.

1.2.4 Eigenvector Delocalization

Ganguly and Srivastava, and later Alon, Ganguly and Srivastava [12, 3] give a perturbation of the LPS graph similar to Kahale's argument, but instead of individual vertices, two trees are added and connected to the graph. By assuming the tree is sufficiently deep and carefully connecting the tree to the rest of the graph, the authors create a graph that is high girth but contains eigenvectors that are localized. These graphs are also lossy vertex expanders. However, they show that these graphs cannot be Ramanujan, but rather have spectral radius at least $(2 + c)\sqrt{d - 1}$ where $c > 0$ is a constant. Alon [2] used eigenvector delocalization to create near-Ramanujan expanders of every degree by perturbing known constructions of Ramanujan or near-Ramanujan graphs. Paredes [23] used similar techniques to remove short cycles in a graph while preserving expansion and uses this to algorithmically create graphs that are near-Ramanujan and also have girth at least $\Omega(\sqrt{\log n})$.

1.2.5 Complexity of Constraint Satisfaction Problems

Proofs that it is hard for even linear degree Sum-of-Squares to refute random 3XOR and 3SAT instances on n variables [13, 25] rely on lossless vertex expansion of some sets in a graph underlying a random instance, which suggests a connection between deterministic algorithms for constructing lossless vertex expanders and algorithms for explicit hard instances for Sum-of-Squares.

2 Preliminaries

2.1 Elementary Graph Theory

▶ **Definition 3.** *The girth $g(G)$ of a graph G is the length of the smallest cycle in G.*

▶ **Definition 4.** *For $G = (V, E)$, the valency of $a \in V$ to $B \subset V$ is $|\Gamma(a) \cap B|$, where $\Gamma(S)$ for $S \subset V$ is the set of neighbors of S in G.*

▶ **Definition 5.** *The ball of radius h around a set $U \subset V$, denoted* $\mathrm{Ball}_h(U)$*, is the set of vertices of distance at most h from U.*

▶ **Definition 6.** *The vertex expansion of a set $U \subset V$ is*

$$\Psi(U) := \frac{|\Gamma(U)|}{|U|}.$$

Similarly, the ϵ-vertex expansion of a graph G is:

$$\Psi_\epsilon(G) = \min_{|U| \leq \epsilon |V|} \Psi(U)$$

where U ranges over subsets of V, and ϵ is an arbitrary constant.

▶ **Definition 7.** *Given a graph G, we use A_G to denote its adjacency matrix. When G is a finite graph on n vertices, the eigenvalues of A_G can be ordered as $\lambda_1(G) \geq \lambda_2(G) \geq \cdots \geq \lambda_n(G)$.*

▶ **Definition 8.** *We use B_G to denote the* nonbacktracking matrix *of a graph G which is a matrix with rows and columns indexed by directed edges of G defined as follows:*

$$B[(u,v),(w,x)] = \begin{cases} 1 & \text{if } v = w \text{ and } u \neq x \\ 0 & \text{otherwise.} \end{cases}$$

▶ **Definition 9.** *The* spectral expansion *of a finite graph G, denoted $\lambda(G)$ is defined as $\max\{\lambda_2(G), -\lambda_n(G)\}$, which can equivalently be described as the "second largest absolute eigenvalue".*

We now state the following standard fact known as the *expander mixing lemma* (see [15, Lemma 2.5]).

▶ **Lemma 10** (Expander Mixing Lemma)**.** *Let G be a d-regular graph on n vertices. For any two subsets of vertices, $S, T \subseteq V(G)$, let $e(S,T)$ be the number of pairs of vertices (x,y) such that $x \in S, y \in T$ and $\{x,y\}$ is an edge in G. Then:*

$$\left| e(S,T) - \frac{d}{n}|S| \cdot |T| \right| \leq \lambda(G)\sqrt{|S| \cdot |T|}.$$

And finally, we state the "irregular Moore bound" of [4] which articulates a tradeoff between the average degree of a graph and its girth.

▶ **Lemma 11.** *Let G be a n-vertex graph with average degree-d. Then*

$$g(G) \leq 2\log_{d-1} n + 2.$$

2.2 Operator Theory

In this section, let V be a countable set and $T : \ell_2(V) \to \ell_2(V)$ be a bounded linear operator.

▶ **Definition 12.** *The* spectrum *of T, which we denote* $\mathsf{spec}(T)$*, is the set of all $\lambda \in$ such that $\lambda I - T$ is not invertible.*

▶ **Definition 13.** *The* spectral radius *of T, which we denote $\rho(T)$ is defined as* $\sup\{|\lambda| : \lambda \in \mathsf{spec}(T)\}$.

▶ **Observation 14.** *The* operator norm *of T, which we write as $\|T\|$ is equal to $\sqrt{\rho(TT^*)}$ where T^* is the adjoint of T.*[2]

▶ **Observation 15.** $\rho(T) = \lim_{\ell\to\infty} \|T^\ell\|^{1/\ell}$.

▶ **Observation 16** (Consequence of [24, Theorem 6])**.** *Suppose T is a self-adjoint operator, and Φ is a basis of $\ell_2(V)$. Then:*

$$\rho(T) = \sup_{\phi\in\Phi} \limsup_{k\to\infty} |\langle \phi, T^k\phi\rangle|^{1/k}.$$

▶ **Observation 17.** *Let A be any principal submatrix of T. Then $\rho(A) \le \rho(T)$.*

▶ **Corollary 18.** *If H is a subgraph of (possibly infinite) graph G, then $\rho(A_H) \le \rho(A_G)$.*

3 Infinite Trees Hanging from a Biregular Graph

Let H be any $(2, d-1)$-biregular graph where the partition with degree-$(d-1)$ vertices is called U and the partition with degree-2 vertices is called V. Let X be the infinite graph constructed from H as follows:

At every vertex in U, the $(d-1)$-regular partition, glue an infinite tree where the root has degree-1 and the remaining vertices have degree-d. At every vertex in V, the 2-regular partition, glue an infinite tree where the root has degree-$(d-2)$ and every other vertex has degree-d.

Note that X is a d-regular infinite graph. The main result of this section is:

▶ **Lemma 19.** $\rho(A_X) \le 2\sqrt{d-1}$.

To prove Lemma 19, we instead turn our attention to the nonbacktracking matrix of X, called B_X. In particular, we bound $\rho(B_X)$ and then employ the Ihara–Bass formula of [7] for infinite graphs to translate the bound on $\rho(B_X)$ into a bound on $\rho(A_X)$.

Thus, we first prove:

▶ **Lemma 20.** $\rho(B_X) \le \sqrt{d-1}$.

We use the following version of the Ihara–Bass formula of [7] for infinite graphs.

▶ **Theorem 21.** *Let G be a (possibly infinite) graph. Then*

$$\mathsf{spec}(B_G) = \{\pm 1\} \cup \{\lambda : (D_G - I) - \lambda A_G + \lambda^2 I \text{ is not invertible}\}.$$

An immediate corollary that we will use is:

▶ **Corollary 22.** *Let G be a d-regular graph. Then $\rho(B_G) \le \sqrt{d-1}$ implies that $\rho(A_G) \le 2\sqrt{d-1}$.*

Proof. If there is μ in $\mathsf{spec}(A_G)$ such that $|\mu| > 2\sqrt{d-1}$, then $\mu I - A_G$ is not invertible. Consequently, by Theorem 21 $\lambda = \frac{\mu+\sqrt{\mu^2-4(d-1)}}{2}$, which is greater than $\sqrt{d-1}$, is in $\mathsf{spec}(B_G)$. ◀

In light of Corollary 22, we see that Lemma 20 implies Lemma 19.

Towards proving Lemma 20, we first make a definition.

---

[2] Since $\ell_2(V)$ comes equipped with the inner product $\langle f, g\rangle := \sum_{v\in V} f(v)g(v)$, T^* is simple the "transpose" of T.

▶ **Definition 23.** *We call a walk W a $(a \times b)$-linkage if it can be split into a segments, each of which is a length-b nonbacktracking walk.*

Proof of Lemma 20. By Observation 15

$$\rho(B_X) = \limsup_{\ell \to \infty} \|B_X^\ell\|^{1/\ell}.$$

Since $\|B_X^\ell\| = \sqrt{\rho(B_X^\ell (B_X^*)^\ell)}$ it suffices to bound $\rho(T)$ where $T := B_X^\ell (B_X^*)^\ell$ is a bounded self-adjoint operator, and hence by Observation 16:

$$\rho(T) = \max_{uv \in \vec{E}(X)} \limsup_{k \to \infty} |\langle 1_{uv}, T^k 1_{uv} \rangle|^{1/k}.$$

The quantity $\langle 1_{uv}, T^k 1_{uv} \rangle$ is bounded by the number of $(2k \times (\ell+1))$-linkages that start and end at vertex u, which we can bound via an encoding argument. In particular, we will give an algorithm to uniquely encode such linkages and bound the total number of possible encodings.

3.1 Encoding Linkages

Each length-$(\ell+1)$ nonbacktracking segment can be broken into 3 consecutive phases (of which some can possibly be empty): the phase where distance to H decreases on each step (Phase 1), the second phase where distance to H does not change on each step (Phase 2), and the third phase where distance to H increases on each step (Phase 3). We further break the third phase into two (possibly empty) subphases – the first subphase where the distance to u decreases on each step (Phase 3a), and the second subphase where the distance to u increases on each step (Phase 3b).

To encode the linkage, for each length-$(\ell+1)$ nonbacktracking we specify four numbers denoting the lengths of Phases 1, 2, 3a, and 3b. Note that Phase 2 is nonempty only if it is contained in H. For each step ab in Phase 2 that goes from U (the $(d-1)$-regular partition) to V (the 2-regular partition) we specify a number i in $[d-1]$ such that b is the ith neighbor of a within H. If the first step ab in Phase 2 is from V to U we specify a number in $[2]$ denoting if b is the first or second neighbor of a. For each step ab in Phase 3b we specify a number i in $[d-1]$ such that b is the ith neighbor of a that does not lie in the path between between u and H.

3.2 Recovering Linkages from Encodings

We recover a linkage from its encoding "segment-by-segment". Suppose the first t segments have been recovered, we show how to recover the $(t+1)$-th segment. Let x be the vertex the walk is at after it has traversed the first t segments. The steps taken in Phase 1 can be recovered from the length of the Phase since there is a unique path from any vertex to H. The steps in Phase 2 alternate between stepping from V to U and from U to V. It is easy to recover the first step of Phase 2 as well as any step from U to V; a step ab from V to U that is not the first step of Phase 2 is uniquely determined by the previous step, since a has 2 neighbors in U and by the nonbacktracking nature of the walk there is only one choice for b. Note that Phase 3a is nonempty only if u is not in H and all the steps are contained in the same branch as u. Since there is a unique shortest path between the start vertex of Phase 3a and u, the steps taken in Phase 3a can be recovered from its length. Finally, it is easy to recover the steps taken in Phase 3b since they are explicitly given in the encoding.

3.3 Counting Encodings

Now we turn our attention to bounding the total number of encodings. For given $\alpha, \beta \geq 0$ such that $\alpha + \beta = 2k(\ell+1)$ we first bound the number of walks such that α steps occur in Phase 2 (i.e. are within H) and β steps occur outside Phase 2 (i.e. are outside H). Let $v_1, v_2, \ldots, v_{2k(\ell+1)}$ be the sequence of vertices visited by the walk in order. By $d(x,y)$ we denote the graphical distance between vertices x and y, and for a set of vertices A, we write $d(x,A) := \min_{y \in A} d(x,y)$. Since $d(v_1, H) = d(v_{2k(\ell+1)}, H)$, $|d(v_i, H) - d(v_{i+1}, H)| \leq 1$ always and $|d(v_i, H) - d(v_{i+1}, H)| = 0$ for every step in Phase 2, the number of steps of the walk that occur in Phase 3 of their respective segments is at most $\frac{\beta}{2}$. In particular, the number of steps that occur in Phase 3b of their respective segments is bounded by $\frac{\beta}{2}$. The following bounds hold:

- The number of possible encodings of the lengths of phases is at most $(\ell+1)^{8k}$.
- The number of possible encodings of the first step of Phase 2 of each segment is at most 2^{2k}.
- The number of possible encodings of the list of U-to-V steps in Phase 2 is at most $(d-1)^{\frac{\alpha+1}{2}}$ because the steps taken in Phase 2 alternate between going from V to U and from U to V.
- The number of possible encodings of the list of steps in Phase 3b is at most $(d-1)^{\frac{\beta}{2}}$.

The above bounds combine to give a bound of

$$(\ell+1)^{8k} 2^{2k} (d-1)^{\frac{\alpha+1}{2}} (d-1)^{\frac{\beta}{2}} \leq (\ell+1)^{8k} 2^{2k} \sqrt{d-1}^{2k(\ell+1)+1}.$$

As there are at most $2k\ell$ choices for (α, β) pairs, the number of $(2k \times (\ell+1))$-linkages is at most

$$2k\ell(\ell+1)^{8k} 2^{2k} \sqrt{d-1}^{2k(\ell+1)+1}.$$

Thus,

$$\rho(T) \leq \limsup_{k \to \infty} \left(2k\ell(\ell+1)^{8k} 2^{2k} \sqrt{d-1}^{2k(\ell+1)+1}\right)^{1/k} = 4(\ell+1)^8 \sqrt{d-1}^{2(\ell+1)}.$$

Consequently,

$$\rho(B_X) \leq \limsup_{\ell \to \infty} \rho(T)^{1/2\ell} \leq \limsup_{\ell \to \infty} \left(4(\ell+1)^8 \sqrt{d-1}^{2(\ell+1)}\right)^{1/2\ell} = \sqrt{d-1}. \qquad \blacktriangleleft$$

4 High-Girth Near-Ramanujan graphs with Lossy Vertex Expansion

We will plant a high girth graph with low spectral radius within a d-regular Ramanujan graph. We will show that such a construction is a spectral expander, but has low vertex expansion. By $u \sim_G v$, we mean that u and v are adjacent in the graph G. We will write $u \sim v$ when the graph is clear from context.

Consider a $(2, d-1)$ biregular bipartite graph $H = (U, V, E)$, with vertex components U and V. U is the degree-$(d-1)$ component and V the degree-2 component. Therefore if we define $\gamma := |U|$, requiring γ to be even, then $|V| = (d-1)\gamma/2$. Call the vertices of U and V $\{u_1, \ldots, u_\gamma\}$ and $\{v_1, \ldots, v_{\gamma(d-1)/2}\}$, respectively. We connect U and V in such a way to maximize the girth of H.

▶ **Lemma 24.**

$$g(H) \geq 2 \log_{d-1} \gamma.$$

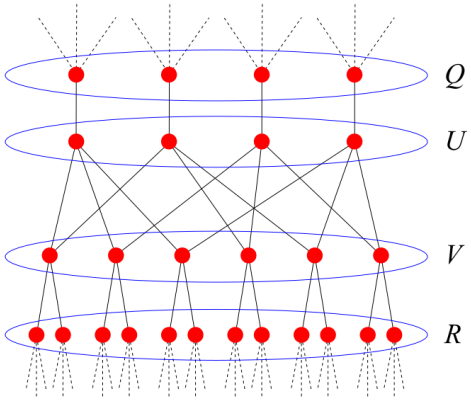


Figure 1 H', with labeled components for $d = 4$, $\gamma = 4$. Note that $\Psi(U) = (d+1)/2$. To create G', we connect Q and R to a well spaced matching in G.

Proof. Because of the valency conditions on H, there is a graph $\tilde{H}$ on γ vertices $\{\tilde{u}_1, \ldots, \tilde{u}_\gamma\}$, where $\tilde{u}_i \sim_{\tilde{H}} \tilde{u}_j$ if and only if $\exists v_k \in H$ such that $u_i \sim_H v_k$ and $u_j \sim_H v_k$. Namely, U corresponds to the vertex set of $\tilde{H}$, and V corresponds to the edge set. $\tilde{H}$ is $d-1$ regular, and, as paths in $\tilde{H}$ of length r correspond to paths of length $2r$ in H, $g(H) = 2g(\tilde{H})$.

By a result of Linial and Simkin [17], there exists a graph $\tilde{H}$ that has girth at least $c \log_{d-2} \gamma$, for any $c \in (0,1)$, assuming γ is even. Therefore by setting $c = \log(d-2)/\log(d-1)$, we have that $g(\tilde{H}) \geq \log_{d-1} \gamma$ and $g(H) \geq 2 \log_{d-1} \gamma$. ◀

We add a new set of vertices $Q = \{q_1, \ldots, q_\gamma\}$ and add a matching between Q and U, adding the edge $q_i u_i$ for $1 \leq i \leq \gamma$. Similarly, we add another set of vertices $R = \{r_{i,j}\}, 1 \leq i \leq \gamma(d-1)/2, 1 \leq j \leq d-2$. For each $1 \leq i \leq \gamma(d-1)/2$, we then add an edge from v_i to each of $r_{i,j}$ for $1 \leq j \leq (d-2)$.

We call H' the graph on $U \cup V \cup Q \cup R$. At this point vertices of U and V have degree-d, and vertices of Q and R have degree-1. Also, note $\Psi(U) = (d+1)/2$. We wish to embed H' into a larger, high girth expander, and show that this new graph maintains high girth and expansion, even though the set U is a lossy vertex expander. Our argument follows that of [16, Section 5], but instead of embedding individual vertices, we will embed H'.

▶ **Theorem 25** (Theorem 1 in more detail). *For every $d = p+1$ for prime $p \geq 3$, there is an infinite family of d-regular graphs $G_m = (V_m, E_m)$ on m vertices, such that $\exists U_m \subset V_m$ with $\Psi(U_m) = (d+1)/2$ for $|U_m| \leq m^{1/3}$, $g(G_m) = (\frac{2}{3} - o_m(1)) \log_{d-1} m$, and such that $\lambda(G_m) \leq 2\sqrt{d-1} + O(1/\log_{d-1} m)$.*

Proof. By the result of Lubotzky, Phillips and Sarnak [18], for such d, there exists an infinite family of d-regular graphs, where graphs of n vertices have girth $(\frac{4}{3} - o_n(1)) \log_{d-1} n$ and have spectral expansion $\leq 2\sqrt{d-1}$.

For a given graph $G = (V, E)$ of this type of size n, we attach H' by removing a matching $M \subset E$, $M = \{(a_{1,1}, a_{1,2}), \ldots, (a_{k,1}, a_{k,2})\}$ for

$$k := \gamma(d-1)(2 + (d-1)(d-2))/4. \tag{1}$$

We take a matching such that the pairwise distance between edges in the matching is maximized in G.

▶ **Lemma 26.** *In a d-regular graph on n vertices, there exists a matching M of size k such that for every pair of edges $(a_{i_1,1}, a_{i_1,2}), (a_{i_2,1}, a_{i_2,2}) \in M$, $i_1 \neq i_2$,*

$$d((a_{i_1,1}, a_{i_1,2}), (a_{i_2,1}, a_{i_2,2})) \geq \log_{d-1} n - \log_{d-1} \gamma - O_n(1).$$

Proof. For a given pair of adjacent vertices $(a_{i,1}, a_{i,2})$, as our graph is d regular, there are at most $1 + d\frac{(d-1)^r - 1}{d-2}$ vertices at distance at most r from $a_{i,1}$, and at most $(d-1)^r$ vertices at distance r from $a_{i,2}$ and distance $r+1$ from $a_{i,1}$. Therefore for any $d \geq 4$, the number of edges at distance at most r from a given edge is less than $4(d-1)^r$. We then greedily add edges by choosing an arbitrary edge with vertices at distance at least r away from all already chosen edges. A kth such edge will exist as long as $4k(d-1)^r \leq n$. For our k given in (1) we can set $r = \log_{d-1} n - \log_{d-1} \gamma - O_n(1)$. ◀

To connect H' to G, we first delete the matching M. Then for every vertex of Q and R, we add $d-1$ edges to the set of vertices of M, connecting to each vertex of M exactly once. Namely, the induced subgraph on $(Q \cup R) \cup M$ is a $(d-1, 1)$ biregular bipartite graph. Call $G' = (V', E')$ the new graph formed from G and H'.

We wish to show that G' remains high girth and a good spectral expander. For the girth of G', cycles are either completely contained in H', completely contained in G, or a mix between the two. Cycles in H' have length at least $2\log_{d-1} \gamma$ by Lemma 24. Cycles in G have length at least $(\frac{4}{3} - o_n(1)) \log_{d-1} n$ by the construction of [18]. For cycles that are a mix of H' and G, we must go from one vertex of H' to another vertex of H' through G. Therefore by Lemma 26, the length of such a cycle is at least $\log_{d-1} n - \log_{d-1} \gamma - O_n(1)$, giving

$$g(G') \geq \min\{2\log_{d-1} \gamma, \log_{d-1} n - \log_{d-1} \gamma - O_n(1)\}.$$

To show that the spectrum is not adversely affected, we follow the argument of [16, Theorem 5.2], with some adjustments. For our new graph, assume that there is an eigenvector $g \perp \mathbf{1}$ corresponding to an eigenvalue $|\mu| > 2\sqrt{d-1}$.

Call A the adjacency matrix of G', and A_G the adjacency matrix of G padded with zeros so it is of the same size as A. Then we have

$$g^* A g = g_G^* A_G g_G + g_{H'}^* A g_{H'} - 2\sum_{i=1}^{k} g(a_{i,1}) g(a_{i,2}) + \sum_{\substack{u \in Q \cup R \\ a_{i,j} \in M \\ u \sim a_{i,j}}} g(u) g(a_{i,j})$$

where g_G and $g_{H'}$ are the projections of g onto G and H', respectively.

We know that

$$|g_G^* A_G g_G| \leq 2\sqrt{d-1}||g_G||^2 + \frac{d}{n}\left(\sum_{u \in G} g(u)\right)^2$$

by decomposing g into parts parallel and perpendicular to the all ones vector.

By a combination of Lemma 19 and Corollary 18, the spectral radius of H' is at most $2\sqrt{d-1}$, and therefore we have

$$|g_G^* A_G g_G| + |g_{H'}^* A g_{H'}| \leq 2\sqrt{d-1}||g||^2 + \frac{d}{n}\left(\sum_{u \in H'} g(u)\right)^2$$

as $\sum_G g(u) = -\sum_{H'} g(u)$, considering $g \perp \mathbf{1}$.

To show that $|\mu| = 2\sqrt{d-1} + O(1/\log n)$, we then need to show

$$\frac{1}{\|g\|^2}\left(\frac{d}{n}\left(\sum_{H'} g_{H'}(u)\right)^2 - 2\sum_{i=1}^{k} g(a_{i,1})g(a_{i,2}) + \sum_{\substack{u\in Q\cup R\\ a_{i,j}\in M\\ u\sim a_{i,j}}} g(u)g(a_{i,j})\right) = O\left(\frac{1}{\log n}\right). \quad (2)$$

The first term of (2) can be bounded as

$$\frac{d}{n}\left(\sum_{H'} g_{H'}(u)\right)^2 \leq \frac{d}{n}|H'|\,\|g_{H'}\|^2 \leq \frac{\gamma(2+(d-1)(d-2))d}{2n}\|g_{H'}\|^2. \quad (3)$$

The second term we can bound as

$$\left|2\sum_{i=1}^{k} g(a_{i,1})g(a_{i,2})\right| \leq \sum_{a_{i,j}\in M} g(a_{i,j})^2. \quad (4)$$

Now we will bound the last term of (2) using the Cauchy-Schwarz inequality.

$$\left|\sum_{\substack{u\in Q\cup R\\ a_{i,j}\in M\\ u\sim a_{i,j}}} g(u)g(a_{i,j})\right| \leq \sqrt{(d-1)\sum_{u\in Q\cup R} g(u)^2}\sqrt{\sum_{a_{i,j}\in M} g(a_{i,j})^2}. \quad (5)$$

We use the following lemma to bound the right hand sides of (4) and (5). The lemma is a generalized version of [16, Lemma 5.1]. The result follows from the same proof, which we reproduce for completeness. Here, for two vectors $a, b \in \mathbb{R}^n$, $a \leq b$ if $\forall i \in [n], a(i) \leq b(i)$.

▶ **Lemma 27** (Lemma 5.1 of [16]). *Consider a graph on a vertex set W, a subset X of W, a positive integer h, and $s \in L^2(W)$. Let X_i be the set of nodes at distance i from X. Assume the following conditions hold:*

(1) *For $h-1 \leq i, j \leq h$, all nodes in X_i have the same number of neighbors in X_j.*

(2) *If $u \in X_{h-1}$ and $v \in X_h$ and $u \sim v$, then $s(u)/s(v)$ does not depend on the choices of u and v.*

(3) *s is nonnegative and $As \leq \mu s$ on $\mathrm{Ball}_{h-1}(X)$, where μ is a positive real number.*

Proof. Let A be the adjacency matrix of W. Let P_{h-1} and $P_h(X)$ be the orthogonal projections onto X_{h-1} and onto X_h, respectively. Let $P_{\leq h-1}$ and $P_{\leq h}(X)$ be the orthogonal projections onto $\mathrm{Ball}_{h-1}(X)$ and $\mathrm{Ball}_h(X)$, respectively. We need to show that

$$\frac{\|P_h g\|^2}{\|P_h s\|^2} \geq \frac{\|P_{h-1} g\|^2}{\|P_{h-1} s\|^2}.$$

Call $A_h = P_{\leq h} A P_{\leq h}$ (so A_h performs the adjacency operator on $\mathrm{Ball}_h(X)$). By the conditions of the lemma, we know that there are constants α, β and γ such that

$$P_h A_h s = \gamma P_h s \quad (6)$$

and

$$A_h P_h s = \alpha P_h s + \beta P_{h-1} s. \quad (7)$$

By assumption,

$$A_h s \le \mu P_{\le h-1} s + \gamma P_h s. \tag{8}$$

Therefore by applying $P_{\le h-1}$ to both sides of (8),

$$\begin{aligned} P_{\le h-1} A_h s &\le \mu P_{\le h-1} s \\ &\le \mu P_{\le h} s - \mu P_h s. \end{aligned}$$

Now we apply A_h to both sides:

$$\begin{aligned} A_h P_{\le h-1} A_h s &\le \mu A_h s - \mu A_h P_h s \\ &\le \mu A_h s - \mu(\alpha P_h s + \beta P_{h-1} s) && \text{by (7)} \\ &\le \left(\mu^2 P_{\le h-1} + \mu(\gamma-\alpha) P_h - \mu\beta P_{h-1}\right) s. && \text{by (8)} \end{aligned}$$

Define the matrix $B := \mu^2 P_{\le h-1} + \mu(\gamma-\alpha)P_h - \mu\beta P_{h-1} - A_h P_{h-1} A_h$. B has no positive entries on the off-diagonal. Take any eigenvector ψ of B. Without loss of generality assume that ψ has a positive entry. Then take $i = \operatorname{argmax}_u \psi(u)/s(u)$. As $\psi \le (\psi(i)/s(i))s$, $(B\psi)(i) \ge (B(\psi(i)/s(i))s)(i)$. The quantity on the right is nonnegative, meaning that the eigenvalue with eigenvector ψ is nonnegative. As ψ was arbitrary, B is positive semidefinite.

Because B is positive semidefinite,

$$g^* A_h P_{\le h-1} A_h g \le g^* \left(\mu^2 P_{\le h-1} + \mu(\gamma-\alpha)P_h - \mu\beta P_{h-1}\right) g. \tag{9}$$

For any orthogonal projection P, $P^2 = P$. Therefore $g^* A_h P_{\le h-1} A_h g = \|P_{\le h-1} A_h g\|^2$. Moreover (9) becomes

$$\|P_{\le h-1} A_h g\|^2 \le \mu^2 \|P_{\le h-1} g\|^2 + \mu(\gamma-\alpha)\|P_h g\|^2 - \mu\beta\|P_{h-1} g\|^2.$$

By assumption, $\|P_{\le h} A_h g\| = \mu \|P_{\le h} g\|$. Therefore

$$(\gamma-\alpha)\|P_h g\|^2 \ge \beta \|P_{h-1} g\|^2. \tag{10}$$

As A_h and P_h are self adjoint, $s^* A_h P_h s = s^* P_h A_h s$, so $\alpha\|P_h s\|^2 + \beta\|P_{h-1}\|^2 = \gamma\|P_h s\|^2$. Combining this with (10), we obtain (11). ◀

For any $g \in L^2(W)$ such that $|Ag(u)| = \mu|g(u)|$ for $u \in \mathrm{Ball}_{h-1}(X)$, we have

$$\frac{\sum_{v\in X_h} g(v)^2}{\sum_{v\in X_h} s(v)^2} \ge \frac{\sum_{v\in X_{h-1}} g(v)^2}{\sum_{v\in X_{h-1}} s(v)^2}. \tag{11}$$

To use the lemma, we set $X_0 = U \cup V$, and h will vary from $2 \le h \le \lfloor r/2 \rfloor$. Assuming that the girth of G' is at least r, the $\lfloor r/2 \rfloor$ neighborhoods of each vertex do not overlap.

Our test vector decays exponentially, with a small adjustment.

$$s(y) = \begin{cases} \frac{1}{(d-1)^{h/2}} & y \in X_{h,U} \\ \frac{2}{\sqrt{d-1}} - \frac{1}{(d-1)^{3/2}} & y \in X_{0,V} \\ \left(\frac{2}{d-2} - \frac{2}{(d-1)(d-2)}\right)\frac{1}{(d-1)^{(h-1)/2}} & y \in X_{h,V}, h \ge 1. \end{cases}$$

For this assignment of values we have $As \le (2\sqrt{d-1})s$. In fact, this inequality is sharp at all coordinates except for $y \in X_{1,V}$.

For this s, we have that $\sum_{y\in X_h} s(y)^2$ is constant for $h = 1,\dots,\lfloor r/2\rfloor$. Also, recall $Q\cup R = X_1$ and $M = X_2$. By Lemma 27, as g corresponds to an eigenvalue $|\mu| > 2\sqrt{d-1}$, the mass on each of first 2 layers of X can only be at most $2/(r-2)$ of the total mass.

Combining (3), (4), and (5), we can bound (2) as

$$\begin{aligned}(2) &\le \frac{\gamma(2+(d-1)(d-2))d}{2n}\|g_{H'}\|^2 + \sum_{a\in X_2} g(a)^2 + \sqrt{(d-1)\sum_{u\in X_1} g(u)^2}\sqrt{\sum_{a\in X_2} g(a)^2}\\ &\le \left(\frac{\gamma(2+(d-1)(d-2))d}{2n} + (1+\sqrt{d-1})\frac{2}{r-2}\right)\|g\|^2.\end{aligned}$$

If we set $\gamma = n^{1/3}$ and $r = \frac{2}{3}\log_{d-1} n - O_n(1)$, for fixed d this becomes

$$O\left(\frac{1}{\log n}\right)\|g\|^2,$$

meaning that $\mu \le 2\sqrt{d-1} + O(1/\log n)$. This also gives the desired bounds on vertex expansion and girth, by setting $U = U_m$. Because $|V'| = (1+o_n(1))n$, the bounds on $\Psi(U_m)$, $g(G')$ and $\lambda(G')$ given in terms of n do not change when they are given in terms of m. ◀

5 Lossless Expansion of Small Sets

In this section, we prove that sufficiently small sets in a high-girth spectral expander expand losslessly.

▶ **Theorem 28** (Theorem 2 in detail). *Let G be a d-regular graph on n vertices with girth at least $2\alpha\log_{d-1} n + 4$. Then for any set S with n^κ vertices,*

$$\frac{|\Gamma(S)|}{|S|} \ge d - \lambda(G) - \frac{d^{2\kappa/\alpha}}{2} - \frac{d}{n^{1-\kappa}}.$$

Proof. Let S be a set of vertices of size n^κ in G. Let e_S denote the number of internal edges within S. Let n_i denote the number of vertices in $\Gamma(S)$ that have i edges from S incident to it. Then: $|\Gamma(S)| = n_1 + n_2 + \cdots + n_d$ and $|E(S,\Gamma(S))| = n_1 + 2n_2 + \cdots + dn_d$. Note that $|E(S,\Gamma(S))|$ is also equal to $d|S| - 2e_S$. Now consider the graph H_S on vertex set S and edge set given by induced edges on S along with new edges introduced by adding an arbitrary spanning tree for every set of i vertices that are neighbors of a vertex in $\Gamma(S)$ with exactly i neighbors in S. The number of edges in H_S is equal to

$$e_S + n_2 + 2n_3 + \cdots + (d-1)n_d = e_S + |E(S,\partial S)| - |\Gamma(S)| = d|S| - e_S - |\Gamma(S)|.$$

The edges in H_S that are not in G correspond to paths of length at most 2 in G. Therefore $g(H_S) \ge \frac{1}{2}g(G) \ge \alpha\log_{d-1} n + 2$. As a consequence of the expander mixing lemma (Lemma 10), $e_S \le \left(\lambda(G) + \frac{d|S|}{n}\right)|S|$. Consequently,

$$|E(H_S)| \ge \left(d - \lambda(G) - \frac{d|S|}{n}\right)|S| - |\Gamma(S)|,$$

which means the average degree is lower bounded by

$$2\left(d - \lambda(G) - \frac{d|S|}{n} - \frac{|\Gamma(S)|}{|S|}\right).$$

Thus by the irregular Moore bound (Lemma 11),

$$g(H_S) \leq \frac{2\log n^\kappa}{\log\left(2\left(d-\lambda(G)-\frac{d|S|}{n}-\frac{|\Gamma(S)|}{|S|}\right)-1\right)}+2$$

and hence

$$\frac{\alpha}{\log(d-1)} \leq \frac{2\kappa}{\log\left(2\left(d-\lambda(G)-\frac{d|S|}{n}-\frac{|\Gamma(S)|}{|S|}\right)-1\right)}.$$

This implies

$$d-\lambda(G)-\frac{d|S|}{n}-\frac{|\Gamma(S)|}{|S|}-\frac{1}{2} \leq \frac{d^{2\kappa/\alpha}}{2},$$

and finally by rearranging the above and plugging in $|S| = n^\kappa$

$$\frac{|\Gamma(S)|}{|S|} \geq d-\lambda(G)-\frac{d^{2\kappa/\alpha}-1}{2}-\frac{d}{n^{1-\kappa}}.$$ ◀

▶ Remark 29. If G is a n-vertex d-regular Ramanujan graph with girth $\frac{4}{3}\log_{d-1} n$ (which is a condition satisfied by the Ramanujan graphs of [18]) then for every set S of size n^κ for $\kappa < 1/3$,

$$\frac{|\Gamma(S)|}{|S|} \geq d(1-o_d(1)).$$

References

1 Noga Alon. Eigenvalues and expanders. *Combinatorica*, 6:83–96, 1986.
2 Noga Alon. Explicit expanders of every degree and size. *Combinatorica*, pages 1–17, 2021.
3 Noga Alon, Shirshendu Ganguly, and Nikhil Srivastava. High-girth near-Ramanujan graphs with localized eigenvectors. *arXiv preprint*, 2019. `arXiv:1908.03694`.
4 Noga Alon, Shlomo Hoory, and Nathan Linial. The Moore bound for irregular graphs. *Graphs and Combinatorics*, 18(1):53–57, 2002.
5 Nalini Anantharaman. Quantum ergodicity on regular graphs. *Communications in Mathematical Physics*, 353(2):633–690, 2017.
6 Nalini Anantharaman and Etienne Le Masson. Quantum ergodicity on large regular graphs. *Duke Math. J.*, 164(4):723–765, 2015.
7 Omer Angel, Joel Friedman, and Shlomo Hoory. The non-backtracking spectrum of the universal cover of a graph. *Transactions of the American Mathematical Society*, 367(6):4287–4318, 2015.
8 Charles Bordenave. A new proof of Friedman's second eigenvalue theorem and its extension to random lifts. In *Annales scientifiques de l'Ecole normale supérieure*, 2019.
9 Shimon Brooks, Etienne Le Masson, and Elon Lindenstrauss. Quantum ergodicity and averaging operators on the sphere. *International Mathematics Research Notices*, 19:6034–6064, 2016.
10 Michael Capalbo, Omer Reingold, Salil Vadhan, and Avi Wigderson. Randomness conductors and constant-degree lossless expanders. In *Proceedings of the 34th Annual ACM Symposium on Theory of Computing*, pages 659–668, 2002.
11 Joel Friedman. A proof of Alon's second eigenvalue conjecture. In *Proceedings of the thirty-fifth annual ACM symposium on Theory of computing*, pages 720–724, 2003.
12 Shirshendu Ganguly and Nikhil Srivastava. On non-localization of eigenvectors of high girth graphs. *International Mathematics Research Notices*, 2018.

13 Dima Grigoriev. Linear lower bound on degrees of positivstellensatz calculus proofs for the parity. *Theoretical Computer Science*, 259(1-2):613–622, 2001.

14 Venkatesan Guruswami, James Lee, and Alexander Razborov. Almost euclidean subspaces of ℓ_1^n via expander codes. In *Proceedings of the nineteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 353–362, 2008.

15 Shlomo Hoory, Nathan Linial, and Avi Wigderson. Expander graphs and their applications. *Bull. Amer. Math. Soc.*, 43(4):439–561, 2018.

16 Nabil Kahale. Eigenvalues and expansion of regular graphs. *Journal of the ACM (JACM)*, 42(5):1091–1106, 1995.

17 Nati Linial and Michael Simkin. A randomized construction of high girth regular graphs. *Random Structures & Algorithms*, 58(2):345–369, 2021.

18 Alexander Lubotzky, Ralph Phillips, and Peter Sarnak. Ramanujan graphs. *Combinatorica*, 8:261–277, 1988.

19 Michael Luby, Michael Mitzenmacher, M. Amin Shokrollahi, and Daniel Spielman. Improved low-density parity-check codes using irregular graphs. *IEEE Trans. Inform. Theory*, 42(2):585–598, 2001.

20 Grigorii Aleksandrovich Margulis. Explicit group-theoretical constructions of combinatorial schemes and their application to the design of expanders and concentrators. *Problemy peredachi informatsii*, 24(1):51–60, 1988.

21 Sidhanth Mohanty, Ryan O'Donnell, and Pedro Paredes. Explicit near-ramanujan graphs of every degree. In *Proceedings of the 52nd Annual ACM SIGACT Symposium on Theory of Computing*, pages 510–523, 2020.

22 Moshe Morgenstern. Existence and explicit constructions of q+ 1 regular ramanujan graphs for every prime power q. *Journal of Combinatorial Theory, Series B*, 62(1):44–62, 1994.

23 Pedro Paredes. Spectrum preserving short cycle removal on regular graphs. In *38th International Symposium on Theoretical Aspects of Computer Science, STACS 2021, March 16-19, 2021, Saarbrücken, Germany (Virtual Conference)*, volume 187 of *LIPIcs*, pages 55:1–55:19, 2021.

24 Gregory Quenell. Notes on an example of McLaughlin, 1996.

25 Grant Schoenebeck. Linear level lasserre lower bounds for certain k-csps. In *2008 49th Annual IEEE Symposium on Foundations of Computer Science*, pages 593–602. IEEE, 2008.

26 Michael Sipser and Daniel Spielman. Expander codes. *IEEE Trans. Inform. Theory*, 42(6, part 1):1710–1722, 1996.

27 Daniel Spielman. Linear-time encodable and decodable error-correcting codes. *IEEE Trans. Inform. Theory*, 42(6, part 1):1723–1731, 1996.

Relational Algorithms for k-Means Clustering

Benjamin Moseley
Carnegie Mellon University, Pittsburgh, PA, USA

Kirk Pruhs
University of Pittsburgh, PA, USA

Alireza Samadian
University of Pittsburgh, PA, USA

Yuyan Wang
Carnegie Mellon University, Pittsburgh, PA, USA

Abstract

This paper gives a k-means approximation algorithm that is efficient in the *relational algorithms model*. This is an algorithm that operates directly on a relational database without performing a join to convert it to a matrix whose rows represent the data points. The running time is potentially exponentially smaller than N, the number of data points to be clustered that the relational database represents.

Few relational algorithms are known and this paper offers techniques for designing relational algorithms as well as characterizing their limitations. We show that given two data points as cluster centers, if we cluster points according to their closest centers, it is NP-Hard to approximate the number of points in the clusters on a general relational input. This is trivial for conventional data inputs and this result exemplifies that standard algorithmic techniques may not be directly applied when designing an efficient relational algorithm. This paper then introduces a new method that leverages rejection sampling and the k-means++ algorithm to construct a $O(1)$-approximate k-means solution.

2012 ACM Subject Classification Theory of computation → Design and analysis of algorithms

Keywords and phrases k-means, clustering, approximation, big-data, databases

Digital Object Identifier 10.4230/LIPIcs.ICALP.2021.97

Category Track A: Algorithms, Complexity and Games

Related Version *Full Version*: `https://arxiv.org/abs/2008.00358`

Funding *Benjamin Moseley*: Supported in part by a Google Research Award, an Infor Research Award, a Carnegie Bosch Junior Faculty Chair and NSF grants CCF-1824303, CCF-1845146, CCF-1733873 and CMMI-1938909.
Kirk Pruhs: Supported in part by NSF grants CCF-1907673, CCF-2036077 and an IBM Faculty Award.

1 Introduction

Kaggle surveys [2] show that the majority of learning tasks faced by data scientists involve *relational data*. Conventional formats usually represent data with multi-dimensional points where each dimension corresponds to a feature of the data. In contrast, a **relational database** consists of tables $T_1, T_2, \ldots, T_m$ where the features could be stored partially in the tables. The columns in each table are a subset of features[1] and the rows are data records for

[1] In relational database context the columns are also referred to as *attributes* but here we call them features per the tradition of broader communities.

these features. The underlying data is represented by the **design matrix** $J = T_1 \bowtie \cdots \bowtie T_m$ where each row in J can be interpreted as a data point. Here the **join** ($\bowtie$) is a binary operator on two tables T_i and T_j. The result of the join is the set of all possible concatenations of two rows from T_i and T_j such that they are equal in their common columns/features. If T_i and T_j have no common columns their join is the cross product of all rows. See Table 1 for an example of join operation on two tables.

Table 1 A join of tables T_1 and T_2. Each has 5 rows and 2 features, sharing f_2. The join has all features from both tables. The rows with $f_2 = x$ in the join is the cross product of all rows with $f_2 = x$ from T_1 and T_2. For example, for $f_2 = 1$, the four rows in $T_1 \bowtie T_2$ has (f_1, f_3) values $\{(1,1),(1,2),(2,1),(2,2)\}$, this is the cross product of $f_1 \in \{1,2\}$ from T_1 and $f_3 \in \{1,2\}$ from T_2.

T_1	
f_1	f_2
1	1
2	1
3	2
4	3
5	4

T_2	
f_2	f_3
1	1
1	2
2	3
5	4
5	5

$T_1 \bowtie T_2$		
f_1	f_2	f_3
1	1	1
1	1	2
2	1	1
2	1	2
3	2	3

Almost all learning tasks are designed for data in matrix format. The current standard practice for a data scientist is the following.

Standard Practice:

1. Extract the data points from the relational database by taking the join of all tables to find the design matrix $J = T_1 \bowtie \cdots \bowtie T_m$.
2. Then interpret each row of J as a point in a Euclidean space and the columns as the dimensions, corresponding to the features of data.
3. Import this design matrix J into a standard algorithm.

A relational database is a highly compact data representation format. The size of J can be exponentially larger than the input size of the relational database [10]. Extracting J makes the standard practice inefficient. Theoretically, there is a potential for exponential speed-up by running algorithms *directly* on the tables in relational data. We call such algorithms **relational algorithms** if their running time is polynomial in the size of tables when the database is *acyclic*. Acyclic databases will be defined shortly. This leads to the following exciting algorithmic question.

The Relational Algorithm Question:

A. Which standard algorithms can be implemented as relational algorithms?

B. For standard algorithms that are *not* implementable by relational algorithms, is there an alternative efficient relational algorithm that has similar performance?

This question has recently been of interest to the community. However, few algorithmic techniques are known. Moreover, we do not have a good understanding of which problems can be solved on relational data and which cannot. Relational algorithm design has a interesting combinatorial structure that requires a deeper understanding.

We design a relational algorithm for k-means. It has a polynomial time complexity for **acyclic** relational databases. The relational database is acyclic if there exists a tree with the following properties. There is exactly one node in the tree for each table. Moreover, for any feature (i.e. column) f, let $V(f)$ be the set of nodes whose corresponding tables contain feature f. The subgraph induced on $V(f)$ must be a connected component. Acyclicity can be easily checked, as the tree can be found in polynomial time if it exists [27].

Luckily, most of the natural database schema are acyclic or nearly acyclic. Answering seemingly simple questions on general (cyclic) databases, such as if the join is empty or not is NP-Hard. For general databases, efficiency is measured in terms of the **fractional hypertree width** of the database (denoted by "fhtw"). The parameter measures how close the database structure is to being acyclic. It is 1 for acyclic databases and larger as the database is farther from being acyclic.

State-of-the-art algorithms for queries as simple as counting the number of rows in the design matrix have linear dependency on n^{fhtw} where n is the *maximum* number of rows in all input tables [7]. Running in time linear in n^{fhtw} is the goal, as fundamental barriers need to be broken to be faster. Notice that this is polynomial time when fhtw is a fixed constant (i.e. nearly acyclic). Our algorithm has linear dependency on n^{fhtw}, matching the state-of-the-art.

Relational Algorithm for k-means. k-means is perhaps the most widely used data mining algorithm (e.g. k-means is one of the few models in Google's BigQuery ML package [1]). The input to the k-means problem consists of a collection S of points in a Euclidean space and a positive integer k. A feasible output is k points $c_1, \ldots, c_k$, which we call **centers**. The objective is to choose the centers to minimize the aggregate squared distance from each original point to its nearest center.

Recall extracting all data points could take time exponential in the size of a relational database. Thus, the problem is to find the cluster centers without fully realizing all of the data points the relational data represents.

[15] was the first paper to give a non-trivial k-means algorithm that works on relational inputs. The paper gives an $O(1)$-approximation. The algorithm's running time has superlinear dependency on k^d when the tables are acyclic and thus is not polynomial. Here k is the number of cluster centers and d is the dimension (a.k.a number of features) of the points. This is equivalently the number of distinct columns in the relational database. For a small number of dimensions, this algorithm is a large improvement over the standard practice and they showed the algorithm gives up to 350x speed up on real data versus performing the query to extract the data points (not even including the time to cluster the output points).

Several questions remain. Is there a relational algorithm for k-means? What algorithmic techniques can we use as building blocks to design relational algorithms? Moreover, how can we show some problems are hard to solve using a relational algorithm?

Overview of Results. The main result of the paper is the following.

▶ **Theorem 1.** *Given an acyclic relational database with tables $T_1, T_2, \ldots T_m$ where the design matrix J has N rows and d columns. Let n be the maximum number of rows in any table. Then there is a randomized algorithm running in time polynomial in d, n and k that computes an $O(1)$ approximate k-means clustering solution with high probability.*

The discussion about the algorithm's time complexity for cyclic databases is left out due to space limits. To illustrate the challenges for finding such an algorithm as described in the prior theorem, even when the database is acyclic, consider the following theorem.

▶ **Theorem 2.** *Given an acyclic relational database with tables $T_1, T_2, \ldots T_m$ where the design matrix J has N rows and d columns. Given k centers $c_1, \ldots, c_k$, let J_i be the set of points in J that are closest to c_i for $i \in [k]$. It is #P-Hard to compute $|J_i|$ for $k \geq 2$ and NP-Hard to approximate $|J_i|$ to any factor for $k \geq 3$.*

We show the proof in Section 2.1. We prove it by reducing a NP-Hard problem to the problem of determining if J_i is empty or not. Counting points closest to a center is a fundamental building block in almost all k-means algorithms. Moreover, we note that performing one iteration of the classic Lloyd's algorithm, that is, to re-compute the centroids of all J_i's, is also $\#P$-Hard. The proof is omitted here.

Together this necessitates the design of new techniques to address the main theorem, shows that seemingly trivial algorithms are difficult relationally, and suggests computing a coreset is the right approach for the problem as it is difficult to cluster the data directly.

Overview of Techniques. We first compute a **coreset** of all points in J. That is, a collection of points with weights such that if we run an $O(1)$ approximation algorithm on this weighted set, we will get a $O(1)$ approximate solution for all of J. To do so, we sample points according to the principle in k-means++ algorithm and assign weights to the points sampled. The number of points chosen will be $\Theta(k \log N)$. Any $O(1)$-approximate weighted k-means algorithm can be used on the coreset to give Theorem 1.

k-means++. k-means++ is a well-known k-means algorithm [9, 8]. The algorithm iteratively chooses centers $c_1, c_2, \ldots$. The first center c_1 is picked uniformly from J. Given that $c_1, \ldots, c_{i-1}$ are picked, a point x is picked as c_i with probability $P(x) = \frac{L(x)}{Y}$ where $L(x) = \min_{j \in [i-1]}(\|x - c_j\|_2^2)$ and $Y = \sum_{x \in J} L(x)$. Here $[i-1]$ denotes $\{1, 2, \ldots, i-1\}$.

Say we sample $\Theta(k \log N)$ centers according to this distribution, which we call the ***k*-means++ distribution**. It was shown in [8] that if we cluster the points by assigning them to their closest centers, the total squared distance between points and their cluster centers is at most $O(1)$ times the optimal k-means cost with high probability. Note that this is not a feasible k-means solution because more than k centers are used. However, leveraging this, the work showed that we can construct a coreset by weighting these centers according to the number of points in their corresponding clusters.

We seek to mimic this approach with a relational algorithm. Let's focus on one iteration where we want to sample the center c_i given $c_1, \ldots, c_{i-1}$ according to the k-means++ distribution. Consider the assignment of every point to its closest center in $c_1, \ldots, c_{i-1}$. Notice that the k-means++ probability is determined by this assignment. Indeed, the probability of a point being sampled is the cost of assigning this point to its closest center $(\min_{j \in [i-1]} \|x - c_j\|_2^2)$ normalized by Y. Y is the summation of this cost over all points.

The relational format makes this distribution difficult to compute without the design matrix J. It is hard to efficiently characterize which points are closest to which centers. The assignment *partitions* the data points according to their closest centers, where each partition may not be easily represented by a compact relational database (unlike J).

A Relational k-means++ Implementation. Our approach will sample every point according to the k-means++ distribution without computing this distribution directly. Instead, we use **rejection sampling** [13], which allows one to sample from a "hard" distribution P using an "easy" distribution Q. Rejection sampling works by sampling from Q first, then reject the sample with another probability used to bridge the gap between Q and P. The process is repeated until a sample is accepted. In our setting, P is the k-means++ distribution, and we need to find a Q which could be sampled from efficiently with a relational algorithm (without computing J). Rejection sampling theory shows that for the sampling to be efficient, Q should be close to P point-wise to avoid high rejection frequency. In the end, we will *perfectly simulate* the k-means++ algorithm.

We now describe the intuition for designing such a Q. Recall that P is determined by the assignment of points to their closest centers. We will approximate this assignment up to a factor of $O(i^2d)$ when sampling the i^{th} center c_i, where d is the number of columns in J. Intuitively, the approximate assignment makes things easier since for any center we can easily find the points assigned to it using an efficient relational algorithm. Then Q is found by normalizing the squared distance between each point and its assigned center.

The approximate assignment is designed as follows. Consider the d-dimensional Euclidean space where the data points in J are located. The algorithm divides space into a **laminar** collection of **hyper-rectangles**[2] (i.e., $\{x \in \mathcal{R}^d : v_j \leq x_j \leq w_j, j = 1, \ldots, d\}$, here x_j is the value for feature f_j). We assign each hyper-rectangle to a center. A point assigns itself to the center that corresponds to the *smallest* hyper-rectangle containing the point.

The key property of hyper-rectangles that benefits our relational algorithm is: we can efficiently represent all points from J inside any hyper-rectangle by removing some entries in each table from the original database and taking the join of all tables. For example, if a hyper-rectangle has constraint $v_j \leq x_j \leq w_j$, we just remove all the rows with value outside of range $[v_j, w_j]$ for column f_j from the tables containing column f_j. The set of points assigned to a given center can be found by adding and subtracting a laminar set of hyper-rectangles, where each hyper-rectangle can be represented by a relational database.

Weighting the Centers. We have sampled a good set of cluster centers. In order to get a coreset we need to assign weights to them. As we have already mentioned, assuming $P \neq \#P$, the weights cannot be computed relationally. In fact, they cannot be approximated up to any factor in polynomial time unless $P = NP$. Rather, we design an alternative relational algorithm for computing the weights. Each weight will not be an approximate individually, but we prove that the weighted centers form an $O(1)$-approximate coreset in aggregate.

The main algorithmic idea is that for each center c_i we generate a collection of hyperspheres around c_i containing geometrically increasing numbers of points. The space is then partitioned using these hyperspheres where each partition contains a portion of points in J. Using the algorithm from [3], we then sample a poly-log sized collection of points from each partition, and use this subsample to estimate the fraction of the points in this partition which are closer to c_i than any other center. The estimated weight of c_i is aggregated accordingly.

Paper Organization. As relational algorithms are relatively new, we begin with some special cases which help the reader build intuition. In Section 2 we give a warm-up by showing how to implement 1-means++ and 2-means++ (i.e. initialization steps of k-means++). In this section, we also prove Theorem 2 as an example of the limits of relational algorithms. In Section 3 we go over background on relational algorithms that our overall algorithm will leverage. In Section 4 we give the k-means++ algorithm via rejection sampling. Section 5 shows an algorithm to construct the weights and then analyze this algorithm.

2 Warm-up: Efficiently Implementing 1-means++ and 2-means++

This section is a warm-up to understand the combinatorial structure of relational data. We will show how to do k-means++ for $k \in \{1, 2\}$ (referred to as 1- and 2-means++) on a simple join structure. We will also show the proof of Theorem 2 which states that counting the number of points in a cluster is a hard problem on relational data.

[2] A laminar set of hyper-rectangles means any two hyper-rectangles from the set either have no intersection, or one of them contains the other.

First, let us consider relationally implementing 1-means++ and 2-means++. For better illustration, we consider a special type of acyclic table structure named **path join**. The relational algorithm used will be generalized to work on more general join structures when we move to the full algorithm in Section 4.

In a path join each table T_i has two features/columns f_i, and f_{i+1}. Table T_i and T_{i+1} then share a common column f_{i+1}. Assume for simplicity that each table T_i contains n rows. The design matrix $J = T_1 \bowtie T_2 \bowtie \ldots \bowtie T_m$ has $d = m + 1$ features, one for each feature (i.e. column) in the tables.

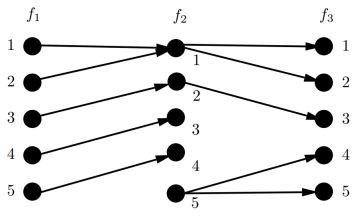
Even with this simple structure, the size of the design matrix J could still be exponential in the size of database - J could contain up to $n^{m/2}$ rows , and $dn^{m/2}$ entries. Thus the standard practice could require time and space $\Omega(mn^{m/2})$ in the worst case.

Table 2 A path join instance where the two tables T_1 and T_2 have $m = 2$ and $n = 5$. This shows T_1, T_2, the design matrix J, and the resulting layered directed graph G. *Every* path from the left most layer to the right most layer of this graph G corresponds to one data point for the clustering problem (i.e. a row of the design matrix).

T_1	
f_1	f_2
1	1
2	1
3	2
4	3
5	4

T_2	
f_2	f_3
1	1
1	2
2	3
5	4
5	5

$J = T_1 \bowtie T_2$		
f_1	f_2	f_3
1	1	1
1	1	2
2	1	1
2	1	2
3	2	3



Graph Illustration of the Design Matrix. Conceptually consider a directed acyclic graph G, where there is one layer of nodes corresponding to each feature $f_i (i = 1, \ldots, d)$, and edges only point from nodes in layer f_i to layer f_{i+1}.

The nodes in G correspond to feature values, and edges in G correspond to rows in tables. There is one vertex v in layer f_i for each value that appears in column f_i in table T_{i-1} or T_i, and one edge pointing from u in layer f_i to v in layer f_{i+1}, if (u, v) is a row in table T_i. Then, there is a one-to-one correspondence between **full paths** in G (paths from layer f_1 to layer f_d) and rows in the design matrix.

A Relational Implementation of 1-means++. Implementing the 1-means++ algorithm is equivalent to *generating a full path uniformly at random from* G. We generate this path by iteratively picking a row from table $T_1, \ldots, T_m$, corresponding to picking an arc pointing from layer f_1 to f_2, f_2 to f_3, ..., such that concatenating all picked rows (arcs) will give a point in J (full path in G).

To sample a row from T_1, for every row $r \in T_1$, consider $r \bowtie J$, which is all rows in J whose values in columns (f_1, f_2) are equivalent to r. Let the function $F_1(r)$ denote the total number of rows in $r \bowtie J$. This is also the number of full paths passing arc r. Then, every $r \in T_1$ is sampled with probability $\frac{F_1(r)}{\sum_{r' \in T_1} F_1(r')}$, notice $\sum_{r' \in T_1} F_1(r')$ is the total number of full paths. Let the picked row be r_1.

After sampling r_1, we can conceptually throw away all other rows in T_1 and focus only on the rows in J that uses r_1 to concatenate with rows from other tables (i.e., $r_1 \bowtie J$). For any row $r \in T_2$, let the function $F_2(r)$ denote the number of rows in $r \bowtie r_1 \bowtie J$, also equivalent to the total number of full paths passing arc r_1 and r. We sample every r with probability

$\frac{F_2(r)}{\sum_{r' \in T_2} F_2(r')}$. Notice that $\sum_{r' \in T_2} F_2(r') = F_1(r_1)$, the number of full paths passing arc r_1. Repeat this procedure until we have sampled a row in the last table T_m: for table T_i and $r \in T_i$, assuming we have sampled $r_1, \ldots, r_{i-1}$ from $T_1, \ldots, T_{i-1}$ respectively, throw away all the other rows in previous tables and focus on $r_1 \bowtie \ldots \bowtie r_{i-1} \bowtie J$. $F_i(r)$ is the number of rows in $r \bowtie r_1 \bowtie \ldots \bowtie r_{i-1} \bowtie J$ and r is sampled with probability proportional to $F_i(r)$. It is easy to verify that every full path is sampled uniformly.

For every table T_i we need to find the function $F_i(\cdot)$ which is defined on all its rows. There are m such functions. For each $F_i(\cdot)$, we can find all $F_i(r)$ values for $r \in T_i$ using a one-pass dynamic programming and then sample according to the values. Repeating this procedure m rounds completes the sampling process. This gives a polynomial time algorithm.

A Relational Implementation for 2-means++. Assume $x = (x_1, \ldots, x_d)$ is the first center sampled and now we want to sample the second center. By k-means++ principles, any row $r \in J$ is sampled with probability $\frac{\|r-x\|^2}{\sum_{r' \in J} \|r'-x\|^2}$. For a full path in G corresponding to a row $r \in J$ we refer to $\|r - x\|^2$ as the **aggregated cost** over all d nodes/features.

Similar to 1-means++, we pick one row in each table from T_1 to T_m and putting all the rows together gives us the sampled point. Assume we have sampled the rows $r_1, r_2, \ldots, r_{i-1}$ from the first $i-1$ tables and we focus on all full paths passing $r_1, \ldots, r_{i-1}$ (i.e., the new design matrix $r_1 \bowtie \ldots \bowtie r_{i-1} \bowtie J$). In 1-means++, we compute $F_i(r)$ which is the total number of full paths passing arc $r_1, \ldots, r_{i-1}, r$ (i.e., $r \bowtie r_1 \bowtie \ldots \bowtie r_{i-1} \bowtie J$.) and sample $r \in T_i$ from a distribution normalized using $F_i(r)$ values. In 2-means++, we define $F_i(r)$ to be the summation of aggregated costs over all full paths which pass arcs $r_1, \ldots, r_{i-1}, r$. We sample $r \in T_i$ from a distribution normalized using $F_i(r)$ values.

It is easy to verify the correctness. Again, each $F_i(\cdot)$ could be computed using a one-pass dynamic programming which gives the values for all rows in T_i when we sample from T_i. This would involve m rounds of such computations and give a polynomial relational algorithm.

2.1 Hardness of Relationally Computing the Weights

Here we prove Theorem 2. We first show that given a set of centers, counting the number of points in J that is closest to any of them is $\#P$-hard. We prove $\#P$-Hardness by a reduction from the well known $\#P$-hard Knapsack Counting problem. The input to the Knapsack Counting problem consists of a set $W = \{w_1, \ldots, w_h\}$ of nonnegative integer weights, and a nonnegative integer L. The output is the number of subsets of W with aggregate weight at most L. To construct the relational instance, for each $i \in [h]$, we define the tables T_{2i-1} and T_{2i} as follows:

T_{2i-1}	
f_{2i-1}	f_{2i}
0	0
0	w_i

T_{2i}	
f_{2i}	f_{2i+1}
0	0
w_i	0

Let centers c_1 and c_2 be arbitrary points such that points closer to c_1 than c_2 are those points p for which $\sum_{i=1}^{d} p_i \leq L$. Then there are 2^h rows in J, since w_i can either be selected or not selected in feature $2i$. The weight of c_1 is the number of points in J closer to c_1 than c_2, which is in turn exactly the number of subsets of W with total weight at most L.

Now we prove the second part of Theorem 2: given an acyclic database and a set of centers $c_1, \ldots, c_k$, it is NP-Hard to approximate the number of points assigned to each center when $k \geq 3$. We prove it by reduction from Subset Sum. In Subset Sum problem, the input

is a set of integers $A = w_1, \ldots, w_m$ and an integer L, the output is true if there is a subset of A such that its summation is L. We create the following acyclic schema. There are m tables. Each table T_i has a single unique column x_i with two rows $w_i, 0$. Then the join of the tables has 2^m rows, and it is a cross product of the rows in different tables in which each row represents one subset of A.

Then consider the following three centers: $c_1 = (\frac{L-1}{m}, \frac{L-1}{m}, \ldots, \frac{L-1}{m})$, $c_2 = (\frac{L}{m}, \ldots, \frac{L}{m})$, and $c_1 = (\frac{L+1}{m}, \frac{L+1}{m}, \ldots, \frac{L+1}{m})$. The Voronoi diagram that separates the points assigned to each of these centers consists of two parallel hyperplanes: $\sum_i x_i = L-1/2$ and $\sum_i x_i = L+1/2$ where the points between the two hyperplanes are the points assigned to c_2. Since all the points in the design matrix have integer coordinates, the only points that are between these two hyperplanes are those points for which $\sum_i x_i = L$. Therefore, the approximation for the number of points assigned to c_2 is non-zero if and only if the answer to Subset Sum is True.

3 Related Work and Background

Related Work on K-means. Constant approximations are known for the k-means problem in the standard computational setting [20, 18]. Although the most commonly used algorithm in practice is a local search algorithm called Lloyd's algorithm, or sometimes confusingly just called "the k-means algorithm". The k-means++ algorithm from [9] is a $\Theta(\log k)$ approximation algorithm, and is commonly used in practice to seed Lloyd's algorithm. Some coreset construction methods have been used before to design algorithms for the k-means problem in other restricted access computational models, including steaming [17, 12], and the MPC model [16, 11], as well as speeding up sequential methods [21, 25].

Relational Algorithms for Learning Problem. Training different machine learning models on relational data has been studied; however, many of the proposed algorithms are not efficient under our definition of a relational algorithm. It has been shown that using repeated patterns in the design matrix, linear regression, and factorization machines can be implemented [23] more efficiently. [19, 24, 5] has improved the relational linear regression and factorization machines for different scenarios. A unified relational algorithm for problems such as linear regression, singular value decomposition and factorization machines proposed in [6]. Algorithms for training support vector machine is studied in [26, 4]. In [14], a relational algorithm is introduced for Independent Gaussian Mixture Models, and they have shown experimentally that this method will be faster than materializing the design matrix.

Relational Algorithm Building Blocks. In the path join scenario, the 1- and 2-means++ sampling methods introduced in subsection 2 have similar procedures: starting with the first table T_1, iteratively evaluate some general function $F_i(\cdot)$ defined on all rows in the table T_i, sample one row r_i according to the distribution normalized from $F_i(\cdot)$. The function $F_i(\cdot)$ for table T_i is defined on the matrix $r_1 \bowtie \ldots \bowtie r_{i-1} \bowtie J$ where J is the design matrix. This matrix is also the design matrix of a new relational database, constructed by throwing away all rows in previous tables apart from the sampled $r_1, \ldots, r_{i-1}$.

We can generalize the computation of $F_i(\cdot)$ functions into a broader class of queries that we know could be implemented efficiently on *any* acyclic relational databases, namely **SumProd queries**. See [7] for more details. In the following lemmas assume the relational database has tables $T_1, \ldots, T_m$ and their design matrix is J, let n be the maximum number of rows in each table T_i, m be the number of tables and d be the number of columns in J.

▶ **Definition 3.** *For the j^{th} feature ($j \in [d]$) let $q_j : \mathbb{R} \to S$ be an efficiently computable function that maps feature values to some set S. Let the binary operations $\oplus$ and $\otimes$ be any operators such that $(S, \oplus, \otimes)$ forms a commutative semiring. The value of $\bigoplus_{x \in J} \bigotimes_{j \in [d]} q_j(x_j)$ is a SumProd query.*

▶ **Lemma 4** ([7])**.** *Any SumProd query can be computed efficiently in time $O(md^2 n^{fhtw} \log(n))$ where fhtw is the fractional hypertree width of the database. For acyclic databases fhtw=1 so the running time is polynomial.*

Despite the cumbersome formal definition of SumProd queries, below we list their key applications used in this paper. With a little abuse of notation, throughout this paper we use $\Psi(n, d, m)$ to denote the worst-case time bound on any SumProd queries.

▶ **Lemma 5.** *Given a point $y \in \mathcal{R}^d$ and a hyper-rectangle $b = \{x \in \mathcal{R}^d : v_i \leq x_i \leq w_i, i = 1, \ldots, d\}$ where v and w are constant vectors, we let $J \cap b$ denote the data points represented by rows of J that also fall into b. Pick any table T_j. Using one single SumProd query we can compute for all $r \in T_j$ the value $\sum_{p \in r \bowtie J \cap b} \|p - y\|_2^2$. The time required is at most that required by one SumProd query, $\Psi(n, d, m)$,*

Lemma 5 is intuitively based on the fact that we can efficiently represent all points in $J \cup b$ by a new relational database, which is constructed by removing some entries in each table from the original database. The following lemma follows by an application of the main result in [3].

▶ **Lemma 6** ([3])**.** *Given a hypersphere $\{x \in \mathcal{R}^d : \|x - y_0\|^2 \leq z_0^2\}$ where y_0 is a given point and z_0 is the radius, a $(1 + \epsilon)$-approximation of the number of points in J that lie inside this hypersphere could be computed in $O\left(\frac{m^6 \log^4 n}{\epsilon^2} \Psi(n, d, m)\right)$ time.*

Notice that a SumProd query could be used to output either a scalar (similar to Lemma 6) or a vector whose entries are function values for every row r in a chosen table T_j (in Lemma 5). We say the SumProd query is **grouped by** T_j in the latter case.

4 The k-means++ Algorithm

In this section, we describe a relational implementation of the k-means++ algorithm. It is sufficient to explain how center c_i is picked given the previous centers $c_1, \ldots, c_{i-1}$. Recall that the k-means++ algorithm picks a point x to be c_i with probability $P(x) = \frac{L(x)}{Y}$ where $L(x) = \min_{j \in [i-1]} \|x - c_j\|_2^2$ and $Y = \sum_{x \in J} L(x)$ is a normalizing constant.

The implementation consists of two parts. The first part, described in Section 4.1, shows how to partition the d-dimensional Euclidean space into a laminar set of hyper-rectangles (referred to as **boxes** hereafter) that are generated around the previous centers. The second part, described in Section 4.2, samples according to the "hard" distribution P using rejection sampling and an "easy" distribution Q.

Conceptually, we assign every point in the design matrix J to an *approximately* nearest center among $c_1, \ldots, c_{i-1}$. This is done by assigning every point in J to one of the centers contained in the *smallest* box this point belongs to. Then Q is derived using the squared distance between the points in J and their assigned centers.

4.1 Box Construction

Here we explain the algorithm for constructing a set of laminar boxes given the centers sampled previously. The construction is completely combinatorial. It only uses the given centers and we don't need any relational operation for the construction.

Algorithm Description. Assume we want to sample the i^{th} point in k-means++. The algorithm maintains two collections $\mathcal{G}_i$ and $\mathcal{B}_i$ of tuples. Each tuple consists of a box and a point in that box, called the **representative** of the box. This point is one of the previously sampled centers. One can think of the tuples in $\mathcal{G}_i$ as "active" ones that are subject to changes and those in $\mathcal{B}_i$ as "frozen" ones that are finalized, thus removed from $\mathcal{G}_i$ and added to $\mathcal{B}_i$. When the algorithm terminates, $\mathcal{G}_i$ will be empty, and the boxes in $\mathcal{B}_i$ will be a laminar collection of boxes that we use to define the "easy" probability distribution Q.

The initial tuples in $\mathcal{G}_i$ consist of one *unit hyper-cube* (side length is 1) centered at each previous center c_j, $j \in [i-1]$, with its representative point c_j. Up to scaling of initial unit hyper-cubes, we can assume that initially no pair of boxes in $\mathcal{G}_i$ intersect. This property of $\mathcal{G}_i$ is maintained throughout the process. Initially $\mathcal{B}_i$ is empty. Over time, the implementation keeps growing the boxes in $\mathcal{G}_i$ in size and moves tuples from $\mathcal{G}_i$ to $\mathcal{B}_i$.

The algorithm repeats the following steps in rounds. At the beginning of each round, there is no intersection between any two boxes in $\mathcal{G}_i$. The algorithm performs a doubling step where it **doubles** every box in $\mathcal{G}_i$. Doubling a box means each of its $d-1$ dimensional face is moved twice as far away from its representative. Mathematically, a box whose representative point is $y \in \mathcal{R}^d$ may be written as $\{x \in \mathcal{R}^d : y_i - v_i \le x_i \le y_i + w_i, i = 1, \ldots, d\}$ $(v_i, w_i > 0)$. This box becomes $\{x \in \mathcal{R}^d : y_i - 2v_i \le x_i \le y_i + 2w_i, i = 1, \ldots, d\}$ after doubling.

After doubling, the algorithm performs the following operations on intersecting boxes until there are none. The algorithm iteratively picks two arbitrary intersecting boxes from $\mathcal{G}_i$. Say the boxes are b_1 with representative y_1 and b_2 with representative y_2. The algorithm executes a **melding** step on (b_1, y_1) and (b_2, y_2), which has the following procedures:

- Compute the smallest box b_3 in the Euclidean space that contains both b_1 and b_2.
- Add (b_3, y_1) to $\mathcal{G}_i$ and delete (b_1, y_1) and (b_2, y_2) from $\mathcal{G}_i$.
- Check if b_1 (or b_2) is a box created by the doubling step at the beginning of the current round and hasn't been melded with other boxes ever since. If so, the algorithm computes a box b_1' (resp. b_2') from b_1 (resp. b_2) by **halving** it. That is, each $d-1$ dimensional face is moved so that its distance to the box's representative is halved. Mathematically, a box $\{x \in \mathcal{R}^d : y_i - v_i \le x_i \le y_i + w_i, i = 1, \ldots, d\}$ $(v_i, w_i > 0)$, where vector y is its representative, becomes $\{x \in \mathcal{R}^d : y_i - \frac{1}{2}v_i \le x_i \le y_i + \frac{1}{2}w_i, i = 1, \ldots, d\}$ after halving. Then (b_1', y_1) (or (b_2', y_2)) is added to $\mathcal{B}_i$. Otherwise do nothing.

Notice that melding decreases the size of $\mathcal{G}_i$.

The algorithm terminates when there is one tuple (b_0, y_0) left in $\mathcal{G}_i$, at which point the algorithm adds a box that contains the whole space with representative y_0 to $\mathcal{B}_i$. Note that during each round of the doubling and melding, the boxes which are added to $\mathcal{B}_i$ are the ones that after doubling were melded with other boxes, and they are added at their shapes before the doubling step.

▶ **Lemma 7.** *The collection of boxes in $\mathcal{B}_i$ constructed by the above algorithm is laminar.*

Proof. Note that right before each doubling step, the boxes in $\mathcal{G}_i$ are disjoint and that is because the algorithm in the previous iteration melds all the boxes that have intersection with each other. We prove by induction that at all time, for every box b in $\mathcal{B}_i$ there exist a box b' in $\mathcal{G}_i$ such that $b \subseteq b'$. Since the boxes added to $\mathcal{B}_i$ in each iteration are a subset of the boxes in $\mathcal{G}_i$ before the doubling step and they do not intersect each other, laminarity of $\mathcal{B}_i$ is a straight-forward consequence.

Initially $\mathcal{B}_i$ is empty and therefore the claim holds. Assume in some arbitrary iteration ℓ this claim holds right before the doubling step, then after the doubling step since every box in $\mathcal{G}_i$ still covers all of the area it was covering before getting doubled, the claim holds.

Furthermore, in the melding step every box b_3 that is resulted from melding of two boxes b_1 and b_2 covers both b_1 and b_2; therefore, b_3 will cover b_1 and b_2 if they are added to $\mathcal{B}_i$, and if a box in $\mathcal{B}_i$ was covered by either of b_1 or b_2, it will be still covered by b_3. ◀

The collection of boxes in $\mathcal{B}_i$ can be thought of as a tree where every node corresponds to a box. The root node is the entire space. In this tree, for any box b', among all boxes included by b', we pick the inclusion-wise *maximal* boxes and let them be the **children** of b'. Thus the number of boxes in $\mathcal{B}_i$ is $O(i)$ since the tree has i leaves, one for each center.

4.2 Sampling

To define our easy distribution Q, for any point $x \in J$, let $b(x)$ be the minimal box in $\mathcal{B}_i$ that contains x and $y(x)$ be the representative of $b(x)$. Define $R(x) = \|x - y(x)\|_2^2$, and $Q(x) = \frac{R(X)}{Z}$ where $Z = \sum_{x \in J} R(x)$ normalizes the distribution. We call $R(x)$ the **assignment cost** for x. We will show how to sample from target distribution $P(\cdot)$ using $Q(\cdot)$ and rejection sampling, and how to implement the this designed sampling step relationally.

Rejection Sampling. The algorithm repeatedly samples a point x with probability $Q(x)$, then either (A) rejects x and resamples, or (B) accepts x as the next center c_i and finishes the sampling process. After sampling x, the probability of accepting x is $\frac{L(x)}{R(x)}$, and that of rejecting x is $1 - \frac{L(x)}{R(x)}$. Notice that here $\frac{L(x)}{R(x)} \leq 1$ since $R(x) = \|x - y(x)\|_2^2 \geq \min_{j \in [i-1]} \|x - c_j\|_2^2$.

If $S(x)$ is the the event of initially sampling x from distribution Q, and $A(x)$ is the event of subsequently accepting x, the probability of choosing x to be c_i in one given round is:

$$\Pr[S(x) \text{ and } A(x)] = \Pr[A(x) \mid S(x)] \Pr[S(x)] = \frac{L(x)}{R(x)} Q(x) = \frac{L(x)}{Z}$$

Thus the probability of x being the accepted sample is proportional to $L(x)$, as desired.

We would like $Q(\cdot)$ to be close to $P(\cdot)$ point-wise so that the algorithm is efficient. Otherwise, the acceptance probability $\frac{L(x)}{R(x)}$ is low and it might keep rejecting samples.

Relational Implementation of Sampling. We now explain how to relationally sample a point x with probability $Q(x)$. The implementation heavily leverages Lemma 5, which states for given box b^* with representative y^*, the cost of assigning all points in $r \bowtie J \cap b^*$ to y^* for each row $r \in T_i$ can be computed in polynomial time using a SumProd query grouped by T_i. Recall that we assign all points in J to the representative of the smallest box they belong to. We show that the total assignment cost is computed by evaluating SumProd queries on the boxes and then adding/subtracting the query values for different boxes.

Following the intuition provided in Section 2, the implementation generates a single row from table $T_1, T_2, \ldots, T_m$ sequentially. The concatenation of these rows (or the join of them) gives the sampled point x. It is sufficient to explain assuming we have sampled $r_1, \ldots, r_{\ell-1}$ from the first $\ell - 1$ tables, how to implement the generation of a row from the next table T_ℓ. Just like 1- and 2-means++ in subsection 2, the algorithm evaluates a function $F_\ell(\cdot)$ defined on rows in T_ℓ using SumProd queries, and samples r with probability $\frac{F_\ell(r)}{\sum_{r' \in T_\ell} F_\ell(r')}$. Again, we focus on $r_1 \bowtie \ldots \bowtie r_{\ell-1} \bowtie J$, denoting the points in J that uses the previously sampled rows. The value of $F_\ell(r)$ is determined by points in $r \bowtie r_1 \bowtie \ldots \bowtie r_{\ell-1} \bowtie J$.

To ensure we generate a row according to the correct distribution Q, we define the function $F_\ell(\cdot)$ as follows. Let $F_\ell(r)$ be the total assignment cost of all points in $r \bowtie r_1 \bowtie \ldots \bowtie r_{\ell-1} \bowtie J$. That is, $F_\ell(r) = \sum_{x \in r \bowtie r_1 \bowtie \ldots \bowtie r_{\ell-1} \bowtie J} R(x)$. Notice that the definition of function $F_\ell(\cdot)$ is very similar to 2-means++ apart from that each point is no longer assigned to a given center, but the representative of the smallest box containing it.

Let $G(r, b^*, y^*)$ denote the cost of assigning all points from $r \bowtie r_1 \bowtie \ldots \bowtie r_{\ell-1} \bowtie J$ that lies in box b^* to a center y^*. By replacing the J in Lemma 5 by $r_1 \bowtie \ldots \bowtie r_{\ell-1} \bowtie J$, we can compute all $G(r, b^*, y^*)$ values in polynomial time using one SumProd query grouped by T_ℓ. The value $F_\ell(r)$ can be expanded into subtraction and addition of $G(r, b^*, y^*)$ terms. The expansion is recursive. For a box b_0, let $H(r, b_0) = \sum_{x \in r \bowtie r_1 \bowtie \ldots \bowtie r_{\ell-1} \bowtie J \cap b_0} R(x)$. Notice that $F_\ell(r) = H(r, b_0)$ if b_0 is the entire Euclidean space. Pick any row $r \in T_\ell$. Assume we want to compute $H(r, b_0)$ for some tuple $(b_0, y_0) \in \mathcal{B}_i$.

Recall that the set of boxes in $\mathcal{B}_i$ forms a tree structure. If b_0 has no children this is the base case - $H(r, b_0) = G(r, b_0, y_0)$ by definition since all points in b_0 must be assigned to y_0. Otherwise, let $(b_1, y_1), \ldots, (b_q, y_q)$ be the tuples in $\mathcal{B}_i$ where $b_1, \ldots, b_q$ are children of b_0. Notice that, by definition all points in $b_0 \setminus (\bigcup_{j \in [q]} b_j)$ is assigned to y_0. Then, one can check that the following equation holds for any r:

$$H(r, b_0) = G(r, b_0, y_0) - \sum_{j \in [q]} G(r, b_j, y_0) + \sum_{j \in [q]} H(r, b_j)$$

Starting with setting b_0 as the entire Euclidean space, the equation above could be used to recursively expand $H(\cdot, b_0) = F_\ell(\cdot)$ into addition and subtraction of $O(|\mathcal{B}_i|)$ number of $G(\cdot, \cdot, \cdot)$ terms, where each term could be computed with one SumProd query by Lemma 5.

Runtime Analysis of the Sampling. We now discuss the running time of the sampling algorithm simulating k-means++. These lemmas show how close the probability distribution we compute is as compared to the k-means++ distribution. This will help bound the running time.

▶ **Lemma 8.** *Consider the box construction algorithm when sampling the i^{th} point in the k-means++ simulation. Consider the end of the j^{th} round where all melding is finished but the boxes have not been doubled yet. Let b be an arbitrary box in $\mathcal{G}_i$ and $h(b)$ be the number of centers in b at this time. Let c_a be an arbitrary one of these $h(b)$ centers. Then:*
A. *The distance from c_a to any $d-1$ dimensional face of b is at least 2^j.*
B. *The length of each side of b is at most $h(b) \cdot 2^{j+1}$.*

Proof. The first statement is a direct consequence of the definition of doubling and melding since at any point of time the distance of all the centers in a box is at least 2^j. To prove the second statement, we define the assignment of the centers to the boxes as following. Consider the centers inside each box b right before the doubling step. We call these centers, the centers assigned to b and denote the number of them by $h'(b)$. When two boxes b_1 and b_2 are melding into box b_3, we assign their assigned centers to b_3.

We prove each side length of b is at most $h'(b)2^{j+1}$ by induction on the number j of executed doubling steps. Since $h'(b) = h(b)$ right before each doubling, this will prove the second statement. The statement is obvious in the base case, $j = 0$. The statement also obviously holds by induction after a doubling step as j is incremented and the side lengths double and the number of assigned boxes don't change. It also holds during every meld step because each side length of the newly created larger box is at most the aggregate maximum side lengths of the smaller boxes that are moved to $\mathcal{B}_i$, and the number of assigned centers in the newly created larger box is the aggregate of the assigned centers in the two smaller boxes that are moved to $\mathcal{B}_i$. Note that since for any box b all the assigned centers to b are inside b at all times, $h'(b)$ is the number of centers inside b before the next doubling. ◀

This lemma bounds the difference of the two probability distributions.

▶ **Lemma 9.** *Consider the box generation algorithm when sampling the ith point in the k-means++ simulation. For all points x, $R(x) \leq O(i^2 d) \cdot L(x)$.*

Proof. Consider an arbitrary point x. Let c_ℓ, $\ell \in [i-1]$, be the center that is closest to x under the 2-norm distance. Assume j is minimal such that just before the $(j+1)$-th doubling round, x is contained in a box b in $\mathcal{G}_i$. We argue about the state of the algorithm at two times, the time s just before doubling round j and the time t just before doubling round $j+1$. Let b be a minimal box in $\mathcal{G}_i$ that contains x at time t, and let y be the representative for box b. Notice that we assign x to the representative of the smallest box in $\mathcal{B}_i$ that contains it, so x will be assigned to y. Indeed, none of the boxes added into $\mathcal{B}_i$ before time t contains x by the minimality of j, and when box b gets added into $\mathcal{B}_i$ (potentially after a few more doubling rounds) it still has the same representative y. By Lemma 8 the squared distance from from x to r is at most $(i-1)^2 d 2^{2j+2}$. So it is sufficient to show that the squared distance from x to c_ℓ is $\Omega(2^j)$.

Let b' be the box in $\mathcal{G}_i$ that contains c_ℓ at time s. Note that x could not have been inside b' at time s by the definition of t and s. Then by Lemma 8 the distance from c_ℓ to the edge of b' at time t is at least 2^{2j-2}, and hence the distance from c_ℓ to x is also at least 2^{2j-2} as x is outside of b'. ◀

The following theorem bounds the running time.

▶ **Theorem 10.** *The expected time complexity for running k' iterations of this implementation of k-means++ is $O(k'^4 dm\Psi(n,d,m))$.*

Proof. When picking center c_i, a point x can be sampled with probability $Q(x)$ in time $O(mi\Psi(n,m,d))$. This is because the implementation samples one row from each of the m tables. To sample one row we evaluate $O(|\mathcal{B}_i|)$ SumProd queries, each in $O(\Psi(n,m,d))$ time. As mentioned earlier $\mathcal{B}_i$ can be thought of as a tree of boxes with $i-1$ leaves, so $|\mathcal{B}_i| = O(i)$.

By Lemma 9, the probability of accepting any sampled x is $\frac{L(x)}{R(x)} = \frac{1}{O(i^2 d)}$. The expected number of sampling from Q until getting accepted is $O(i^2 d)$. Thus the expected time of finding c_i is $O(i^3 dm\Psi(n,m,d))$. Summing over $i \in [k']$, we get $O(k'^4 dm\Psi(n,m,d))$. ◀

5 Weighting the Centers

Our algorithm samples a collection C of $k' = \Theta(k \log N)$ centers using the k-means++ sampling described in the prior section. We give weights to the centers to get a coreset.

Ideally, we would compute the weights in the standard way. That is, let w_i denote the number of points that are closest to point c_i among all centers in C. These pairs of centers and weights (c_i, w_i) are known to form a coreset. Unfortunately, as stated in Theorem 2, computing such w_i's even approximately is NP hard. Instead, we will find a different set of weights which still form a coreset and are computable.

Next we describe a relational algorithm to compute a collection W' of weights, one weight $w'_i \in W'$ for each center $c_i \in C$. The proof that the centers with these alternative weights (c_i, w'_i) also form a coreset is postponed until Section 6.

Algorithm for Computing Alternative Weights. Initialize the weight w'_i for each center $c_i \in C$ to zero. In the d-dimensional Euclidean space, for each center $c_i \in C$, we generate a collection of hyperspheres (also named **balls**) $\{B_{i,j}\}_{j \in [\lg N]}$, where $B_{i,j}$ contains approximately 2^j points from J. The space is then partitioned into $\{B_{i,0}, B_{i,1} - B_{i,0}, B_{i,2} - B_{i,1}, \ldots\}$. For each partition, we will sample a small number of points and use this sample to estimate

the number of points in this partition that are closer to c_i than any other centers, and thus aggregating w'_i by adding up the numbers. Fix small constants $\epsilon, \delta > 0$. The following steps are repeated for $j \in [\lg N]$:

- Let $B_{i,j}$ be a ball of radius $r_{i,j}$ centered at c_i. Find a $r_{i,j}$ such that the number of points in $J \cap B_{i,j}$ lies in the range $[(1-\delta)2^j, (1+\delta)2^j]$. This is an application of Lemma 6.
- Let τ be a constant that is at least 30. A collection $T_{i,j}$ of $\frac{\tau}{\epsilon^2} k'^2 \log^2 N$ "test" points are independently sampled following the same **approximately uniform** distribution with replacement from every ball $B_{i,j}$. Here an "approximately uniform" distribution means one where every point p in $B_{i,j}$ is sampled with a probability $\gamma_{p,i,j} \in [(1-\delta)/|B_{i,j}|, (1+\delta)/|B_{i,j}|]$ on each draw. This can be accomplished efficiently similar to the techniques used in Lemma 6 from [3]. We leave out the details due to space limit.
- Among all sampled points $T_{i,j}$, find $S_{i,j}$, the set of points that lie in the **"donut"** $D_{i,j} = B_{i,j} - B_{i,j-1}$. Then the cardinality $s_{i,j} = |S_{i,j}|$ is computed.
- Find $t_{i,j}$, the number of points in $S_{i,j}$ that are closer to c_i than any other center in C.
- Compute the ratio $f'_{i,j} = \frac{t_{i,j}}{s_{i,j}}$ (if $s_{i,j} = t_{i,j} = 0$ then $f'_{i,j} = 0$).
- If $f'_{i,j} \geq \frac{1}{2k'^2 \log N}$ then w'_i is incremented by $f'_{i,j} \cdot 2^{j-1}$, else w'_i stays the same.

At first glance the algorithm appears naive: w'_i can be significantly underestimated if in some donuts only a small portion of points are closest to c_i, making the estimation inaccurate based on sampling. However, we prove the following theorem which shows that the alternative weights computed by our algorithm actually form a coreset.

▶ **Theorem 11.** *The centers C, along with the computed weights W', form an $O(1)$-approximate coreset with high probability.*

The running time of a naive implementation of this algorithm would be dominated by sampling of the test points. Sampling a single test point can be accomplished with m applications of the algorithm from [3] and setting the approximation error to $\delta = \epsilon/m$. Recall the running time of the algorithm from [3] is $O\left(\frac{m^6 \log^4 n}{\delta^2} \Psi(n,d,m)\right)$. Thus, the time to sample all test points is $O\left(\frac{k'^2 m^9 \log^6 n}{\epsilon^4} \Psi(n,d,m)\right)$. Substituting for k', and noting that $N \leq n^m$, we obtain a total time for a naive implementation of $O\left(\frac{k^2 m^{11} \log^8 n}{\epsilon^4} \Psi(n,d,m)\right)$.

6 Analysis of the Weighting Algorithm

The goal in this subsection is to prove Theorem 11 which states that the alternative weights form an $O(1)$-approximate coreset with high probability. Throughout our analysis, "with high probability" means that for any constant $\rho > 0$ the probability of the statement not being true can be made less than $\frac{1}{N^\rho}$ asymptotically by appropriately setting the constants in the algorithm.

Intuitively, if a decent fraction of the points in each donut are closer to center c_i than any other center, then Theorem 11 can be proven by using a straight-forward application of Chernoff bounds to show that each alternate weight w'_i is likely close to the true weight w_i. The conceptual difficultly is if only a very small portion of points in a donut $D_{i,j}$ are closer to c_i than any other points, in which case the estimated $f'_{i,j} < \frac{1}{2k'^2 \log N}$ and thus the "uncounted" points in $D_{i,j}$ would contribute no weight to the computed weight w'_i. We call this the **undersampled** case. If many donuts around a center i are undersampled, the computed weight w'_i may well poorly approximate the actual weight w_i.

To address this, we need to prove that omitting the weight from these uncounted points does not have a significant impact on the objective value. We break our proof into four parts. The first part, described in subsubsection 6.1, involves conceptually defining a fractional weight w_i^f for each center $c_i \in C$. Each point has a weight of 1, and instead of giving all this weight to its closest center, we allow fractionally assigning the weight to various "near" centers. w_i^f is then the aggregated weight over all points for c_i. The second part, described in subsubsection 6.2, establishes various properties of the fractional weight that we will need. The third part, described in subsubsection 6.3, shows that each fractional weight w_i^f is likely to be closely approximated the computed weight w_i'. The fourth part, described in subsubsection 6.4, shows that the fractional weights of the centers in C form a $O(1)$-approximate coreset. Subsubsection 6.4 also contains the proof of Theorem 11.

6.1 Defining the Fractional Weights

To define the fractional weights we first define an auxiliary directed acyclic graph $G = (S, E)$ where there is one node in S corresponding to each row in J. For the rest of this section, with a little abuse of notation, we use S to denote both the nodes in graph G, and the set of d-dimensional data points in the design matrix. Let p be an arbitrary point in $S - C$. Let $\alpha(p)$ denote the subscript of the center closest to p, i.e., if $c_i \in C$ is closest to p then $\alpha(p) = i$. Let $D_{i,j}$ be the donut around c_i that contains p. If $D_{i,j}$ is not undersampled then p will have one outgoing edge (p, c_i). Therefore, let us now assume that $D_{i,j}$ is undersampled. Defining the outgoing edges from p in this case is a bit more complicated.

Let $A_{i,j}$ be the points $q \in D_{i,j}$ that are closer to c_i than any other center in C (i.e., $\alpha(q) = i$). If $j = 1$ then $D_{i,1}$ contains only the point p, and the only outgoing edge from p goes to c_i. Therefore, let us now assume $j > 1$. Let c_h the center that is closest to the most points in $D_{i,j-1}$, the next donut in toward c_i from $D_{i,j}$. That is $c_h = \arg\max_{c_j \in C} \sum_{q \in D_{i,j-1}} \mathbb{1}_{\alpha(q)=c_j}$. Let $M_{i,j-1}$ be points in $D_{i,j-1}$ that are closer to c_h than any other center. That is, $M_{i,j-1}$ is the collection of $q \in D_{i,j-1}$ such that $\alpha(q) = h$. Then there is a directed edge from p to each point in $M_{i,j-1}$. Before defining how to derive the fractional weights from G, let us take a detour to note that G is acyclic.

▶ **Lemma 12.** *G is acyclic.*

Proof. Consider a directed edge $(p, q) \in E$, and c_i be the center in C that p is closest to, and $D_{i,j}$ the donut around c_i that contains p. Then, since $p \in D_{i,j}$ it must be the case that $\|p - c_i\|_2^2 > r_{i,j-1}$. Since $q \in B_{i,j-1}$ it must be the case that $\|q - c_i\|_2^2 \leq r_{i,j-1}$. Thus $\|p - c_i\|_2^2 > \|q - c_i\|_2^2$. Thus, the closest center to q must be closer to q than the closest center to p is to p. Thus as one travels along a directed path in G, although identify of the closest center can change, the distance to the closest center must be monotonically decreasing. Thus, G must be acyclic. ◀

We explain how to compute a fractional weight w_p^f for each point $p \in S$ using the network G. Initially, each w_p^f is set to 1. Then conceptually these weights flow toward the sinks in G, splitting evenly over all outgoing edges at each vertex. More formally, the following flow step is repeated until is no longer possible to do so:

Flow Step. Let $p \in S$ be an arbitrary point that currently has positive fractional weight and that has positive outdegree h in G. Then for each directed edge (p, q) in G increment w_q^f by w_p^f/h. Finally, set w_p^f to zero.

As the sinks in G are exactly the centers in C, the centers in C will be the only points that end up with positive fractional weight. Thus, we use w_i^f to refer to the resulting fractional weight on center $c_i \in C$.

6.2 Properties of the Fractional Weights

Let $f_{i,j}$ be the fraction of points that are closest to c_i among all centers in C in this donut $D_{i,j} = B_{i,j} - B_{i,j-1}$. We show in Lemma 13 and 14 that with high probability, either the estimated ratio is a good approximation of $f_{i,j}$, or the real ratio $f_{i,j}$ is very small.

We show in Lemma 16 that the maximum flow through any node is bounded by $1+\epsilon$ when N is big enough. This follows by induction because each point has $\Omega(k' \log N)$ neighbors and every point can have in degree from one set of nodes per center. We further know every point that is not uncounted actually contributes to their centers' weight.

▶ **Lemma 13.** *With high probability, either* $|f_{i,j} - f'_{i,j}| \leq \epsilon f_{i,j}$ *or* $f'_{i,j} \leq \frac{1}{2k'^2 \log N}$.

▶ **Lemma 14.** *If* $f_{i,j} > \frac{1+\epsilon}{2k'^2 \log N}$ *then with high probability* $f'_{i,j} \geq \frac{1}{2k'^2 \log N}$.

The proofs of Lemmas 13 and 14 are omitted; see [22] for the full version of this work.

We now seek to bound the fractional weights computed by the algorithm. Let $\Delta_i(p)$ denote the total weight received by a point $p \in S \setminus C$ from other nodes (including the initial weight one on p). Furthermore, let $\Delta_o(p)$ denote the total weight sent by p to all other nodes. Notice that in the flow step $\Delta_o(p) = \Delta_i(p)$ for all p in $S \setminus C$.

▶ **Lemma 15.** *Let* $\Delta_i(p)$ *denote the total weight received by a point* $p \in S \setminus C$ *from other nodes (including the initial weight one on* p*). Furthermore, let* $\Delta_o(p)$ *denote the total weight sent by* p *to all other nodes. With high probability, for all* $q \in S$, $\Delta_i(q) \leq 1 + \frac{1+2\epsilon}{\log N} \max_{p:(p,q)\in E} \Delta_o(p)$.

Proof. Fix the point q that redirects its weight (has outgoing arcs in G). Consider its direct predecessors: $P(q) = \{p : (p, q) \in E\}$. Partition $P(q)$ as follows: $P(q) = \bigcup_{i=1,\ldots,k'} P_{c_i}(q)$, where $P_{c_i}(q)$ is the set of points that have flowed their weights into q, but c_i is actually their closest center in C. Observe the following. The point q can only belong to one donut around c_i. Due to this, $P_{c_i}(q)$ is either empty or contains a set of points in a single donut around c_i that redirect weight to q.

Fix $P_{c_i}(q)$ for some c_i. If this set is non-empty suppose this set is in the j-th donut around c_i. Conditioned on the events stated in Lemma 13 and 14, since the points in $P_{c_i}(q)$ are undersampled, we have $|P_{c_i}(q)| \leq \frac{(1+\epsilon)2^{j-1}}{2k'^2 \log N}$. Consider any $p \in P_{c_i}(q)$. Let β_i be the number of points that p charges its weight to (this is the same for all such points p). It is the case that β_i is at least $\frac{(1-\delta)2^{j-1}}{2k'}$ since p flows its weights to the points that are assigned to the center that has the most number of points assigned to it from c_i's $(j-1)$th donut.

Thus, q receives weight from $|P_{c_i}(q)| \leq \frac{(1+\epsilon)2^{j-1}}{2k'^2 \log N}$ points and each such point gives its weight to at least $\frac{(1-\delta)2^{j-1}}{2k'}$ points with equal split. The total weight that q receives from points in $P_{c_i}(q)$ is at most the following.

$$\begin{aligned} &\frac{2k'}{(1-\delta)2^{j-1}} \sum_{p \in P_{c_i}(q)} \Delta_o(p) \\ \leq &\frac{2k'}{(1-\delta)2^{j-1}} \sum_{p \in P_{c_i}(q)} \max_{p \in P_{c_i}(q)} \Delta_o(p) \end{aligned}$$

$$\leq \frac{2k'}{(1-\delta)2^{j-1}} \cdot \frac{(1+\epsilon)\cdot 2^{j-1}}{2k'^2 \log N} \max_{p\in P_{c_i}(q)} \Delta_o(p) \qquad [|P_{c_i}(q)| \leq \tfrac{(1+2\epsilon)2^{j-1}}{2k'^2 \log N}]$$
$$\leq \frac{1+2\epsilon}{k' \log N} \max_{p\in P_{c_i}(q)} \Delta_o(p) \qquad [\delta \leq \tfrac{\epsilon}{2} \leq \tfrac{1}{10}]$$

Switching the max to $\max_{p:(p,q)\in E} \Delta_o(p)$, summing over all centers $c_i \in C$ and adding the original unit weight on q gives the lemma. ◀

The following crucial lemma bounds the maximum weight that a point can receive.

▶ **Lemma 16.** *Fix η to be a constant smaller than $\frac{\log(N)}{10}$ and $\epsilon < 1$. Say that for all $q \in S \setminus C$ it is the case that $\Delta_o(q) = \eta\Delta_i(q)$. Then, with high probability for any $p \in S \setminus C$ it is the case that $\Delta_i(p) \leq 1 + \frac{2\eta}{\log N}$.*

Proof. We can easily prove this by induction on nodes. The lemma is true for all nodes that have no incoming edges in G. Now assume it is true for all nodes whose longest path that reaches them in G has length $t-1$. Now we prove it for nodes whose longest path that reaches then in G is t. Fix such a node q. For any node p such that $(p,q) \in E$, by induction we have $\Delta_i(p) \leq 1 + \frac{2\eta}{\log N}$, so $\Delta_o(p) \leq 2(1 + \frac{2\eta}{\log N})$. By Lemma 15, $\Delta_i(q) \leq 1 + \frac{1+2\epsilon}{\log N} \max_{p:(p,q)\in E} \Delta_o(p) \leq 1 + \left(\frac{\eta(1+2\epsilon)}{\log N}\right)\left(1 + \frac{2\eta}{\log N}\right) = 1 + \frac{\eta}{\log N} + \frac{\eta}{\log N} \cdot \frac{2(1+2\epsilon)\eta+2\epsilon}{\log N} \leq 1 + \frac{2\eta}{\log N}$. ◀

6.3 Comparing Alternative Weights to Fractional Weights

It only remains to bound the cost of mapping points to the centers they contribute weight to. This can be done by iteratively charging the total cost of reassigning each node with the flow. In particular, each point will only pass its weight to nodes that are closer to their center. We can charge the flow through each node to the assignment cost of that node to its closest center, and argue that the cumulative reassignment cost bounds the real fractional assignment cost. Further, each node only has $1+\epsilon$ flow going through it. This will be sufficient to bound the overall cost in Lemma 18.

▶ **Lemma 17.** *With high probability, for every center c_i, it is the case that the estimated weight w'_i computed by the weighting algorithm is $(1 \pm 2\epsilon)w_i^f$ where w_i^f is the fractional weight of i.*

The proofs of Lemmas 17 is omitted; see [22] for the full version of this work.

6.4 Comparing Fractional Weights to Optimal

Next, we bound the total cost of the fractional assignment defined by the flow. According to the graph G, any point $p \in S$ and $c_i \in C$, we let $\omega(p, c_i)$ be the fraction of weights that got transferred from p to c_i. Naturally we have $\sum_{c_i\in C} \omega(p, c_i) = 1$ for any $p \in S$ and the fractional weights $w_i^f = \sum_{p\in S} \omega(p, c_i)$ for any $c_i \in C$.

▶ **Lemma 18.** *Let ϕ_{opt} be the optimal k-means cost on the original set S. With high probability, it is the case that:*

$$\sum_{p\in S} \sum_{c_i\in C} \omega(p, c_i)\|p - c_i\|^2 \leq 160(1+\epsilon)\phi_{opt}$$

Proof. Let $\phi^* = \sum_{p \in S} \|p - c_{\alpha(p)}\|^2$. Consider any $p \in S$ and center c_i such that $\omega(p, c_i) > 0$. Let P be any path from p to c_i in G. If node p's only outgoing arc is to its closest center $c_{\alpha(p)} = c_i$, then $P = p \to c_i$, we have $\sum_{c \in C} \omega(p, c)\|p - c\|^2 = \|p - c_{\alpha(p)}\|^2$. Otherwise assume $P = p \to q_1 \to q_2 \to \ldots \to q_\ell \to c_i$. Note that the closest center to q_ℓ is c_i. Let $\Delta(P)$ be the fraction of the original weight of 1 on p that is given to c_i along this path according to the flow of weights. As we observed in the proof of Lemma 12, we have $\|p - c_{\alpha(p)}\| > \|q_1 - c_{\alpha(p)}\| \geq \|q_1 - c_{\alpha(q_1)}\| > \|q_2 - c_{\alpha(q_1)}\| \geq \|q_2 - c_{\alpha(q_2)}\| > \ldots > \|q_\ell - c_{\alpha(q_\ell)}\|$. This follows because for any arc (u, v) in the graph, v is in a donut closer to $c_{\alpha(u)}$ than the donut u is in, and v is closer to $c_{\alpha(v)}$ than $c_{\alpha(u)}$.

We use the relaxed triangle inequality for squared ℓ_2 norms. For any three points x, y, z, we have $\|x - z\|^2 \leq 2(\|x - y\|^2 + \|y - z\|^2)$. Thus, we bound $\|p - c_i\|^2$ by

$$\begin{aligned}
\|p - c_i\|^2 &= \|p - c_{\alpha(p)} + c_{\alpha(p)} - q_1 + q_1 - c_i\|^2 \\
&\leq 2\|p - c_{\alpha(p)} + c_{\alpha(p)} - q_1\|^2 + 2\|q_1 - c_i\|^2 && \text{[relaxed triangle inequality]} \\
&\leq 2(\|p - c_{\alpha(p)}\| + \|c_{\alpha(p)} - q_1\|)^2 + 2\|q_1 - c_i\|^2 && \text{[triangle inequality]} \\
&\leq 8\|p - c_{\alpha(p)}\|^2 + 2\|q_1 - c_i\|^2 && [\|p - c_{\alpha(p)}\| \geq \|c_{\alpha(p)} - q_1\|].
\end{aligned}$$

Applying the prior steps to each q_i gives the following.

$$\|p - c_i\|^2 \leq 8(\|p - c_{\alpha(p)}\|^2 + \sum_{j=1}^{\ell} 2^j \|q_j - c_{\alpha(q_j)}\|^2)$$

Let $\mathcal{P}_q(j)$ be the set of all paths P that reach point q using j edges. If $j = 0$, it means P starts with point q. We seek to bound $\sum_{j=0}^{\infty} 2^j \sum_{P \in \mathcal{P}_q(j)} \Delta(P)\|q - c_{\alpha(q_j)}\|^2$. This will bound the charge on point q above over all path P that contains it.

Define a weight function $\Delta'(p)$ for each node $p \in S \setminus C$. This will be a new flow of weights like Δ, except now the weight increases at each node. In particular, give each node initially a weight of 1. Let $\Delta'_o(p)$ be the total weight leaving p. This will be evenly divided among the nodes that have outgoing edges from p. Define $\Delta'_i(p)$ to be the weight incoming to p from all other nodes plus one, the initial weight of p. Set $\Delta'_o(p)$ to be $2\Delta'_i(p)$, twice the incoming weight.

Lemma 16 implies that the maximum weight of any point p is $\Delta'_i(p) \leq 1 + \frac{4}{\log N}$. Further notice that for any q it is the case that $\Delta'_i(q) = \sum_{j=0}^{\infty} 2^j \sum_{P \in \mathcal{P}_q(j)} \Delta(P)$. Letting $\mathcal{P}(p, c_i)$ be the set of all paths that start at p to center c_i. Notice such paths correspond to how p's unit weight goes to c_i. We have $\omega(p, c_i) = \sum_{P \in \mathcal{P}(p, c_i)} \Delta(P)$. Let $\mathcal{P}$ denote the set of all paths, $\ell(P)$ denote the length of path P (number of edges on P) , and let $P(j)$ denote the jth node on path P. Thus we have the following.

$$\begin{aligned}
&\sum_{p \in S} \sum_{c_i \in C} \omega(p, c_i)\|p - c_i\|^2 = \sum_{p \in S} \sum_{c_i \in C} \sum_{P \in \mathcal{P}(p, c_i)} \Delta(P)\|p - c_i\|^2 \\
\leq\ & 8 \sum_{p \in S} \sum_{c_i \in C} \sum_{P \in \mathcal{P}(p, c_i)} \Delta(P)\Big(\sum_{j=0}^{\ell(p)-1} 2^j \|P(j) - c_{\alpha(P(j))}\|^2\Big) \\
=\ & 8 \sum_{P \in \mathcal{P}} \Delta(P)\Big(\sum_{j=0}^{\ell(p)-1} 2^j \|P(j) - c_{\alpha(P(j))}\|^2\Big) \\
=\ & 8 \sum_{q \in S} \sum_{j=0}^{+\infty} \sum_{P \in \mathcal{P}_q(j)} 2^j \Delta(P)\|q - c_{\alpha(q)}\|^2 = 8 \sum_{q \in S} \Delta'_i(q)\|q - c_{\alpha(q)}\|^2 \\
\leq\ & \sum_{q \in S} 8(1 + \frac{4}{\log N})\|q - c_{\alpha(q)}\|^2 = 8(1 + \frac{4}{\log N})\phi^*
\end{aligned}$$

Lemma 18 follows because if $k' \geq 1067k \log N$, $\phi^* \leq 20\phi_{opt}$ with high probability by Theorem 1 in [8]. ◀

Finally, we prove that finding any $O(1)$-approximation solution for optimal weighted k-means on the set (C, W') gives a constant approximation for optimal k-means for the original set S. Let $W^f = \{w_1^f, \ldots, w_{k'}^f\}$ be the fractional weights for centers in C. Let $\phi^*_{W^f}$ denote the optimal weighted k-means cost on (C, W^f), and $\phi^*_{W'}$ denote the optimal weighted k-means cost on (C, W'). We first prove that $\phi^*_{W^f} = O(1)\phi_{\text{OPT}}$, where ϕ_{OPT} denote the optimal k-means cost on set S.

▶ **Lemma 19.** *Let (C, W^f) be the set of points sampled and the weights collected by fractional assignment ω. With high probability, we have $\phi^*_{W^f} = O(1)\phi_{\text{OPT}}$.*

Proof. Consider the cost of the fractional assignment we've designed. For $c_i \in C$, the weight is $w_i^f = \sum_{p\in S} \omega(p, c_i)$. Denote the k-means cost of ω by $\phi_\omega = \sum_{p\in S}\sum_{c\in C} \omega(p, c)\|p - c\|^2$. By Lemma 18, we have that $\phi_\omega \leq 160(1 + \epsilon)\phi_{\text{OPT}}$.

Intuitively, in the following we show $\phi^*_{W^f}$ is close to ϕ_ω. As always, we let C_{OPT} denote the optimal centers for k-means on set S. For a set of points X with weights $Y : X \to \mathbb{R}^+$ and a set of centers Z, we let $\phi_{(X,Y)}(Z) = \sum_{x\in X} Y(x) \min_{z\in Z} \|x - z\|^2$ denote the cost of assigning the weighted points in X to their closest centers in Z. Note that $\phi^*_{W^f} \leq \phi_{(C,W^f)}(C_{\text{OPT}})$ since C_{OPT} is chosen with respect to S.

$$\begin{aligned}
\phi^*_{W^f} \leq \phi_{(C,W^f)}(C_{\text{OPT}}) &= \sum_{c_i\in C}\sum_{p\in S} \min_{c\in C_{\text{OPT}}} \omega(p, c_i)\|c_i - c\|^2 && [w_i^f = \textstyle\sum_{p\in S}\omega(p, c_i)] \\
&\leq \sum_{c_i\in C}\sum_{p\in S} \min_{c\in C_{\text{OPT}}} \omega(p, c_i) \cdot 2(\|p - c_i\|^2 + \|p - c\|^2) && [\text{relaxed triangle inequality}] \\
&= 2\phi_\omega + 2\phi_{\text{OPT}} \leq 322(1 + \epsilon)\phi_{\text{OPT}} &&
\end{aligned}$$ ◀

Using the mentioned lemmas, we can prove the final approximation guarantee.

Proof of Theorem 11. Using Lemma 17, we know $w'_i = (1 \pm 2\epsilon)w_i^f$ for any center c_i. Let C'_k be k centers for (C, W') that is a γ-approximate for optimal weighted k-means. Let C^f_{OPT} be the *optimal* k centers for (C, W^f), and C'_{OPT} optimal for (C, W'). We have $\phi_{(C,W^f)}(C'_k) \leq (1 + 2\epsilon)\phi_{(C,W')}(C'_k)$ for the reason that the contribution of each point grows by at most $(1+2\epsilon)$ due to weight approximation. Using the same analysis, $\phi_{(C,W')}(C^f_{\text{OPT}}) \leq (1+2\epsilon)\phi^*_{W^f}$. Combining the two inequalities, we have

$$\begin{aligned}
\phi_{(C,W^f)}(C'_k) &\leq (1 + 2\epsilon)^2\phi_{(C,W')}(C'_k) \leq (1 + 2\epsilon)^2\gamma\phi^*_{W'} && \\
&\leq (1 + 2\epsilon)^2\gamma\phi_{(C,W')}(C^f_{\text{OPT}}) && [\text{by optimality of } \phi^*_{W'}] \qquad (1) \\
&\leq (1 + 2\epsilon)^3\gamma\phi^*_{W^f} \leq 322\gamma(1 + 2\epsilon)^4\phi_{\text{OPT}} && [\text{using Lemma 19}]
\end{aligned}$$

Let $\phi_S(C'_k) = \sum_{p\in S} \min_{c\in C'_k} \|p - c\|^2$. For every point $p \in S$, to bound its cost $\min_{c\in C'_k} \|p - c\|^2$, we use multiple relaxed triangle inequalities for every center $c_i \in C$, and take the weighted average of them using $\omega(p, c_i)$.

$$\begin{aligned}
\phi_S(C'_k) &= \sum_{p\in S} \min_{c\in C'_k} \|p - c\|^2 = \sum_{p\in S}\sum_{c_i\in C} \omega(p, c_i) \min_{c\in C'_k} \|p - c\|^2 && [\textstyle\sum_{c_i\in C}\omega(p, c_i) = 1] \\
&\leq \sum_{p\in S}\sum_{c_i\in C} \omega(p, c_i) \min_{c\in C'_k} 2(\|p - c_i\|^2 + \|c_i - c\|^2) && [\text{relaxed triangle inequality}] \\
&= 2\phi_\omega + 2\phi_{(C,W^f)}(C'_k) && [\textstyle\sum_{p\in S}\omega(p, c_i) = w_i^f] \\
&\leq 2\phi_\omega + 2 \cdot 322\gamma(1 + 2\epsilon)^4\phi_{\text{OPT}} && [\text{inequality (1)}] \\
&\leq 2 \cdot 160(1 + \epsilon)\phi_{\text{OPT}} + 2 \cdot 322\gamma(1 + 2\epsilon)^4\phi_{\text{OPT}} && [\text{Lemma 18}] \\
&= O(\gamma)\phi_{\text{OPT}} &&
\end{aligned}$$ ◀

References

1 https://cloud.google.com/bigquery-ml/docs/bigqueryml-intro.

2 Kaggle machine learning and data science survey. https://www.kaggle.com/kaggle/kaggle-survey-2018, 2018.

3 Mahmoud Abo-Khamis, Sungjin Im, Benjamin Moseley, Kirk Pruhs, and Alireza Samadian. Approximate aggregate queries under additive inequalities. In *Symposium on Algorithmic Principles of Computer Systems (APOCS)*, pages 85–99. SIAM, 2021.

4 Mahmoud Abo-Khamis, Sungjin Im, Benjamin Moseley, Kirk Pruhs, and Alireza Samadian. A relational gradient descent algorithm for support vector machine training. In *Symposium on Algorithmic Principles of Computer Systems (APOCS)*, pages 100–113. SIAM, 2021.

5 Mahmoud Abo Khamis, Hung Q Ngo, XuanLong Nguyen, Dan Olteanu, and Maximilian Schleich. Ac/dc: in-database learning thunderstruck. In *Second Workshop on Data Management for End-To-End Machine Learning*, page 8. ACM, 2018.

6 Mahmoud Abo Khamis, Hung Q. Ngo, XuanLong Nguyen, Dan Olteanu, and Maximilian Schleich. In-database learning with sparse tensors. In *ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems*, pages 325–340, 2018.

7 Mahmoud Abo Khamis, Hung Q. Ngo, and Atri Rudra. Faq: Questions asked frequently. In *Proceedings of the 35th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems*, PODS '16, page 13–28, New York, NY, USA, 2016. Association for Computing Machinery. doi:10.1145/2902251.2902280.

8 Ankit Aggarwal, Amit Deshpande, and Ravi Kannan. Adaptive sampling for k-means clustering. In *International Conference on Approximation Algorithms for Combinatorial Optimization Problems*, pages 15–28. Springer, 2009.

9 David Arthur and Sergei Vassilvitskii. k-means++: the advantages of careful seeding. In *ACM-SIAM Symposium on Discrete Algorithms*, pages 1027–1035, 2007.

10 Albert Atserias, Martin Grohe, and Dániel Marx. Size bounds and query plans for relational joins. In *IEEE Symposium on Foundations of Computer Science*, pages 739–748, 2008.

11 Bahman Bahmani, Benjamin Moseley, Andrea Vattani, Ravi Kumar, and Sergei Vassilvitskii. Scalable k-means++. *Proceedings of the VLDB Endowment*, 5(7):622–633, 2012.

12 Vladimir Braverman, Gereon Frahling, Harry Lang, Christian Sohler, and Lin F. Yang. Clustering high dimensional dynamic data streams. In *International Conference on Machine Learning*, pages 576–585, 2017.

13 George Casella, Christian P Robert, Martin T Wells, et al. Generalized accept-reject sampling schemes. In *A Festschrift for Herman Rubin*, pages 342–347. Institute of Mathematical Statistics, 2004.

14 Zhaoyue Cheng and Nick Koudas. Nonlinear models over normalized data. In *2019 IEEE 35th International Conference on Data Engineering (ICDE)*, pages 1574–1577. IEEE, 2019.

15 Ryan R. Curtin, Benjamin Moseley, Hung Q. Ngo, XuanLong Nguyen, Dan Olteanu, and Maximilian Schleich. Rk-means: Fast clustering for relational data. In Silvia Chiappa and Roberto Calandra, editors, *The 23rd International Conference on Artificial Intelligence and Statistics, AISTATS 2020, 26-28 August 2020, Online [Palermo, Sicily, Italy]*, volume 108 of *Proceedings of Machine Learning Research*, pages 2742–2752. PMLR, 2020.

16 Alina Ene, Sungjin Im, and Benjamin Moseley. Fast clustering using mapreduce. In *SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 681–689, 2011.

17 Sudipto Guha, Adam Meyerson, Nina Mishra, Rajeev Motwani, and Liadan O'Callaghan. Clustering data streams: Theory and practice. *IEEE Transactions on Knowledge and Data Engineering*, 15(3):515–528, 2003.

18 Tapas Kanungo, David M. Mount, Nathan S. Netanyahu, Christine D. Piatko, Ruth Silverman, and Angela Y. Wu. A local search approximation algorithm for k-means clustering. *Computational Geometry*, 28(2-3):89–112, 2004.

19 Arun Kumar, Jeffrey Naughton, and Jignesh M. Patel. Learning generalized linear models over normalized data. In *ACM SIGMOD International Conference on Management of Data*, pages 1969–1984, 2015.

20 Shi Li and Ola Svensson. Approximating k-median via pseudo-approximation. *SIAM J. Comput.*, 45(2):530–547, 2016. doi:10.1137/130938645.

21 Adam Meyerson, Liadan O'Callaghan, and Serge A. Plotkin. A k-median algorithm with running time independent of data size. *Machine Learning*, 56(1-3):61–87, 2004.

22 Benjamin Moseley, Kirk Pruhs, Alireza Samadian, and Yuyan Wang. Relational algorithms for k-means clustering, 2020. arXiv:2008.00358.

23 Steffen Rendle. Scaling factorization machines to relational data. In *Proceedings of the VLDB Endowment*, volume 6(5), pages 337–348. VLDB Endowment, 2013.

24 Maximilian Schleich, Dan Olteanu, and Radu Ciucanu. Learning linear regression models over factorized joins. In *Proceedings of the 2016 International Conference on Management of Data*, SIGMOD '16, pages 3–18. ACM, 2016.

25 Christian Sohler and David P. Woodruff. Strong coresets for k-median and subspace approximation: Goodbye dimension. In *Symposium on Foundations of Computer Science*, pages 802–813, 2018.

26 Keyu Yang, Yunjun Gao, Lei Liang, Bin Yao, Shiting Wen, and Gang Chen. Towards factorized svm with gaussian kernels over normalized data.

27 Clement Tak Yu and Meral Z Ozsoyoglu. An algorithm for tree-query membership of a distributed query. In *Computer Software and The IEEE Computer Society's Third International Applications Conference*, pages 306–312. IEEE, 1979.

Testing Dynamic Environments: Back to Basics

Yonatan Nakar
Tel Aviv University, Israel

Dana Ron
Tel Aviv University, Israel

Abstract

We continue the line of work initiated by Goldreich and Ron (*Journal of the ACM, 2017*) on testing dynamic environments and propose to pursue a systematic study of the complexity of testing basic dynamic environments and local rules. As a first step, in this work we focus on dynamic environments that correspond to elementary cellular automata that evolve according to threshold rules.

Our main result is the identification of a set of conditions on local rules, and a meta-algorithm that tests evolution according to local rules that satisfy the conditions. The meta-algorithm has query complexity $\text{poly}(1/\epsilon)$, is non-adaptive and has one-sided error. We show that all the threshold rules satisfy the set of conditions, and therefore are $\text{poly}(1/\epsilon)$-testable. We believe that this is a rich area of research and suggest a variety of open problems and natural research directions that may extend and expand our results.

2012 ACM Subject Classification Theory of computation → Streaming, sublinear and near linear time algorithms

Keywords and phrases Property Testing

Digital Object Identifier 10.4230/LIPIcs.ICALP.2021.98

Category Track A: Algorithms, Complexity and Games

Related Version *Full Version*: https://arxiv.org/abs/2105.00759

1 Introduction

Property testing [12, 5] is the study of algorithms that distinguish between objects that have a given property and those that are far from having the property, by performing a small number of queries to the object. Goldreich and Ron [6] initiated the study of testing *dynamic environments*, which introduces a temporal aspect to property testing. In this context, the entity being tested changes with time, and is referred to as an *environment*.

Starting from some initial *configuration* (say, a vector or a matrix), the environment is supposed to evolve according to a prespecified *local* rule. The rule is local in the sense that the value associated with each location in the environment at time t is determined by the values of nearby locations at time $t-1$. The goal of the testing algorithm is then to distinguish between the case that the environment indeed evolves according to the rule, and the case in which the evolution significantly strays from obeying the rule. To this end, the algorithm can query the value of any location of the environment at any of the available time steps, as long as it does not "go back in time". Namely, the algorithm cannot choose to query a location at time t after it has queried some location at time $t' > t$. We refer to this as the *time-conforming* requirement. The aim is to design time-conforming algorithms with low query complexity.

Goldreich and Ron [6] investigate the complexity landscape of testing dynamic environments from multiple angles. From a hardness perspective, they show that there are dynamic environments whose testing requires high query complexity and running time, and that adaptivity and time-conformity are relevant constraints which can significantly impact the query complexity. However, as we discuss in Section 1.4, relatively little is known regarding positive results for testing specific rules.

In our quest for understanding which natural families of dynamic environments can be tested efficiently, we propose to first "go back to the basics" and study testing in the simplest of dynamic environments. Namely, in this work we consider environments defined by one-dimensional configurations, which evolve according to local rules that are functions of the current location and its two immediate neighbors. These dynamic environments, originally introduced by von Neumann [13], have been extensively studied under the name of *Elementary Cellular Automata* [14] (see definition in Section 1.1). While these environments can be described in simple terms, they are nevertheless able to capture complex behavior.[1] Cellular automata have played a role in various research fields and applications. Examples include modeling physical [2] and chemical [7] systems, VLSI design [3], music generation [1], analyzing plant population dynamics [15], forest fire spread [17], city traffic [11], urban sprawl [8], and more.

As we discuss in Section 1.4, there are several hardness results (both regarding the query complexity and the running time) for testing dynamic environments that correspond to one-dimensional cellular automata (over non-binary alphabets) [6]. Hence, in order to obtain efficient algorithms, it is necessary to restrict the rules considered. In the current work, our main focus is on perhaps the most basic and natural rules, defined by threshold functions. Such functions have received much attention within the study of propagation of information/influence in networks (see, e.g., the review paper of Peleg [10], and the recent Ph.D. thesis of Zehmakan [16] and references within).

Our testers are based on a general meta-algorithm which works for rules that satisfy a set of conditions that we define. In essence, the conditions capture a certain type of behavior leading to ultimate convergence. This behavior induces a global structure on the environment which we exploit in our meta-algorithm.

We hope this work can serve as a basis for further extensions and generalizations, some of which we discuss shortly in Section 1.5.

1.1 Testing basic evolution rules

We now formally define the problems we study. We use $[m]$ to denote the set $\{0, ..., m-1\}$. For two integers n and m, let $\texttt{ENV} : [m] \times \mathbb{Z}_n \to \{0,1\}$ denote the evolving environment, and for any $t \in [m]$ let $\texttt{ENV}_t : \mathbb{Z}_n \to \{0,1\}$ (the environment at time t) be defined by $\texttt{ENV}_t(i) = \texttt{ENV}(t,i)$. In general, we refer to a function $\sigma : \mathbb{Z}_n \to \{0,1\}$ as a *configuration*. When convenient, we may view σ as a (cyclic) binary string of length n.

For a function (evolution rule) $\rho : \{0,1\}^3 \to \{0,1\}$, we say that $\texttt{ENV}$ *evolves according to* ρ, if for every $i \in \mathbb{Z}_n$ and $t > 0$, we have that $\texttt{ENV}_t(i) = \rho(\texttt{ENV}_{t-1}(i-1), \texttt{ENV}_{t-1}(i), \texttt{ENV}_{t-1}(i+1))$, where all operations are modulo n. We use $\mathcal{E}^{\rho}_{m,n}$ to denote the set of environments $\texttt{ENV} : [m] \times \mathbb{Z}_n \to \{0,1\}$ that evolve according to ρ. As in [6], we employ the standard notion of distance used in property testing and say that $\texttt{ENV} : [m] \times \mathbb{Z}_n \to \{0,1\}$ is ϵ-*far from evolving according to* ρ (ϵ-far from $\mathcal{E}^{\rho}_{m,n}$) if $|\{(t,i) : \texttt{ENV}(t,i) \neq \texttt{ENV}'(t,i)\}| > \epsilon mn$ for every $\texttt{ENV}' \in \mathcal{E}^{\rho}_{m,n}$.[2]

[1] Some rules are even Turing complete [4].

[2] In the context of dynamic environments, this notion of distance can be interpreted as capturing "measurement errors" due to some noise process. Namely, it can be viewed as allowing the testing algorithm to accept not only "perfect" environments, but also environments that correspond to a correct evolution with a bounded fraction of corruptions. Also note that being ϵ-far from evolving according to ρ does not simply translate to there being an ϵ-fraction of pairs (t,i) for which $\texttt{ENV}_t(i) \neq \rho(\texttt{ENV}_{t-1}(i-1), \texttt{ENV}_{t-1}(i), \texttt{ENV}_{t-1}(i+1))$ (which would be trivial to test).

Given n, m, and a distance parameter $\epsilon \in (0,1)$, a testing algorithm for evolution according to a rule ρ should distinguish with constant success probability between the case that an environment ENV belongs to $\mathcal{E}^{\rho}_{m,n}$ and the case that it is ϵ-far from $\mathcal{E}^{\rho}_{m,n}$. To this end, the algorithm is given query access to ENV, where a query on a pair (t,i) cannot follow any query on (t',i') for $t' > t$. We are interested in bounding both the total number of queries performed by the algorithm (as a function of ϵ, and possibly m and n) and the maximum number of queries it performs at any time step (which we refer to as its *temporal query complexity*).

1.2 Our results

We identify several conditions on local rules (which are formally stated in Section 3), such that if a rule ρ satisfies these conditions, then evolution according to ρ can be tested with query complexity $\text{poly}(1/\epsilon)$ with one-sided error. Our testers have the advantage that they are non-adaptive, and therefore, in particular, time-conforming.

▶ **Theorem 1.** *Let Ψ be the set of conditions specified in Section 3. For every rule ρ that satisfies the conditions in Ψ, it is possible to test evolution according to ρ by performing $O(1/\epsilon^4)$ queries. Furthermore, the testing algorithm is non-adaptive and has one-sided error.*

To establish Theorem 1, we present a *meta-algorithm* for testing evolution and prove its correctness for rules that satisfy the aforementioned conditions (the set Ψ). It is a meta-algorithm in the sense that it is based on certain subroutines that are rule-specific (but have a common functionality of detecting violations of evolution according to the tested rule). We provide a high-level discussion of the conditions and the algorithm in Section 1.3.

Our main application of the meta-algorithm is to the natural family of threshold rules.

▶ **Definition 1.** *We say that a rule $\rho : \{0,1\}^3 \to \{0,1\}$ is a* threshold *rule if there exist a threshold integer $0 \leq b \leq 3$ and a bit $\alpha \in \{0,1\}$ such that $\rho(\beta_1,\beta_2,\beta_3) = \alpha$ if and only if $\beta_1 + \beta_2 + \beta_3 \geq b$.*

We prove:

▶ **Theorem 2.** *For each threshold rule ρ, evolution according to ρ can be tested with query complexity $O(1/\epsilon^4)$. Furthermore, the testing algorithm is non-adaptive and has one-sided error.*

We also show that the conditions hold for two additional (non-threshold) rules, so the applicability of our meta-algorithm is more general (for details, see full version of this paper [9]). We believe that appropriate (perhaps more complex) variants of our algorithm can be used to test an even larger variety of basic local rules (see Section 1.5), where we conjecture that this is true for all rules that ultimately converge. Interestingly, while the two additional rules are not threshold rules as per Definition 1, they can be represented as weighted threshold rules (which are a subclass of ultimately converging rules).

1.3 The high-level ideas behind our results

In this high-level discussion, we assume for simplicity that $m \geq n$ (the case $m < n$ can be essentially reduced to this case).

1.3.1 Convergence, final/non-final locations and prediction functions

To give an intuition on the convergence behavior that our conditions capture, it is useful to first discuss the notion of *ultimate convergence*. A rule ρ ultimately converges if, for any initial configuration $\texttt{ENV}_0$, an environment evolving according to ρ converges after a bounded number of steps to either a single final configuration or to a constant number of configurations between which it alternates. For example, consider the majority rule (threshold 2). Unless the initial configuration is $(01)^{n/2}$, the environment ultimately converges to some configuration that consists of blocks of 0s and 1s of size at least 2 each (and if it is $(01)^{n/2}$, then it alternates between $(01)^{n/2}$ and $(10)^{n/2}$).

Once an environment converges, testing is straightforward since we can easily predict the values of locations in future time steps and then verify that indeed they hold the predicted values (or else we reject). The issue, however, is that convergence is not ensured to be reached after a small number of time steps.[3] In other words, knowing that a rule ultimately converges cannot be exploited directly. Hence, the challenge is to identify and formalize conditions that allow for "pre-convergence prediction". Namely, conditions that imply the ability to predict future values of locations based on the current values of these and other locations (before convergence is reached).

In this context, our conditions try to formalize the idea that rules exhibit a certain *local* convergence, which "expands" with time. The first ingredient of our approach is the observation that, in the case of the majority rule, if at any time step t, $\texttt{ENV}_t(i) \in \{\texttt{ENV}_t(i-1), \texttt{ENV}_t(i+1)\}$, then $\texttt{ENV}_{t'}(i) = \texttt{ENV}_t(i)$ for any $t' > t$ (operations are modulo n). We say in such a case that location i is *final* at time t (in $\texttt{ENV}$). Otherwise it is *non-final*. Crucially for us, whether a location i is final or not at a certain time step depends solely on its local neighborhood at that time (and can hence be verified with a constant number of queries).

An important property of a location being final at time t (in addition to converging to their final value, up to alternations), is the aforementioned expansion (or "transmission of finality"). Namely, a location i that is non-final at time t becomes final at time $t+1$ if either $i-1$ or $i+1$ is final at time t (possibly both). Furthermore, it cannot become final if both its neighbors are non-final. Another related property of final locations is that (under certain circumstances), they can be used to predict the values of locations that become final in the future, based on a (rule-specific) *prediction function*. A similar statement holds for non-final locations (though the circumstances are different).

1.3.2 The meta-algorithm: the grid and violating pairs

Based on these properties (which are formalized in the conditions we introduce), our (meta) algorithm works in two stages. In the first stage, it queries the environment at time $t_1 = \Theta(\epsilon m)$ on $O(1/\epsilon^2)$ equally spaced locations, which we refer to as *the grid locations*, and their local neighborhoods. This allows the algorithm to determine which of the grid locations are final at time t_1 and which are non-final. If the answers it gets are not consistent with any environment that evolves according to ρ (in which case we say that the grid is *not feasible*), then it rejects.

In its second stage, the algorithm uniformly samples $O(1/\epsilon)$ random time-location pairs (t, i) and queries $\texttt{ENV}_t$ on i and its local neighborhood. It then checks whether the answers are consistent with the answers to queries in the first stage (on the grid locations and their neighborhoods) or constitute a *violation*. The definition of consistency/violation is based on the aforementioned prediction functions of the tested rule.

---

[3] In fact, there are initial configurations that require $\Omega(n)$ steps before they ultimately converge.

One may have hoped that such a consistency check is sufficient, in the sense that all (or almost all) pairs (t, i) can be predicted based on the answers to the queried grid locations. Unfortunately, this is not the case. There are (possibly many) pairs (t, i) whose 0/1 values are not determined given the first-stage answers. However, we show that such pairs are constrained in a different way (in environments that evolve according to ρ): their location must have become final by time $t_2 = t_1 + \Delta$, where Δ is the distance between grid location. Hence, for each selected pair (t, i), the algorithm also queries $\texttt{ENV}_{t_2}$ on location i (and its neighborhood) and checks consistency with the queried locations at time t_2.

1.3.3 On the analysis of the algorithm and "backward prediction"

To show that the algorithm always accepts environments that evolve according to the tested rule ρ, we prove that our definition of violation is such that there are no violations in such environments (assuming ρ satisfies the aforementioned conditions). The more involved part of the analysis is proving that if the environment $\texttt{ENV}$ is ϵ-far from evolving according to ρ, then the algorithm will detect this with probability at least 2/3. To this end, we prove the contrapositive statement. Namely, we show that if the algorithm accepts with probability at least 2/3, then there exists an environment that evolves according to ρ and is ϵ-close to $\texttt{ENV}$. This is done by showing that we can construct an initial configuration $\texttt{ENV}'_0$, such that if we let it evolve according to ρ, resulting in an environment $\texttt{ENV}' \in \mathcal{E}^{\rho}_{m,n}$, then we can upper bound the number of pairs (t, i) such that $\texttt{ENV}_t(i) \neq \texttt{ENV}'_t(i)$ by ϵmn.

Here we build on a useful property of the prediction functions, by which they allow us a certain "prediction back in time". Namely (for t_1 and t_2 as mentioned above), we use the queried grid locations at time t_1 as well as some locations at time t_2 (which have not been queried) to determine values of locations at the earlier time 0 in $\texttt{ENV}'$. We prove that this can be done in a way that ensures that $\texttt{ENV}'$ agrees with $\texttt{ENV}$ on all pairs (t, i) that are not violating.

1.4 A short overview of the results in [6]

As stated earlier, the study of testing dynamic environments was initiated by Goldreich and Ron [6], who present several general results as well as analyze two natural specific rules. We first provide a short overview of their main general results.

They prove that the query complexity of testing (one-dimensional) rules may have high query complexity. Specifically, they show that there exists a constant $c > 0$ and an evolution rule $\rho : \Sigma^3 \to \Sigma$ such that any tester of evolution according to ρ requires $\Omega(n^c)$ queries.[4] They also prove that testing dynamic environments may be NP-Hard, provided that the temporal query complexity is "significantly sublinear" (where $f(x)$ is significantly sublinear if $f(x) < x^{1-\Omega(1)}$). More precisely, they show that for every constant $c > 0$ there exists an evolution rule $\rho : \Sigma^3 \to \Sigma$ such that no (time-conforming) polynomial-time testing algorithm with temporal query complexity n^{1-c} can test whether n-sized environments evolve according to ρ (assuming $\mathcal{NP} \not\subseteq \mathcal{BPP}$). Their general results also include a theorem concerning the usefulness of adaptivity in testing dynamic environments, a study of the relation between testing and learning dynamic environments, and a result on the power of being non time-conforming.

[4] Observe that it is possible to test the evolution according to any rule ρ over configurations of size n by performing $O(n + 1/\epsilon)$ queries (n queries to the initial configuration and $O(1/\epsilon)$ uniformly selected queries elsewhere). To get sublinear temporal query complexity, a total of $O(n/\epsilon)$ uniformly selected queries suffice (by applying a simple union bound over all possible initial configurations).

Goldreich and Ron [6] also provide testers for evolution according to two specific (classes of) rules. The first is the class of linear rules, which in the binary 1-dimensional case corresponds to the XOR rule in elementary cellular automata. They show that for any $d \geq 1$ and any field Σ of prime order, there exists a constant $\gamma < d$ such that the following holds. For any linear rule $\rho : \Sigma^{3^d} \to \Sigma$ there exists a time-conforming oracle machine of (total) time complexity $\text{poly}(1/\epsilon) \cdot n^\gamma$ that tests the consistency of an evolving environment with respect to ρ. Furthermore, the tester is non-adaptive and has one-sided error.

Their second specific positive result, loosely stated, captures fixed-speed movement of objects in one-dimension such that colliding objects stop forever. They present a (time-conforming) algorithm of (total) time complexity $\text{poly}(1/\epsilon)$ that tests the consistency of evolving environments with respect to that rule.

1.5 Future directions

Basic dynamic environments. A natural question that arises is whether a more nuanced version of the set of conditions formalized in this paper and the meta-algorithm can be defined and proved to work for other rules in the realm of basic dynamic environments. Indeed, preliminary results suggest that several other rules that ultimately converge exhibit behaviors that "resemble" the ones captured by our conditions. This leads us to the following conjecture.

▶ **Conjecture.** *If a rule ρ ultimately converges, then it is poly$(\frac{1}{\epsilon})$-testable.*

While our meta-algorithm does not apply to rules that do not ultimately converge, there are natural rules that fall under this category (the XOR rule for instance). This raises the question of whether $\text{poly}(1/\epsilon)$ testers exist for such rules. The answer is that there are $\text{poly}(1/\epsilon)$-testable rules that do not ultimately converge, but as we'll see, the question should be slightly rephrased.[5] To give one example, for the rule ρ defined as $\rho(x, y, z) = x$, each configuration is simply a copy of the previous configuration, shifted one location to the right. That is, while an environment evolving according to this rule does not, technically, ultimately converge, this rule is trivially $\text{poly}(1/\epsilon)$-testable. However, this particular rule and other rules that are capable of producing such "shifting behaviors" also have the property of not being *symmetric* (i.e., it does not hold that $\rho(x, y, z) = \rho(z, y, x)$ for every x, y, z). Hence, one way to rephrase the question is restricting it to symmetric rules.

▶ **Open Problem 1.** *Are there any symmetric rules that do not ultimately converge and are poly$(1/\epsilon)$-testable?*

Another way to rephrase this question is to define a more general notion of ultimate convergence. Specifically, we say that a rule ρ *ultimately converges up to a shift* if, for any initial configuration $\texttt{ENV}_0$, an environment evolving according to ρ converges after a bounded number of steps to a constant number of configuration *equivalence classes* between which it alternates, where two configurations are *equivalent* if they are equal *up to a shift*.

▶ **Open Problem 2.** *Are there any non-symmetric rules that do not ultimately converge up to a shift and are poly$(1/\epsilon)$-testable?*

As mentioned in Section 1.4, it has been shown in [6] that the XOR rule is sublinearly testable. However, the query complexity of the tester depends on the size of the environment and is only mildly sublinear (the complexity is $O(n^{0.8})$ for an environment of size n). This

[5] We thank one of the anonymous reviewers of this paper for pointing this out.

raises the question of whether there exists a tester for the XOR rule with significantly lower query complexity (maybe even polylogarithmic). Another question that can be raised is whether there are other symmetric rules, ones that do not ultimately converge, that can be tested with a sublinear query complexity that depends on the size of the environment.

▶ **Open Problem 3.** *Which symmetric rules that do not ultimately converge can be tested with query complexity that is sublinear in (but strictly grows with) the size of the environment?*

More general dynamic environments. Building on the ideas for testing basic dynamic environments, it may be possible to venture into more general environments. One such generalization is to consider rules that depend on more than just the three locations constituting the immediate neighborhood. Other generalizations are to environments and rules over non-binary values, higher dimensions, and environments that evolve on more general graphs.

Non-deterministic rules. We also suggest considering local rules that are non-deterministic in the sense that given some configuration, the rule allows several configurations to follow. An example of one such rule, which can be thought of as a relaxation of the OR rule, is the rule in which each value is restricted to be monotonically non-decreasing with respect to the previous values at the location's neighborhood.

Missing details

Due to space constraints, not all details appear in this extended abstract, and can be found in the full version of this paper [9].

2 Preliminaries

In addition to the basic definitions provided in Section 1.1 regarding testing dynamic environments, here we introduce several more definitions and notations.

In all that follows, when performing operations on locations $i \in \mathbb{Z}_n$, these operations are modulo n. For a pair of locations $i, j \in \mathbb{Z}_n$ we use $[i, j]$ to denote the sequence $i, i+1, \ldots, j$ (so that it is possible that $j < i$).

▶ **Definition 2.** *For a location $i \in \mathbb{Z}_n$ and an integer r, the r-neighborhood of i, denoted $\Gamma_r(i)$, is the sequence $[i-r, i+r]$. For a set of locations $I \subseteq \mathbb{Z}_n$, we let $\Gamma_r(I)$ denote the set of locations in the union of sequences $[i-r, i+r]$ taken over all $i \in I$.*

▶ **Definition 3.** *For an integer n and a local rule ρ, let $\mathcal{M}_\rho(n)$ denote the (deterministic) state machine that is defined as following. Each state of $\mathcal{M}_\rho(n)$ corresponds to a different configuration $\sigma : \mathbb{Z}_n \to \{0,1\}$. If a state corresponds to a configuration σ, then it has a single transition going to the state corresponding to the configuration that results from applying ρ to σ.*

The period of $\mathcal{M}_\rho(n)$, denoted $p_\rho(n)$, is the longest size of a (directed) cycle in $\mathcal{M}_\rho(n)$. If there exists a constant p such that $p_\rho(n) = p$ ($p_\rho(n) \leq p$) for every sufficiently large n, then we say that ρ has period (at most) p, and that ρ ultimately converges.

Observe that for every $\mathcal{M}_\rho(n)$, each strongly connected component in $\mathcal{M}_\rho(n)$ is either a single state with no edges in the component or a cycle (where in particular, the cycle may be a self-loop). For example, if ρ is the OR function, then it has period 1 (as it contains

only two cycles: one is a self-loop for the state corresponding to the configuration 1^n and the other is a self loop for the state corresponding to the configuration 0^n). On the other hand, there are rules, such as XOR, for which $p_\rho(n) = \Omega(n)$.

▶ **Definition 4.** *For two locations $i, i' \in \mathbb{Z}_n$, we let $\overrightarrow{\text{dist}}(i, i') = i' - i$ denote the directed distance from i to i, and let $\text{dist}(i, i') = \min\{\overrightarrow{\text{dist}}(i, i'), \overrightarrow{\text{dist}}(i', i)\}$ denote the (undirected) distance.*

Note that since operations on locations are modulo n, we have that $\overrightarrow{\text{dist}}(i, i') \leq n - 1$, while $\text{dist}(i, i') \leq n/2$ for all $i, i' \in \mathbb{Z}_n$.

▶ **Definition 5.** *For $t \in [m]$ and $i \in \mathbb{Z}_n$, we refer to (t, i) as a time-location pair (or simply pair).*

Given two locations, $i, i' \in \mathbb{Z}_n$ and two time steps $t, t' \in [m]$ where $t > t'$, we say that the pair (t, i) descends from the pair (t', i') if $\text{dist}(i, i') \leq t - t'$. We say that (t, i) is a descendant of (t', i') and that (t', i') is an ancestor of (t, i).

▶ **Definition 6.** *For an integer r, an r-pattern is a string in $\{0, 1\}^r$.*

3 The Conditions

Let $\rho : \{0, 1\}^3 \to \{0, 1\}$ be a local rule. We present several conditions, such that if they all hold, then the rule ρ can be tested with $\text{poly}(1/\epsilon)$ queries. These conditions capture properties of local rules that can be exploited by our (meta) algorithm.

The conditions are defined with respect to a constant (integer) k (which depends on ρ, but for the sake of simplicity we suppress the dependence on ρ and use k rather than k_ρ), and a partition of all $(2k + 1)$-patterns.[6] The partition is denoted by $(\mathcal{F}_\rho, \overline{\mathcal{F}}_\rho)$, where $\mathcal{F}$ stands for *final* and $\overline{\mathcal{F}}$ for *non-final*.

We shall say that a pair (t, i) is final (respectively, non-final) *with respect to* ENV *and* ρ if $\texttt{ENV}_t(\Gamma_k(i)) \in \mathcal{F}_\rho$ (respectively, $\overline{\mathcal{F}}_\rho$). Roughly speaking, if (t, i) is final (with respect to ENV and ρ), then location i does not change from time t and onward (or, more generally, $\texttt{ENV}_{t'}(i)$ for $t' > t$ can be predicted based on $\texttt{ENV}_t(i)$). Furthermore, if (t, i) is non-final, then $(t + 1, i)$ is final if and only if $(t, i - 1)$ or $(t - 1, i + 1)$ is final (so that finality is "infectious").

In our statements of the conditions, we make use of the parity function, which we denote by $\text{parity} : \mathbb{N} \to \{0, 1\}$ (so that $\text{parity}(x) = 1$ if x is odd and $\text{parity}(x) = 0$ if x is even).

Before each of the conditions is stated formally, we give a short, informal description. It will also be useful to have a running example of a specific rule ρ, which is the majority rule. Namely, $\text{MAJ}(\beta_1, \beta_2, \beta_3) = 1$ for any three bits $\beta_1, \beta_2, \beta_3$, if and only if $\beta_1 + \beta_2 + \beta_3 \geq 2$. For the majority rule, $k = 1$, and $\mathcal{F}_{\text{MAJ}} = \{111, 110, 011, 000, 001, 100\}$ (so that $\overline{\mathcal{F}}_{\text{MAJ}} = \{101, 010\}$).

The first condition says that if a location is final, then it remains final.

▶ **Condition 1.** *Let $\texttt{ENV} \in \mathcal{E}^\rho_{m,n}$ be an environment that evolves according to ρ. For any time step $t \in [m - 1]$ and location $i \in \mathbb{Z}_n$, if $\texttt{ENV}_t(\Gamma_k(i)) \in \mathcal{F}_\rho$, then $\texttt{ENV}_{t+1}(\Gamma_k(i)) \in \mathcal{F}_\rho$.*

Indeed, for the majority rule, if $\texttt{ENV}_t(\Gamma_1(i)) = 111$, then $\texttt{ENV}_{t+1}(\Gamma_1(i)) = 111 \in \mathcal{F}_{\text{MAJ}}$, if $\texttt{ENV}_t(\Gamma_1(i)) = 110$, then $\texttt{ENV}_{t+1}(\Gamma_1(i)) \in \{110, 111\} \subset \mathcal{F}_{\text{MAJ}}$, and if $\texttt{ENV}_t(\Gamma_1(i)) = 110$, then $\texttt{ENV}_{t+1}(\Gamma_1(i)) \in \{110, 111\} \subset \mathcal{F}_{\text{MAJ}}$ (analogous statements hold for $\texttt{ENV}_t(\Gamma_1(i)) \in \{000, 001, 100\}$).

[6] For the local rules we apply our conditions to, k is either 0 or 1, but using a variable parameter k will hopefully allow to extend our results more easily.

The second condition says that if a location is non-final, then it can become final in one time step if and only if it has a final neighbor.

▶ **Condition 2.** *Let* $\texttt{ENV} \in \mathcal{E}^{\rho}_{m,n}$ *be an environment that evolves according to* ρ*. For any time step* $t \in [m-1]$ *and location* $i \in \mathbb{Z}_n$*, if* $\texttt{ENV}_t(\Gamma_k(i)) \in \overline{\mathcal{F}}_\rho$*, then* $\texttt{ENV}_{t+1}(\Gamma_k(i)) \in \mathcal{F}_\rho$ *if and only if* $\texttt{ENV}_t(\Gamma_k(i-1)) \in \mathcal{F}_\rho$ *or* $\texttt{ENV}_t(\Gamma_k(i+1)) \in \mathcal{F}_\rho$ *(or both).*

For the majority rule, consider the case that $\texttt{ENV}_t(\Gamma_1(i)) = 101$ (so that it belongs to $\overline{\mathcal{F}}_{\text{MAJ}}$). In this case, $\texttt{ENV}_t(\Gamma_1(i-1)) \in \{110, 010\}$ and $\texttt{ENV}_t(\Gamma_1(i+1)) \in \{011, 010\}$. If $\texttt{ENV}_t(\Gamma_1(i-1)) = 110$ (which belongs to $\mathcal{F}_{\text{MAJ}}$), then $\texttt{ENV}_{t+1}(\Gamma_1(i)) \in \{110, 111\} \subset \mathcal{F}_{\text{MAJ}}$, and the case that $\texttt{ENV}_t(\Gamma_1(i+1)) = 011$ is analogous. On the other hand, if both $\texttt{ENV}_t(\Gamma_1(i-1)) = 010$ and $\texttt{ENV}_t(\Gamma_1(i+1)) = 010$ (so that they both belong to $\overline{\mathcal{F}}_{\text{MAJ}}$), then $\texttt{ENV}_{t+1}(\Gamma_1(i)) = 010$ (and it belongs to $\overline{\mathcal{F}}_{\text{MAJ}}$ as well). Note that, if for every location $i \in \mathbb{Z}_n$, it holds that $\texttt{ENV}_0(\Gamma_1(i)) \in \{010, 101\}$ (that is, every location in the initial configuration is non-final), then no location would ever become final throughout the evolution of the rule. In particular, in this case the environment alternates between $(01)^{n/2}$ and $(10)^{n/2}$, where all the locations are non-final.

The first two conditions intuitively imply that one can determine whether certain locations are final or non-final using particular "past" locations that are known to be final or non-final. The next two conditions capture the idea that the actual *values* at certain locations (and not only whether or not they are final) can also be determined based on past locations.

In particular, the third condition captures how values at locations that are final at a certain time step can be predicted using a function that depends on "past" final locations from which they descend (and to which they are closest).

▶ **Condition 3.** *Let* $\texttt{ENV} \in \mathcal{E}^{\rho}_{m,n}$ *be an environment that evolves according to* ρ*. There exists a function* $f_\rho : \{0,1\}^3 \to \{0,1\}$ *for which the following holds. First,* f_ρ *is the XOR of its first argument and a subset of the other two arguments. Second, let* (t,i) *and* (t',i') *be any two pairs such that* (t,i) *descends from* (t',i')*,* $\texttt{ENV}_t(\Gamma_k(i)), \texttt{ENV}_{t'}(\Gamma_k(i')) \in \mathcal{F}_\rho$*, and for every* $i'' \neq i'$ *satisfying* $\text{dist}(i,i'') \leq \text{dist}(i,i')$ *it holds that* $\texttt{ENV}_{t'}(\Gamma_k(i'')) \in \overline{\mathcal{F}}_\rho$*. Then*

$$\texttt{ENV}_t(i) = f_\rho(\texttt{ENV}_{t'}(i'), \mathit{parity}(t-t'), \mathit{parity}(\text{dist}(i,i'))) \ .$$

For the majority rule, f_{MAJ} is simply the identity function on its first argument, namely, $f_{\text{MAJ}}(\beta, \cdot, \cdot) = \beta$.

The fourth condition captures how locations that are non-final at a certain time step can be predicted using a function that depends on "past" non-final locations from which they descend (conditioned on there not being any final location among its ancestors in that past time step).

▶ **Condition 4.** *Let* $\texttt{ENV} \in \mathcal{E}^{\rho}_{m,n}$ *be an environment that evolves according to* ρ*. There exists a function* $h_\rho : \overline{\mathcal{F}}_\rho \times \{0,1\} \times \mathbb{Z}_n \to \overline{\mathcal{F}}_\rho$ *for which the following holds. First,* h_ρ *is reversible in the sense that for each fixed* $\tau \in \overline{\mathcal{F}}_\rho$*,* $\beta \in \{0,1\}$ *and* $\ell \in \mathbb{Z}_n$*, there exists a unique* τ' *such that* $h_\rho(\tau', \beta, \ell) = \tau$*. Second, let* (t,i) *and* (t',i') *be any two pairs such that* (t,i) *descends from* (t',i')*,* $\texttt{ENV}_t(\Gamma_k(i)), \texttt{ENV}_{t'}(\Gamma_k(i')) \in \overline{\mathcal{F}}_\rho$*, and* $\texttt{ENV}_{t'}(\Gamma_k(i'')) \in \overline{\mathcal{F}}_\rho$ *for every* i'' *such that* (t,i) *descends from* (t',i'')*. Then*

$$\texttt{ENV}_t(\Gamma_k(i)) = h_\rho(\texttt{ENV}_{t'}(\Gamma_k(i')), \mathit{parity}(t-t'), \overrightarrow{\text{dist}}(i',i)) \ .$$

For the majority rule, $h_{\text{MAJ}}(010, \beta, x) = 010$ if $\beta \oplus \mathsf{parity}(x) = 0$ and $h_{\text{MAJ}}(010, \beta, x) = 101$ if $\beta \oplus \mathsf{parity}(x) = 0$. Similarly, $h_{\text{MAJ}}(101, \beta, x) = 101$ if $\beta \oplus \mathsf{parity}(x) = 0$ and $h_{\text{MAJ}}(101, \beta, x) = 010$ if $\beta \oplus \mathsf{parity}(x) = 1$.

The additional two conditions presented below are a bit more involved than Conditions 1–4, and perhaps initially less intuitive. They do not play a role in the definition of the meta algorithm, but are applied in the proof of Lemma 2 (and we recommend that the reader return to them in that context). In a nutshell, they allow us to show that if our testing algorithm accepts the environment $\texttt{ENV}$ with high constant probability, then there exists an environment $\texttt{ENV}'$ that evolves according to ρ and is relatively close to $\texttt{ENV}$. In particular, they aid us in defining the initial configuration $\texttt{ENV}'_0$ based on $\texttt{ENV}_t$ for some appropriate time step t.

▶ **Condition 5.** *Let $\sigma : \mathbb{Z}_n \to \{0,1\}$ be a configuration and let $[x,y]$ be an interval of locations such that $\sigma(\Gamma_k(x)) \in \mathcal{F}_\rho$ and $\sigma(\Gamma_k(y)) \in \mathcal{F}_\rho$. There exists a configuration $\widetilde{\sigma} \in \mathbb{Z}_n$, which differs from σ only on locations inside $[x,y]$, for which the following holds: For every $i \in [x,y]$ we have that $\widetilde{\sigma}(\Gamma_k(i)) \in \mathcal{F}_\rho$, and if $\sigma(\Gamma_k(i)) \in \mathcal{F}_\rho$, then $\widetilde{\sigma}(i) = \sigma(i)$.*

This condition also covers the special case in which $y = x$ and we interpret $[x,y]$ as $x, x+1, \ldots, x+n$ (with a slight abuse of notation).

▶ **Condition 6.** *Let $\sigma : \mathbb{Z}_n \to \{0,1\}$ be a configuration and $z \in \mathbb{Z}_n$ such that $\sigma(\Gamma_k(z)) \in \overline{\mathcal{F}}_\rho$. Let $\nu \in \{\tau_{k+1} : \tau \in \mathcal{F}_\rho\}$ and $\gamma, \gamma' \in \{0,1\}$. There exists a configuration $\widetilde{\sigma} : \mathbb{Z}_n \to \{0,1\}$ for which the following hold. There is a location $z' \in [z+1, z+2k+1]$ where $\widetilde{\sigma}(\Gamma_k(z')) \in \mathcal{F}_\rho$, and $f_\rho(\widetilde{\sigma}(z'), \gamma, \mathsf{parity}(z'-z) \oplus \gamma') = \nu$. Furthermore, for every $i \in [z+1, z'-1]$ it holds that $\widetilde{\sigma}(\Gamma_k(i)) \in \overline{\mathcal{F}}_\rho$, and for every $i \notin [z+k, z'+k]$, $\widetilde{\sigma}(i) = \sigma(i)$.*

A (symmetric) variant of the above should also hold if we replace $z' \in [z+1, z+2k+1]$ by $z' \in [z-2k-1, z-1]$, $i \in [z+1, z'-1]$ by $i \in [z'+1, z-1]$, and $i \notin [z+k, z'+k]$ by $i \notin [z'-k, z-k]$.

4 The Meta-Algorithm

In this section, we present a meta-algorithm for testing evolution of local rules that satisfy the sufficient conditions (specified in Section 3). Here we give an algorithm whose complexity is $\lceil n/m \rceil \cdot \mathrm{poly}(1/\epsilon)$ and, in the full version of this paper [9], we explain how to remove the dependence on n/m.

In order to precisely describe our meta-algorithm, we need to first define a particular set of locations that we designate as *the 1-dimensional grid* and the notion of *violating time-location pairs* with respect to the 1-dimensional grid. The 1-dimensional grid is defined in Section 4.1 and the notion of violating pairs is defined in Section 4.2. Then, in Section 4.3, we describe our meta-algorithm.

In the full version of this paper [9], we show that these conditions hold for all (non-trivial) threshold rules, as well as a couple of additional rules.

4.1 The grid

In this subsection we introduce the notion of a one-dimensional "grid", which will be a central building block of the meta algorithm (and its analysis). Recall that a configuration is a function $\sigma : \mathbb{Z}_n \to \{0,1\}$. A *partial* configuration is a function $\sigma' : \mathbb{Z}_n \to \{0,1\} \cup \perp$, which will serve us to denote restrictions of configurations to a subset of the locations.

Let $\Delta = \frac{\epsilon^2}{b_0} \cdot \min\{n, m\}$ where b_0 is a sufficiently large constant. We assume for simplicity that Δ and n/Δ are both integers. Let $G \subseteq \mathbb{Z}_n$ (the *grid*) be the set of locations $\{j \cdot (n/\Delta)\}_{j=0}^{n/\Delta - 1}$.

As we shall see in Section 4.3, our algorithm queries the tested environment on all grid locations and their k-neighborhoods at a specific time step t_1 (which will be set subsequently).

Let $\texttt{ENV}_t[G]$ be the partial configuration that agrees with $\texttt{ENV}_t$ on all locations in $\{\Gamma_k(g) : g \in G\}$ and is $\perp$ elsewhere.

▶ **Definition 7.** *Given a time step $t > 0$, we say that the partial configuration $\texttt{ENV}_t[G]$ induced by the k-neighborhoods of the grid locations at time t is* feasible *with respect to a rule ρ, if there exists an environment $\texttt{ENV}'$ that evolves according to ρ and such that $\texttt{ENV}'_t(i) = \texttt{ENV}_t(i)$ for every $i \in \Gamma_k(G)$. We say in such a case that $\texttt{ENV}'$ is a* feasible completion *of $\texttt{ENV}_t[G]$ with respect to ρ.*

▶ **Definition 8.** *Given a pair of grid locations $g_1, g_2 \in G$, a time step t and a subset $\mathcal{S} \subset \{0,1\}^{2k+1}$, if for every grid location $g \in G \cap [g_1, g_2]$ it holds that $\texttt{ENV}_t(\Gamma_k(g)) \in S$, then we say that the interval $[g_1, g_2]$ is an S grid interval with respect to $\texttt{ENV}_t$. We say that $[g_1, g_2]$ is a maximal $\mathcal{S}$ grid interval with respect to $\texttt{ENV}_t$, if both $\texttt{ENV}_t(g_1 - \Delta)$ and $\texttt{ENV}_t(g_2 + \Delta)$ do not belong to $\mathcal{S}$.*

In particular, we shall be interested in the case that $\mathcal{S}$ is $\mathcal{F}_\rho$ or $\overline{\mathcal{F}}_\rho$. Note that a grid interval $[g_1, g_2]$ contains all the locations between g_1 and g_2, and not just the grid locations. Also note that if $\texttt{ENV}_t(\Gamma_k(g)) \in S$ for every $g \in G$, then by Definition 8, there is no maximal $\mathcal{S}$ grid interval with respect to $\texttt{ENV}_t$ (we shall deal with such cases separately).

4.2 Violating Pairs

Let ρ be a fixed local rule that satisfies all the conditions stated in Section 3. Let $t_1 = \frac{b_1 \Delta}{\epsilon}$, where b_1 is a sufficiently large constant and Δ is as defined in Section 4.1. Let $t_2 = t_1 + \Delta$. We now define the concept of a violating pair $(t, i) \in [m] \times \mathbb{Z}_n$ with respect to $\texttt{ENV}_{t_1}$. Generally speaking, these are pairs in the environment $\texttt{ENV}$ whose values are inconsistent with evolving according to the rule ρ given the values at the grid locations at time t_1. The definition of a violating pair serves us later by allowing our algorithm to reject when it encounters one, which, as we prove, happens with high constant probability if $\texttt{ENV}$ is ϵ-far from evolving according to the rule ρ.

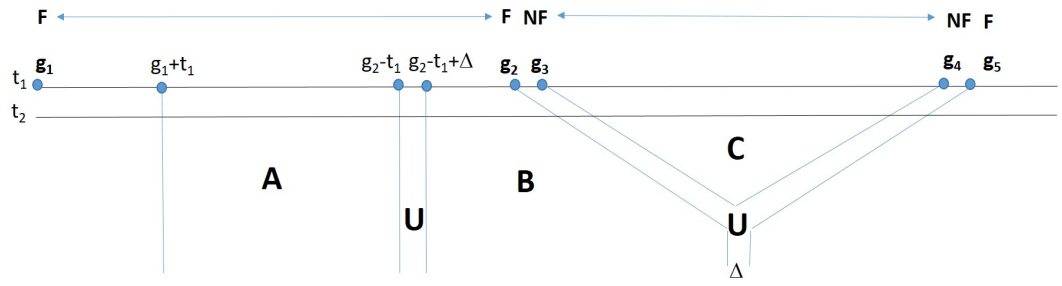


Figure 4.1 An illustration for the sets A, B, C, and U. Here $[g_1, g_2]$ is a maximal $\mathcal{F}_\rho$ grid interval, $g_3 = g_2 + \Delta$, where $[g_3, g_4]$ is a maximal $\overline{\mathcal{F}}_\rho$ grid interval, and $g_5 = g_4 + \Delta$ is an endpoint of a maximal $\mathcal{F}_\rho$ grid-interval. The area marked by A corresponds to pairs (t, i) such that $t > t_2$ and $i \in [g_1 + t_1, g_2 - t_1]$. These pairs are supposed to be final. The area marked by B corresponds to pairs (t, i) such that $t > t_2$, $i \in [g_2 - t_1 + \Delta, g_2 + (t - t_1)]$, and $\text{dist}(g_2, i) < \text{dist}(g_5, i) - \Delta$. These pairs are supposed to be final too. The area marked by C corresponds to pairs (t, i) such that $t > t_2$, $i \in [g_3, g_4]$, and (t, i) neither descends from $(t_1, g_3 + 1)$ nor from $(t_1, g_4 - 1)$. These pairs are supposed to be non-final. Finally, the areas marked by U correspond to pairs (t, i) such that $t > t_2$ and one of the following holds: **(1)** $i \in [g_2 - t_1 + 1, g_2(i) - t_1 + \Delta]$; **(2)** $i \in [g_3, g_4]$ and either **(a)** (t, i) descend from $(t_1, g_3 - \Delta)$ or $(t_1, g_4 + \Delta)$ and $|\text{dist}(g_3, i) - \text{dist}(g_4, i)| \leq \Delta$, or **(b)** (t, i) does not descend from either $(t_1, g_3 - \Delta)$ or $(t_1, g_4 + \Delta)$ but it descends from either (t_1, g_3) or (t_1, g_4).

We next define three disjoint sets of time-location pairs, denoted A, B and C, and for each of these three sets we state conditions under which the pair is considered to be a violating pair with respect to $\texttt{ENV}_{t_1}[G]$. The proof that the three sets are pairwise disjoint appears in Section 5, and for an illustration, see Figure 4.1.

If $\texttt{ENV}_{t_1}(\Gamma_k(g)) \in \mathcal{F}_\rho$ for every $g \in G$, then $A = \{(t,i) : t_2 < t < m,\ i \in \mathbb{Z}_n\}$. Otherwise, A is the set of pairs (t,i) where $t_2 < t < m$ and $i \in \mathbb{Z}_n$ such that there exists a maximal $\mathcal{F}_\rho$ grid interval $[g_1(i), g_2(i)]$ with respect to $\texttt{ENV}_{t_1}$ for which $i \in [g_1(i) + t_1, g_2(i) - t_1]$.

▶ **Definition 9.** *A pair $(t,i) \in A$ is said to be a violating pair with respect to $\texttt{ENV}_{t_1}[G]$, if at least one of the following requirements does not hold. (1) $\texttt{ENV}_{t_2}(\Gamma_k(i)) \in \mathcal{F}_\rho$. (2) $\texttt{ENV}_t(\Gamma_k(i)) \in \mathcal{F}_\rho$. (3) $\texttt{ENV}_t(i) = f_\rho(\texttt{ENV}_{t_2}(i), \mathit{parity}(t-t_2), 0)$ where f_ρ is the function referred to in Condition 3.*

Let B be the set of pairs (t,i), where $t_2 < t < m$ and $i \in \mathbb{Z}_n$ for which the following holds. First, there exists a maximal $\mathcal{F}_\rho$ grid interval $[g_1(i), g_2(i)]$ with respect to $\texttt{ENV}_{t_1}$ such that either $i \in [g_1(i) - (t - t_1), g_1(i) + t_1 - \Delta - 1]$ or $i \in [g_2(i) - t_1 + \Delta + 1, g_2(i) + (t - t_1)]$. Second, for every other maximal $\mathcal{F}_\rho$ grid interval $[g_1', g_2']$ (with respect to $\texttt{ENV}_{t_1}$), if $i \in [g_1(i) - (t - t_1), g_1(i) + t_1 - \Delta - 1]$, then $\mathrm{dist}(g_1(i), i) < \mathrm{dist}(g_2', i) - \Delta$, and if $i \in [g_2(i) - t_1 + \Delta + 1, g_2(i) + (t - t_1)]$, then $\mathrm{dist}(g_2(i), i) < \mathrm{dist}(g_1', i) - \Delta$.

▶ **Definition 10.** *A pair $(t,i) \in B$ is said to be a violating pair with respect to $\texttt{ENV}_{t_1}[G]$, if at least one of the following requirements does not hold. (1) $\texttt{ENV}_t(\Gamma_k(i)) \in \mathcal{F}_\rho$. (2) Let $[g_1(i), g_2(i)]$ be the maximal $\mathcal{F}_\rho$ grid interval ensured by the definition of B given (t,i). Let $g(i)$ be the grid location in $G \cap ([g_1(i), g_1(i) + t_1 - \Delta] \cup [g_2(i), g_2(i) - t_1 + \Delta])$ that is closest to i (if there are two such grid locations, then select the one closer to $g_1(i)$). Then $\texttt{ENV}_t(i) = f_\rho(\texttt{ENV}_{t_1}(g(i)), \mathit{parity}(t - t_1), \mathit{parity}(\mathrm{dist}(i, g(i))))$, where f_ρ is the function referred to in Condition 3.*

If $\texttt{ENV}_{t_1}(\Gamma_k(g)) \in \overline{\mathcal{F}}_\rho$ for every $g \in G$, then $C = \{(t,i) : t_2 < t < m,\ i \in \mathbb{Z}_n\}$. Otherwise, C is the set of pairs (t,i) where $t_2 < t < m$ and $i \in \mathbb{Z}_n$ for which the following holds. First, there exists a maximal $\overline{\mathcal{F}}_\rho$ grid interval $[g_1(i), g_2(i)]$ with respect to $\texttt{ENV}_{t_1}$ such that $i \in [g_1(i), g_2(i)]$. Second, the pair (t,i) neither descends from the pair $(t_1, g_1(i) + 1)$ nor from the pair $(t_1, g_2(i) - 1)$.

▶ **Definition 11.** *A pair $(t,i) \in C$ is said to be a violating pair with respect to $\texttt{ENV}_{t_1}$, if at least one of the following requirements does not hold. (1) $\texttt{ENV}_t(\Gamma_k(i)) \in \overline{\mathcal{F}}_\rho$. (2) Let $g(i) \in G$ be a grid location satisfying $\mathrm{dist}(g(i), i) < \Delta$ (if there are two such grid locations, then select the one closer to $g_1(i)$). Then $\texttt{ENV}_t(\Gamma_k(i)) = h_\rho(\texttt{ENV}_{t_1}(\Gamma_k(g(i))), \mathit{parity}(t - t_1), \overrightarrow{\mathrm{dist}}(g(i), i))$.*

Finally, we define the set U of *uncertain* pairs (t,i), for which we cannot determine, given $\texttt{ENV}_{t_1}[G]$ and the corresponding pairs (t_2, i), whether they are violating or not.

▶ **Definition 12.** *The set U consists of all pairs $(t,i) \in \mathbb{Z}_n \times [m]$ such that $t > t_2$ and $(t,i) \notin A \cup B \cup C$.*

In Section 5 show that the number of pairs (t,i) belonging to the set U is relatively small, provided that $\texttt{ENV}_{t_1}[G]$ is feasible.

4.3 The testing algorithm

Recall that Let $\Delta = \frac{\epsilon^2}{b_0} \cdot \min\{n, m\}$, $t_1 = \frac{b_1 \Delta}{\epsilon}$, and $t_2 = t_1 + \Delta$ (where b_0 and b_1 are constants that will be set in the analysis).

Algorithm 1 The testing algorithm.

Tester for evolution according to a rule ρ

1. Query $\texttt{ENV}_{t_1}$ on all locations in $\Gamma_k(G)$. If $\texttt{ENV}_{t_1}[G]$ is infeasible with respect to ρ, reject.
2. Select uniformly at random $\Theta(\frac{1}{\epsilon})$ pairs (t,i) where $i \in \mathbb{Z}_n$ and $t_2 < t < m$.
   For each selected pair (t,i), query $\texttt{ENV}_t(\Gamma_k(i))$ and $\texttt{ENV}_{t_2}(\Gamma_k(i))$.
3. If some pair selected in Step 2 is a violating pair with respect to ρ, then reject.
   Otherwise, accept.

▶ **Theorem 3.** *Let ρ be any local rule that satisfies Conditions 1–6. Algorithm 1 is a one-sided error non-adaptive testing algorithm for evolution according to ρ whose query complexity is $O(\lceil n/m \rceil/\epsilon^2)$.*

The bound on the query complexity of the algorithm follows from the fact that the number of queries performed in Step 1 is $O(n/\Delta) = O(\lceil n/m \rceil/\epsilon^2)$ (recall that k is a constant), and the number of queries performed in Step 3 is $O(1/\epsilon)$. The correctness of the algorithm follows from the next two lemmas. We prove Lemma 1 in Section 6 and Lemma 2 in Section 7.

▶ **Lemma 1** (Completeness of the meta-algorithm)**.** *Let ρ be any local rule that satisfies Conditions 1–6. If the environment* ENV *evolves according to ρ, then the algorithm accepts with probability 1.*

▶ **Lemma 2** (Soundness of the meta-algorithm)**.** *Let ρ be any local rule that satisfies Conditions 1–6. If the environment* ENV *is ϵ-far from evolving according to ρ, then the algorithm rejects with probability at least $2/3$.*

5 Observations and simple claims

In this subsection we present several observations and simple claims that will be used in our proofs of Lemma 1 and Lemma 2.

The first two observations are directly implied by Conditions 1 and 2.

▶ **Observation 1.** *Let ρ be a local rule that satisfies Conditions 1 and 2, $\texttt{ENV} \in \mathcal{E}^{\rho}_{m,n}$ an environment that evolves according to ρ and $(t,i) \in [m] \times \mathbb{Z}_n$. If (t,i) has an ancestor (t',i') such that $\texttt{ENV}_{t'}(\Gamma_k(i')) \in \mathcal{F}_\rho$, then $\texttt{ENV}_t(\Gamma_k(i)) \in \mathcal{F}_\rho$.*

Note that Observation 1 implies that if $\texttt{ENV}_t(\Gamma_k(i)) \in \overline{\mathcal{F}}_\rho$, then $\texttt{ENV}_{t'}(\Gamma_k(i')) \in \overline{\mathcal{F}}_\rho$ for every ancestor (t',i') of (t,i).

▶ **Observation 2.** *Let ρ be a local rule that satisfies Conditions 1 and 2, $\texttt{ENV} \in \mathcal{E}^{\rho}_{m,n}$ an environment that evolves according to ρ and $(t,i) \in [m] \times \mathbb{Z}_n$, $t \leq n/2$. If $\texttt{ENV}_t(\Gamma_k(i)) \in \mathcal{F}_\rho$, then the location i belongs to an interval whose size is at least $2t$ such that $\texttt{ENV}'_t(\Gamma_k(j)) \in \mathcal{F}_\rho$ for every location j in this interval.*

The observation below directly follows from Observation 2 (as well as the definition of the grid G and Definitions 7 and 8).

▶ **Observation 3.** *Let ρ be a local rule that satisfies Conditions 1 and 2. Suppose that $\texttt{ENV}_t[G]$ for $t \leq n/2$ is feasible with respect to ρ. Then for every $[g_1,g_2]$ that is a maximal $\mathcal{F}_\rho$ grid interval with respect to $\texttt{ENV}_t$, the number of locations in $[g_1,g_2]$ is at least $2t - \Delta$.*

The next observation follows directly from Observation 1 (as well as the definition of G and Definitions 7 and 8).

▶ **Observation 4.** *Let ρ be a local rule that satisfies Conditions 1 and 2. Suppose that $\texttt{ENV}_t[G]$ for $t \leq n/2$ is feasible with respect to ρ and let $g \in G$ be such that $\texttt{ENV}_t(\Gamma_k(g)) \in \mathcal{F}_\rho$. If $\texttt{ENV}'$ is a feasible completion of $\texttt{ENV}_t[G]$ (with respect to ρ), then $\texttt{ENV}'_{t'}(\Gamma_k(i)) \in \mathcal{F}_\rho$ for every $t' \geq t + \Delta$ and $i \in [g - \Delta, g + \Delta]$.*

Claim 5, stated next, also deals with feasible completions.

▷ Claim 5. Let ρ be a local rule that satisfies Conditions 1 and 2. Suppose that $\texttt{ENV}_t[G]$ is feasible with respect to ρ for $t \geq \Delta$ and let $g \in G$ be such that both $\texttt{ENV}_t(\Gamma_k(g)) \in \overline{\mathcal{F}}_\rho$ and $\texttt{ENV}_t(\Gamma_k(g + \Delta)) \in \overline{\mathcal{F}}_\rho$. If $\texttt{ENV}'$ is a feasible completion of $\texttt{ENV}_t[G]$ (with respect to ρ), then $\texttt{ENV}'_t(\Gamma_k(i)) \in \overline{\mathcal{F}}_\rho$ for every $i \in [g, g + \Delta]$.

Recall the definitions of the sets A, B and C from Section 4.2.

▷ Claim 6. The sets A, B, and C are pairwise disjoint.

In the last claim of this subsection, we bound the size of the set U of uncertain pairs (as defined in Definition 12).

▷ Claim 7. If $\texttt{ENV}_{t_1}[G]$ is feasible (with respect to ρ), then $|U| \leq \frac{5\epsilon}{b_1}mn$ (where b_1 is the constant in the setting of $t_1 = \frac{b_1 \Delta}{\epsilon}$).

We note that Claim 7 does not depend on the setting of Δ, but only on the definition of t_1 as a function of Δ (as well as the definition of the grid G, which, too is defined based on Δ, and in turn is used to determine U).

6 Proof of Lemma 1: Completeness of the meta-algorithm

Let ρ be any local rule that satisfies Conditions 1–6 (where in this proof we do not make use of Conditions 5 and 6, which are provided in the next subsection), and let $\texttt{ENV} \in \mathcal{E}^\rho_{m,n}$ be a dynamic environment that evolves according to ρ. The only steps in which our algorithm may reject are Step 1 and Step 3. The grid is feasible by definition, and hence the algorithm does not reject in Step 1. To show that it also does not reject in Step 3, we show that there are no violating pairs with respect to $\texttt{ENV}_{t_1}[G]$. Recall that each violating pair belongs to one of the three sets A, B, or C (as defined in Section 4.2). Specifically, we next show that in each of the three cases ($(t,i) \in A$, $(t,i) \in B$, and $(t,i) \in C$), the requirements (specified in Section 4.2) for (t,i) being a non-violating pair hold. In what follows, if we say that a pair (t,i) is final (similarly, non-final), then we mean with respect to $\texttt{ENV}$, and when we refer to maximal grid intervals, it is always with respect to $\texttt{ENV}_{t_1}$, and violations are always with respect to $\texttt{ENV}_{t_1}[G]$.

Pairs $(t,i) \in A$. By the definition of A, $t > t_2$ and there exists a grid location $g(i) \in G$ such that $\text{dist}(i, g(i)) \leq \Delta$ and $\texttt{ENV}_{t_1}(\Gamma_k(g(i))) \in \mathcal{F}_\rho$ (this holds both in the case that $\texttt{ENV}_{t_1}(\Gamma_k(g)) \in \mathcal{F}_\rho$ for every $g \in G$ and in the case that there exists a maximal $\mathcal{F}_\rho$ grid interval $[g_1(i), g_2(i)]$ such that $i \in [g_1(i) + t_1, g_2(i) - t_1]$.) By Observation 4, both $\texttt{ENV}_{t_2}(\Gamma_k(i)) \in \mathcal{F}_\rho$ and $\texttt{ENV}_t(\Gamma_k(i)) \in \mathcal{F}_\rho$. Turning to the third requirement, by Condition 3, applied with $t' = t_2$ and $i' = i$, we get that $\texttt{ENV}_t(i) = f_\rho(\texttt{ENV}_{t_2}(i), \mathsf{parity}(t - t_2), 0)$. Therefore, all three requirements on pairs in A hold, and hence (t,i) is not a violating pair.

Pairs $(t,i) \in B$. By the definition of B, $t > t_2$ and there exists a maximal $\mathcal{F}_\rho$ grid interval $[g_1(i), g_2(i)]$ with respect to $\texttt{ENV}_{t_1}$ such that either $i \in [g_1(i) - (t-t_1), g_1(i) + t_1 - \Delta - 1]$ or $i \in [g_2(i) - t_1 + \Delta + 1, g_2(i) + (t-t_1)]$. Furthermore, for every other maximal $\mathcal{F}_\rho$ grid interval $[g'_1, g'_2]$ (with respect to $\texttt{ENV}_{t_1}$), if $i \in [g_1(i) - (t-t_1), g_1(i) + t_1 - \Delta - 1]$, then $\text{dist}(g_1(i), i) < \text{dist}(g'_2, i) - \Delta$, and if $i \in [g_2(i) - t_1 + \Delta + 1, g_2(i) + (t-t_1)]$, then $\text{dist}(g_2(i), i) < \text{dist}(g'_1, i) - \Delta$. Let $g(i)$ be the grid location closest to i in $G \cap ([g_1(i), g_1(i) + t_1 - \Delta] \cup [g_2(i) - t_1 + \Delta, g_2(i)])$ (as defined in the second requirement concerning (non-)violating pairs $(t,i) \in B$).

We claim that (t,i) descends from $(0, g(i))$. To see why, first consider the case in which $i \in [g_1(i) - (t-t_1), g_1(i)] \cup [g_2(i), g_2(i) + (t-t_1)]$. In this case, either $g(i) = g_1(i)$ or $g(i) = g_2(i)$, which means that $\text{dist}(i, g(i)) \leq t - t_1 \leq t$. Second, consider the case in which $i \in [g_1(i), g_1(i) + t_1 - \Delta - 1] \cup [g_2(i) - t_1 + \Delta + 1, g_2(i)]$. In this case, the grid location closest to i in $G \cap ([g_1(i), g_1(i) + t_1 - \Delta] \cup [g_2(i) - t_1 + \Delta, g_2(i)])$ is within a distance of at most Δ from the location i. Hence, $\text{dist}(i, g(i)) \leq \Delta \leq t$. Therefore, in any case, $\text{dist}(i, g(i)) \leq t$, and thus the pair (t,i) descends from the pair $(0, g(i))$.

Assume (without loss of generality) that $g(i) \in [g_2(i) - t_1 + \Delta, g_2(i)]$. Since $\texttt{ENV}_{t_1}(\Gamma_k(g_2(i) + \Delta)) \in \overline{\mathcal{F}}_\rho$ (as $[g_1(i), g_2(i)]$ is a maximal final grid interval), we know (by Observation 1) that $\texttt{ENV}_0(\Gamma_k(j)) \in \overline{\mathcal{F}}_\rho$ for every $j \in [g_2(i) + \Delta - t_1, g_2(i) + \Delta + t_1]$. However, since $\texttt{ENV}_{t_1}(\Gamma_k(g_2(i)) \in \mathcal{F}_\rho$, there must be some location $\ell \in [g_2(i) - t_1, g_2(i) + \Delta - t_1 - 1]$ such that $\texttt{ENV}_0(\Gamma_k(\ell)) \in \mathcal{F}_\rho$. Among the locations ℓ that satisfy these conditions, let ℓ^* be the one that minimizes $\text{dist}(\ell, g_2(i) + \Delta - t_1)$, so that for every $\ell' \in [\ell^* + 1, g_2(i) + \Delta - t_1]$ we have that $\texttt{ENV}_0(\Gamma_k(\ell')) \in \overline{\mathcal{F}}_\rho$. Hence, for every $i'' \neq g(i)$ satisfying $\text{dist}(g(i), i'') \leq \text{dist}(g(i), \ell^*)$ it holds that $\texttt{ENV}_0(i'') \in \overline{\mathcal{F}}_\rho$. Additionally, since $g(i) \in [g_2(i) - t_1 + \Delta, g_2(i)]$ and $\ell^* \in [g_2(i) - t_1, g_2(i) + \Delta - t_1 - 1]$, it must hold that $\text{dist}(g(i), \ell^*) \leq t_1$, which means that the pair $(t_1, g(i))$ descends from the pair $(0, \ell^*)$. Also, both $\texttt{ENV}_{t_1}(g(i)) \in \mathcal{F}_\rho$ and $\texttt{ENV}_0(\ell^*) \in \mathcal{F}_\rho$. Thus, we can apply Condition 3 for the two pairs $(0, \ell^*)$ and $(t_1, g(i))$ to get that $\texttt{ENV}_{t_1}(g(i)) = f_\rho(\texttt{ENV}_0(\ell^*), \mathsf{parity}(t_1), \mathsf{parity}(\text{dist}(\ell^*, g(i))))$.

Since the pair (t,i) descends from the pair $(t_1, g(i))$, and the pair $(t_1, g(i))$ descends from the pair $(0, \ell^*)$, it holds that the pair (t,i) must also descend from the pair $(0, \ell^*)$. Additionally, both both $\texttt{ENV}_t(i) \in \mathcal{F}_\rho$ and $\texttt{ENV}_0(\ell^*) \in \mathcal{F}_\rho$. Also, by the second requirement on (t,i), involving other maximal $\mathcal{F}_\rho$ grid intervals $[g'_1, g'_2]$, for every $i'' \neq i$ satisfying $\text{dist}(i, i'') \leq \text{dist}(i, \ell^*)$ it holds that $\texttt{ENV}_0(i'') \in \overline{\mathcal{F}}_\rho$. Thus, we can apply Condition 3 for the two pairs $(0, \ell^*)$ and (t,i) to get that $\texttt{ENV}_t(i) = f_\rho(\texttt{ENV}_0(\ell^*), \mathsf{parity}(t), \mathsf{parity}(\text{dist}(\ell^*, i)))$. But then, since f_ρ is the XOR of its first argument and a subset of the other two, and $\mathsf{parity}(t - t_1) = \mathsf{parity}(t_1) \oplus \mathsf{parity}(t)$ as well as $\mathsf{parity}(\text{dist}(g(i), i))) = \mathsf{parity}(\text{dist}(\ell^*, g(i))) \oplus \mathsf{parity}(\text{dist}(\ell^*, i))$, we get that $\texttt{ENV}_t(i) = f_\rho(\texttt{ENV}_{t_1}(g(i)), \mathsf{parity}(t - t_1), \mathsf{parity}(\text{dist}(g(i), i)))$.

Pairs $(t,i) \in C$. There are two cases (where in both $t > t_2$). The first is that $\texttt{ENV}_{t_1}(\Gamma_k(g)) \in \overline{\mathcal{F}}_\rho$ for every $g \in G$ (so that i may be any location in $\mathbb{Z}_n$). In the second case there exists a maximal $\overline{\mathcal{F}}_\rho$ grid interval $[g_1(i), g_2(i)]$ such that $i \in [g_1(i), g_2(i)]$, and (t,i) does not descend from either $(t_1, g_1(i) - 1)$ or $(t_1, g_2(i) + 1)$, which implies that for every $j \in \mathbb{Z}_n$, if the pair (t,i) descends from (t_1, j), then $j \in [g_1(i), g_2(i)]$. In both cases, by Claim 5, all ancestors (t_1, j) of (t,i) satisfy $\texttt{ENV}_{t_1}(\Gamma_k(j)) \in \overline{\mathcal{F}}_\rho$. By Observation 1 this implies that $\texttt{ENV}_t(\Gamma_k(i)) \in \overline{\mathcal{F}}_\rho$, so that the first requirement is met. As for the second requirement, since the grid location $g(i)$ defined in the second requirement is such that $(t_1, g(i))$ is an ancestor of (t,i) (and $\texttt{ENV}_{t_1}(\Gamma_k(g(i))) \in \overline{\mathcal{F}}_\rho$), we can apply Condition 4 (with $t' = t_1$ and $i' = g$) to get that $\texttt{ENV}_t(\Gamma_k(i)) = h_\rho(\texttt{ENV}_{t_1}(\Gamma_k(g(i))), \mathsf{parity}(t - t_1), \overrightarrow{\text{dist}}(g(i), i)$, as required.

We've shown that under the premise of the lemma, there is no pair $(t,i) \in A \cup B \cup C$ that is a violating pair. Thus, our algorithm cannot reject at Step 3.

7 Proof of Lemma 2: Soundness of the meta-algorithm

Let $\texttt{ENV}$ be any environment that is ϵ-far from evolving according to ρ, where ρ is a local rule that satisfies Conditions 1–6. If $\texttt{ENV}_{t_1}[G]$ is infeasible with respect to ρ, then the algorithm rejects (in Step 1). Hence, we assume from now on that $\texttt{ENV}_{t_1}[G]$ is feasible.

We claim that the number of violating pairs with respect to $\texttt{ENV}_{t_1}[G]$ is at least $\frac{\epsilon}{b_2}mn$, where $b_2 > 1$ is a constant. Lemma 2 follows, since the algorithm selects $s = \Theta(1/\epsilon)$ pairs (in Step 2), and rejects if any of them is found to be a violating pair (in Step 3). Hence, the probability that the algorithm rejects is at least $1 - (1 - \epsilon/b_2)^s$, which is at least $2/3$ for $s \geq 2b_2/\epsilon$.

Suppose by way of contradiction that there are less than $\frac{\epsilon}{b_2}mn$ violating pairs. We show how, based on $\texttt{ENV}$ (to be precise, $\texttt{ENV}_{t_1}[G]$ and $\texttt{ENV}_{t_2}$) we can define an environment $\texttt{ENV}'$ for which the following holds. First, $\texttt{ENV}'$ evolves according to ρ. Second, $\texttt{ENV}'$ differs from $\texttt{ENV}$ on at most ϵmn pairs $(t, i) \in \mathbb{Z}_n \times [m]$. But this contradicts the premise that $\texttt{ENV}$ is ϵ-far from evolving according to ρ. Details follow in the next subsections.

We first provide all details (in Section 7.1 and Section 7.2) under the assumption that there exist grid locations $g \in G$ for which $\texttt{ENV}_{t_1}(\Gamma_k(g)) \in \mathcal{F}_\rho$ as well as grid locations $g' \in G$ for which $\texttt{ENV}_{t_1}(\Gamma_k(g')) \in \overline{\mathcal{F}}_\rho$. We discuss (in the full version of this paper [9]) the two special cases for which either $\texttt{ENV}_{t_1}(\Gamma_k(g)) \in \mathcal{F}_\rho$ for every $g \in G$ or $\texttt{ENV}_{t_1}(\Gamma_k(g)) \in \overline{\mathcal{F}}_\rho$ for every $g \in G$, which we refer to as the *homogeneous* cases.

7.1 The definition of $\texttt{ENV}'$

To construct the dynamic environment $\texttt{ENV}'$, we define its initial configuration $\texttt{ENV}'_0$, and then apply the local rule ρ for $m - 1$ steps. Hence, $\texttt{ENV}'$ evolves according to ρ by construction. The initial configuration $\texttt{ENV}'_0$ is defined with respect to a configuration σ on which we perform several transformations to obtain $\texttt{ENV}'_0$. We define the configuration σ by specifying the value of $\sigma(i)$ for each location $i \in \mathbb{Z}_n$ as explained next. In what follows, whenever we refer to maximal $\overline{\mathcal{F}}_\rho$ grid intervals (similarly, maximal $\mathcal{F}_\rho$ grid intervals), it is with respect to $\texttt{ENV}_{t_1}$.

We shall make use of a function $h_\rho^{\leftarrow} : \overline{\mathcal{F}}_\rho \times \{0,1\} \times \mathbb{Z}_n$ (based on h_ρ – see Condition 4). Recall that by Condition 4, for each fixed $\tau \in \overline{\mathcal{F}}_\rho$, $\beta \in \{0,1\}$ and $\ell \in \mathbb{Z}_n$, there exists a unique τ' such that $h_\rho(\tau', \beta, \ell) = \tau$.

▶ **Definition 8.** *For any $\tau \in \overline{\mathcal{F}}_\rho$, $\beta \in \{0,1\}$ and $\ell \in \mathbb{Z}_n$, $h_\rho^{\leftarrow}(\tau, \beta, \ell)$ equals the (unique) pattern τ' for which $h_\rho(\tau', \beta, \ell) = \tau$.*

We also make the following observation, based on Condition 3, by which f_ρ is the XOR of its first argument and a subset of the other two.

▶ **Observation 9.** *For any $\beta_1, \beta_2, \beta_3 \in \{0,1\}$, if $f_\rho(\beta_1, \beta_2, \beta_3) = \beta'_1$, then $f_\rho(\beta'_1, \beta_2, \beta_3) = \beta_1$. Furthermore, for any $\beta'_2, \beta'_3 \in \{0,1\}$, $f_\rho(f_\rho(\beta_1, \beta_2, \beta_3), \beta'_2, \beta'_3) = f_\rho(\beta_1, \beta_2 \oplus \beta'_2, \beta_3 \oplus \beta'_3)$, and in particular, $f_\rho(f_\rho(\beta_1, \beta_2, \beta_3), \beta_2, \beta_3) = \beta_1$.*

For each maximal $\overline{\mathcal{F}}_\rho$ grid interval $[g_1, g_2]$, let $J(g_1, g_2) = [g_1 - t_1 - k, g_2 + t_1 + k]$ and let J be the union over all such sets. We also define $J_1(g_1, g_2) = [g_1 - t_1, g_2 + t_1]$ (for each $\overline{\mathcal{F}}_\rho$ grid interval $[g_1, g_2]$), and let $J_1 \subset J$ be the union over all such sets.

We first establish two simple claims.

▷ Claim 10. Let ρ be any local rule that satisfies Conditions 1–4. For every two maximal $\overline{\mathcal{F}}_\rho$ grid interval $[g_1, g_2]$ and $[g'_1, g'_2]$, we have that $J(g_1, g_2) \cap J(g'_1, g'_2) = \emptyset$.

▷ Claim 11. Let ρ be any local rule that satisfies Conditions 1–4. Let $\texttt{ENV}''$ be any environment that is a feasible extension of $\texttt{ENV}_{t_1}[G]$ with respect to ρ, and let $[g_1, g_2]$ be a maximal $\overline{\mathcal{F}}_\rho$ grid interval (with respect to $\texttt{ENV}_{t_1}[G]$). Then $\texttt{ENV}''_0(\Gamma_k(i)) \in \overline{\mathcal{F}}_\rho$ for every $i \in J_1(g_1, g_2)$. Furthermore, $\texttt{ENV}''_0(\Gamma_k(i)) = h^{\leftarrow}_\rho(\texttt{ENV}_{t_1}(\Gamma_k(g)), \mathsf{parity}(t_1), \overrightarrow{\mathrm{dist}}(i, g))$ for any $g \in G \cap [g_1, g_2]$ and every ancestor $(0, i)$ of (t_1, g).

Observe that Claim 11 implies that $\texttt{ENV}''_0$ is *uniquely* determined by $\texttt{ENV}_{t_1}[G]$ on all location in J for every $\texttt{ENV}''$ that is a feasible extension of $\texttt{ENV}_{t_1}[G]$ (with respect to ρ). Based on this observation, we start by setting the locations of σ that belong to J as in such $\texttt{ENV}''_0$. In particular we have that $\sigma(\Gamma_k(i)) \in \overline{\mathcal{F}}_\rho$ for every $i \in J_1$, and furthermore,

$$\forall i \in J_1, g \in G \text{ s.t. } (t_1, g) \text{ descends from } (0, i) \text{ and,}$$
$$\sigma(\Gamma_k(i)) = h^{\leftarrow}_\rho(\texttt{ENV}_{t_1}(\Gamma_k(g)), \mathsf{parity}(t_1), \overrightarrow{\mathrm{dist}}(i, g)) \ . \tag{7.1}$$

Turning to the locations not yet set in σ, for each location $i \in \mathbb{Z}_n \setminus J$, $\sigma(i) = f_\rho(\texttt{ENV}_{t_2}(i), \mathsf{parity}(t_2), 0)$. Note that by Observation 9, $f_\rho(\sigma(i), \mathsf{parity}(t_2), 0) = \texttt{ENV}_{t_2}(i)$.

We next explain how we modify σ so as to obtain $\texttt{ENV}'_0$ using Condition 5 and Condition 6. The modifications are performed (strictly) within the following set of intervals $\mathcal{S}$.

$$\mathcal{S} = \left\{ \left[a = g_1 - \Delta + t_1, b = g_2 + \Delta - t_1\right] \ : \ [g_1, g_2] \text{ is a maximal } \mathcal{F}_\rho \text{ grid interval} \right\} \ . \tag{7.2}$$

The intervals in $\mathcal{S}$ are clearly disjoint (as each is a sub-interval of a different maximal $\mathcal{F}_\rho$ grid interval), and by Observation 3, each is non-empty. Note that for each maximal $\mathcal{F}_\rho$ grid interval $[g_1, g_2]$, we have that $g_1 - \Delta$ and $g_2 + \Delta$ are endpoints of maximal $\overline{\mathcal{F}}_\rho$ grid interval. Therefore, $a, b \in J_1$ for each interval $[a, b] \in \mathcal{S}$, and by the setting of σ and Claim 11, $\sigma(\Gamma_k(a)), \sigma(\Gamma_k(b)) \in \overline{\mathcal{F}}_\rho$.

For each $[a, b] \in \mathcal{S}$ and the corresponding $[g_1, g_2]$, let $\alpha(a, b) = \texttt{ENV}_{t_1}(g_1)$, $\beta(a, b) = \texttt{ENV}_{t_1}(g_2)$, $\gamma(a, b) = \mathsf{parity}(t_1)$, $\gamma'(a, b) = \mathsf{parity}(t_1 - \Delta)$. We shall apply Condition 5 and Condition 6 to modify σ on all $[a, b] \in \mathcal{S}$ "in parallel" as described next, and set $\texttt{ENV}'_0$ to be the resulting configuration.

For each $[a, b] \in \mathcal{S}$ we first apply Condition 6 with $z = a$, $\nu = \alpha(a, b)$, $\gamma = \gamma(a, b)$ and $\gamma' = \gamma'(a, b)$. We let $a' = z'$ (recall that $z' \in [z + 1, z + 2k + 1]$ and $\widetilde{\sigma}(\Gamma_k(z')) \in \mathcal{F}_\rho$). Next we apply Condition 6 in its second (symmetric) variant with $z = b$, $\nu = \beta(a, b)$, $\gamma = \gamma(a, b)$ and $\gamma' = \gamma'(a, b)$. We let $b' = z'$ (recall that in this variant, $z' \in [z - 2k - 2, z - 1]$, and here too $\widetilde{\sigma}(\Gamma_k(z')) \in \mathcal{F}_\rho$). Finally we apply Condition 5 on the modified configuration with $x = a'$ and $y = b'$.

7.2 The distance between ENV and ENV′

In this subsection we show that based on the counter-assumption regarding the number of violating pairs, the number of pairs $(t, i) \in \mathbb{Z}_n \times [m]$ on which $\texttt{ENV}$ and $\texttt{ENV}'$ differ is at most ϵmn. To this end we show that each (t, i) such that $\texttt{ENV}_t(i) \neq \texttt{ENV}'_t(i)$ belongs to one of the following sets:

1. The set of pairs (t, i) for which $0 \leq t \leq t_2$.
2. The uncertainty set U.
3. The set of (t, i) pairs where (t, i) is a violation with respect to $\texttt{ENV}_{t_1}[G]$.

By the setting of t_2 (t_1) and Δ, the number of pairs in the first set is at most $\frac{(b_1+1)\epsilon}{b_0} mn$. By Claim 7, $|U| \leq \frac{5\epsilon}{b_1} mn$. By our counter-assumption, the number of violating pairs is at most $\frac{\epsilon}{b_2} mn$. Setting $b_1 = 15$, $b_0 = 48$ and $b_2 = 3$, we get a total of at most ϵmn pairs, as claimed.

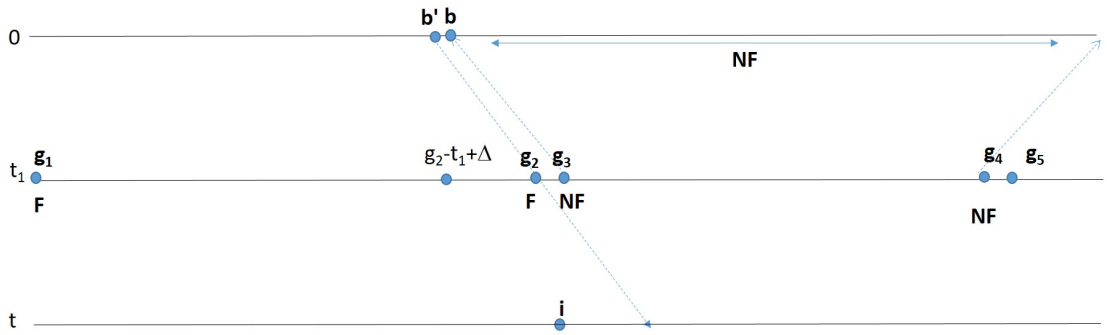


Figure 7.1 An illustration for the setting of $\texttt{ENV}_0$. As in Figure 4.1, $[g_1, g_2]$ is a maximal $\mathcal{F}_\rho$ grid interval, $g_3 = g_2 + \Delta$, where $[g_3, g_4]$ is a maximal $\overline{\mathcal{F}}_\rho$ grid interval, and $g_5 = g_4 + \Delta$ is an endpoint of a maximal $\mathcal{F}_\rho$ grid-interval. The maximal $\overline{\mathcal{F}}_\rho$ grid interval $[g_3, g_4]$ is used to set the locations between $g_3 - t_1 = g_2 - t_1 + \Delta$ and $g_4 + t_1$ (more precisely, between $g_3 - t_1 - k$ and $g_4 + t_1 + k$ based on Claim 11. The location $b = g_2 - t_1 + \Delta$ is an endpoint of an interval in $\mathcal{S}$, and the location b' is determined by the application of Condition 6. The values in the k-neighborhood of b' are set so that the evolution of ρ will result in the $\texttt{ENV}_{t_1}(g_2)$ at time t_1. The pair (t, i) belongs to the set B.

To establish the claim that each (t, i) for which $\texttt{ENV}_t(i) \neq \texttt{ENV}'_t(i)$ belongs to one of the above three sets, we prove the contrapositive. Suppose the pair (t, i) is not in the uncertainty set U and that $t > t_2$. It follows that $(t, i) \in A \cup B \cup C$. We show that for each of the three types of pairs ($(t, i) \in A$, $(t, i) \in B$, and $(t, i) \in C$), if the pair (t, i) is not a violating pair with respect to $\texttt{ENV}_{t_1}[G]$, it must hold that $\texttt{ENV}_t(i) = \texttt{ENV}'_t(i)$.

Pairs $(t, i) \in A$. By the definition of A, there exists a maximal $\mathcal{F}_\rho$ grid interval $[g_1(i), g_2(i)]$ (with respect to $\texttt{ENV}_{t_1}$) for which $i \in [g_1(i) + t_1, g_2(i) - t_1]$. Since (t, i) is not a violating pair with respect to $\texttt{ENV}_{t_1}[G]$, it must hold that $\texttt{ENV}_{t_2}(\Gamma_k(i)), \texttt{ENV}_t(\Gamma_k(i)) \in \mathcal{F}_\rho$ and that $\texttt{ENV}_t(i) = f_\rho(\texttt{ENV}_{t_2}(i), \mathsf{parity}(t - t_2), 0)$. Since $i \in [g_1(i) + t_1, g_2(i) - t_1]$, we know that $i \notin J$. Hence, by the definition of the configuration σ, we have that $\sigma(i) = f_\rho(\texttt{ENV}_{t_2}(i), \mathsf{parity}(t_2), 0)$ and that $\sigma(\Gamma_k(i)) \in \mathcal{F}_\rho$. Let $[a(i), b(i)] = [g_1(i) - \Delta + t_1, g_2(i) + \Delta - t_1]$, so that $i \in I(a(i), b(i))$. By the definition of E'_0, based on Condition 5 we have that $\texttt{ENV}'_0(i) = \sigma(i)$ and $\texttt{ENV}'_0(\Gamma_k(i)) \in \mathcal{F}_\rho$. Since $\texttt{ENV}'$ evolves according to ρ, by Condition 3,

$$\texttt{ENV}'_{t_2}(i) = f_\rho(\texttt{ENV}'_0(i), \mathsf{parity}(t_2), 0) = f_\rho(f_\rho(\texttt{ENV}_{t_2}(i), \mathsf{parity}(t_2), 0), \mathsf{parity}(t_2), 0) = \texttt{ENV}_{t_2}(i)$$

where the last equality follows from (the second part of) Observation 9. Additionally, by Condition 1, $\texttt{ENV}'_{t_2}(i) \in \mathcal{F}_\rho$. Therefore, by Condition 3,

$$\texttt{ENV}'_t(i) = f_\rho(\texttt{ENV}'_{t_2}(i), \mathsf{parity}(t - t_2), 0) = f_\rho(\texttt{ENV}_{t_2}(i), \mathsf{parity}(t - t_2), 0) = \texttt{ENV}_t(i)\ .$$

Pairs $(t, i) \in B$. By the definition of B, there exists a maximal $\mathcal{F}_\rho$ grid interval $[g_1(i), g_2(i)]$ with respect to $\texttt{ENV}_{t_1}$ such that either $i \in [g_1(i) - (t - t_1), g_1(i) + t_1 - \Delta - 1]$ or $i \in [g_2(i) - t_1 + \Delta + 1, g_2(i) + (t - t_1)]$. Assume (without loss of generality) that the latter holds. By the definition of B we also know that for every other maximal $\mathcal{F}_\rho$ grid interval $[g'_1, g'_2]$ it holds that $\mathrm{dist}(i, g_2(i)) < \mathrm{dist}(i, g'_1), \mathrm{dist}(i, g'_2) - \Delta$. Let $g(i)$ be the grid location closest to i in $[g_2(i) - t_1 + \Delta, g_2(i)]$. Since $i \in [g_2(i) - t_1 + \Delta + 1, g_2(i) + (t - t_1)]$, necessarily, $g(i) \in [g_2(i) - t_1, g_2(i)]$. For the sake of conciseness, in what follows we shall use g_1, g_2, and g as a shorthand for $g_1(i)$, $g_2(i)$ and $g(i)$, respectively.

Since $[g_1, g_2]$ is a maximal $\mathcal{F}_\rho$ grid interval, $[a = g_1 - \Delta + t_1, b = g_2 + \Delta - t_1] \in \mathcal{S}$. Hence, to obtain $\texttt{ENV}'_0$ from the configuration σ, we invoked Condition 6 (the symmetric version) with $z = b$, $\nu = \texttt{ENV}_{t_1}(g_2)$, $\gamma = \mathsf{parity}(t_1)$, and $\gamma' = \mathsf{parity}(t_1 - \Delta) = \mathsf{parity}(\mathrm{dist}(g_2, b))$. By Condition 6, letting $b' = z'$, $\texttt{ENV}_{t_1}(g_2) = f_\rho(\texttt{ENV}'_0(b'), \mathsf{parity}(t_1), \mathsf{parity}(\mathrm{dist}(g_2, b')))$. By Observation 9,

$$\texttt{ENV}'_0(b') = f^{\leftarrow}_\rho(\texttt{ENV}_{t_1}(g_2), \mathsf{parity}(t_1), \mathsf{parity}(\mathrm{dist}(g_2, b'))) \ . \tag{7.3}$$

As $b' \in [b - 2k - 1, b - 1]$, which by the setting of b implies that $b' \in [g_2 + \Delta - t_1 - 2k - 1, g_2 + \Delta - t_1 - 1]$, we have that $(0, b')$ is an ancestor of (t_1, j) for every $j \in [g_2 - t_1, g_2]$. In particular this holds for the grid location g (that is closest to i in $G \cap [g_2 - t_1, g_2]$). Since (t, i) descends from (t_1, g), we get that (t, i) also descends from $(0, b')$.

We claim that for every $b'' \neq b'$ with $\mathrm{dist}(i, b'') < \mathrm{dist}(i, b')$ it holds that $\texttt{ENV}'_0(b'') \in \overline{\mathcal{F}}_\rho$. To verify this, let $[g_3, g_4]$ be the maximal $\overline{\mathcal{F}}_\rho$ grid interval where $g_3 = g_2 + \Delta$, and let $g_5 = g_4 + \Delta$, so that g_5 is the endpoint of a maximal $\mathcal{F}_\rho$ grid interval. By the definition of $\texttt{ENV}'_0$ (based on σ and Condition 6) we have that $\texttt{ENV}'_0(\Gamma_k(j)) \in \overline{\mathcal{F}}_\rho$ for every $j \in [b' + 1, b - 1] \cup J_1(g_3, g_4) = [b' + 1, g_4 + t_1]$. Since (by the second requirements on pairs in B), $\mathrm{dist}(i, g_2) < \mathrm{dist}(i, g_5) - \Delta$ and $b' \in [g_2 - t_1 + \Delta - 2k - 1, g_2 - t_1 + \Delta - 1]$, we have that $\texttt{ENV}'_0(b'') \in \overline{\mathcal{F}}_\rho$ for every $b'' \neq b'$ with $\mathrm{dist}(i, b'') < \mathrm{dist}(i, b')$. Therefore, we can apply Condition 3 to obtain that

$$\begin{aligned}
&\texttt{ENV}'_t(i) = f_\rho(\texttt{ENV}_0(b'), \mathsf{parity}(t), \mathsf{parity}(\mathrm{dist}(i, b'))) \\
&= f_\rho(f_\rho(\texttt{ENV}_{t_1}(g_2), \mathsf{parity}(t_1), \mathsf{parity}(\mathrm{dist}(g_2, b'))), \mathsf{parity}(t), \mathsf{parity}(\mathrm{dist}(i, b'))) &(7.4)\\
&= f_\rho(\texttt{ENV}_{t_1}(g_2), \mathsf{parity}(t - t_1), \mathsf{parity}(\mathrm{dist}(i, g_2))) &(7.5)
\end{aligned}$$

where the last equality follows from Observation 9.

Consider first the case that $g = g_2$. Since the pair (t, i) is not a violating pair,

$$\texttt{ENV}_t(i) \ = \ f_\rho(\texttt{ENV}_{t_1}(g_2), \mathsf{parity}(t - t_1), \mathsf{parity}(\mathrm{dist}(i, g_2))) \ , \tag{7.6}$$

and hence in this case, $\texttt{ENV}_t(i) = \texttt{ENV}'_t(i)$, as desired. We next turn to the case that $g \neq g_2$. We claim that since $\texttt{ENV}_{t_1}[G]$ is feasible,

$$\texttt{ENV}_{t_1}(g) = f_\rho(\texttt{ENV}'_0(b'), \mathsf{parity}(t_1), \mathsf{parity}(\mathrm{dist}(g, b'))) \ . \tag{7.7}$$

Conditioned on Equality 7.7 holding, the argument is the same as for the case that $g = g_2$ (replacing g_2 with g in Equations (7.4)–(7.6)).

To verify Equation (7.7), we introduce the notion of a *source* for a final pair. Let $\texttt{ENV}''$ be an environment that evolves according to ρ, and (t', i') a final pair with respect to $\texttt{ENV}''$ and ρ. We say that $(0, b'')$ is the *source* of (t', i') (at time 0) in $\texttt{ENV}''$ if $(0, b'')$ is an ancestor of (t', i'), is final, and $\mathrm{dist}(b'', i') < \mathrm{dist}(j, i')$ for every other final $(0, j)$. Consider any feasible extension E'' of $\texttt{ENV}_{t_1}[G]$. By Claim 11 and the discussion above, the source $(0, b'')$ of (t_1, g_2) (at time 0 in $\texttt{ENV}''$) must satisfy $b'' \in [g_2 - t_1, g_2 - t_1 + \Delta - 1]$. Furthermore, $(0, b'')$ must also be the source of (t_1, g') for every grid location $g' \in [g_2 - t_1, g_2]$. Therefore, for each such grid location, $\texttt{ENV}_{t_1}(g') = \texttt{ENV}''_{t_1}(g') = f_\rho(\texttt{ENV}''_0(b''), \mathsf{parity}(t_1), \mathsf{parity}(\mathrm{dist}(g', b'')))$, where $\texttt{ENV}''_0(b'') = f^{\leftarrow}_\rho(\texttt{ENV}_{t_1}(g_2), \mathsf{parity}(t_1), \mathsf{parity}(\mathrm{dist}(g_2, b'')))$. If f_ρ and $f^{\leftarrow}_\rho$ do no depend on their third argument, then, by Equation (7.3), $\texttt{ENV}'_0(b') = \texttt{ENV}''_0(b'')$ and if they do, then $\texttt{ENV}'_0(b') = \texttt{ENV}''_0(b'') \oplus \mathsf{parity}(\mathrm{dist}(b', b''))$. In either case, Equation (7.7) follows.

Pairs $(t, i) \in C$. By the definition of C, there exists a maximal $\overline{\mathcal{F}}_\rho$ grid interval $[g_1(i), g_2(i)]$ such that $g_1(i) \leq i \leq g_2(i)$. Additionally, the pair (t, i) does not descend from either the pair $(t_1, g_1(i) - 1)$ or from the pair $(t_1, g_2(i) + 1)$. Let $g(i)$ be the grid location defined in Definition 11 (of violating pairs in C), so that $g(i) \in G \cap [g_1(i), g_2(i)]$ and $\mathrm{dist}(i, g(i)) < \Delta$.

By the definition of $\texttt{ENV}'_0$ (based on σ – recall Equation (7.1)), we have that $\texttt{ENV}'_0(\Gamma_k(i)) = h^{\leftarrow}_\rho(\texttt{ENV}_{t_1}(\Gamma_k(g(i))), \mathsf{parity}(t_1), \overrightarrow{\mathrm{dist}}(i, g(i)))$. By the definition of $h^{\leftarrow}_\rho$ (Definition 8), this implies that $\texttt{ENV}_{t_1}(\Gamma_k(g(i))) = h_\rho(\texttt{ENV}'_0(\Gamma_k(i)), \mathsf{parity}(t_1), \overrightarrow{\mathrm{dist}}(i, g(i)))$. Since $\texttt{ENV}'_0(\Gamma_k(j)) \in \overline{\mathcal{F}}_\rho$ for every location $j \in J(g_1(i)), g_2(i))$, and the environment $\texttt{ENV}'$ evolves according to ρ where ρ satisfies Condition 4, we know that $\texttt{ENV}'_{t_1}(\Gamma_k(g(i))) = h_\rho(\texttt{ENV}'_0(\Gamma_k(i)), \mathsf{parity}(t_1), \overrightarrow{\mathrm{dist}}(i, g(i)))$. Hence, $\texttt{ENV}'_{t_1}(\Gamma_k(g(i))) = \texttt{ENV}_{t_1}(\Gamma_k(g(i)))$. Furthermore, using in addition the fact that (t, i) does not descend from either $(t_1, g_1(i) - 1)$ or $(t_1, g_2(i) + 1)$, we get all ancestors of (t_1, j) of (t, i) satisfy $\texttt{ENV}'_{t_1}(\Gamma_k(j)) \in \overline{\mathcal{F}}_\rho$, so that $\texttt{ENV}'_t(\Gamma_k(i)) = h_\rho(\texttt{ENV}'_{t_1}(\Gamma_k(g(i))), \mathsf{parity}(t - t_1), \overrightarrow{\mathrm{dist}}(g(i), i)))$. Since (t, i) is not a violating pair, $\texttt{ENV}_t(\Gamma_k(i)) = h_\rho(\texttt{ENV}_{t_1}(\Gamma_k(g(i))), \mathsf{parity}(t - t_1), \overrightarrow{\mathrm{dist}}(g(i), i))$, and using $\texttt{ENV}'_{t_1}(\Gamma_k(g(i))) = \texttt{ENV}_{t_1}(\Gamma_k(g(i)))$ we get that $\texttt{ENV}'_t(i) = \texttt{ENV}_t(i)$.

References

1 D. Burraston and E. Edmonds. Cellular automata in generative electronic music and sonic art: a historical and technical review. *Digital Creativity*, 16(3):165–185, 2005.

2 Bastien C. and Michel D. *Cellular Automata Modelling of Physical Systems*. Cambridge University Press, 1998.

3 P. P. Chaudhuri, D. R. Chowdhury, S. Nandi, and S. Chattopadhyay. *Additive Cellular Automata Theory and Applications. Vol. 1*. IEEE Press, 1997. IEEE Press advances in circuits and systems series.

4 M. Cook. Universality in elementary cellular automata. *Complex systems*, 15(1):1–40, 2004.

5 O. Goldreich, S. Goldwasser, and D. Ron. Property testing and its connection to learning and approximation. *Journal of the ACM*, 45(4):653–750, 1998.

6 O. Goldreich and D. Ron. On learning and testing dynamic environments. *Journal of the ACM*, 64(3):1–90, 2017.

7 L. Kier, C. Cheng, and P. Seybold. Cellular automata models of chemical systems. *SAR and QSAR in Environmental Research*, 11(2):79–102, 2000.

8 A. Mustafa, A. Heppenstall, H. Omrani, I. Saadi, M. Cools, and J. Teller. Modelling built-up expansion and densification with multinomial logistic regression, cellular automata and genetic algorithm. *Computers, Environment and Urban Systems*, 67:147–156, 2018.

9 Y. Nakar and D. Ron. Testing dynamic environments: Back to basics. *arXiv preprint*, 2021. `arXiv:2105.00759`.

10 D. Peleg. Local majorities, coalitions and monopolies in graphs: a review. *Theoretical Computer Science*, 282(2):231–257, 2002.

11 D. A Rosenblueth and C. Gershenson. A model of city traffic based on elementary cellular automata. *Complex Systems*, 19(4):305, 2011.

12 R. Rubinfeld and M. Sudan. Robust characterization of polynomials with applications to program testing. *SIAM Journal on Computing*, 25(2):252–271, 1996.

13 J. von Neumann. The general and logical theory of automata. *Cerebral Mechanisms of Behavior: The Hixon Symposium*, pages 1–41, 1951.

14 S. Wolfram. *A new kind of science*, volume 5. Wolfram media Champaign, IL, 2002.

15 J. Xu, B. Gu, Y. Guo, J. Chang, Y. Ge, Y. Min, and X. Jin. A cellular automata model for population dynamics simulation of two plant species with different life strategies. In *2010 IEEE International Conference on Intelligent Systems and Knowledge Engineering*, 2010.

16 A. N. Zehmakan. *On the spread of information through graphs*. PhD thesis, ETH Zurich, 2019.

17 Z. Zheng, W. Huang, S. Li, and Y. Zeng. Forest fire spread simulating model using cellular automaton with extreme learning machine. *Ecological Modelling*, 348:33–43, 2017.

Decision Problems for Second-Order Holonomic Recurrences

Eike Neumann
Max Planck Institute for Software Systems, Saarland Informatics Campus, Saarbrücken, Germany

Joël Ouaknine
Max Planck Institute for Software Systems, Saarland Informatics Campus, Saarbrücken, Germany

James Worrell
Department of Computer Science, Oxford University, UK

Abstract

We study decision problems for sequences which obey a second-order holonomic recurrence of the form $f(n+2) = P(n)f(n+1) + Q(n)f(n)$ with rational polynomial coefficients, where P is non-constant, Q is non-zero, and the degree of Q is smaller than or equal to that of P. We show that existence of infinitely many zeroes is decidable. We give partial algorithms for deciding the existence of a zero, positivity of all sequence terms, and positivity of all but finitely many sequence terms. If Q does not have a positive integer zero then our algorithms halt on almost all initial values $(f(1), f(2))$ for the recurrence. We identify a class of recurrences for which our algorithms halt for all initial values. We further identify a class of recurrences for which our algorithms can be extended to total ones.

2012 ACM Subject Classification Mathematics of computing → Discrete mathematics

Keywords and phrases holonomic sequences, Positivity Problem, Skolem Problem

Digital Object Identifier 10.4230/LIPIcs.ICALP.2021.99

Category Track A: Algorithms, Complexity and Games

Funding *Joël Ouaknine*: ERC grant AVS-ISS (648701) and DFG grant 389792660 as part of TRR 248 (see `https://perspicuous-computing.science`).
James Worrell: EPSRC Fellowship EP/N008197/1.

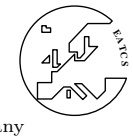
1 Introduction

A sequence $(f(n))_{n\geq 1}$ of real numbers is called *holonomic* or *P-finite* if its terms satisfy an algebraic equation of the form

$$P_r(n)f(n+r) + P_{r-1}(n)f(n+r-1) + \cdots + P_0(n)f(n) = 0,$$

where $P_0, \ldots, P_r \in \mathbb{R}[X]$ are polynomials, not all zero. The number r is called the *order* of the recurrence. When all polynomials $P_r, \ldots, P_0$ are constant we recover the familiar example of ordinary linear recurrence sequences. Alternatively, holonomic sequences are characterised as the coefficients of formal power series which satisfy a non-trivial homogeneous linear ordinary differential equation with polynomial coefficients [10]. Strikingly, holonomic sequences with rational polynomial coefficients can be tested for equality automatically [11]. This allows for automatic proving of highly non-trivial special function identities with numerous applications in mathematics and the sciences [7].

It is natural to ask if holonomic sequences can be automatically tested for inequality as well. This reduces to the problem of deciding whether all terms of a given holonomic sequence $(f(n))_n$ are positive. In full generality this question seems to be completely out of reach, even in the case where the polynomials $P_r, \ldots, P_0$ are all constant. While this problem, often called the Positivity Problem, is widely believed to be decidable in the constant coefficient



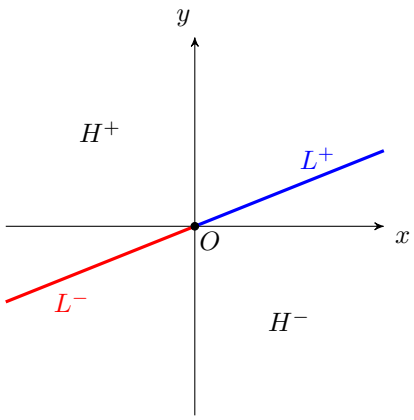



Figure 1 Example of a partition of the space of initial values into the five regions identified in Theorem 1. The asymptotic dynamic behaviour of the signs of a sequence is completely determined by the region containing the initial values.

case, a feasible decision method for linear recurrences of order six or higher would entail major breakthroughs in Diophantine approximation [6]. Nonetheless, decision methods are known for constant coefficient linear recurrence sequences of order up to five.

There is hence some hope that one can obtain decidability results on the Positivity Problem for low-order holonomic sequences as well. In this paper we investigate the problem for sequences satisfying a second-order holonomic recurrence of the form

$$f(n+2) = P(n)f(n+1) + Q(n)f(n) \tag{1}$$

with P non-constant and $0 \leq \deg Q \leq \deg P$. This constitutes arguably the simplest class of non-trivial instances of the Positivity Problem. By a straightforward reduction, our results extend to the larger class of holonomic recurrences of the form

$$R(n)f(n+2) = P(n)f(n+1) + Q(n)f(n)$$

with P non constant, R without positive integer zeroes, and $0 \leq \deg R + \deg Q \leq \deg P$.

We study the possible behaviours that a sequence satisfying a recurrence of the form (1) may exhibit as $n \to \infty$. Up to shifting the recurrence by finitely many terms we may assume that Q does not have any positive integer zeroes. We then show that the plane of initial values $(f(1), f(2)) \in \mathbb{R}^2$ decomposes into five disjoint pieces – the origin O, two rays L^+ and L^-, and two open half-planes H^+ and H^- – such that the behaviour of the sequence $(f(n))_{n\geq 1}$ for large n depends only on the piece that contains the initial values and the signs of the leading coefficients of P and Q. See Figure 1 for a graphical illustration. Depending on these data, unless the sequence is identically zero, it will be eventually strictly positive, strictly negative, or alternating between strictly positive and strictly negative. Moreover, we can compute on each of the five pieces for any initial value a number N such that the sequence $(f_n)_n$ has the described behaviour for all $n \geq N$.

Up to potentially shifting the recurrence by finitely many terms, the line $L = L^+ \cup L^- \cup O$ has a well-defined slope, given by the (necessarily convergent) continued fraction $-\mathrm{K}_{n=1}^{\infty} \frac{Q(n)}{P(n)}$. This allows us to approximate the line L numerically to any given finite precision. We can hence determine, by means of a potentially non-terminating algorithm, if a given pair of rational initial values $(f(1), f(2)) \in \mathbb{Q}^2$ is outside the line and in that case determine the behaviour of the sequence for large n. This yields a partial algorithm for deciding the Positivity Problem and related problems on the class of sequences satisfying recurrences of the form (1).

While we do not obtain a total algorithm for all second-order holonomic recurrences of the form (1), we identify a class of recurrences for which the slope of the line L is an effectively computable rational number. In this case we can extend our algorithm to a total one. One can effectively check if a given recurrence belongs to this class.

Our algorithm is also total for the class of all recurrences such that the slope of the line L is irrational. We establish non-trivial effective criteria that guarantee this.

Related Work. Decidability of the Positivity Problem for second-order holonomic sequences is investigated in [4]. It is shown that for sequences with linear polynomial coefficients, the Positivity Problem reduces to the problem of deciding the equality of certain effectively given quantities, closely related to periods [5] whose equality is conjectured to be decidable [5, Conjecture 1].

Gerhold and Kauers [2] give a partial algorithm for deciding Positivity for general holonomic sequences based on symbolic methods from real algebraic geometry. To the best of our knowledge its precise termination behaviour is not known, even for low-order sequences. Further practical partial algorithms in the spirit of [2] are introduced in [3, 8, 9]. In those papers sufficient termination criteria are given for recurrences of the form

$$P_r(n)f(n+r) + \cdots + P_0(n)(f) = 0$$

with $\deg P_0 = \deg P_r$ and $\deg P_j \leq \deg P_0$ for all $j \leq r$. This situation is disjoint from the one we investigate. Similarly to our main result, one of the algorithms in [3] is shown to terminate for all second-order recurrences of the above form on almost all initial values. All further termination criteria established in the aforementioned papers put restrictions on the eigenvalues of the holonomic recurrence but no restrictions on the initial values. The algorithms may fail to converge for all initial values of a recurrence that does not meet the restrictions on the eigenvalues.

Key contributions. There remains a dearth of algorithmic results on positivity and inequality problems for second-order (and higher) holonomic sequences. The present paper makes two substantial contributions to these outstanding open problems: (i) we identify a large class of second-order holonomic recurrences for which we can precisely characterise all the possible asymptotic behaviours (Theorem 1); and (ii) building upon this, we identify a substantial subclass of holonomic sequences for which we exhibit total algorithms for the Positivity and Skolem problems.

2 Results

Let us first introduce the decision problems we seek to investigate. The Skolem Problem is the problem of deciding for a given recurrence of the form (1) and for given initial values $f(1), f(2)$ if the induced recurrence sequence $(f(n))_n$ has a zero. The Infinite Zero Problem asks if the sequence $(f(n))_n$ thus given has infinitely many zeroes. The Positivity Problem asks if all terms of the sequence $(f(n))_n$ are positive. The Ultimate Positivity Problem is the problem of deciding if there exists an index N such that all terms $f(n)$ with $n \geq N$ are positive.

Our results are best stated for holonomic recurrences that are normalised in the following sense. A second-order holonomic recurrence $f(n+2) = P(n)f(n+1) + Q(n)f(n)$ with P non-constant and $0 \leq \deg Q \leq \deg P$ is said to be *in normal form* if $P, Q \in \mathbb{Z}[X]$ are integer polynomials such that P has a positive leading coefficient, P and Q have no positive real zeroes, $P(n)^2 + 4Q(n) > 0$ for all positive integers n, and there is no prime number p such that p divides all coefficients of P and p^2 divides all coefficients of Q.

For the most part, the above assumptions on P and Q do not present essential restrictions: Given any second-order holonomic recurrence of the form (1) we can effectively compute integers c and N such that the holonomic recurrence $g(n+2) = cP(N+n)g(n+1)+c^2Q(N+n)g(n)$ is in normal form. For any given pair of initial values $f(1), f(2) \in \mathbb{Q}$, we can effectively compute $g(1) = c^{N+1}f(N+1)$ and $g(2) = c^{N+2}f(N+2)$. The sequence $(g(n))_n$ is then equal to $(c^n f(n))_{n>N}$. Thus, the behaviour of $(f(n))_n$ is easily deduced from that of $(g(n))_n$ and the finite sequence $f(1), \dots, f(N)$. In particular, the above mentioned decision problems reduce in this way to their specialisation to recurrences in normal form. However, since in general there may exist initial values for which our algorithm is not guaranteed to terminate, we also need to understand which initial values for the original recurrence get mapped to such ones. We will discuss this below, after we have stated our main results.

It is worth pointing out that our results extend to holonomic recurrences of the form $R(n)f(n+2) = P(n)f(n+1) + Q(n)f(n)$ with P non constant, R without positive integer zeroes, and $0 \leq \deg R + \deg Q \leq \deg P$. Indeed, if $(f(n))_n$ satisfies a recurrence of this form then the sequence $g(n) = R(1) \cdot \dots \cdot R(n)f(n)$ satisfies the recurrence $g(n+2) = R(n+2)P(n)f(n+2) + R(n+2)R(n+1)Q(n)$, which falls within the class we investigate. Note that up to shifting the recurrence appropriately we may assume that $R(n)$ has constant (non-zero) sign, so that the behaviour of $(f(n))_n$ is easily deduced from that of $(g(n))_n$.

Our first result is a complete classification of the possible behaviours that a holonomic recurrence of the form (1) may exhibit for large n. We say that a sequence $(x_n)_n$ of real numbers is *eventually positive* if there exists an $N \in \mathbb{N}$ such that $x_n > 0$ for all $n \geq N$. We say that it is *eventually negative* if there exists an $N \in \mathbb{N}$ such that $x_n < 0$ for all $n \geq N$. We say that it is *eventually alternating* if there exists an $N \in \mathbb{N}$ such that $x_N \neq 0$ and $\operatorname{sgn}(x_{n+1}) = -\operatorname{sgn}(x_n)$ for all $n \geq N$. We say that it is *eventually zero* if there exists an $N \in \mathbb{N}$ such that $x_N = 0$ for all $n \geq N$. In each of these cases we call any admissible choice for N a *witness* for the respective behaviour.

▶ **Theorem 1.** Let $f(n+2) = P(n)f(n+1) + Q(n)f(n)$ be a holonomic recurrence in normal form. Then there exists a partition of $\mathbb{R}^2$ into five pieces, the origin O, two rays L^+ and L^-, and two open half-planes H^+ and H^-, such that for all pairs of initial values $(f(1), f(2)) \in \mathbb{R}^2$ we have:

1. If $(f(1), f(2)) \in O$ then the sequence is constant equal to zero.
2. If $(f(1), f(2)) \in H^+$ then the sequence is eventually positive.
3. If $(f(1), f(2)) \in H^-$ then the sequence is eventually negative.
4. If $(f(1), f(2)) \in L^+$ then the sequence is eventually positive if the leading coefficient Q is negative, and eventually alternating if the coefficient is positive.
5. If $(f(1), f(2)) \in L^-$ then the sequence is eventually negative if the leading coefficient Q is negative, and eventually alternating if the coefficient is positive.

We will call the line $L = O \cup L^+ \cup L^-$ the *critical line* of the holonomic recurrence. We can compute its slope to any given finite precision thanks to the following continued fraction representation:

▶ **Proposition 2.** Let $f(n+2) = P(n)f(n+1) + Q(n)f(n)$ be a holonomic recurrence in normal form. Then we can compute a number N such that for the shifted recurrence $g(n+2) = P(n+N)g(n+1) + Q(n+N)g(n)$, in the notation of Theorem 1, the line $L = O \cup L^+ \cup L^-$ has slope

$$-\operatorname*{K}_{n=N}^{\infty} \frac{Q(n)}{P(n)} = -\cfrac{Q(N)}{P(N) + \cfrac{Q(N+1)}{P(N+1) + \dots}}.$$

Theorem 1 suggests the following computational problem: given a holonomic recurrence in normal form and initial values $(f(1), f(2)) \in \mathbb{Q}^2$, report whether the sequence $(f(n))_n$ thus defined is eventually positive, eventually negative, eventually alternating, or eventually zero and output a witness N for this. Let us call this the *Ultimate Sign Problem*. By Theorem 1 this problem is well-defined. It is clear that the Skolem Problem, the Positivity Problem, the Ultimate Positivity Problem, and the Infinite Zero Problem reduce to this.

Unfortunately we only obtain a partial computability result:

▶ **Theorem 3.** There exists an algorithm which takes as input a holonomic recurrence $f(n+2) = P(n)f(n+1) + Q(n)f(n)$ in normal form, together with a pair $(f(1), f(2)) \in \mathbb{Q}^2$ of rational initial values and halts if and only if $(f(1), f(2)) \notin L^+ \cup L^-$. Upon halting the algorithm reports if $(f(1), f(2))$ is zero, belongs to H^+, or belongs to H^-, and in the latter two cases returns a number N such that the sequence $(f(n))_n$ has constant sign for all $n \geq N$.

Theorem 3 yields a total algorithm for deciding the Infinite Zero Problem and partial algorithms for deciding the Skolem Problem, the Positivity Problem, and the Ultimate Positivity Problem. The set of problem instances where the algorithm does not halt is "small" in the sense that it is contained in a set of codimension one.

While we do not obtain a total algorithm in general, there are special instances for which we do. To describe these instances we need to introduce further concepts. The *companion matrix* of the holonomic recurrence (1) is the matrix

$$M(n) = \begin{pmatrix} 0 & 1 \\ Q(n) & P(n) \end{pmatrix}.$$

Note that we have

$$\begin{pmatrix} f(n) \\ f(n+1) \end{pmatrix} = \prod_{j=1}^{n-1} M(j) \begin{pmatrix} f(1) \\ f(2) \end{pmatrix}.$$

Its n^{th} *characteristic polynomial* is given by

$$z^2 - P(n)z - Q(n).$$

For holonomic recurrences in normal form, the discriminant of this polynomial is by definition strictly positive for all $n \in \mathbb{N}$. Hence the characteristic polynomial has two distinct real roots

$$\lambda_1(n) = \tfrac{1}{2}\left(P(n) + \left(P(n)^2 + 4Q(n)\right)^{1/2}\right)$$

and

$$\lambda_2(n) = \tfrac{1}{2}\left(P(n) - \left(P(n)^2 + 4Q(n)\right)^{1/2}\right).$$

In the case where $\lambda_2(n)$ is a constant function of n we can compute the slope of the line L, yielding a total algorithm.

▶ **Proposition 4.** Let $f(n+2) = P(n)f(n+1) + Q(n)f(n)$ be a holonomic recurrence in normal form. If $\lambda_2(n) = \lambda_2$ is a constant function of n then, in the notation of Theorem 1,

$$L^+ \cup L^- \cup O = \left\{(x, y) \in \mathbb{R}^2 \mid y = \lambda_2 x\right\}.$$

▶ **Corollary 5.** The Ultimate Sign Problem is computable for the class of all holonomic recurrences in normal form which have the additional property that $\lambda_2(n)$ is a constant function of n.

The following criterion allows us to check whether λ_2 is constant, increasing, or decreasing:

▶ **Proposition 6.** Write $P(n) = a_d n^d + \cdots + a_0$, $Q(n) = b_d n^d + \cdots + b_0$ with $a_d > 0$. Let

$$\chi_j = \det \begin{pmatrix} b_d & b_j \\ a_d & a_j \end{pmatrix}$$

for $j = 1, \ldots, d-1$. Let

$$\chi_0 = \det \begin{pmatrix} b_d & b_0 \\ a_d & a_0 \end{pmatrix} + b_d^2/a_d$$

The function λ_2 is either constant, strictly monotonically increasing for sufficiently large n, or strictly monotonically decreasing for large n. It is constant if and only if $\chi_0 = \cdots = \chi_d = 0$. It is decreasing if and only if there exists a j_0 such that $\chi_{j_0} > 0$ and $\chi_j = 0$ for $j > j_0$. It is increasing if and only if there exists a j_0 such that $\chi_{j_0} < 0$ and $\chi_j = 0$ for $j > j_0$.

We also obtain a total algorithm for the Ultimate Sign Problem in the case where the critical line contains no rational points. We collect some sufficient conditions that guarantee this.

▶ **Theorem 7.** Let $f(n+2) = P(n)f(n+1) + Q(n)f(n)$ be a holonomic recurrence in normal form with integer polynomial coefficients $P(n) = a_d n^d + \cdots + a_1 n + a_0$ and $Q(n) = b_d n^d + \cdots + b_1 n + b_0$. Then the critical line L contains no non-trivial rational points if any of the following sufficient conditions is met:

1. $b_d = 0$.
2. $|\operatorname{lcof}(Q)/\operatorname{lcof}(P)| < 1$.
3. $|\operatorname{lcof}(Q)/\operatorname{lcof}(P)| = 1$ and $\lambda_2(n)$ is non-constant, positive and increasing for large n.
4. $|\operatorname{lcof}(Q)/\operatorname{lcof}(P)| = 1$ and $\lambda_2(n)$ is non-constant, negative and decreasing for large n.
5. $|\operatorname{lcof}(Q)/\operatorname{lcof}(P)| = 1$ and $\lambda_2(n)$ is non-constant, positive, decreasing for large n, and

$$\begin{cases} |a_0 + b_0 - 1| < 3a_1 & \text{if } d = 1, \\ |a_{d-1} + b_{d-1}| < (d+2)a_d & \text{otherwise.} \end{cases}$$

6. $|\operatorname{lcof}(Q)/\operatorname{lcof}(P)| = 1$ and $\lambda_2(n)$ is non-constant, negative, increasing for large n, and

$$\begin{cases} |a_0 - b_0 + 1| < 3a_1 & \text{if } d = 1, \\ |a_{d-1} - b_{d-1}| < (d+2)a_d & \text{otherwise.} \end{cases}$$

Finally, let us discuss how the reduction to normal form affects the termination behaviour of our algorithm. Let $f(n+2) = P(n)f(n+1) + Q(n)f(n)$ be a holonomic recurrence of the form (1), and let $g(n+2) = cP(n+N)g(n+1) + c^2Q(n+N)g(n)$ be a recurrence in normal form as above. The map which sends initial values $(f(1), f(2))$ for the original recurrence to the initial values $(c^{N+1}f(N+1), c^{N+2}f(N+2))$ for the recurrence in normal form is a linear map $A\colon \mathbb{R}^2 \to \mathbb{R}^2$. The map A is bijective if and only if Q does not have any positive integer zeroes. If Q does have positive integer zeroes then the kernel of A is one-dimensional. In the cases where we have a total algorithm for the Ultimate Sign Problem for the recurrence in normal form the reduction of course yields a total algorithm. Assume that we only have a

partial algorithm that halts outside the union of the two rays L^+ and L^-. If Q does not have any positive integer zeroes, then applying the partial algorithm for the Ultimate Sign Problem after the reduction yields an algorithm that halts outside the union of the two rays $A^{-1}(L^+)$ and $A^{-1}(L^-)$. Thus, the behaviour of the algorithm is unchanged. If Q has positive integer zeroes then either A sends all initial values outside its kernel into the union of the rays L^+ and L^-, or it sends all such initial values into the union of H^+ and H^-. In the latter case we obtain a total algorithm, but in the former case we obtain an algorithm that only halts on the one-dimensional kernel of A. Thus, in the former scenario the dimension of the set of inputs which lead to termination decreases by one.

Let us illustrate some of our results with the help of a simple example.

▶ **Example 8.** Consider the holonomic recurrence

$$f(n+2) = (n-1)f(n+1) + nf(n).$$

We have $\lambda_1(n) = n$ and $\lambda_2(n) = -1$ for all $n \in \mathbb{N}$. Proposition 4 allows us to easily compute the critical line:

$$L = \left\{(x, y) \in \mathbb{R}^2 \mid x = -y\right\}.$$

Using elementary linear algebra we can explicitly compute the n^{th} term of the sequence:

$$f(n) = (-1)^n \frac{f(1) - f(2)}{2} + \left(\frac{n!}{n+1} + \sum_{k=1}^{n-1} \frac{(-1)^{n-k} k!}{(k+1)(k+2)}\right) \frac{f(1) + f(2)}{2}.$$

We hence have $H^+ = \left\{(x, y) \in \mathbb{R}^2 \mid x > -y\right\}$ and $H^- = \left\{(x, y) \in \mathbb{R}^2 \mid x < -y\right\}$.

Thus, if we fix $f(1) > 0$ and let $f_t(2) = -f(1) + t$ with $t \in [0, 1]$ the sequence with initial values $(f(1), f_t(2))$ is eventually positive for all $t > 0$. For sufficiently small $t > 0$ the sequence will alternate between positive and negative values a finite number $N(t)$ of times before attaining only positive values. We have $N(t) \to \infty$ as $t \to 0$. For $t = 0$ the sequence alternates between positive and negative values forever, a witness for this being given by $N(0) = 1$.

If we let the sequence start at the index $n = 0$ then the matrix product has the following closed form:

$$\prod_{j=0}^{n} M(j) = \begin{pmatrix} 0 & 1 \\ 0 & -1 \end{pmatrix}.$$

Thus, every initial value gets mapped onto the critical line for the same recurrence with starting index $n = 1$. The sequence is alternating for all initial values. Since λ_2 is a constant function our algorithm is total and hence able to detect this.

3 Proof of the Results

3.1 Preliminaries

Consider a second-order holonomic recurrence $f(n+2) = P(n)f(n+1) + Q(n)f(n)$ in normal form. Let $M(n)$ be its companion matrix. Using that $M(n)$ has two distinct real eigenvalues for all n, write $M(n) = S(n)D(n)S(n)^{-1}$, where

$$D(n) = \begin{pmatrix} \lambda_2(n) & 0 \\ 0 & \lambda_1(n) \end{pmatrix}, \qquad S(n) = \begin{pmatrix} 1 & 1 \\ \lambda_2(n) & \lambda_1(n) \end{pmatrix},$$

$$S(n)^{-1} = \frac{1}{\lambda_1(n) - \lambda_2(n)} \begin{pmatrix} \lambda_1(n) & -1 \\ -\lambda_2(n) & 1 \end{pmatrix}.$$

Then we have:

$$M(n)M(n-1)\cdot\cdots M(k) = S(n)D(n)S(n)^{-1}S(n-1)D(n-1)S(n-1)^{-1}\cdot\cdots S(k)D(k)S(k)^{-1}.$$

Intuitively, the products $S(n+1)^{-1}S(n)$ are very close to the identity matrix for large n, but we need to study the error terms precisely. Thus, define real-valued functions $\varepsilon_{i,j}(n)$ by:

$$S(n+1)^{-1}S(n) = \begin{pmatrix} 1+\varepsilon_{1,1}(n) & \varepsilon_{1,2}(n) \\ \varepsilon_{2,1}(n) & 1+\varepsilon_{2,2}(n) \end{pmatrix}.$$

More explicitly:

$$\varepsilon_{1,1}(n) = \frac{\lambda_2(n+1)-\lambda_2(n)}{\lambda_1(n+1)-\lambda_2(n+1)} \qquad \varepsilon_{1,2}(n) = \frac{\lambda_1(n+1)-\lambda_1(n)}{\lambda_1(n+1)-\lambda_2(n+1)}$$
$$\varepsilon_{2,1}(n) = \frac{\lambda_2(n)-\lambda_2(n+1)}{\lambda_1(n+1)-\lambda_2(n+1)} \qquad \varepsilon_{2,2}(n) = \frac{\lambda_1(n)-\lambda_1(n+1)}{\lambda_1(n+1)-\lambda_2(n+1)}$$

We want to study the product $M(n)M(n-1)\cdot\cdots\cdot M(k)$. To this end, define functions $a(k,n), b(k,n), c(k,n)$, and $d(k,n)$ via:

$$\prod_{j=k}^{n} M(j) = S(n)\begin{pmatrix} a(k,n) & b(k,n) \\ c(k,n) & d(k,n) \end{pmatrix} S(k)^{-1}.$$

Define functions `stay-small`, `switch-big`, `switch-small`, and `stay-big` as follows:

$$\texttt{stay-small}(n) = \lambda_2(n+1)(1+\varepsilon_{1,1}(n)) \qquad \texttt{switch-big}(n) = \lambda_2(n+1)\varepsilon_{1,2}(n)$$
$$\texttt{switch-small}(n) = \lambda_1(n+1)\varepsilon_{2,1}(n) \qquad \texttt{stay-big}(n) = \lambda_1(n+1)(1+\varepsilon_{2,2}(n)).$$

A straightforward calculation then shows that we have recursive equations:

$$\begin{aligned} a(k,n+1) &= \texttt{stay-small}(n)a(k,n) + \texttt{switch-big}(n)c(k,n) \\ b(k,n+1) &= \texttt{stay-small}(n)b(k,n) + \texttt{switch-big}(n)d(k,n) \\ c(k,n+1) &= \texttt{stay-big}(n)c(k,n) + \texttt{switch-small}(n)a(k,n) \\ d(k,n+1) &= \texttt{stay-big}(n)d(k,n) + \texttt{switch-small}(n)b(k,n). \end{aligned} \tag{2}$$

By definition we have the following initial values:

$$a(k,k) = \lambda_2(k) \qquad b(k,k) = 0 \qquad c(k,k) = 0 \qquad d(k,k) = \lambda_1(k).$$

The next three lemmas constitute the key steps in the proof of Theorem 1. We defer their technical proof to Section 3.4.

▶ **Lemma 9.**

1. The function $\lambda_1(n)$ is positive, strictly monotonically increasing, and satisfies $\lambda_1(n) = \Theta(P(n))$ as $n \to \infty$.
2. The function $\lambda_2(n)$ is either positive for all n or negative for all n. It is either constant, strictly monotonically decreasing for large n, or strictly monotonically increasing for large n. It satisfies $\lambda_2(n) = \Theta(n^{\deg Q - \deg P})$ as $n \to \infty$.
3. We have $\texttt{stay-big}(n) = \Theta(P(n))$ as $n \to \infty$.
4. We have $\texttt{switch-big}(n) = \Theta(n^{\deg Q - \deg P - 1})$ as $n \to \infty$.
5. We have $\texttt{stay-small}(n) = \Theta(n^{\deg Q - \deg P})$ as $n \to \infty$.
6. If λ_2 is constant then $\texttt{switch-small} = 0$ for all n. Otherwise $\texttt{switch-small} = O(n^{-2})$ and $\texttt{switch-small} = \Omega(n^{1-3\deg P})$ as $n \to \infty$.

▶ **Lemma 10.** We can compute a number $K \in \mathbb{N}$ such that for all $k \geq K$ there exists $N \in \mathbb{N}$ such that $d(k,n) > 0$ for all $n \geq N$.

▶ **Lemma 11.** Let K be as in Lemma 10. Let $k \geq K$ be fixed and $n \geq k+5$ such that $d(k,n') > 0$ for all $n' \geq n$. Then $|a(k,n)/d(k,n)| \in O(1/nP(n))$ and $|b(k,n)/d(k,n)| \in O(1/nP(n))$.

▶ **Lemma 12.** Let K be as in Lemma 10. Let $k \geq K$ be fixed and $n \geq k+3$ such that $d(k,n') > 0$ for all $n' \geq n$. Then the sequence $c(k,n)/d(k,n)$ converges to a limit $L(k)$ as $n \to \infty$. We have $L(k) = O(1/k^2P(k)^2)$ as $k \to \infty$. The number $L(k)$ is equal to zero if λ_2 is constant. If λ_2 is decreasing and positive or increasing and negative then the number $L(k)$ is positive. If λ_2 is increasing and positive or decreasing and negative then the number $L(k)$ is negative. Moreover, for any given $p \in \mathbb{N}$ we can compute a rational number $\tilde{L} \in \mathbb{Q}$ with $|\tilde{L} - L(k)| < 2^{-p}$.

3.2 Proof of Theorem 1

With the asymptotic behaviour of the matrix entries being established, we can study the asymptotic behaviour of the sequence $(f(n))_n$. Using Lemmas 10 and 12 we can compute a number K such that for all $k \geq K$, the number $L(k)$ is defined and $1 - L(k) > 0$. By definition of $M(n)$ we have for all $k \geq K$:

$$\begin{aligned}\begin{pmatrix} f(n) \\ f(n+1) \end{pmatrix} &= \prod_{j=k}^{n-1} M(j) \begin{pmatrix} f(k) \\ f(k+1) \end{pmatrix} \\ &= S(n) \begin{pmatrix} a(k,n-1) & b(k,n-1) \\ c(k,n-1) & d(k,n-1) \end{pmatrix} S(k)^{-1} \begin{pmatrix} f(k) \\ f(k+1) \end{pmatrix}.\end{aligned}$$

By calculating the right hand side explicitly we obtain:

$$\begin{aligned}&(\lambda_1(k) - \lambda_2(k))f(n) = \\ &\quad \big(a(k,n-1)\,(\lambda_1(k)f(k) - f(k+1)) \qquad + b(k,n-1)\,(f(k+1) - \lambda_2(k)f(k)) \\ &\quad + c(k,n-1)\,(\lambda_1(k)f(k) - f(k+1)) \qquad + d(k,n-1)\,(f(k+1) - \lambda_2(k)f(k))\big).\end{aligned}$$

Using Lemma 11 we obtain:

$$(\lambda_1(k) - \lambda_2(k))\tfrac{f(n)}{d(k,n-1)} = \tag{3}$$

$$\tfrac{c(k,n-1)}{d(k,n-1)}\,(\lambda_1(k)f(k) - f(k+1)) + (f(k+1) - \lambda_2(k)f(k)) + O(1/nP(n)) \tag{4}$$

Passing to the limit as $n \to \infty$ we obtain, using Lemma 12:

$$\lim_{n\to\infty} (\lambda_1(k) - \lambda_2(k))\tfrac{f(n)}{d(k,n-1)} = (1 - L(k))f(k+1) + (\lambda_1(k)L(k) - \lambda_2(k))f(k). \tag{5}$$

Let $\ell(k) = (1 - L(k))f(k+1) + (\lambda_1(k)L(k) - \lambda_2(k))f(k)$. Note that this defines a straight line for all k, since $1 - L(k) \neq 0$ by assumption.

Now there are two cases:

1. There exists a $k \geq K$ such that $\ell(k) \neq 0$. In this case the sign $f(n)$ is eventually constant and the same as that of $\ell(k)$. It follows from Lemma 12 that $\ell(k)$ is computable. This together with the estimate (4) with an effective constant for the $O(1/nP(n))$ term allows us to compute an index N such that the sign of $f(n)$ is equal to that of $\ell(k)$ for all $n \geq N$.

2. We have $\ell(k) = 0$ for all $k \geq K$. Then the sequence satisfies the first-order recurrence relation

$$f(k+1) = \frac{\lambda_2(k) - \lambda_1(k)L(k)}{1 - L(k)} f(k)$$

for all $k \geq K$. In particular, if λ_2 is negative for all k then the sequence $(f(n))_{n \geq K}$ is zero or alternating, and if λ_2 is positive then the sign of every sequence element $f(n)$ with $n \geq K$ is equal to that of $f(K)$.

Now, since $Q(n)$ is assumed to have no integer zeroes, the matrices $M(n)$ are non-singular for all n. It follows with the above that if $\ell(n) \neq 0$ then the behaviour of the sequence as $n \to \infty$ is robust under small perturbations, while if $\ell(n) \neq 0$ then the behaviour changes under arbitrarily small perturbations of the initial values. It follows that for all $n \geq K$, the matrix $M(n)$ sends the line $L_n = \{(x, y) \in \mathbb{R}^2 \mid (1 - L(n))y + (\lambda_1(n)L(n) - \lambda_2(n))x\}$ to the line L_{n+1}. Hence $\ell(k) = 0$ for some $k \geq K$ if and only if $\ell(k) = 0$ for all $k \geq K$. Thus, the first case in the above case alternative occurs if and only if $\ell(K) \neq 0$. Also note that since the matrices $M(j)$ are all invertible, if the sequence $(f(n))_n$ is eventually zero then it is everywhere zero. It follows that in the second case alternative above the sequence is either eventually alternating or eventually has constant sign.

Let $h = M(K-1) \cdot \cdots \cdot M(1)$. Let

$$\begin{aligned}
H^+ &= h^{-1}\left(\{(x, y) \in \mathbb{R}^2 \mid (1 - L(K))y + (\lambda_1(K)L(K) - \lambda_2(K))x > 0\}\right) \\
H^- &= h^{-1}\left(\{(x, y) \in \mathbb{R}^2 \mid (1 - L(K))y + (\lambda_1(K)L(K) - \lambda_2(K))x < 0\}\right) \\
L^+ &= h^{-1}\left(\{(x, y) \in \mathbb{R}^2 \mid (1 - L(K))y + (\lambda_1(K)L(K) - \lambda_2(K))x = 0, x > 0\}\right) \\
L^- &= h^{-1}\left(\{(x, y) \in \mathbb{R}^2 \mid (1 - L(K))y + (\lambda_1(K)L(K) - \lambda_2(K))x = 0, x < 0\}\right).
\end{aligned}$$

Theorems 1 and 3 follow.

Proposition 2 is now proved as follows: For $n \geq K$, let $S(n)$ denote the slope of the line $L_n = \{(x, y) \in \mathbb{R}^2 \mid (1 - L(n))y + (\lambda_1(n)L(n) - \lambda_2(n))x\}$. Using that $M(n)$ maps L_n onto L_{n+1} we obtain the equation $S(n) = -\frac{Q(n)}{P(n) - S(n+1)}$. This yields $S(K) = -\mathrm{K}_{m=K}^{\infty} \frac{Q(m)}{P(m)}$, and this is the slope of the critical line of the recurrence shifted by K.

If λ_2 is a constant function of n then $L_n = 0$ for all n, as is readily seen from the recursive equation (2) for $c(n, k)$. Thus, $\ell(K) = y - \lambda_2 x$. Since the vector $(1, \lambda_2)$ is an eigenvector for all matrices $M(1), \ldots, M(K-1)$ we have $h^{-1}(\{(x, y) \mid y - \lambda_2 x = 0\}) = \{(x, y) \mid y - \lambda_2 x = 0\}$. It follows that we have $L^+ \cup L^- \cup O = \{(x, y) \mid y = \lambda_2 x\}$. This establishes Proposition 4. Corollary 5 follows together with the above discussion.

Let us now prove Proposition 6. Write $P(n) = a_d n^d + \cdots + a_0$ and $Q(n) = b_d n^d + \cdots + b_0$ with $a_d > 0$. Then the limit of $\lambda_2(n)$ as $n \to \infty$ is equal to $-b_d/a_d$. This follows for instance from the series representation (10) in Section 3.4. We have seen in Lemma 9 that λ_2 is either constant or strictly monotone. It follows that λ_2 is decreasing or constant if and only if $\lambda_2(n) \geq -b_d/a_d$ for all sufficiently large n, with λ_2 being decreasing if and only if the inequality is strict. By writing out the definition of $\lambda_2(n)$ and applying basic algebra we obtain that this is equivalent to:

$$P(n) + 2b_d/a_d \geq \left(P(n)^2 + 4Q(n)\right)^{\frac{1}{2}}.$$

For sufficiently large n the expressions on both sides are positive, so the inequality is equivalent to the same inequality with both sides squared:

$$P(n)^2 + 4P(n)b_d/a_d + 4b_d^2/a_d^2 \geq P(n)^2 + 4Q(n).$$

This is further equivalent to the inequality:

$$P(n)b_d - a_dQ(n) + b_d^2/a_d \geq 0.$$

Proposition 6 follows.

3.3 Proof of Theorem 7

By the proof of Theorem 1 the critical line is, up to potentially shifting the recurrence, given by the equation

$$(1 - L(1))f(2) + (\lambda_1(1)L(1) - \lambda_2(1))f(2).$$

If the equation has a non-zero rational solution then it has a non-zero integer solution. Thus, assume that the equation has an integer solution $(f(1), f(2))$ with $f(1)$ and $f(2)$ not both zero. Then the recurrence sequence $(f(n))_n$ satisfies

$$f(n+1) = \frac{\lambda_2(n) - L(n)\lambda_1(n)}{1 - L(n)} f(n). \tag{6}$$

If $\deg Q < \deg P$ or $\deg Q = \deg P$ and $|\operatorname{lcof}(Q)| < |\operatorname{lcof}(P)|$ then it follows from Lemma 9 that $|\lambda_2(n)| \to c$ with $0 \geq c < 1$ as $n \to \infty$. It follows from (6) that $f(n) \in o(1)$. But since $(f(n))_n$ is an integer sequence it follows that $f(n) = 0$ for all large n. But then $f(n) = 0$ for all n by Theorem 1. Hence, the only integer solution is $(0, 0)$.

It remains to consider the cases where $\deg Q = \deg P$ and $|\operatorname{lcof}(Q)| = |\operatorname{lcof}(P)|$.

We claim that in the case where λ_2 is negative and decreasing or positive and increasing the sequence $(f(n))_n$ is bounded. In this case $|\lambda_2(n)| < 1$ for all n. It follows from (6) and Lemma 12 that there exists a constant c such that $|f(n)| \leq \prod_{k=1}^{n}(1 + \frac{c}{k^2})|f(1)|$. Boundedness of the sequence follows by taking logarithms on both sides and noting that the sequence $\sum_{n=1}^{\infty} n^{-2}$ converges.

We claim that if λ_2 is positive and decreasing and $d \geq 2$ then $|f(n)| \leq Cn^{\left|\frac{a_{d-1}+b_{d-1}}{a_d}\right|}$. By assumption on λ_2 and Lemma 12 the number $L(n)$ is positive, so that we have for all sufficiently large n:

$$\left|\frac{\lambda_2(n) - L(n)\lambda_1(n)}{1 - L(n)}\right| \leq \lambda_2(n)$$

There exist constants c and c' such that for all sufficiently large n we have:

$$|\lambda_2(n)| \leq 1 + \left|\frac{b_{d-1} + a_{d-1}}{a_d}\right| \frac{1}{n - c'} + \frac{c}{n^2}.$$

It follows that, for sufficiently large M and all $N \geq M$ we have:

$$f(N) \leq |f(M)| \prod_{n=M}^{N} \left(1 + \left|\frac{b_{d-1} + a_{d-1}}{a_d}\right| \frac{1}{n - c'} + \frac{c}{n^2}\right).$$

Taking logarithms on both sides we obtain:

$$\log f(N) \leq \log |f(M)| + \sum_{n=M}^{N} \log \left(1 + \left|\frac{b_{d-1} + a_{d-1}}{a_d}\right| \frac{1}{n - c'} + \frac{c}{n^2}\right)$$

Using $\log(1+x) = x + O(x^2)$ we obtain:

$$\log f(N) \le \log|f(M)| + \sum_{n=M}^{N} \left(\left| \frac{b_{d-1}+a_{d-1}}{a_d} \right| \frac{1}{n-c'} + \frac{c}{n^2} + O(1/n^2) \right) \tag{7}$$

$$\le \log|f(M)| + \left| \frac{b_{d-1}+a_{d-1}}{a_d} \right| \log(N) + C', \tag{8}$$

where C' is a constant.

We have used in (7) that $\sum_{n=1}^{N} \frac{1}{n} \le \log(N) + \gamma + 1$ for all large N, where $\gamma \approx 0.5572\ldots$ is the Euler-Mascheroni constant. Since $\sum_{n=1}^{N} \frac{1}{n}$ diverges, it follows that $\sum_{n=M}^{N} \frac{1}{n} \le \log(N)$ for large M. Now apply the exponential function to both sides of (8):

$$f(N) \le |f(M)| N^{\left| \frac{b_{d-1}+a_{d-1}}{a_d} \right|} e^{C'}.$$

This proves the claim. An analogous argument shows that $f(N) \in O\left(N^{\left| \frac{b_{d-1}+a_{d-1}}{a_d} \right|}\right)$ if λ_2 is negative and increasing and $d \ge 2$. Analogous claims hold for the case that $d = 1$. Now, by assumption we have

$$L(n) = \frac{\lambda_2(n) f(n) - f(n+1)}{\lambda_1(n) f(n) - f(n+1)}. \tag{9}$$

By our previous considerations, the denominator of this expression is in $O(\frac{1}{P(n)})$ if $\lambda_2(n)$ is positive and increasing or if $\lambda_2(n)$ is negative and decreasing. If $\lambda_2(n)$ is, say, positive and decreasing and $d \ge 2$ then the expression is in $O\left(n^{\left| \frac{b_{d-1}+a_{d-1}}{a_d} \right|} P(n)\right)$. Similarly for the other cases we consider. Thus, if $\lambda_2(n)$ is positive and increasing or negative and decreasing, or for instance if $d \ge 2$ and it is positive and decreasing and $\left| \frac{b_{d-1}+a_{d-1}}{a_d} \right| < \deg P + 2$, then the numerator of (9) has to be in $o(1)$.

In other words, we have a sequence of pairs of non-zero integers (x_n, y_n) with $|x_n|, |y_n| \le n^p$ for some positive integer p such that $|\lambda_2(n) x_n - y_n| \to 0$ as $n \to \infty$. We may assume that $p \ge 2$. Now use the series representation (10) of $\lambda_2(n)$ computed in Section 3.4:

$$\begin{aligned}
&|\lambda_2(n) x_n - y_n| \\
&= \left| -x_n \sum_{m=1}^{\infty} \frac{2^{m-1} Q(n)^m}{m!} \prod_{j=1}^{m-1} (1-2j)\, P(n)^{1-2m} - y_n \right| \\
&= \left| x_n \sum_{m=1}^{2p} \frac{2^{m-1} Q(n)^m}{m!} \prod_{j=1}^{m-1} (1-2j)\, P(n)^{1-2m} + y_n + O\left(n^p Q(n)^{2p} / P(n)^{4p-1}\right) \right| \\
&= \left| P(n)^{1-4p} \left(x_n \left(\sum_{m=1}^{2p} \frac{2^{m-1} Q(n)^m}{m!} \prod_{j=1}^{m-1} (1-2j)\, P(n)^{4p-2m} \right) + y_n P(n)^{4p-1} \right) \right| \\
&\qquad + O\left(n^p Q(n)^{2p} / P(n)^{4p-1}\right).
\end{aligned}$$

In order for this to converge to zero we must have

$$x_n \left(\sum_{m=1}^{2p} \frac{2^{m-1} Q(n)^m}{m!} \prod_{j=1}^{m-1} (1-2j)\, P(n)^{4p-2m} \right) + y_n P(n)^{4p-1} \in o(P(n)^{4p-1}).$$

But the left hand side is an integer linear combination of two polynomials of degree $(4p-1)\deg P$. By assumption, their leading coefficients are either equal or additive inverses of each other – depending on the sign of $\lambda_2(n)$. It follows that $x_n = y_n$ or $x_n = -y_n$ for all sufficiently large n. Let us without loss of generality assume that the former holds true.

Then, for all sufficiently large n the lines

$$L_n = \{(x,y) \mid (\lambda_1(n)L(n) - \lambda_2(n))x + (1 - L(n)) = 0\}$$

are all equal to the diagonal $x = y$. Since the matrix $M(n)$ sends the line L_n line onto the line L_{n+1}, it follows that the vector $(1,1)$ is an eigenvector for every $M(n)$. The corresponding eigenvalue must be $\lambda_2(n)$. But since the vector gets mapped to itself it follows that $\lambda_2(n) = 1$ for all n, contradicting our initial assumption.

3.4 Proof of Lemmas 11 and 12

We will expand the terms $a(k,n)$, $b(k,n)$, $c(k,n)$, and $d(k,n)$ into large sums based on the above recursive equations. It will be convenient to describe these sums with the help of a finite automaton. Consider the finite automaton $\mathcal{A}$ over the alphabet $\Sigma = \{\texttt{stay-big}, \texttt{switch-small}, \texttt{stay-small}, \texttt{switch-big}\}$ defined in Figure 2.

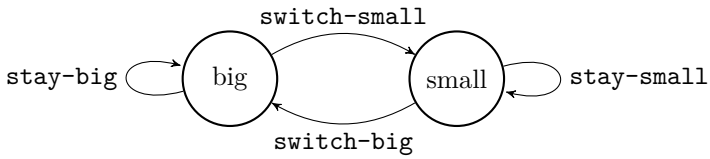


Figure 2 The automaton $\mathcal{A}$.

Let $[\text{small} \to \text{small}] \subseteq \Sigma^*$ denote the set of all words that are accepted by $\mathcal{A}$ with initial state "small" and accepting state "small". Define the sets $[\text{small} \to \text{big}]$, $[\text{big} \to \text{small}]$, and $[\text{big} \to \text{big}]$ analogously.

For each symbol $s \in \Sigma$, let $[\![s]\!] \colon \mathbb{N} \to \mathbb{R}$ be the obvious function associated with it. For a word $w = w_1 \cdot \cdots \cdot w_s$ over the alphabet Σ and $n \geq s$, let $[\![w]\!](n) = [\![w_1]\!](n) \cdot \cdots \cdot [\![w_s]\!](n-s+1)$.

For $k \leq n$, let

$$A(k,n) = [\text{small} \to \text{small}] \cap \Sigma^{n-k} \qquad B(k,n) = [\text{small} \to \text{big}] \cap \Sigma^{n-k}$$
$$C(k,n) = [\text{big} \to \text{small}] \cap \Sigma^{n-k} \qquad D(k,n) = [\text{big} \to \text{big}] \cap \Sigma^{n-k}$$

From the above recursive equations and initial values we obtain for all $n > k$:

$$a(k,n) = \lambda_2(k) \sum_{w \in A(k,n)} [\![w]\!](n-1) \qquad b(k,n) = \lambda_1(k) \sum_{w \in B(k,n)} [\![w]\!](n-1)$$
$$c(k,n) = \lambda_2(k) \sum_{w \in C(k,n)} [\![w]\!](n-1) \qquad d(k,n) = \lambda_1(k) \sum_{w \in D(k,n)} [\![w]\!](n-1).$$

We study the asymptotic behaviour of the quotients $a(k,n)/d(k,n)$, $b(k,n)/d(k,n)$, and $c(k,n)/d(k,n)$. In order to do so, we first need to study the asymptotic behaviour of the functions $\texttt{stay-big}, \texttt{switch-small}, \texttt{stay-small}, \texttt{switch-big}$. In the sequel we will denote these functions by $\texttt{stb}, \texttt{sws}, \texttt{sts}, \texttt{swb}$ for short.

Recall that we have

$$\lambda_1(n) = \tfrac{1}{2}\left(P(n) + \left(P(n)^2 + 4Q(n)\right)^{1/2}\right)$$

and

$$\lambda_2(n) = \tfrac{1}{2}\left(P(n) - \left(P(n)^2 + 4Q(n)\right)^{1/2}\right)$$

The Taylor series expansion of $h(x) = x^{1/2}$ about $x = P(n)^2$ is

$$x^{1/2} = \sum_{m=0}^{\infty} \frac{(x - P(n)^2)^m}{2^m m!} \left(\prod_{j=1}^{m-1}(1-2j)\right) P(n)^{1-2m}.$$

Hence:

$$\left(P(n)^2 + 4Q(n)\right)^{1/2} = \sum_{m=0}^{\infty} \frac{2^m Q(n)^m}{m!} \left(\prod_{j=1}^{m-1}(1-2j)\right) P(n)^{1-2m}. \tag{10}$$

Let

$$\rho(n) = \sum_{m=1}^{\infty} \frac{2^{m-1} Q(n)^m}{m!} \prod_{j=1}^{m-1}(1-2j)\, P(n)^{1-2m}.$$

Then we have

$$\lambda_1(n) = P(n) + \rho(n),$$

and

$$\lambda_2(n) = -\rho(n).$$

Note that the series

$$\sum_{m=1}^{\infty} \frac{2^{m-1} Q(n)^m}{m!} \prod_{j=1}^{m-1}(1-2j)\, P(n)^{1-2m}$$

is majorised by the geometric series $P(n)\sum_{m=1}^{\infty}\left(\frac{4Q(n)}{P(n)^2}\right)^m$. In particular we have

$$\sum_{m=k}^{\infty} \frac{2^{m-1} Q(n)^m}{m!} \prod_{j=1}^{m-1}(1-2j)\, P(n)^{1-2m} = O\left(\frac{Q(n)^k}{P(n)^{2k-1}}\right).$$

Let us now prove Lemma 9.

Proof of Lemma 9. It is clear that $\lambda_1(n)$ is positive and monotonically increasing. It follows easily from the series representation (10) that $\lambda_1(n)$ has the claimed asymptotic behaviour.

If $Q(n)$ is negative for all n then $\left(P(n)^2 + 4Q(n)\right)^{1/2} < P(n)$ and $\lambda_2(n)$ is positive for all n. If $Q(n)$ is positive for all n then $\left(P(n)^2 + 4Q(n)\right)^{1/2} > P(n)$ and $\lambda_2(n)$ is negative for all n. It follows easily from the series representation (10) that $\lambda_2(n)$ has the claimed asymptotic behaviour.

The claimed asymptotic behaviour of the functions `stb`, `swb`, and `sts` is easily verified.

It remains to prove that λ_2 is either constant or monotone and to study the asymptotic behaviour of $\mathtt{sws}$. To this end we compute the derivative of $\lambda_2(z)$ for $z \in \mathbb{R}$:

$$\lambda_2'(z) = \frac{P'(z)(P(z)^2 + 4Q(z))^{1/2} - P(z)P'(z) - 2Q'(z)}{2(P(z)^2 + 4Q(z))^{-1/2}}.$$

Let $A(z) = P'(z)(P(z)^2 + 4Q(z))^{1/2}$ and $B(z) = P(z)P'(z) - 2Q'(z)$. Then $A(z)^2$ and $B(z)^2$ are polynomials. Hence, if the functions $A(z)$ and $B(z)$ are not equal everywhere, then there exists a positive constant c such that $|A(z)^2 - B(z)^2| \geq c$ for all sufficiently large z. Thus,

$$|A(z) - B(z)| = |A(z)^2 - B(z)^2|/|A(z) + B(z)| \geq c/|A(z) + B(z)|.$$

This already establishes that λ_2 is either constant or monotone.

We have $\mathtt{sws} \in \Theta(\lambda_2(n) - \lambda_2(n+1))$. If λ_2 is constant then clearly $\mathtt{sws} = 0$. Assume now that λ_2 is not constant. The upper bound $\mathtt{sws} \in O(n^{-2})$ can be deduced from the easily established fact that if the degree of P and Q is bounded by d, then the degree of $Q(n)P(n+1) - Q(n+1)P(n)$ is bounded by $2d - 2$.

From the series representation (10) we obtain $|A(z)+B(z)| \in O(z^{2\deg P - 1})$ and $2(P(z)^2 - 4Q(z))^{-1/2} \in O(z^{\deg P})$. It follows that $|\lambda_2'(z)| \in \Omega(z^{1-3\deg P})$. Then, by the mean value theorem, $|\lambda_2(n) - \lambda_2(n+1)| \in \Omega(n^{1-3\deg P})$. ◀

The signs of the coefficients $\mathtt{stb}, \mathtt{sws}, \mathtt{swb}, \mathtt{sts}$ can be easily deduced from Lemma 9. They depend on the behaviour of the function $\lambda_2(n)$. Recall that this function is either constant, positive, or negative and either increasing or decreasing. This leads to four possible sign configurations, indicated in Figures 3 - 7.

We will treat the case where λ_2 is constant and the case where λ_2 is non-constant separately. Let us focus on the latter case for now. Let $D(k,n)^+$ denote the set of words in $D(k,n)$ which contain each of the symbols $\mathtt{sws}$ and $\mathtt{sts}$ an even number of times. Let $D(k,n)^-$ denote its complement. Clearly, for every word $w \in D(k,n)^+$, the number $[\![w]\!](n)$ is positive.

The following proposition is trivial but useful when comparing sums over large index sets:

▶ **Proposition 13.** Let A and B be finite sets of positive real numbers. Let $\mu\colon A \to B$ be a function. Assume that $a/\mu(a) < \varepsilon$ for some $\varepsilon > 0$ and $\mu^{-1}(b)$ contains at most c elements. Then

$$\sum_{a \in A} a / \sum_{b \in B} b < c\varepsilon.$$

▶ **Lemma 14.** Assume that λ_2 is non-constant. For all sufficiently large k and $n \geq k$ we have

$$\sum_{w \in D(k,n)^+} [\![w]\!](n-1) > 2 \sum_{w \in D(k,n)^-} |[\![w]\!](n-1)|$$

Proof. Let $D(k,n)_1$ denote the set of words with an odd number of $\mathtt{sws}$. Let $D(k,n)_2$ denote the set of words with an even number of $\mathtt{sws}$. Consider the map $\mu\colon D(k,n)_1 \to D(k,n)_2$ defined as follows: For a word $w \in D(k,n)_1$ there exist unique words p, r such that $w = p \cdot \mathtt{sws} \cdot r$ with $\mathtt{sws}$ not occurring in r. Let $\mu(w) = p \cdot \mathtt{stb}^{|r|+1}$.

For $w = p \cdot \mathtt{sws} \cdot r$ we have

$$\mu^{-1}(\mu(w)) = \left\{ p \cdot \mathtt{sws} \cdot \mathtt{sts}^j \cdot \mathtt{swb} \cdot \mathtt{stb}^{|r|-j-1} \mid j \in \{0, \ldots, |r|-1\} \right\}.$$

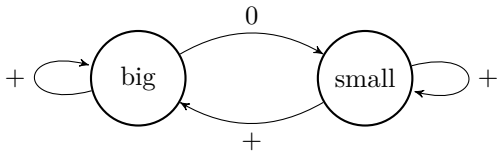


Figure 3 λ_2 constant.

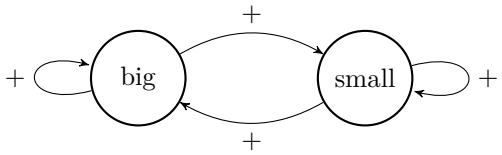


Figure 4 λ_2 positive and decreasing.

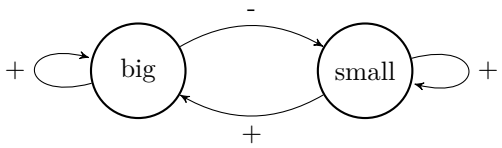


Figure 5 λ_2 positive and increasing.

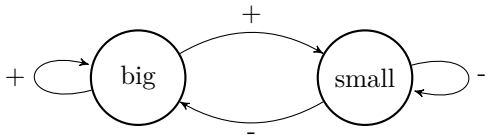


Figure 6 λ_2 negative and increasing.

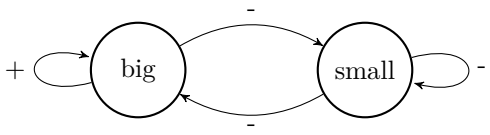


Figure 7 λ_2 negative and decreasing.

We have

$$\begin{aligned}
&\sum_{v\in\mu^{-1}(\mu(w))} |[\![v]\!](n-1)|/|[\![\mu(w)]\!](n-1)| \\
&= \sum_{j=0}^{n-k-|p|-2} \left| \frac{\mathtt{sws}(n-|p|-1)\cdot\prod_{l=1}^{j}\mathtt{sts}(n-|p|-l-1)\cdot\mathtt{swb}(n-|p|-j-2)\cdot\prod_{l=k}^{n-|p|-j-3}\mathtt{stb}(l)}{\mathtt{stb}(n-|p|-1)\cdot\cdots\cdot\mathtt{stb}(k)} \right| \\
&\leq \sum_{j=0}^{n-k-|p|-2} \left| \frac{c/(n-|p|-1)^2\cdot c^j\cdot c/(n-|p|-j-2)}{P(n-|p|-1)\cdot\cdots\cdot P(n-|p|-j-2)} \right|
\end{aligned}$$

for some constant c. Now, for large k we have $P(k) > c$ and we can estimate:

$$\sum_{j=0}^{n-k-|p|-2} \left| \frac{c/(n-|p|-1)^2 \cdot c^j \cdot c/(n-|p|-j-2)}{P(n-|p|-1)\cdot\dots\cdot P(n-|p|-j-2)} \right|$$
$$\leq \frac{c^2}{k(n-|p|-1)^2} \sum_{j=0}^{n-k-|p|-2} \left(\frac{c}{P(k)}\right)^j$$
$$\leq \frac{c^2}{k^3} \frac{P(k)}{P(k)-c}.$$

It follows that

$$\sum_{w\in D(k,n)_1} |[\![w]\!](n-1)| / \sum_{w\in D(k,n)_2} |[\![w]\!](n-1)| \leq \frac{c^2}{k^3}\frac{P(k)}{P(k)-c}.$$

Define a map $\sigma\colon D(k,n)_2 \to D(k,n)^+$ as follows: For a word $w \in D(k,n)_2$, if w contains the symbol $\mathtt{sts}$ an even number of times, let $\sigma(w) = w$. If w contains the symbol $\mathtt{sts}$ an odd number of times then there exists a unique integer $e \geq 1$ and unique words p, q such that

$$w = p \cdot \mathtt{sws} \cdot \mathtt{sts}^e \cdot q$$

and q does not contain the symbol $\mathtt{sts}$. Now, let

$$\sigma(w) = p \cdot \mathtt{stb} \cdot \mathtt{sws} \cdot \mathtt{sts}^{e-1} \cdot q.$$

Then σ is a well-defined map of type $D(k,n)_2 \to D(k,n)^+$. Every word in $D(k,n)^+$ has at most two preimages under σ.

Let $w = p \cdot \mathtt{sws} \cdot \mathtt{sts}^e \cdot q \in D(k,n)_2$ be a word which contains the symbol $\mathtt{sts}$ an odd number of times. Then

$$|[\![w]\!](n-1)| / [\![\sigma(w)]\!](n-1) = \frac{|\mathtt{sws}(n-|p|-1)\cdot\mathtt{sts}(n-|p|-2)|}{|\mathtt{stb}(n-|p|-1)\cdot\mathtt{sws}(n-|p|-2)|} \leq \frac{c}{P(k)}$$

with the same constant $c > 0$ as above. It follows that

$$\sum_{w\in D(k,n)_2} |[\![w]\!](n-1)| / \sum_{w\in D(k,n)^+} [\![w]\!](n-1) \leq 2.$$

Further, letting $D(k,n)_2^-$ denote the set of words in $D(k,n)_2$ which contain the symbol $\mathtt{sts}$ an odd number of times, the same estimate shows that

$$\sum_{w\in D(k,n)_2^-} |[\![w]\!](n-1)| / \sum_{w\in D(k,n)^+} [\![w]\!](n-1) \leq \frac{c}{P(k)}.$$

Thus, in total, we have:

$$\frac{\sum_{w\in D(k,n)^-} |[\![w]\!](n-1)|}{\sum_{w\in D(k,n)^+} [\![w]\!](n-1)}$$
$$\leq \frac{\sum_{w\in D(k,n)_2^-} |[\![w]\!](n-1)| + \sum_{w\in D(k,n)_1} |[\![w]\!](n-1)|}{\sum_{w\in D(k,n)^+} [\![w]\!](n-1)}$$
$$= \frac{\sum_{w\in D(k,n)_2^-} |[\![w]\!](n-1)|}{\sum_{w\in D(k,n)^+} [\![w]\!](n-1)} + \frac{\sum_{w\in D(k,n)_1} |[\![w]\!](n-1)|}{\sum_{w\in D(k,n)_2} |[\![w]\!](n-1)|} \cdot \frac{\sum_{w\in D(k,n)_2} |[\![w]\!](n-1)|}{\sum_{w\in D(k,n)^+} [\![w]\!](n-1)}$$
$$\leq \frac{c}{P(k)} + 2\frac{c^2}{k^3}\frac{P(k)}{P(k)-c}.$$

The right hand side converges to zero as $k \to \infty$. In particular it is smaller than $1/2$ for all sufficiently large k, which yields the claim. ◀

Lemma 14 immediately implies Lemma 10. The computability of the constant K is obtained by observing that we can effectively find all constants that appear in the estimates in the proof. We are now ready to prove Lemmas 11 and 12.

Proof of Lemma 11. We only prove the claim for $b(k,n)$. The claim for $a(k,n)$ is proved analogously.

Let us first assume that λ_2 is non-constant. Then by Lemma 14 it suffices to show that

$$\sum_{w\in B(k,n)} |[\![w]\!](n-1)| / \sum_{w\in D(k,n)} |[\![w]\!](n-1)| = O(1/nP(n)).$$

By Lemma 9 we have $\mathtt{sws} \in \Omega(n^{1-3\deg P})$. Define a map $\mu\colon B(k,n) \to D(k,n)$ as follows: for a word $w = p\cdot q$ in $B(k,n)$ with $|p| = 5$, let $\mu(w) = \mathtt{stb}^4 \cdot s \cdot q$, where $s = \mathtt{sws}$ if $q \in [\text{small} \to \text{big}]$ and $s = \mathtt{stb}$ if $q \in [\text{big} \to \text{big}]$. Then the set $\mu^{-1}(\mu(w))$ contains at most 16 elements. By Proposition 13 it suffices to show that $|[\![w]\!](n-1)|/|[\![\mu(w)]\!](n-1)| \in O(1/nP(n))$.

Consider two cases. The first case is that $w = p\cdot q$ with $p \in [\text{small} \to \text{small}]$, $|p| = 5$. Note that $|[\![p]\!](n-1)|$ is smaller than $[\![\mathtt{sts}^5]\!](n-1)$ or $[\![\mathtt{swb}\cdot\mathtt{stb}^3\cdot\mathtt{sws}]\!](n-1)$. Now, $\mu(\mathtt{sts}^5\cdot q) = \mathtt{stb}^4\cdot\mathtt{sws}\cdot q$, with

$$\begin{aligned}\frac{[\![\mathtt{sts}^5\cdot q]\!](n-1)}{[\![\mathtt{stb}^4\cdot\mathtt{sws}\cdot q]\!](n-1)} &= \frac{\mathtt{sts}(n-1)\cdot\cdots\cdot\mathtt{sts}(n-5)}{\mathtt{stb}(n-1)\cdot\cdots\cdot\mathtt{stb}(n-4)\cdot\mathtt{sws}(n-5)}\\ &= \frac{\Theta(n^{5(\deg P-\deg Q)})}{\Theta(n^{4\deg P}n^{1-3\deg P})}\\ &= O\left(\frac{1}{nP(n)}\right).\end{aligned}$$

It remains to check the other possibility. $\mu(\mathtt{swb}\cdot\mathtt{stb}^3\cdot\mathtt{sws}\cdot q) = \mathtt{stb}^4\cdot\mathtt{sws}\cdot q$, with

$$\frac{[\![\mathtt{swb}\cdot\mathtt{stb}^3\cdot\mathtt{sws}\cdot q]\!](n-1)}{[\![\mathtt{stb}^4\cdot\mathtt{sws}\cdot q]\!](n-1)} = \frac{\mathtt{swb}(n-1)}{\mathtt{stb}(n-1)} = \frac{O(1/n)}{\Theta(P(n))} = O\left(\frac{1}{nP(n)}\right).$$

The second case is that $w = p\cdot q$ with $p \in [\text{small} \to \text{big}]$, $|p| = 5$. Again, $|[\![p]\!](n-1)|$ is smaller than $[\![\mathtt{swb}\cdot\mathtt{stb}^4]\!](n-1)$ or $[\![\mathtt{sts}^4\cdot\mathtt{swb}]\!](n-1)$. We have $\mu(\mathtt{swb}\cdot\mathtt{stb}^4\cdot q) = \mathtt{stb}^5\cdot q$ with

$$\frac{[\![\mathtt{swb}\cdot\mathtt{stb}^4\cdot q]\!](n-1)}{[\![\mathtt{stb}^5\cdot q]\!](n-1)} = \frac{\mathtt{swb}(n-1)}{\mathtt{stb}(n-1)} = O\left(\frac{1}{nP(n)}\right)$$

and $\mu(\mathtt{sts}^4\cdot\mathtt{swb}\cdot q) = \mathtt{stb}^5\cdot q$ with

$$\frac{[\![\mathtt{sts}^4\cdot\mathtt{swb}\cdot q]\!](n-1)}{[\![\mathtt{stb}^5\cdot q]\!](n-1)} = \frac{\mathtt{sts}(n-1)\cdot\cdots\cdot\mathtt{sts}(n-4)\cdot\mathtt{swb}(n-5)}{\mathtt{stb}(n-1)\cdot\cdots\cdot\mathtt{stb}(n-5)} = O\left(\frac{1}{nP(n)}\right).$$

This proves the claim.

It remains to examine the case where λ_2 is constant. In this case, $\mathtt{sws} = 0$, so that

$$d(k,n) = \mathtt{stb}(n-1)\cdot\cdots\cdot\mathtt{stb}(k)\lambda_1(k).$$

We have

$$B(k,n) = \left\{\mathtt{sts}^e\cdot\mathtt{swb}\cdot\mathtt{stb}^{n-k-e} \mid e\in\{0,\dots,n-k\}\right\},$$

so that

$$b(k,n) = \lambda_2(k)\sum_{e=0}^{n-k}\left(\prod_{j=1}^{e}\mathtt{sts}(n-j)\right)\mathtt{swb}(n-e-1)\left(\prod_{j=e+1}^{n-k}\mathtt{stb}(n-j-1)\right).$$

It follows that

$$b(k,n)/d(k,n) = \frac{\lambda_2(k)}{\lambda_1(k)} \sum_{e=0}^{n-k} \frac{\left(\prod_{j=1}^{e} \mathtt{sts}(n-j)\right) \mathtt{swb}(n-e-1)}{\prod_{j=0}^{e} \mathtt{stb}(n-j-1)}.$$

Now, by Lemma 9 there exists a positive constant c such that $|\mathtt{sts}(n-j)| \leq c$, $|\mathtt{swb}(n-e-1)| \leq c/(n-e)$, and $|\mathtt{stb}(n-j-1)| \geq P(n)/c$. Thus:

$$\begin{aligned}
|b(k,n)/d(k,n)| &\leq \sum_{e=0}^{n-k} \frac{c^{2e+1}}{(n-e)P(n)\cdot P(n-1)\cdot\dots\cdot P(n-e)} \\
&\leq \frac{c}{nP(n-k)} \sum_{e=0}^{n-k} \frac{(c^2)^e}{(n-1)\cdot\dots\cdot(n-e)} \\
&\leq \frac{c}{nP(n-k)} \sum_{e=0}^{n-k} \frac{(c^2)^e}{e!} \\
&\leq \frac{c\exp(c^2)}{nP(n-k)} \\
&= O(1/nP(n)).
\end{aligned}$$

◀

Proof of Lemma 12. If $\mathtt{sws} = 0$ then $c(k,n) = 0$ for all k and n, so that the claim is trivial.

Let us hence assume that $\mathtt{sws} \neq 0$. Define a map $\mu\colon C(k,n) \to D(k,n)$ as follows: Let $w = p \cdot q \in C(k,n)$ with $|q| = 3$. If $p \in [\text{big} \to \text{big}]$ then let $\mu(w) = p \cdot \mathtt{stb}^3$. If $p \in [\text{big} \to \text{small}]$ then let $\mu(w) = p \cdot \mathtt{swb} \cdot \mathtt{stb}^2$. One easily verifies that $|[\![w]\!](n-1)/[\![\mu(w)]\!](n-1)| \in O(1/k^2P(k))$ with a constant that does not depend on n. It follows from Lemma 14, Lemma 9, and Proposition 13 that

$$|c(k,n)/d(k,n)| = O(1/k^2P(k)^2) \tag{11}$$

with a constant that does not depend on n. In particular, for all fixed k the sequence $(c(k,n)/d(k,n))_n$ is bounded.

Now, by (2) we have:

$$\begin{aligned}
\frac{c(k,n+1)}{d(k,n+1)} &= \frac{\mathtt{stb}(n)c(k,n) + \mathtt{sws}(n)a(k,n)}{\mathtt{stb}(n)d(k,n) + \mathtt{sws}(n)b(k,n)} \\
&= \frac{\mathtt{stb}(n)c(k,n)}{\mathtt{stb}(n)d(k,n) + \mathtt{sws}(n)b(k,n)} + \frac{\mathtt{sws}(n)a(k,n)}{\mathtt{stb}(n)d(k,n) + \mathtt{sws}(n)b(k,n)}.
\end{aligned}$$

Thus,

$$\begin{aligned}
&\left|\frac{c(k,n+1)}{d(k,n+1)} - \frac{c(k,n)}{d(k,n)}\right| \\
&\leq \frac{\mathtt{swb}(n)c(k,n)}{\mathtt{swb}(n)d(k,n)} \frac{\mathtt{sws}(n)b(k,n)}{\mathtt{stb}(n)d(k,n) + \mathtt{sws}(n)b(k,n)} + \frac{\mathtt{sws}(n)a(k,n)}{\mathtt{stb}(n)d(k,n)} \\
&= O(1)O\left(\tfrac{1}{nP(n)^2}\right) + O\left(\tfrac{1}{nP(n)^2}\right) \\
&= O\left(\tfrac{1}{nP(n)^2}\right).
\end{aligned}$$

It follows that the distance $\left|\frac{c(k,n+m)}{d(k,n+m)} - \frac{c(k,n)}{d(k,n)}\right|$ is majorised by the tail of a convergent series. The convergence of $\left(\frac{c(k,n)}{d(k,n)}\right)_n$ follows.

The asymptotics of $L(k)$ follow from (11). The computability of $L(k)$ to any given finite precision is obtained by observing that we can make all implicit constants in the above estimates into explicit ones, yielding explicit error estimates.

It remains to compute the sign of $L(k)$. The case where λ_2 is constant is trivial. For the other cases, note that essentially the same argument as in Lemma 14 shows that the sum $\sum_{w\in C(k,n)}[\![w]\!](n-1)$ is dominated by the terms with an even number of `sts` and an odd number of `sws`. The number of `swb` is even in all such terms, so that the sign of the sum $\sum_{w\in C(k,n)}[\![w]\!](n-1)$ for sufficiently large n is the sign of $\mathtt{sws}(k)$. Since $c(k,n) = \lambda_2(k)\sum_{w\in C(k,n)}[\![w]\!](n-1)$, the sign of $c(k,n)$ is the sign of $\lambda_2(k)\cdot\mathtt{sws}(k)$. Since $d(k,n)$ is positive for large n it follows that the sign of $L(k)$ is the sign of $\lambda_2(k)\cdot\mathtt{sws}(k)$. ◀

References

1 Jason P. Bell, Stanley N. Burris, and Karen Yeats. On the set of zero coefficients of a function satisfying a linear differential equation. *Mathematical Proceedings of the Cambridge Philosophical Society*, 153(2):235–247, 2012.

2 Stefan Gerhold and Manuel Kauers. A procedure for proving special function inequalities involving a discrete parameter. In Manuel Kauers, editor, *Symbolic and Algebraic Computation, International Symposium ISSAC 2005, Beijing, China, July 24-27, 2005, Proceedings*, pages 156–162. ACM, 2005.

3 Manuel Kauers and Veronika Pillwein. When can we detect that a P-finite sequence is positive? In Wolfram Koepf, editor, *Symbolic and Algebraic Computation, International Symposium, ISSAC 2010, Munich, Germany, July 25-28, 2010, Proceedings*, pages 195–201. ACM, 2010.

4 George Kenison, Oleksiy Klurman, Engel Lefaucheux, Florian Luca, Pieter Moree, Joël Ouaknine, Markus A. Whiteland, and James Worrell. On positivity and minimality for second-order holonomic sequences. *CoRR*, abs/2007.12282, 2020. `arXiv:2007.12282`.

5 Maxim Kontsevich and Don Zagier. Periods. In *Mathematics unlimited—2001 and beyond*, pages 771–808. Springer, Berlin, 2001.

6 Joël Ouaknine and James Worrell. Positivity problems for low-order linear recurrence sequences. In Chandra Chekuri, editor, *Proceedings of the Twenty-Fifth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2014, Portland, Oregon, USA, January 5-7, 2014*, pages 366–379. SIAM, 2014.

7 Marko Petkovšek, Herbert S. Wilf, and Doron Zeilberger. *A = B*. A.K. Peters, 1997.

8 Veronika Pillwein. Termination conditions for positivity proving procedures. In *Proceedings of the 38th International Symposium on Symbolic and Algebraic Computation*, ISSAC '13, page 315–322, New York, NY, USA, 2013. Association for Computing Machinery.

9 Veronika Pillwein and Miriam Schussler. An efficient procedure deciding positivity for a class of holonomic sequences. *ACM Communications in Computer Algebra*, 49(3):90–93, 2015. Extended abstract of the poster presentation at ISSAC 2015.

10 R.P. Stanley. Differentiably finite power series. *European Journal of Combinatorics*, 1(2):175–188, 1980.

11 Doron Zeilberger. A holonomic systems approach to special functions identities. *Journal of Computational and Applied Mathematics*, 32(3):321–368, 1990.

New Sublinear Algorithms and Lower Bounds for LIS Estimation

Ilan Newman
University of Haifa, Israel

Nithin Varma
University of Haifa, Israel

Abstract

Estimating the length of the longest increasing subsequence (LIS) in an array is a problem of fundamental importance. Despite the significance of the LIS estimation problem and the amount of attention it has received, there are important aspects of the problem that are not yet fully understood. There are no better lower bounds for LIS estimation than the obvious bounds implied by testing monotonicity (for adaptive or nonadaptive algorithms). In this paper, we give the first nontrivial lower bound on the complexity of LIS estimation, and also provide novel algorithms that complement our lower bound.

Specifically, we show that for every $\epsilon \in (0, 1)$, every nonadaptive algorithm that outputs an estimate of the LIS length in an array of length n to within an additive error of ϵn has to make $\log^{\Omega(\log(1/\epsilon))} n$ queries. Next, we design nonadaptive LIS estimation algorithms whose complexity decreases as the number of distinct values, r, in the array decreases. We first present a simple algorithm that makes $\tilde{O}(r/\epsilon^3)$ queries and approximates the LIS length with an additive error bounded by ϵn. This algorithm has better complexity than the best previously known adaptive algorithm (Saks and Seshadhri; 2017) for the same problem when $r \ll \operatorname{poly}\log(n)$. We use our algorithm to construct a nonadaptive algorithm with query complexity $\tilde{O}(\sqrt{r} \cdot \operatorname{poly}(1/\lambda))$ that, when the LIS is of length at least λn, outputs a multiplicative $\Omega(\lambda)$-approximation to the LIS length. Our algorithm improves upon the state of the art nonadaptive LIS estimation algorithm (Rubinstein, Seddighin, Song, and Sun; 2019) in terms of the approximation guarantee.

Finally, we present a $O(\log n)$-query nonadaptive erasure-resilient tester for monotonicity. Our result implies that lower bounds on erasure-resilient testing of monotonicity does not give good lower bounds for LIS estimation. It also implies that nonadaptive tolerant testing is strictly harder than nonadaptive erasure-resilient testing for the natural property of monotonicity.

2012 ACM Subject Classification Theory of computation → Streaming, sublinear and near linear time algorithms

Keywords and phrases longest increasing subsequence, monotonicity, distance estimation, sublinear algorithms

Digital Object Identifier 10.4230/LIPIcs.ICALP.2021.100

Category Track A: Algorithms, Complexity and Games

Related Version *Full Version*: `https://arxiv.org/abs/2010.05805`

Funding *Ilan Newman*: Supported by The Israel Science Foundation, grant number 497/17.
Nithin Varma: Supported by The Israel Science Foundation, grant number 497/17 and by the PBC Fellowship for Postdoctoral Fellows by the Israeli Council of Higher Education.

1 Introduction

Estimating the length of the longest increasing subsequence (LIS) in an array is a problem of fundamental importance. For arrays of length n, one can solve this problem exactly in time $O(n \log n)$ using dynamic programming [9] or patience sorting [2]. Approximating the length of the LIS has also been well-studied, and there are several sublinear-time

algorithms [15, 1, 19, 18] for this task. In the approximation task, for a real-valued array A of size n, the goal is to estimate the length of the LIS within an additive error (of ϵn) or multiplicative error. An additive ϵn-approximation algorithm for this problem can also be used to estimate, with the same approximation guarantee, the Hamming distance of A to the closest sorted array[1] (a.k.a. distance to monotonicity).

Early sublinear-time algorithms for LIS estimation [15, 1] provided multiplicative $(2+o(1))$-approximation for the distance to monotonicity, and thereby, additive $\frac{n}{2}$-approximation to the length of the LIS. Saks and Seshadhri [19] made a major improvement to the state of the art, and presented an algorithm that approximates the LIS length to within an additive error of ϵn for arbitrary $\epsilon \in (0,1)$. All these algorithms have query complexity polylogarithmic[2] in n for constant ϵ. Subsequently, Rubinstein, Seddighin, Song, and Sun [18] presented a nonadaptive algorithm that computes a multiplicative $\Omega(\lambda^3)$-approximation to the LIS length, with query complexity $\tilde{O}(\sqrt{n}\cdot\text{poly}(1/\lambda))$, where λ is the ratio of the LIS length to n. In a very recent work (independent and parallel to ours), Mitzenmacher and Seddighin [11] developed a sublinear algorithm for LIS estimation with query complexity $\tilde{O}(n^{1-\Omega(\epsilon)}\cdot\text{poly}(1/\lambda))$ that obtains an approximation ratio of $\Omega(\lambda^\epsilon)$ for arbitrary $\epsilon \in (0,1)$.

Despite the significance of the LIS estimation problem and the amount of attention it has received, there are important aspects of the problem that are not yet fully understood. There is no better lower bound on the query complexity of LIS estimation, for adaptive or nonadaptive algorithms, other than the obvious bound of $\Omega(\log n)$ implied by monotonicity testing [8]. Another issue is to investigate whether the input length n is the right parameter to express the complexity of LIS estimation algorithms. In other words, it is unknown whether there are other input parameters that capture the fine-grained complexity of LIS estimation by making use of the underlying combinatorics of the problem.

In this paper we address both these issues. We prove the first nontrivial lower bound on the query complexity of nonadaptive algorithms for additive error LIS estimation. We also design nonadaptive LIS estimation algorithms whose query complexity is parameterized in terms of the number of distinct values in the input array.

Lower Bound for LIS Estimation. We show that there is no nonadaptive algorithm that approximates the LIS length to arbitrary additive error and has query complexity polylogarithmic in n. Specifically, for arbitrary constant $\epsilon \in (0,1)$, every nonadaptive LIS estimation algorithm that has an additive error bounded by ϵn has to make $\log^{\Omega(\log(1/\epsilon))} n$ queries. Interestingly, our lower bound construction uses ideas from the lower bound [4] on the query complexity of 1-sided error nonadaptive testers for the property of $(k,\ldots,2,1)$-freeness. This is the first lower bound that improves upon the obvious lower bound of $\Omega(\log n)$.

One general approach for proving lower bounds on the complexity of LIS estimation was proposed by Dixit, Raskhodnikova, Thakurta, and Varma [6], who showed that lower bounds for erasure-resilient testing of monotonicity provides lower bounds for estimating the distance to monotonicity up to an additive error. We prove that this method cannot provide a nontrivial lower bound for LIS estimation, by showing a $O(\log n)$-query *nonadaptive* algorithm for erasure-resilient monotonicity testing.

---

[1] It is necessary and sufficient to modify the values that do not belong to an LIS to make the array sorted.

[2] The query complexity of the algorithm by Saks and Seshadhri [19] depends on the approximation parameter ϵ as $O((1/\epsilon)^{1/\epsilon})$ and hence is within aforementioned bound only if ϵ is constant. In particular, the query complexity ceases to be sublinear as as soon as ϵ is $O(1/\log(n))$.

Sublinear Algorithms for LIS Estimation. Our starting point here is to understand the dependence of the query complexity of LIS estimation on the range size of an input array. This is a major direction of study for the simpler problem of monotonicity testing, since the only tight lower bound [8] holds for exponential range. Recently, Pallavoor, Raskhodnikova, and Varma [14], and Belovs [3], gave efficient algorithms for monotonicity testing whose query complexity beats the above lower bound when range size is small. There were no explicit results on LIS estimation for limited range size before our work.[3] In this paper, we give efficient *nonadaptive* LIS estimation algorithms whose complexity is parameterized by r, the number of distinct values in the array, which is always at most the range size. Our algorithms improve upon the state of the art algorithms in both complexity and approximation guarantee when the range is small.

We first show a $\tilde{O}(r/\epsilon^3)$-query nonadaptive algorithm for LIS estimation, of additive error ϵn, for arbitrarily small ϵ. In particular, when the LIS length is a constant fraction of n, our algorithm can be used to get a multiplicative $(1 \pm \epsilon)$-approximation for the LIS length. We add that our algorithm is the only sublinear nonadaptive algorithm giving this approximation guarantee when $r = o(n)$. Furthermore, when $r = o(\log^k n)$ (for an appropriate power k), our algorithm outperforms the adaptive algorithm of Saks and Seshadhri [19], not only in terms of the dependence of query complexity on the input size n, but also in terms of its dependence on the approximation parameter ϵ. Hence, our algorithm bridges the gap between the known $\Omega(\text{poly} \log n)$-query algorithm for the general range and the $O(1)$-query algorithm for the Boolean range.

An additional main result of this paper is a $\tilde{O}(\sqrt{r})$-query nonadaptive algorithm that gives a multiplicative approximation to the LIS length even when the LIS is relatively small. Namely, the algorithm makes $\tilde{O}(\sqrt{r} \cdot \text{poly}(1/\lambda))$ queries and outputs a multiplicative $\Omega(\lambda)$-aproximation to the LIS length, where λ denotes the LIS length normalized by the input length. This is an improvement over the algorithm by Rubinstein, Seddighin, Song, and Sun [18], which makes $\tilde{O}(\sqrt{n} \cdot \text{poly}(1/\lambda))$ nonadaptive queries and outputs a multiplicative $\Omega(\lambda^3)$-approximation to the LIS length. Our algorithm improves upon [18] in terms of approximation guarantee (even in the general case of $r = n$) as well as query complexity (when $r \ll n$), and further, works for any value of r. Finally, the query complexity of our algorithm is always better than that of the recent LIS estimation algorithm by Mitzenmacher and Seddighin [11] that outputs a multiplicative $\Omega(\lambda^\epsilon)$-approximation to the LIS length for arbitrary $\epsilon \in (0, 1)$.[4]

Separating Distance Estimation from Erasure-Resilient Testing. As mentioned before, a general method for proving lower bounds on distance estimation (or tolerant testing [15]) is via proving lower bounds on erasure-resilient testing [6].

Our nonadaptive erasure-resilient tester for monotonicity with complexity $O(\log n)$ and our lower bound on the query complexity of nonadaptive algorithms for LIS estimation imply that nonadaptive tolerant testing is strictly harder than nonadaptive erasure-resilient testing for the natural property of monotonicity, thereby making progress towards solving an open question raised by Raskhodnikova, Ron-Zewi, and Varma [16].

[3] For the case of Boolean arrays, Berman, Raskhodnikova, and Yaroslavtsev [5] showed that one can approximate the LIS length to within an additive error of ϵn by making $O(1/\epsilon^2)$ queries.

[4] We point out that the LIS estimation algorithm of Mitzenmacher and Seddighin [11] uses the algorithm of Rubinstein et al. [18] as a subroutine. By using our algorithm instead, the query complexity of the algorithm of Mitzenmacher and Seddighin [11] can be improved.

1.1 Discussion of Results and Overview of Techniques

In this section, we state our results more formally, and provide an overview of the techniques used to prove them. We use ideas from [18], [12] and [4]. Given a real-valued array A of length n, an LIS in A is the longest nondecreasing sequence of values in A. In other words, the LIS is a largest cardinality set $\mathcal{L}$ of indices such that for $u, v \in \mathcal{L}$, we have $u < v$ if and only if $A[u] \leq A[v]$. We abuse notation and also use the term LIS to denote $|\mathcal{L}|$ when this is clear from the context. A real-valued array of length n can be equivalently viewed as a function from $[n]$ to the reals. Adopting this view, we use the term *monotone array* to refer to a sorted array. Throughout, we denote by r, the number of (or a guaranteed upper bound on) distinct values in the array. That is, $r = |R|$ for $R = \{A[i] : \ i \in [n]\}$. Thus, for the unrestricted case it is assumed that $r = n$.

1.1.1 Lower bound on the query complexity of nonadaptive LIS estimation algorithms

Our first result proves that there is no nonadaptive algorithm that approximates the LIS length in an array of length n to within an additive error of ϵn and has query complexity polylogarithmic in n, for arbitrary constant $\epsilon \in (0, 1)$.

▶ **Theorem 1.1.** *For every $\epsilon \in (0, 1)$, every nonadaptive algorithm that on an array A of length n, outputs an additive ϵn-approximation to the length of the LIS in A, has to make $\log^{\Omega(\log(1/\epsilon))} n$ queries.*

We note that this is the first lower bound on LIS estimation that is not directly implied by the lower bound for testing monotonicity [8].

To prove our lower bound, we construct two distributions with different LIS lengths such that every deterministic nonadaptive algorithm distinguishing the distributions with probability at least 2/3, has query complexity $\log^{\omega(1)}(n)$. More specifically, for every natural number h, we construct distributions $\mathcal{D}_0^{(h)}$ and $\mathcal{D}_1^{(h)}$ that are supported on inputs whose LIS lengths differ by $\exp(-h)$. We then prove that every deterministic nonadaptive algorithm that takes input from the union of the supports of $\mathcal{D}_0^{(h)}$ and $\mathcal{D}_1^{(h)}$, and aims to correctly identify the distribution from which the input is taken, either fails for most inputs or makes $\Omega(\log^h n)$ queries. Interestingly, our lower bound construction uses ideas from the lower bound of Ben-Eliezer, Canonne, Letzter, and Waingarten [4] on the query complexity of 1-sided error nonadaptive testers for the property of $(k, \ldots, 2, 1)$-freeness, where an array A of length n is $(k, \ldots, 2, 1)$-free if there are no k indices $i_1 < i_2 < \cdots < i_k$ such that $A[i_1] > A[i_2] > \cdots > A[i_k]$.

Using reductions from erasure-resilient testing

As mentioned before, a general method for proving lower bounds on distance estimation is via proving lower bounds on erasure-resilient testing [6].

▶ **Definition 1.2** (Erasure-resilient monotonicity tester). *Given $\epsilon, \alpha \in (0, 1)$ and a real-valued array A containing at most α-fraction of* erased *values*[5]*, the goal of an α-erasure-resilient ϵ-tester for monotonicity is to determine whether A can be completed to a monotone array or whether every completion of A has Hamming distance at least ϵn to monotonicity.*

[5] Erasures are made adversarially before the tester makes its queries and the tester is unaware of the location of the erasures. A tester that queries the value at an erased location is returned a special symbol $\perp$.

Dixit, Raskhodnikova, Thakurta and Varma [6] observed that the complexity of erasure-resilient (ER) testing a property, falls in between the complexity of standard testing the property and estimating the distance to that property (with additive error). Hence, a lower bound on the complexity of ER testing monotonicity implies the same lower bound for estimating the LIS length up to an additive error. The only previously known ER tester for monotonicity [6] is adaptive and has query complexity $O(\log(n)/\epsilon)$. Hence, a nontrivial lower bound for (adaptive) LIS estimation cannot be obtained this way.

We present a *nonadaptive* ER tester that makes $O(\log n)$ queries and works for all fraction of erasures. This makes the results on ER testing monotonicity tight, and also shows that one cannot obtain a lower bound for LIS estimation via ER testing.

▶ **Theorem 1.3.** *Let $\epsilon, \alpha \in (0,1)$ such that $\alpha + \epsilon < 1$. There exists a nonadaptive α-erasure-resilient ϵ-tester for monotonicity that makes $O\left(\frac{\log n}{\epsilon^2} + \frac{1}{\epsilon^3}\right)$ queries for n-length arrays.*

The ER testers designed by Dixit et al. [6] for various properties, are all either adaptive, or obtained by repeating a (standard) tester that makes independent and uniformly distributed queries. Our tester is different, and is in this sense, the first nontrivial nonadaptive ER tester for a natural property. Consider an array A of length n with at most α fraction of erasures, where $\alpha \in [0,1)$. Our tester samples an index $s \in [n]$ uniformly at random and does a randomized binary search for s on the array as if it were monotone. It queries the array values on these indices, and looks for violations to monotonicity on the search path to s. In case there are no erasures, this is a good strategy to detect a violation to monotonicity [7]. However, when values at a constant fraction of indices are erased, it could be the case that most of the values on the search path are erased. We show that a slightly modified version of this tester can be used for testing monotonicity. Specifically, our tester, in addition to querying the values along the binary search path, also queries the indices in a small constant-sized interval around the search point s. To analyze this modified tester, we rely on a combinatorial lemma by Newman, Rabinovich, Rajendraprasad, and Sohler [12]. A nonerased index $x \in [n]$ is γ-deserted for $\gamma \in (0,1)$ if there exists an interval $I \subseteq [n]$ such that $x \in I$ and at most γ fraction of the values in I are nonerased. Roughly speaking, the lemma implies that the fraction of γ-deserted indices in A is proportional to $\gamma \cdot \alpha$. Using this, we are able to argue that, with high probability, the index s that we sample as the search point is not γ-deserted (for an appropriate choice of γ) and that it forms a violation with enough other nonerased indices, so as to ensure a high probability of success.

1.1.2 Parameterized and nonadaptive algorithms for LIS estimation

We present efficient nonadaptive LIS estimation algorithms. The novelty is that we parameterize the complexity of LIS estimation algorithms in terms of the number of distinct values r in an array. We first show an LIS estimation algorithm with query complexity $\tilde{O}(r)$.

▶ **Theorem 1.4.** *There exists a nonadaptive algorithm that, given a real-valued array A of length n containing at most r distinct values, and a parameter $\epsilon \in (0,1)$, makes $\tilde{O}(r/\epsilon^3)$ queries and outputs, with probability at least $2/3$, an estimate for the LIS size that is accurate to within additive ϵn-error. Moreover, the queries of the algorithm are uniformly and independently distributed, and the algorithm runs in time $\tilde{O}(r/\epsilon^3)$.*

We mention that the approximation guarantee provided by the algorithm is quite strong and holds even for non-constant error parameter ϵ. It matches the approximation guarantee of the adaptive LIS estimation algorithm by Saks and Seshadhri [19], which makes polylog(n)

queries when $\epsilon = \theta(1)$. In particular, when the length of the LIS $\mathcal{L}$ is a constant fraction of n, our algorithm can be used to get a multiplicative $(1 \pm \epsilon)$-approximation for the LIS length. We add that our algorithm is the only nonadaptive sublinear algorithm giving this approximation guarantee as soon as $r = o(n)$. Furthermore, when $r = o(\log^k n)$ (for an appropriate power k), our algorithm performs much better than the algorithm of Saks and Seshadhri, not only in terms of the dependence of query complexity on the input size n, but also in terms of the dependence on the approximation parameter ϵ.

The high level idea of the algorithm is that it is enough to restrict attention to special subarrays that are *dense* and *nice*, as elaborated in the following. Let $\mathcal{L}$ be a fixed unknown LIS in the input array. A subarray is dense if a constant fraction of its indices belong to $\mathcal{L}$, and it is nice if the LIS takes at most one distinct value in the subarray. Informally, we divide the array into $O(r/\epsilon)$ subarrays. This will make most dense subarrays nice with respect to $\mathcal{L}$ (for an appropriate density parameter). We then sample $O(\log r)$ indices in each subarray to find the values that are "typical" in each subarray.

Our goal is to output as an estimate for $|\mathcal{L}|$, the size of $\mathcal{L}'$, which is the restriction of $\mathcal{L}$ to such typical values. This will naturally be an underestimate, but with a small additive error. To estimate the size of $\mathcal{L}'$, we consider all possible increasing sequences of the typical values, taking one value from each subarray. Since most subarrays are nice, the size of $\mathcal{L}'$ restricted to such a sequence of values is quite close to $|\mathcal{L}'|$. Finally, for a given nice subarray A_i, the largest subsequence in A_i that takes one given value v can be easily determined – this is just the distance to the array taking the value v everywhere.

Next, we use the above $\tilde{O}(r)$-query algorithm to obtain a nonadaptive LIS estimation algorithm with query complexity $\tilde{O}(\sqrt{r})$.

▶ **Theorem 1.5.** *There exists a nonadaptive algorithm that, given a real-valued array A of length n containing at most r distinct values and $|\mathrm{LIS}(A)| = \lambda \cdot n$, makes $\tilde{O}(\sqrt{r} \cdot \mathrm{poly}(1/\lambda))$ queries and outputs, with probability at least $2/3$, an estimate est such that $\Omega(\lambda \cdot |\mathrm{LIS}(A)|) \leq \mathsf{est} \leq O(|\mathrm{LIS}(A)|)$. Moreover, the algorithm runs in time $\tilde{O}(r \cdot \mathrm{poly}(1/\lambda))$.*

As mentioned before, this result is an improvement over a recent LIS estimation algorithm by Rubinstein, Seddighin, Song and Sun [18], in terms of the approximation guarantee. Additionally, the complexity of our algorithm improves as the number of distinct values in the input array decreases. Another advantage of our algorithm (also that of [18]) is that its query complexity is sublinear, even if λ is sub-constant.

Our $\tilde{O}(\sqrt{r})$-query nonadaptive algorithm is somewhat complicated. In the following, we present a high-level description of the algorithm. We denote the input array by A and use $\mathcal{L}$ to denote a fixed LIS in A. We visualize the array values as points in an $r \times n$ grid G_n. The vertical axis of G_n represents the range R of the array and is labeled with the at most r distinct array values in increasing order and the horizontal axis is labeled with the indices in $[n]$. We refer to an index-value pair in the grid as a point. The grid has n points, to which we do not have direct access. We use queries to the array to form some *approximate* picture of the location of points in this grid, and use it to estimate $|\mathcal{L}|$.

The main idea is to build, in $\tilde{O}(\sqrt{r})$ queries, a data structure that possesses enough information to compute an estimate est, which is a lower bound on $|\mathcal{L}|$ and is also a reasonably good approximation. Roughly speaking, the first step in building this data structure is the following. We divide the $r \times n$ grid G_n into y^* rows and x columns that partitions G_n into a $y^* \times x$ grid G' of boxes, where $y^* = \Theta(\sqrt{r})$ and $x = \Theta(\sqrt{r})$. Specifically, we divide the interval $[n]$ into x contiguous subarrays. For $i \in [x]$, let D_i denote the i-th subarray. Additionally, we divide the range R into y^* contiguous intervals of array values, where for $j \in [y^*]$, we use I_j to denote the j-th interval when the intervals are sorted in the nondecreasing order of values. The set of boxes in G' is then $\{(D_i, I_j) : \ i \in [x], j \in [y^*]\}$.

For simplicity, we assume that $r = n$ for the rest of the high-level description. The $y^* \times x$ grid of boxes G' induces a poset on the y^*x boxes, which is similar to the natural poset defined on G_n. Namely, for two boxes in G' (or for two points in G_n), we have $(D_i, I_j) \preceq (I_t, D_s)$ (or $(i, j) \leq (t, s)$) if $i \leq s$ and $j \leq t$. The points in $\mathcal{L}$ form a chain in the above poset in G_n. Conversely, each chain in the poset G_n forms an increasing subsequence in the array A. Further, the boxes in G' through which $\mathcal{L}$ passes also forms a chain in the poset in G'. On the other hand, every chain of boxes in the poset in G' induces a number of chains in the poset in G_n, but of possibly quite different lengths. Our strategy is to find a small collection of chains in the poset in G' that cover all boxes through which the fixed $\mathcal{L}$ passes, and then to estimate the length of an LIS in each of these chains of boxes.

Let $I \subseteq R$ be a subset of the range R of values and B be a subarray of A. The density of the box (B, I), denoted by $\mathsf{den}(B, I)$, is defined to be the fraction of indices in the subarray B whose values belong to the interval I. In other words, for each box $(D_i, I_j) \in G'$, its density $\mathsf{den}(D_i, I_j)$ is the fraction of indices in the subarray D_i whose values land in the interval I_j. For $\beta < 1$, a box (D_i, I_j) is said to be β-dense, if $\mathsf{den}(D_i, I_j) \geq \beta$. There can be at most $\frac{1}{\beta}$ boxes that are β-dense in any particular subarray D_i.

Suppose that we know (a good approximation of) the density of every box in G' (this is what we require from our data structure, and this will be achieved via sampling). Then, we may restrict our attention to the at most x/β dense boxes in G' and compute the LIS only in the corresponding part of G_n. This is obviously an underestimate of the size of $\mathcal{L}$, but one that can be afforded; deleting every box that is not β-dense from the chain of boxes that $\mathcal{L}$ passes through causes the deletion at most βn points from $\mathcal{L}$.

We note that the same global idea is also used in the algorithms of [18] and (implicitly) also of [19], but in a completely different setting (and grid sizes) which makes the first one weaker in term of approximation guarantees, and the second one necessarily adaptive.

Next, in order to further reduce the number of possible chains of boxes in which we need to compute LIS, we note that we can delete large antichains of boxes from G', while not decreasing the LIS size by much. For this, we first consider a finer partition of each dense box into dense cells of nearly equal densities, and then define a poset on the set of all dense cells in the whole array. We then remove large antichains from this latter poset and argue that the removal of dense cells participating in these antichains does not *hurt* the LIS too much. Finally, by using Dilworth's theorem, we are able to obtain a collection of a *constant* number of chains in G', that covers the restriction of $\mathcal{L}$ to the undeleted boxes.

The next idea is to estimate the LIS in each of the constantly many remaining chains. This results in a loss of a multiplicative constant factor in the LIS size estimation.

At this point, we have reduced the problem to the estimation of the LIS in a given fixed chain of β-dense boxes in G'. Such a chain can be partitioned into two chains, one that contains only strictly horizontal chains on disjoint subarrays, and the other that contains only strictly vertical chains on disjoint interval ranges (see Figure 3). We will estimate the LIS in each, losing possibly another multiplicative 2-factor, which we are prepared to accept.

The final idea is the following. For the vertical going chain, one can just sample a constant number of vertically going subchains, estimate the LIS length in each one of them, and use these estimates to estimate the LIS length in the vertical chain. By the Hoeffding bound, this will be a good approximation. When $r = n$, estimating the LIS in a single vertical going subchain is trivial; we just query all $n/x = \tilde{O}(\sqrt{n})$ points in the subarray D_i corresponding to that subchain. For smaller r, this is not possible, and what we do is to reduce to the algorithm implied in Theorem 1.4, using the fact that most vertically going chains span a small range (this later fact will have to be argued from the way the data structure is formed).

For horizontally going chains, we will need a bit more from our data structure. The partition G' of G_n will be such that every layer formed by $i \in [y^*]$ contains either a small fraction of points from $\mathcal{L}$, or it contains only one range value. This is the only place in which we actually make use of the fact that $y^* = \Omega(\sqrt{n})$, which lower bounds the query complexity of the algorithm. Having this guarantee on the grid G', it would have been enough to sample a constant number of horizontally going subchains, and estimate the LIS within. Further, by the guarantee above, each horizontal layer in G' contains only a small number of distinct values. This implies that we could again employ our algorithm of Theorem 1.4. However, this will make the whole algorithm adaptive (as one has to "locate" the horizontal segments). Instead, we show that we can concentrate on short (spanning a constant number of boxes) subchains, which will allow us to employ the algorithm given by Theorem 1.4 nonadaptively.

1.1.3 Separating erasure-resilient testing from tolerant testing

Tolerant testing is a generalization of property testing [17, 10] defined by Parnas, Ron and Rubinfeld [15]. Specifically, a $(\delta, \epsilon + \delta)$-tolerant tester of monotonicity distinguishes, with probability at least $2/3$, between the cases that the distance of A to monotonicity is less than δn and at least $(\epsilon + \delta)n$, where $\epsilon, \delta \in (0, 1)$.

It has been observed [15] that a tolerant tester for a property is equivalent to an algorithm for estimating the distance to that property with an additive error guarantee. Hence, the task of estimating the LIS up to an additive error is equivalent to tolerant monotonicity testing. This allows us to restate Theorem 1.1 in terms of tolerant testing as follows.

▶ **Theorem 1.6.** *For every $\epsilon \in (0, 1)$, there exists a constant $\delta \in (0, 1)$ such that every nonadaptive 2-sided error $(\delta, \delta + \epsilon)$-tolerant tester of monotonicity has query complexity $\log^{\Omega(\log(1/\epsilon))} n$.*

Theorem 1.6 and Theorem 1.3 together imply that for the property of monotonicity, nonadaptive tolerant testing is strictly harder than nonadaptive ER testing, and also significantly less efficient than adaptive tolerant testing. Our results make progress towards answering the open question raised by Raskhodnikova, Ron-Zewi, and Varma [16] on the existence of natural properties for which one can show a separation between tolerant testing and ER testing in terms of query complexity.

1.2 Organization

We set our notations in Section 2. The proof for an important special case of our lower bound on the query complexity of nonadaptive LIS estimation (Theorem 1.6) is presented in Section 3 and the proof in its full generality is deferred to the full version [13]. Our nonadaptive and parameterized algorithm for multiplicative error LIS estimation (Theorem 1.5) and its analysis are presented in Section 4. Our algorithm for additive error LIS estimation (Theorem 1.4), and our nonadaptive erasure-resilient monotonicity tester (Theorem 1.3) can both be found in the full version [13]. All omitted proofs can also be found in the full version [13].

2 Notations and Preliminaries

For a natural number n, we use $[n]$ to denote the set $\{1, 2, \ldots, n\}$. For a real-valued array A of length n, we use $A[i]$ to denote the i-th entry of A for $i \in [n]$. For $x \leq y \in [n]$ we denote by $[x, y]$ the set $\{x, x + 1, \ldots, y\}$. The array A is monotone if for every two indices $u, v \in [n]$

such that $u < v$, we have $A[u] \leq A[v]$. If A is not monotone, two indices $u, v \in [n]$ are said to violate monotonicity if $u < v$ and $A[u] > A[v]$. For $\epsilon \in (0, 1)$, we say that A is ϵ-far from monotone if the values on at least $\epsilon \cdot n$ indices need to be modified to get a monotone array. A is ϵ-close to monotone if there is a way to modify the values on fewer than $\epsilon \cdot n$ indices to get a monotone array. For a parameter $\alpha \in [0, 1)$, we say that A is α-erased, if at most α fraction of the array values evaluate to a special symbol $\perp$. An assignment of values to the erased points in an array is called a completion. An α-erased array is monotone if there exists a completion that is monotone; it is ϵ-far from monotone if every completion is ϵ-far from monotone. We assume that algorithms accesses an input array A via an oracle; that is when the algorithm makes a query $i \in [n]$, the oracle returns a special symbol $\perp$ if the array value at index i is erased, and $A[i]$ otherwise. An algorithm is *adaptive* if its queries depend on the answers to its previous queries, and is *nonadaptive* otherwise. A partially ordered set (poset) is a set $\mathcal{P}$ associated with a reflexive, transitive, antisymmetric order relation $\preceq$ on its elements. We denote the poset by $\langle \mathcal{P}, \preceq \rangle$. A chain in $\langle \mathcal{P}, \preceq \rangle$ of length k is a sequence of elements $x_1 \preceq x_2 \preceq \cdots \preceq x_k$. An antichain is a set $S \subseteq \mathcal{P}$ such that for $u, v \in S$ neither $u \preceq v$ nor $v \preceq u$.

3 Lower Bounds for Nonadaptive LIS Estimation

In this section, we prove our lower bounds (Theorem 1.1) on the query complexity of 2-sided error nonadaptive algorithms for estimating the distance of real-valued arrays of length n from monotonicity up to an additive error of $\epsilon \cdot n$ for some constant $\epsilon \in (0, 1)$. Equivalently, our lower bounds also hold for algorithms that $(\delta, \epsilon + \delta)$-tolerant test monotonicity for constants $\delta, \epsilon \in (0, 1)$. Interestingly, our lower bounds use ideas from the lower bound on the query complexity of 1-sided error nonadaptive testers for the property of $(k, \ldots, 2, 1)$-freeness [4], where an array A of length n is $(k, \ldots, 2, 1)$-free if there are no k indices $i_1 < i_2 < \cdots < i_k$ such that $A[i_1] > A[i_2] > \cdots > A[i_k]$.

An algorithm is said to be *comparison-based* if its decisions are based only on the ordering relation between the queried values, and not on the values themselves. The following Lemma 3.1, which follows from the work of Fischer [8], states that it is enough to restrict our attention to comparison-based algorithms.

▶ **Lemma 3.1** ([8]). *There is an optimal comparison-based algorithm for computing an additive ϵn-approximation to the LIS in real-valued arrays of length n for all constant $\epsilon \in (0, 1)$.*

Even though Fischer [8] proves the above statement in the context of testing monotonicity in the standard model, his proof also works for the case of tolerant testing monotonicity, and in turn for LIS estimation. In the rest of this section, we restrict our attention to comparison-based algorithms for LIS estimation.

3.1 An $\Omega(\log^2 n)$ Lower Bound

As a starting point, we prove an $\Omega(\log^2 n)$ lower bound. Throughout this section, we assume that n is of the form 2^{2^x} for some natural number x. We prove the lower bound using Yao's method. Specifically, we describe two distributions $\mathcal{D}_0$ and $\mathcal{D}_1$ over real-valued arrays of length n, with different distances to monotonicity (Lemma 3.2), and show that every deterministic nonadaptive comparison-based algorithm distinguishing these distributions with probability at least $2/3$, has to make $\Omega(\log^2 n)$ queries (Lemma 3.3).

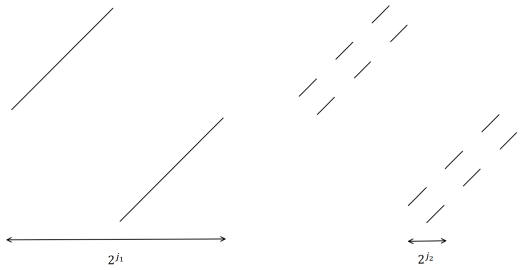


Figure 1 The relative values in j_1-blocks of $\mathcal{D}_0$ either look like the left or right diagrams above, with equal probability.

For ease in describing our distributions, we first define some notation. We think of the indices of an array of length n as the leaves of an ordered binary tree $\mathcal{T}$ of height $\log(n)+1$. We associate bit positions in the $\log n$-bit representation of the numbers in $[n]$ with the non-leaf nodes of $\mathcal{T}$. The root is associated with the most significant bit (or the bit position $\log n$). Every node at distance $i \in [\log(n)-1]$ from the root is associated with bit position $\log(n)-i$. The bit position associated with a node is also referred to as its *level* (and the level of the root is $\log n$). The edges connecting a node with its left and right children are labeled 0 and 1, respectively. Clearly, the string obtained by concatenating all the edge labels on a root-to-leaf path in $\mathcal{T}$ gives the binary representation of the index corresponding to the leaf. For two indices $x, y \in [n]$ (which are, by definition, the leaves in $\mathcal{T}$), we use $\mathsf{LCA}(x,y)$ to denote the lowest common ancestor of x and y in $\mathcal{T}$. For $j \in [\log n]$ there are obviously $n/2^j$ subtrees of $\mathcal{T}$, each rooted at level j. In a left to right ordering, these $n/2^j$ subtrees partition $[n]$ into blocks of size 2^j. The "the ℓ-th j-block" is the ℓ-th block from left with size 2^j.

The distributions $\mathcal{D}_0$ and $\mathcal{D}_1$

We first describe the steps that are common to constructing the distributions $\mathcal{D}_0$ and $\mathcal{D}_1$. Sample numbers $j_1, j_2 \in [\log n]$ such that $j_1 < \log(n) - 14$, and $j_2 < j_1 - 14$. We refer to the numbers j_1, j_2 as the *scales* of the distributions.

We start with the monotone array A in which $A[u] = u$ for all $u \in [n]$. Swap the array values between the left and right halves of every j_1-block. See the left part of Figure 1 to see how the relative values in each j_1-block of the array will look like at this point. For $\ell \in [n/2^{j_1}]$, let B_ℓ denote the ℓ-th j_1-block.

- **Distribution** $\mathcal{D}_0$: Independently for each $\ell \in [n/2^{j_1}]$:
  1. with probability $\frac{1}{2}$, for each j_2-block inside B_ℓ, swap the array values between the left and right halves of that j_2-block.
- **Distribution** $\mathcal{D}_1$: Independently for each $\ell \in [n/2^{j_1}]$:
  1. with probability $\frac{1}{2}$, for each j_2-block inside the left half of B_ℓ, swap the array values between the left and right halves of that j_2-block,
  2. with the remaining probability $\frac{1}{2}$, for each j_2-block inside the right half of B_ℓ, swap the array values between the left and right halves of that j_2-block.

The relative values taken by the array A on indices in an arbitrary j_1-block in distributions $\mathcal{D}_0$ and $\mathcal{D}_1$ can be visualized as in Figures 1 and 2. We note that in both $\mathcal{D}_0, \mathcal{D}_1$, all values in the ℓ-th j_1-block are smaller than the values in the $(\ell+1)$-th j_1-block for all $\ell \in [(n/2^{j_1})-1]$.

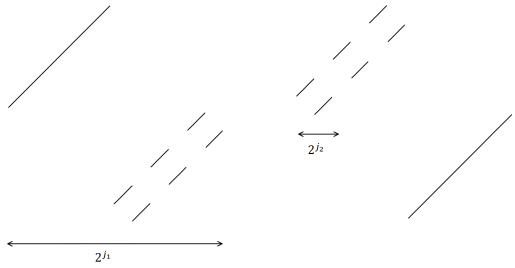


Figure 2 The relative values in j_1-blocks of $\mathcal{D}_1$ either look like the left or right diagrams above, with equal probability.

▶ **Lemma 3.2.** *The distance to monotonicity of every array sampled from $\mathcal{D}_0$ is, with probability $1 - \delta/2$, within $\frac{5}{8} \pm \delta$, where $\delta \leq \frac{1}{2^6}$. The distance to monotonicity of every array sampled from $\mathcal{D}_1$ is equal to $1/2$.*

Proof. Consider an array A sampled from one of the distributions. Let $j_1 > j_2$ be the scales used. First, observe that there are no violations to monotonicity across j_1-blocks. Therefore, it is enough to focus on repairing individual j_1-blocks and making them monotone (without inducing new violations). Consider a j_1-block B_ℓ for $\ell \in [n/2^{j_1}]$.

Assume that A is constructed from $\mathcal{D}_0$, and that B_ℓ is such that the values in the left and right halves of every j_2-block in B_ℓ are swapped (happens with probability $\frac{1}{2}$ for that block). Then we need to modify the values of at least $3/4$ fraction of indices in B_ℓ to make it monotone, since B_ℓ contains an exact cover by disjoint decreasing subsequences, each of size 4. Further, it is easy to see that by correcting a $3/4$ fraction of indices we can make B_ℓ monotone. In case swapping of values is done for none of the j_2-blocks in B_ℓ (happens with probability $\frac{1}{2}$), then we can repair B_ℓ if and only if we modify the values on half the indices in B_ℓ. Therefore, the expected distance to monotonicity of each block B_ℓ, and thereby, of A is equal to $5/8$. Note that the specific values of the scales did not matter in the above.

Let $\delta = 1/2^6$. We now show that the distance of A from monotonicity is $\frac{5}{8} \pm \delta$, with high probability. For a block $B_\ell, \ell \in [n/2^{j_1}]$, let $\mathsf{dist}(B_\ell)$ denote the distance of the block from monotonicity, normalized by the block length 2^{j_1}. We can see that the random variable $(\sum_\ell \mathsf{dist}(B_\ell))/(n/2^{j_1})$ corresponds to the normalized Hamming distance of A from monotonicity. By Hoeffding's bound, we have, $\Pr\left[\left|\frac{\sum_\ell \mathsf{dist}(B_\ell)}{n/2^{j_1}} - \frac{5}{8}\right| > \frac{1}{2^6}\right] \leq \frac{2}{\exp(8)} \leq \frac{1}{2^7} = \frac{\delta}{2}$.

Assume now that A is constructed from $\mathcal{D}_1$. For $\ell \in [n/2^{j_1}]$, if the swap of values happens within every j_2-block in the left half of B_ℓ, then we can repair B_ℓ by setting every value in the left half of that block to the smallest value in the right half of the same block. An analogous repair can be done if the swap happens in the right hand side of the block. In both cases, we only change values of at most half the number of indices in each block. The reader can easily convince themselves that at least half the values per block need to be changed to make a j_1-block monotone, which concludes the proof for the given scales. As before, the argument is independent of the choice of the scales. ◀

▶ **Lemma 3.3.** *Every comparison-based nonadaptive deterministic algorithm that, with probability at least $2/3$, distinguishes the distributions $\mathcal{D}_0$ and $\mathcal{D}_1$ has to make $\Omega(\log^2 n)$ queries.*

Proof. Consider an arbitrary deterministic comparison-based nonadaptive algorithm T that makes $o(\log^2 n)$ queries and aims to distinguish $\mathcal{D}_0$ and $\mathcal{D}_1$. Let $Q \subseteq [n]$ denote the set of queries that T makes.

Consider an array A sampled according to one of the distributions. Recall that $\mathcal{T}$ denotes an ordered binary tree whose leaves are the indices of A. Let $j_1, j_2 \in [\log n]$ such that $j_2 < j_1$ denote the scales used while constructing A. Let E denote the (bad) event that Q contains four indices $w < x < y < z \in [n]$ such that for some $\ell \in [n/2^{j_1}]$, (1) Each of $\{w, x, y, z\}$ belongs to the same ℓ-th j_1-block, (2) $\mathsf{LCA}(w, x)$ and $\mathsf{LCA}(y, z)$ are both nodes in $\mathcal{T}$ at level j_2, and (3) $\mathsf{LCA}(x, y)$ is at level j_1. By an argument of Ben-Eliezer, Canonne, Letzter, and Waingarten [4], the probability, over the choice of the scales j_1, j_2, of the event E is at most $1/3$. In the rest of the proof, we fix the scales (j_1, j_2) for which E does not happen.

Let $x, y \in Q$ be such that $\mathsf{LCA}(x, y)$ is at level j_2 in $\mathcal{T}$. In the rest of the proof, for simplicity, we refer to such queries as being j_2-*cousins*. Let B be a j_1-block, and let $Q_L(B)$ be the queries in Q that are in the left half of B, and $Q_R(B)$ the queries in Q that are in the right half of B. By our conditioning, for each j_1-block B, either all j_2-cousins in B belong to $Q_L(B)$ or to $Q_R(B)$ but not both. Consider the half of B that does not contain any j_2-cousins. In the algorithm's view, the array values in that half are increasing, whether A is sampled from $\mathcal{D}_0$ or $\mathcal{D}_1$.

Assume that A is sampled from $\mathcal{D}_0$. We show that there is a way to sample an array A' from $\mathcal{D}_1$ such that the algorithm's view on A and A' are identical. The scales of A' have to be identical to those of A. We only need to specify how swapping of values is done inside the j_1-blocks, as part of constructing A'.

Note that we only need to consider the j_1-blocks in which at least two queries fall. Consider such a block B, and assume that $Q_R(B)$ contains no j_2-cousins. Consider the case that swapping of values was done within every j_2-block inside the block B while constructing A. Then, in A', we swap the values only within the j_2-blocks inside the left half of B. In the other case that no swapping of values (within j_2-blocks) was done while constructing A, we swap the values only within the j_2-blocks inside the right half of B. It is easy to see that the relative values within block B in the array A' are consistent with that of an array sampled from $\mathcal{D}_1$. One can make similar arguments about coupling the arrays A and A' on blocks B such that the only occurrences of indices $x, y \in Q \cap B$ that are j_2-cousins are in the right half of B.

We conclude that for any scales j_1, j_2 for which E does not happen, the view of the algorithm making queries Q is identical on A' and A. Hence the algorithm cannot distinguish between the case that the array is sampled from $\mathcal{D}_0$ or from $\mathcal{D}_1$ for such scales. As this is true for any scales for which E does not happen, this concludes the proof.

Observe that the only place in the analysis where we made use of the bound on the number of queries is in arguing that the event E happens with low probability. ◀

3.2 A $\log^{\omega(1)} n$ Lower Bound

Next, we strengthen the $\Omega(\log^2 n)$ lower bound and prove Theorem 1.1. We make use of the idea of Ben-Eliezer et al. [4] for lower bounding query complexity of nonadaptive detection of larger forbidden order patterns. The idea is to use more than just two scales. This, in turn, makes the difference between the distances of arrays sampled from the two distributions smaller, which is why we only get a $\log^{\omega(1)} n$ lower bound.

Let $h \geq 2$ be an integer parameter. We describe the two distributions $\mathcal{D}_0^{(h)}$ and $\mathcal{D}_1^{(h)}$ on real-valued arrays, such that no comparison-based deterministic nonadaptive algorithm that makes $o(\log^h n)$ queries can distinguish between the distributions with high probability. Further, the distance to monotonicity of each array sampled from $\mathcal{D}_0^{(h)}$ will be significantly different than that of arrays sampled from $\mathcal{D}_1^{(h)}$.

The distributions are defined recursively, where, for the base case $h = 2$, let $\mathcal{D}_0^{(2)}$ and $\mathcal{D}_1^{(2)}$ be equal to $\mathcal{D}_0$ and $\mathcal{D}_1$ defined with scales j_1, j_2 as in Section 3.1, respectively. The details of the distributions and the proof of lower bound for the general case is much more technical and it is deferred to the full version [13].

4 Parameterized and Nonadaptive Algorithm for LIS Estimation

Our final goal in this paper is to present a sublinear algorithm that, for an array of length n containing at most r distinct values, approximates the LIS length within a bounded multiplicative error (Theorem 1.5). Our algorithm is described in the following subsections, and it uses as a subroutine, the algorithm guaranteed by Theorem 1.4 that approximates the LIS length within a bounded additive error. The description and analysis of the latter algorithm can be found in the full version of the paper [13].

4.1 $\tilde{O}(\sqrt{r})$-Query Nonadaptive Algorithm

Let $\mathcal{L}$ denote the set of points in an arbitrary and fixed LIS in the input array A. For simplicity of the presentation, we assume that our algorithm knows a lower bound λn on $|\mathcal{L}|$. Disregarding this assumption, the algorithm will output, with high probability, a lower bound estimate of the size of an increasing sequence in A. If λn is indeed a bound as assumed, it will be guaranteed that the estimate is within the multiplicative error that is stated. Hence λ can be checked by running the algorithm, in parallel, for a geometrically decreasing sequence of λ's. The reader may think of $\lambda < 1$ as a small constant (although the algorithm works for $\lambda = o(1)$ as well).

Throughout this section, we visualize the array values as points in an $r \times n$ grid G_n. The vertical axis of G_n represents the range R of the array and is labeled with the at most r distinct array values in increasing order and the horizontal axis is labeled with the indices in $[n]$. We refer to an index-value pair in the grid as a point. The grid has n points, to which we do not have direct access.

We divide the $r \times n$ grid G_n into y^* rows and x columns that partitions G_n into a $y^* \times x$ grid G' of boxes, where $y^* = \Theta(\sqrt{r})$ and $x = \Theta(\sqrt{r})$. Specifically, we divide the interval $[n]$ into x contiguous subintervals. For $i \in [x]$, let D_i denote the subarray induced by the indices in the i-th subinterval. Additionally, we divide the range R into y^* contiguous intervals of array values, where for $j \in [y^*]$, we use I_j to denote the j-th interval when the intervals are sorted in the nondecreasing order of values. The set of boxes in G' is then $\{(D_i, I_j) : \ i \in [x], j \in [y^*]\}$.

The $y^* \times x$ grid of boxes G' induces a poset $\langle \mathcal{P}, \preceq \rangle$ on the $y^* x$ boxes, which is similar to the natural poset defined on G_n. Namely, for two boxes in G' (or for two points in G_n), we have $(D_i, I_j) \preceq (D_s, I_t)$ (or $(i, j) \leq (s, t)$) if $i \leq s$ and $j \leq t$. The points in $\mathcal{L}$ form a chain in the above poset in G_n. Further, each chain in the poset in G_n forms an increasing subsequence in the array A. The boxes in G' through which $\mathcal{L}$ passes also forms a chain in the poset in G'. Every chain of boxes in the poset in G' induces a number of chains in the poset in G_n, but of possibly quite different lengths.

▶ **Definition 4.1** (Density of a box). *Let $I \subseteq R$ be a subset of the range R of values and B be a subarray of A. The density of the box (B, I), denoted by $\mathsf{den}(B, I)$, is defined to be the fraction of indices in the subarray B whose values belong to the interval I.*

In other words, for each box $(D_i, I_j) \in G'$, its density $\mathsf{den}(D_i, I_j)$ is the fraction of points in the subarray D_i that land in the box (D_i, I_j). For $\beta < 1$ (that the reader can think of as a small constant), a box (D_i, I_j) is said to be β-dense, if $\mathsf{den}(D_i, I_j) \geq \beta$.

4.1.1 Forming the grid G' of boxes

Our goal in this section is to describe a procedure that determines the grid G' of boxes. Specifically, as we do not know the range R and only know an upper bound r on its size $|R|$, we start by forming an approximation of R and an approximation of the densities of subinterval ranges in R in the array A. To do this, we first partition R into $\tilde{O}(\sqrt{r})$ sub-ranges called *layers*. For the sake of generality, we describe the procedure for a subarray B of A.

More generally, given a subarray B, and a parameter y, our goal is to partition the range R into roughly y intervals of roughly equal densities, where the densities are with respect to B. We note that although the size r of the range R might be relatively large, it is possible that some values appear in B much more frequently than others. One of our goals is to identify such values and well-approximate their densities. We now define a "nice" partition as follows. Given y and B, a nice y-partition of the values in B is a partition $R = \cup_{i=1}^{y^*} I_i$, if for each $i \in [y^*]$, either I_i contains only one value v_i and $\mathsf{den}(B, I_i) \geq \frac{1}{2y}$, or $\mathsf{den}(B, I_i) \leq \frac{2}{y}$. In the former case, we call I_i a single-valued layer. In the latter, we say that I_i is a multi-valued layer (although in an extreme case it might contain only one value). We also require $y^* \leq 2y$.

Next, we describe our procedure $\textsc{Layering}(B, y, t)$ that forms a y-nice partition of a subarray B, along with a good approximation of the densities of the single-valued layers. This is quite technical, although standard. We advise the reader to avoid it on first reading, and assume that, when needed, we have a nice partition along with a good approximation to the densities of layers.

Algorithm 1 Layering.

procedure $\textsc{Layering}(B, y, t)$

Goal: To divide the set of array values in the subarray B into roughly y contiguous intervals of roughly equal densities. The parameter t is used to control the success probability.

Sample a set of $\ell = t \cdot y \log y$ indices S from B, uniformly at random and independently. ▷ Note that a value v is expected to appear in proportion to its density in the array. Hence the collection of values obtained is a multiset of size ℓ.

We sort the multiset of values $V = \{B[p] : p \in S\}$ to form a strictly increasing sequence $\mathsf{seq} = (v'_1 < \ldots < v'_q)$, where with each $i \in [q]$ we associate a weight w_i that equals the multiplicity of v'_i in the multiset V of values. ▷ Note that $\sum_{i \in [q]} w_i = \ell$.

We now partition the sequence $W = (w_1, \ldots, w_q)$ into maximal disjoint contiguous subsequences $W_1, \ldots W_{y^*}$ such that for each $j \in [y^*]$, either $\sum_{w \in W_j} w < 2t \log y$, or W_j contains only one member w for which $w > t \log y$.

Note that this can be done greedily as follows. If $w_1 > t \log y$ then W_1 will contain only w_1, otherwise W_1 will contain the maximal subsequence $(w_1, \ldots, w_i)$ whose sum is at most $2t \log y$. We then delete the members of W_1 from W and repeat the process. For $i \in [y^*]$, let $w(W_i)$ denote the total weight in W_i.

Correspondingly, we obtain a partition of the sequence seq of sampled values into at most y^* subsequences $\{S_i\}_{i \in [y^*]}$. Some subsequences contain only one value of weight at least $t \log y$ and are called *single-valued*. The remaining subsequences are called *multi-valued*.

For a subsequence S_i, let $\alpha_i = \min(S_i)$ and $\beta_i = \max(S_i)$. Let $\beta_0 = -\infty$. Note that $\alpha_i \leq \beta_i$ and $\beta_{i-1} < \alpha_i$ for all $i \in [y^*]$.

For $i \in [y^*]$, we associate with the subsequence S_i, an interval (layer) $I_i \subseteq R$, where $I_i = (\beta_{i-1}, \beta_i]$, and an approximate density $\widetilde{\mathsf{den}}(B, I_i) = w(W_i)/\ell$.

end procedure

Let $\{I_i\}_{i=1}^{y^*}$ be the set of layers that are created by a call to LAYERING(B, y, t). Recall that $w(W_i) \geq t \log y$ if I_i is a single-valued layer and $w(W_i) < 2t \log y$ if I_i is multi-valued, where W_i denotes the sum of multiplicities of the values in the sample S_i.

For a multi-valued layer I_i, let E_i denote the event that $\mathsf{den}(B, I_i) < \frac{4}{y}$. For a single-valued layer I_i such that $\mathsf{den}(B, I_i) \geq \frac{1}{2y}$, let E_i denote the event that $\frac{\mathsf{den}(B,I_i)}{2} \leq \widetilde{\mathsf{den}}(B, I_i) \leq \frac{3}{2}\mathsf{den}(B, I_i)$. For a single-valued layer I_i such that $\mathsf{den}(B, I_i) < \frac{1}{2y}$, let E_i be the event that $\widetilde{\mathsf{den}}(B, I_i) \leq \frac{3}{2}\mathsf{den}(B, I_i)$. Let $E = \bigcap_{i=1}^{y^*} E_i$. The following claim asserts that the layering above well-represents the structure of the range w.r.t. B.

▷ Claim 4.2. LAYERING(B, y, t) returns a collection of intervals $\{I_i\}_{i=1}^{y^*}$ such that, $y^* \leq 2y$, and $\Pr(E) = 1 - \exp(\Omega(-t))$.

We now define the grid G' of boxes as follows. We first use the procedure LAYERING on the original array, $B = A$, with parameters $y = \frac{\sqrt{r}}{\epsilon}$ and $t = O(1)$, where the value of t is set to ensure a success probability of 99/100 in Claim 4.2. This defines the set of y^* layers that partitions R as $R = \bigcup_{i \in y^*} I_i$. Next we partition $[n]$ into $x = \epsilon \cdot \sqrt{r}$ contiguous intervals $D_1, \ldots D_x$ each of size n/x, which defines G' as the grid of boxes $\{(D_i, I_j) : (j, i) \in [y^*] \times [x]\}$ in the $r \times n$ grid, some of which may be empty, while some may contain many points.

We set $\beta = \epsilon^3 \lambda$. Next, our goal is to find all the β-dense boxes in G' by making $\tilde{O}(\sqrt{r})$ queries and then restrict our attention only to these boxes. As described in the high level overview Introduction, doing this will not make the LIS in this restricted array too short. This is made formal in the following claim.

▷ Claim 4.3. The number of points in $\mathcal{L}$ that belong to boxes that are not β-dense is at most $\beta n \cdot (1 + 2y/x)$.

We do not know which boxes are β-dense. We approximate this by sampling, and this is formally presented below as algorithm GRIDDING. The algorithm assumes the partition of $[n]$ and of the range R as above. As before, $t = O(1)$ in this procedure can be set appropriately to ensure a large constant success probability.

Algorithm 2 GRIDDING.

```
1: procedure GRIDDING(A, {I_j}_{j∈[y*]}, {D_i}_{i∈[x]}, β)
2:     for i ∈ [x] do
3:         Sample ℓ = t · (1/β) · log(x/β) indices from D_i uniformly and independently at random.
4:         for j ∈ [y*] do
5:             Label box (D_i, I_j) as dense if and only if the values on at least (3/4)βℓ points
   from the sample fall into the box; namely, if for at least (3/4)βℓ indices sampled from D_i,
   the values are in I_j.
6:         end for
7:     end for
8: end procedure
```

Let D be the event that all β-dense boxes are tagged as *dense* by the procedure, and that every box that is not $\beta/8$-dense is not tagged as dense.

▷ Claim 4.4. $\Pr[D] \geq 1 - \exp(-\Omega(t))$.

From now on, we assume that we have the grid G' for which the events E and D hold. This is the initialization of our data structure as described in the Introduction. We now refine the data structure as follows.

A β-dense box may have density that is anything in $[\beta, 1]$. For a better approximation guarantee, we need to identify the regions with density nearly equal to β. To achieve this, we perform the following finer layering using the procedure LAYERING on each dense box.

Finer layering of each dense box. For each $i \in [x]$, call LAYERING on the array D_i with $y = 1/\beta$ and $t' = \Theta(\log(x/\beta))$. Here, we do not collapse the single-valued intervals into a single layer, but rather just leave them as different layers of the same value and density β.

Let $\{I'_k\}_{k \in [y_i^*]}$ be the set of intervals returned by the procedure. We restrict our attention to the boxes (D_i, I'_k) contained in some β-dense box (D_i, I_j), and call them β-dense cells.

Fix D_i. The number of β-dense cells in D_i is at most $2y = 2/\beta$. Claim 4.2 asserts that, with probability $1 - \exp(-\Omega(t')) = 1 - \frac{\beta}{100x}$, each D_i is layered so that each β-dense cell has true density at most $3\beta/2$ and at least $\beta/8$. Additionally, the portion of a β-dense box that is not covered by β-dense cells has true density smaller than β. This implies that for all $i \in [x]$ this event happens with probability at least $99/100$. We denote this event by F, and assume in what follows that F happens.

4.1.2 Chain reduction

In this section, we define a poset over dense cells and argue that in order to well-approximate the LIS, it is enough to restrict our attention to LIS's in a few chains in this poset.

Since dense cells, by definition, are contained inside dense boxes, we denote dense cells using triplets (D_i, I_j, k), where this triplet denotes the k-th dense cell inside the dense box $(D_i, I_j), i \in [\epsilon\sqrt{r}], j \in [\sqrt{r}/\epsilon]$.

Recall that there is a poset $\langle \mathcal{P}, \preceq \rangle$ on the dense boxes. Now, we define another poset $\langle \mathcal{P}^\star, \preceq^\star \rangle$ whose elements are the (at most) $2x/\beta$ dense cells. The order relation $\preceq^\star$ is defined by $(D_i, I_j, k) \preceq^\star (D_{i'}, I_{j'}, k')$ if and only if either $(D_i, I_j) \neq (D_{i'}, I_{j'})$ and $(D_i, I_j) \preceq (D_{i'}, I_{j'})$, or if $j' = j, i' = i$ and $k \leq k'$. Note that the poset $\preceq^\star$ is not consistent with a grid poset, it rather inherits the order from $\mathcal{P}$ for cells in different boxes.

Let $\mathcal{L}_1$ be the LIS $\mathcal{L}$ restricted to dense boxes, let $C(\mathcal{L}_1, \mathcal{P})$ be the set of dense boxes in which $\mathcal{L}_1$ passes, and let $C(\mathcal{L}_1, \mathcal{P}^\star)$ be the set of dense cells in which $\mathcal{L}_1$ passes. We observe that $C(\mathcal{L}_1, \mathcal{P})$ and $C(\mathcal{L}_1, \mathcal{P}^\star)$ are chains in the corresponding posets.

Our goal now is to show that there are a small number of chains in $\mathcal{P}^\star$ that cover $C(\mathcal{L}_1, \mathcal{P}^\star)$. This is done as follows.

- For parameter $\tau = 5/\lambda$, repeatedly remove antichains of size larger than τ from $\mathcal{P}^\star$. Here, by removing, we mean the deletion of the points in the cells of the corresponding antichain from the array[6].

Let the resulting poset be denoted by $\mathcal{P}^{\star\star}$. The maximum antichain in this poset has size at most τ, and Dilworth's theorem implies that there is a decomposition of $\mathcal{P}^{\star\star}$ into at most τ chains. These chains, being made of dense cells, is naturally extended to at most τ (possibly intersecting) chains of the poset $\mathcal{P}$. Let these chains be $C_1, \ldots, C_\tau$. We bound the "loss" to the LIS incurred by chain reduction in Claim 4.5.

▷ Claim 4.5. Conditioned on the events $E \cup D \cup F$, the number of points in $\mathcal{L}_1$ that does not belong to $\cup_{i=1}^{\tau} C_i$ is at most $4n/\tau$.

[6] For a single-valued cell $a = (D_i, I_j, k)$ taking the value v, there might be other points with value v in the dense box containing a. When we remove a from $\mathcal{P}^\star$, we "mentally" remove some $\beta n/x$ points of value v from the box (D_i, I_j). This is not algorithmically done, but will just be used in the analysis.

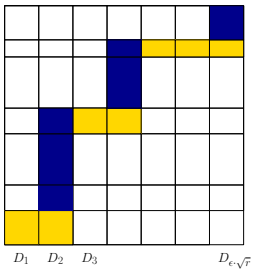


Figure 3 A staircase like chain, and its decomposition into two chains, one that contains only horizontal blocks and one that contains only vertical blocks. In each of the chains, no two blocks share a layer or a subarray.

Let $\mathcal{L}_2$ denote the LIS $\mathcal{L}_1$ after chain reduction. The following claim is straightforward.

▷ Claim 4.6. There exists $i \in [\tau]$ such that $|\text{LIS}(C_i)| \geq \frac{|\mathcal{L}_2|}{\tau}$.

We point out that no query is made at this stage.

4.1.3 Estimating the LIS restricted to poset chains

Let $\mathcal{P}'$ denote the poset obtained from $\mathcal{P}$ after removing the large antichains in $\mathcal{P}^\star$. At this point, we have covered the poset $\mathcal{P}'$ using at most τ chains $C_1, \ldots, C_\tau$. This reduces the LIS estimation to estimating the LIS in one of these chains.

In what follows, we fix such one chain C, and denote by $\mathcal{L}(C)$, the LIS in the array restricted to C. The chain C is composed of a sequence of horizontal and vertical blocks, arranged in a staircase manner (see Figure 3), where a horizontal block is a sequence of contiguous boxes in the chain from the same layer, and a vertical block is a sequence of contiguous dense boxes in the chain that belong to the same subarray.

Let H_1, H_2 be two maximal horizontal blocks in C. Blocks H_1, H_2 have a subarray D_i in common if there are boxes $(D_i, I_j) \in H_1$ and $(D_i, I_s) \in H_2$. In particular, there is a vertical block between them. Two horizontal chains have at most one subarray in common, and if this happens, then the common subarray D_i defines the rightmost box of the "lower" horizontal chain (the horizontal chain in the lower layer) and the leftmost box of the "upper" horizontal chain. We conclude that if we arrange the horizontal blocks from bottom to top as $H_1, \ldots H_s$, and remove the rightmost box from H_i if it has a common subarray with H_{i+1}, we get a sequence of horizontal blocks in which no two share a subarray. We use C_H to denote this subchain of C. Notice that $C_V = C \setminus C_H$ is a chain that contains only vertical blocks, where no two share a layer (see Figure 3, as how the whole chain C decomposes into a chain of horizontal blocks and a chain of vertical blocks).

To estimate the size of $\mathcal{L}(C)$, we estimate the LIS within C_H and C_V separately, and use the larger for the size estimate for $\mathcal{L}(C)$. In the following, we denote $\mathcal{L}_H$ and $\mathcal{L}_V$ for $\text{LIS}(C_H)$ and $\text{LIS}(C_V)$, respectively. The main advantage of this decomposition of C into C_H and C_V is given by the following observation.

▶ **Observation 4.7.** *For any chain C, we have $|\mathcal{L}_H| = \sum_{B \in C_H} |LIS(B)|$, where the sum is over the horizontal blocks B in C_H. A similar statement holds for C_V in which case, horizontal blocks are replaced with vertical blocks.*

The observation leads to an immediate adaptive way to approximate the lengths of $\mathcal{L}_H$ and $\mathcal{L}_V$. We sample a constant number of blocks from the chain, estimate the LIS in each block, and normalize to estimate the LIS in the entire chain. By the Hoeffding bound, we can

see that the estimate is accurate enough with high probability. The adaptivity is needed to locate each block, and to estimate the LIS within a block (horizontal and vertical), which we did not yet specify how to do. To avoid adaptivity, we will rely on the fact that if $\mathcal{L}_V$ is large, then it must contain a large number of small vertical blocks. Thus, sampling uniformly in $[n]$ will hit such blocks frequently enough to facilitate the Hoeffding bound above. Further, for the estimation of LIS within a short vertical block, we will use the algorithm guaranteed by Theorem 1.4. For horizontal blocks, we need some further relaxations. We will show below that we may restrict ourselves to short horizontal blocks, due to the choice of parameters in the formation of the grid G'. This again will facilitate the use of the algorithm guaranteed by Theorem 1.4. The details now follow.

Estimating the length of LIS in a horizontal chain. A horizontal block belonging to a multi-valued layer is referred to as a multi-valued horizontal block, and a horizontal block belonging to a single-valued layer is a single-valued horizontal block. We treat these horizontal blocks separately.

Let $m = \epsilon/\lambda^2$. Let $\mathcal{L}'_H$ denote the restriction of $\mathcal{L}_H$ after deleting multi-valued horizontal blocks containing more than m boxes. We first show that the length of $\mathcal{L}'_H$ is not much smaller than $\mathcal{L}_H$.

▷ Claim 4.8. $|\mathcal{L}'_H| \leq |\mathcal{L}_H| - 4\lambda^2\epsilon n$.

Let C'_H denote the chain obtained after removing multi-valued horizontal blocks containing more than m boxes. If there are at most $\phi = \epsilon\lambda^2 y$ multi-valued horizontal blocks in the chain C'_H, then, by removing all of them, we end up losing only $\phi \cdot \frac{4n}{y} \leq 4n\epsilon\lambda^2$ points (as each multi-valued layer has density at most $4/y$). If there are at least ϕ multi-valued horizontal blocks in C'_H, then the average number of values in such a horizontal block is at most $\frac{r}{\phi} \leq \frac{r}{\epsilon\lambda^2 y} = \frac{\sqrt{r}}{\lambda^2}$ by our choice of y. That is, with probability at least $1 - \frac{1}{100\log(\tau)}$, a uniformly random multi-valued horizontal block in C'_H contains at most $\frac{100\sqrt{r}\log(\tau)}{\lambda^2}$ values. Thus, we have reduced the problem to estimating the LIS in a collection of (possibly very long) single-valued horizontal blocks and several short multi-valued horizontal blocks containing $O(\frac{\sqrt{r}\log(\tau)}{\lambda^2})$ values.

In the following, we use the term *segment* to denote a subarray composed of $2m$ subarrays $\{D_i, D_{i+1}, \ldots, D_{i+2m-1}\}$ for some $i \in [x-2m+1]$. A segment is said to contain a multi-valued horizontal block H if all the subarrays forming H are contained in the segment.

Fix $r' = \frac{100\sqrt{r}\log(\tau)}{\lambda^2}$. Our algorithm for estimating the length of $\mathcal{L}_H$ is as follows:

1. Sample $t\log^2(\tau)$ uniformly random segments.
2. For each sampled segment B, query $s = \Theta\left(\frac{m\log(\tau)}{\beta} \cdot \frac{r'}{\epsilon^3\lambda^6}\log\left(\frac{r'^2}{\epsilon\lambda}\right)\right)$ points uniformly and independently at indexes from B and run the algorithm given by Theorem 1.4 with parameters r' (for the number of distinct values) and $\epsilon\lambda^2$ (for approximation guarantee) using the samples that fall into the multi-valued horizontal block H contained in the segment B, if any.
3. Estimate the contribution to the LIS from multi-valued horizontal blocks by summing the answers returned by the algorithm in the previous steps and then normalizing appropriately.
4. Estimate the contribution to the LIS from single-valued horizontal blocks by summing the estimates of the densities of all single-valued horizontal blocks in C_H (as we already know these estimates from the Gridding stage).
5. Output an estimate L_H of the length of $\mathcal{L}_H$ by summing the above two estimates.

Clearly, the contribution to $\mathcal{L}_H$ from single-valued horizontal blocks is estimated within multiplicative $(1 \pm \frac{1}{2})$-error, by our conditioning on the event F. We show the following.

▷ Claim 4.9. With probability $1 - O(\frac{\log(\tau)}{\tau^2})$, the contribution to $\mathcal{L}_H$ from multi-valued horizontal blocks is estimated within an additive error of $\epsilon\lambda^2 n$.

Estimating the length of the LIS in a vertical chain. Let $\nu = \epsilon\lambda^2$. We may assume that the vertical chain is composed of at least $\nu \cdot x$ vertical blocks, for otherwise, we can abandon the entire vertical chain by incurring a "loss" to the LIS amounting to at most $\nu \cdot n$ points. Additionally, since the boxes from different vertical blocks belong to different layers, using a similar averaging argument as before, we can show that with probability at least $1 - \frac{1}{100\log(\tau)}$, a uniformly random vertical block contains at most $\frac{100\sqrt{\tau}\log(\tau)}{\lambda^2}$ distinct values.

Therefore, in order to estimate the length of the LIS in the vertical chain, we sample $O(\log(\tau))$ subarrays $D_i, i \in [x]$ and run the pseudo-solution-based LIS estimation algorithm, restricted to the vertical box, if any, that belongs to this subarray while making sure that the success probability is at least $1 - \frac{1}{100\log\tau}$ and the error parameter is $\epsilon\lambda^2$. The details of how to implement this procedure nonadaptively are identical to how we implemented the estimation of the LIS in C_H in the preceding section. The query complexity is also identical.

▷ Claim 4.10. With probability $1 - O(\frac{\log(\tau)}{\tau^2})$, we estimate the contribution of vertical blocks to within an additive error of $\epsilon\lambda^2 n$.

4.1.4 Correctness, approximation guarantee, and query complexity

In this section, we complete the analysis of our algorithm and finish the proof of Theorem 1.5.

Success probability. The probability that any of Layering, Gridding and Finer Gridding fail is at most 3/100. For a specific chain of boxes, by Claims 4.9 and 4.10, we know that estimating the length of LIS within them is within the approximation guarantee with probability at least $1 - O(\frac{\log(\tau)}{\tau^2})$. By a union bound over all τ chains, we can see that the probability of incorrectly estimating the LIS length in some chain is at most 1/100. Thus, overall, the failure probability is at most a small constant.

Query complexity. The query complexity is clearly $\tilde{O}(\sqrt{r} \cdot \text{poly}(1/\lambda))$ from the description of the algorithm.

Approximation guarantee. Consider a fixed true LIS $\mathcal{L}$. The loss to $\mathcal{L}$ due to ignoring boxes that are not β-dense ($\beta = \epsilon^2\lambda$) is at most $\epsilon^3\lambda n + \epsilon\lambda n$. The loss to $\mathcal{L}$ due to antichain removal is at most $4n/\tau$, which is equal to $4\lambda n/5$. The resulting increasing sequence has length at least $|\mathcal{L}| - \epsilon^3\lambda n - \epsilon\lambda n - 4\lambda n/5$, which is at least $(1 - \epsilon^3 - \epsilon - 4/5) \cdot |\mathcal{L}|$, since, by our assumption $|\mathcal{L}| \geq \lambda n$. After chain decomposition, the length of the LIS in the best chain is at least $(1 - \epsilon^3 - \epsilon - 4/5) \cdot |\mathcal{L}|/\tau$, which is equal to $\frac{\lambda}{5} \cdot |\mathcal{L}| \cdot (1/5 - \epsilon^3 - \epsilon)$. Since we split the chains into horizontal and vertical chains, we further lose a factor of 2, and the resulting LIS length becomes $\frac{\lambda}{10} \cdot |\mathcal{L}| \cdot (1/5 - \epsilon^3 - \epsilon)$. In case of horizontal chains, we additionally lose a $9\epsilon\lambda^2 n$ and in the case of vertical chains, we additionally lose $\epsilon\lambda^2 n$. That is the length of LIS in the (best) horizontal chain is at least $\frac{\lambda}{10} \cdot |\mathcal{L}| \cdot (1/5 - \epsilon^3 - 11\epsilon)$. Finally, using Claims 4.9 and 4.10, we can see that we estimate the lengths of the best horizontal and vertical chains to within a constant multiplicative factor. Overall, the approximation guarantee is multiplicative $\Omega(\lambda)$.

References

1. Nir Ailon, Bernard Chazelle, Seshadhri Comandur, and Ding Liu. Estimating the distance to a monotone function. *Random Structures & Algorithms*, 31(3):371–383, 2007. doi:10.1002/rsa.20167.
2. David Aldous and Persi Diaconis. Longest increasing subsequences: from patience sorting to the Baik-Deift-Johansson theorem. *Bull. Amer. Math. Soc.*, 34:413–432, 1999. doi:10.1090/S0273-0979-99-00796-X.
3. Aleksandrs Belovs. Adaptive lower bound for testing monotonicity on the line. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques, APPROX/RANDOM 2018*, pages 31:1–31:10, 2018. doi:10.4230/LIPIcs.APPROX-RANDOM.2018.31.
4. Omri Ben-Eliezer, Clément L. Canonne, Shoham Letzter, and Erik Waingarten. Finding monotone patterns in sublinear time. In *60th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2019*, pages 1469–1494, 2019. doi:10.1109/FOCS.2019.000-1.
5. Piotr Berman, Sofya Raskhodnikova, and Grigory Yaroslavtsev. L_p-testing. In David B. Shmoys, editor, *Symposium on Theory of Computing, STOC 2014*, pages 164–173. ACM, 2014. doi:10.1145/2591796.2591887.
6. Kashyap Dixit, Sofya Raskhodnikova, Abhradeep Thakurta, and Nithin Varma. Erasure-resilient property testing. *SIAM J. Comput.*, 47(2):295–329, 2018. doi:10.1137/16M1075661.
7. Funda Ergün, Sampath Kannan, Ravi Kumar, Ronitt Rubinfeld, and Mahesh Viswanathan. Spot-checkers. *Journal of Computer and System Sciences*, 60(3):717–751, 2000.
8. Eldar Fischer. On the strength of comparisons in property testing. *Inf. Comput.*, 189(1):107–116, 2004. doi:10.1016/j.ic.2003.09.003.
9. Michael L. Fredman. On computing the length of longest increasing subsequences. *Discret. Math.*, 11(1):29–35, 1975. doi:10.1016/0012-365X(75)90103-X.
10. Oded Goldreich, Shafi Goldwasser, and Dana Ron. Property testing and its connection to learning and approximation. *Journal of the ACM*, 45(4):653–750, 1998.
11. Michael Mitzenmacher and Saeed Seddighin. Improved sublinear time algorithm for longest increasing subsequence. In Dániel Marx, editor, *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms, SODA 2021, Virtual Conference, January 10 - 13, 2021*, pages 1934–1947. SIAM, 2021. doi:10.1137/1.9781611976465.115.
12. Ilan Newman, Yuri Rabinovich, Deepak Rajendraprasad, and Christian Sohler. Testing for forbidden order patterns in an array. *Random Struct. Algorithms*, 55(2):402–426, 2019.
13. Ilan Newman and Nithin Varma. New sublinear algorithms and lower bounds for LIS estimation. *CoRR*, abs/2010.05805, 2021. arXiv:2010.05805.
14. Ramesh Krishnan S. Pallavoor, Sofya Raskhodnikova, and Nithin Varma. Parameterized property testing of functions. *ACM Trans. Comput. Theory*, 9(4):17:1–17:19, 2018. doi:10.1145/3155296.
15. Michal Parnas, Dana Ron, and Ronitt Rubinfeld. Tolerant property testing and distance approximation. *Journal of Computer and System Sciences*, 6(72):1012–1042, 2006.
16. Sofya Raskhodnikova, Noga Ron-Zewi, and Nithin M. Varma. Erasures vs. errors in local decoding and property testing. In *Proceedings of the Innovations in Theoretical Computer Science Conference, (ITCS) 2019*, pages 63:1–63:21, 2019.
17. Ronitt Rubinfeld and Madhu Sudan. Robust characterizations of polynomials with applications to program testing. *SIAM Journal on Computing*, 25(2):252–271, 1996.
18. Aviad Rubinstein, Saeed Seddighin, Zhao Song, and Xiaorui Sun. Approximation algorithms for LCS and LIS with truly improved running times. In *60th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2019*, pages 1121–1145, 2019. doi:10.1109/FOCS.2019.00071.
19. Michael E. Saks and C. Seshadhri. Estimating the longest increasing sequence in polylogarithmic time. *SIAM Journal on Computing*, 46(2):774–823, 2017.

Optimal-Time Queries on BWT-Runs Compressed Indexes

Takaaki Nishimoto
RIKEN Center for Advanced Intelligence Project, Tokyo, Japan

Yasuo Tabei
RIKEN Center for Advanced Intelligence Project, Tokyo, Japan

Abstract

Indexing highly repetitive strings (i.e., strings with many repetitions) for fast queries has become a central research topic in string processing, because it has a wide variety of applications in bioinformatics and natural language processing. Although a substantial number of indexes for highly repetitive strings have been proposed thus far, developing compressed indexes that support various queries remains a challenge. The *run-length Burrows-Wheeler transform* (RLBWT) is a lossless data compression by a reversible permutation of an input string and run-length encoding, and it has received interest for indexing highly repetitive strings. LF and ϕ^{-1} are two key functions for building indexes on RLBWT, and the best previous result computes LF and ϕ^{-1} in $O(\log \log n)$ time with $O(r)$ words of space for the string length n and the number r of runs in RLBWT. In this paper, we improve LF and ϕ^{-1} so that they can be computed in a constant time with $O(r)$ words of space. Subsequently, we present *OptBWTR (optimal-time queries on BWT-runs compressed indexes)*, the first string index that supports various queries including locate, count, extract queries in optimal time and $O(r)$ words of space.

2012 ACM Subject Classification Theory of computation → Data compression

Keywords and phrases Compressed text indexes, Burrows-Wheeler transform, highly repetitive text collections

Digital Object Identifier 10.4230/LIPIcs.ICALP.2021.101

Category Track A: Algorithms, Complexity and Games

Related Version *Full Version*: `https://arxiv.org/abs/2006.05104`

1 Introduction

A string index represents a string in a compressed format that supports locate queries (i.e., computing all the positions at which a given pattern appears in a string). The *FM-index* [10, 11] is an efficient string index on a lossless data compression called the *Burrows-Wheeler transform (BWT)* [5], which is a reversible permutation of an input string. In particular, locate queries can be efficiently computed on an FM-index by performing a *backward search*, which is an iterative algorithm for computing an interval corresponding to the query on a *suffix array* (SA) [19] storing all the suffixes of an input string in lexicographical order. The FM-index performs locate queries in $O(m + occ)$ time with $O(n(\frac{\log \sigma}{\log n} + \frac{1}{s}))$ words of space for a string T of length n, query string of length m, alphabet size σ, parameter s, and number occ of occurrences of a query in T [3].

A *highly repetitive string* is a string including many repetitions. Examples include the human genome, version-controlled documents, and source code in repositories. A significant number of string indexes on various compressed formats for highly repetitive strings have been proposed thus far (e.g., SLP-index [8], LZ-indexes [6, 12], BT-indexes [7, 21]). For a large collection of highly repetitive strings, the most powerful and efficient compressed format is the run-length (RL) Burrows Wheeler transform (RLBWT) [5], which is a BWT compressed

by run-length encoding. Mäkinen et al. [18] presented an RLBWT-based string index, named the RLFM-index, that solves locate queries by executing a backward search algorithm on RLBWT. While the RLFM-index can solve locate queries in $O(r + n/s)$ words of space in $O((m + s \cdot occ)(\frac{\log \sigma}{\log \log r} + (\log \log n)^2))$ time for the number r of runs in the RLBWT of T and parameter $s \geq 1$, the size of the index depends on the string length. Recently, Gagie et al. [13] presented the r-index, which can reduce the space usage of the RLFM-index to one linearly proportional to the number of runs in RLBWT. The r-index can solve locate queries space efficiently with only $O(r)$ words of space and in $O(m \log \log_w(\sigma + (n/r)) + occ \log \log_w(n/r))$ time for a machine word size $w = \Theta(\log n)$. If the r-index allows $O(r \log \log_w(\sigma + n/r))$ words of space to be used, it can solve locate queries in the optimal time, $O(m + occ)$. Although there are other important queries including count query, extract query, decompression and prefix search for various applications to string processing, no previous string index can support various queries in addition to locate queries based on RLBWT in an optimal time with only $O(r)$ words of space. That is, developing a string index for various queries in an optimal time with $O(r)$ words of space remains a challenge.

Contribution. In this paper, we present *OptBWTR (optimal-time queries on BWT-runs compressed indexes)*, the first string index that supports various queries including locate, count, extract queries in optimal time and $O(r)$ words of space for the number r of runs in RLBWT. LF and ϕ^{-1} are important functions for string indexes on RLBWT. The best previous data structure computes LF and ϕ^{-1} in $O(\log \log_w(n/r))$ time with $O(r)$ words of space [13]. In this paper, we present a novel data structure that can compute LF and ϕ^{-1} in constant time and $O(r)$ words of space. Subsequently, we present OptBWTR that supports the following five queries in optimal time and $O(r)$ words of space.

- **Locate query:** OptBWTR can solve a locate query on an input string in $O(r)$ words of space and $O(m \log \log_w \sigma + occ)$ time, which is optimal for strings with polylogarithmic alphabets (i.e., $\sigma = O(\text{polylog } n)$).
- **Count query:** OptBWTR can return the number of occurrences of a query string on an input string in $O(r)$ words of space and $O(m \log \log_w \sigma)$ time, which is optimal for polylogarithmic alphabets.
- **Extract query:** OptBWTR can return substrings starting at a given position bookmarked beforehand in a string in $O(1)$ time per character and $O(r + b)$ words of space, where b is the number of bookmarked positions. Resolving extract queries is sometimes called the *bookmarking problem* [12, 9].
- **Decompression:** OptBWTR decompresses the original string of length n in optimal time (i.e., $O(n)$). This is the first linear-time decompression algorithm for RLBWT in $O(r)$ words of working space.
- **Prefix search:** OptBWTR can return the strings in a set D that include a given pattern as their prefixes in optimal time (i.e., $O(m + occ')$) and $O(r')$ words of space, where occ' is the number of output strings and r' is the number of runs in the RLBWT of a string made by concatenating the strings in D.

The state-of-the-art string indexes for each type of query are summarized in Table 1.

This paper is organized as follows. In Section 2, we introduce the important notions used in this paper. Section 3 presents novel data structures for computing LF and ϕ^{-1} in constant time. Section 4 presents a data structure supporting a modified version of a backward search on RLBWT. The backward search leverages the two data structures introduced in Section 3. Sections 5 and 6 present OptBWTR that supports all five queries mentioned above by leveraging the modified backward search, LF, and ϕ^{-1}.

Table 1 Summary of space and time for (i) locate and (ii) count queries, (iii) extract queries (a.k.a the bookmarking problem), (iv) decompression of BWT or RLBWT and (v) prefix searches for each query, where n is the length of the input string T, m is the length of a given string P, occ is the number of occurrences of P in T, σ is the alphabet size of T, $w = \Theta(\log n)$ is the machine word size, r is the number of runs in the RLBWT of T, s is a parameter, g is the size of a compressed grammar deriving T, b is the number of input positions for the bookmarking problem, $G = \max\{1, \log^* g - \log^*(\frac{g}{b} - \frac{b}{g})\}$, D is a set of strings of total length n, occ' is the number of strings in D such that each string has P as a prefix and r' is the number of runs in the RLBWT of a string made by concatenating the strings in D.

(i) Locate query	Space (words)	Time
RLFM-index [18]	$O(r + n/s)$	$O((m + s \cdot occ)(\frac{\log \sigma}{\log \log r} + (\log \log n)^2))$
r-index [13]	$O(r)$	$O(m \log \log_w(\sigma + (n/r)) + occ \log \log_w(n/r))$
	$O(r \log \log_w(\sigma + (n/r)))$	$O(m + occ)$
	$O(rw \log_\sigma \log_w n)$	$O(\lceil m \log(\sigma)/w \rceil + occ)$
OptBWTR	$O(r)$	$O(m \log \log_w \sigma + occ)$

(ii) Count query	Space (words)	Time
RLFM-index [18]	$O(r)$	$O(m(\frac{\log \sigma}{\log \log r} + (\log \log n)^2))$
r-index [13]	$O(r)$	$O(m \log \log_w(\sigma + (n/r)))$
	$O(r \log \log_w(\sigma + (n/r)))$	$O(m)$
	$O(rw \log_\sigma \log_w n)$	$O(\lceil m \log(\sigma)/w \rceil)$
OptBWTR	$O(r)$	$O(m \log \log_w \sigma)$

(iii) Extract query	Space (words)	Time per character	Overhead
Gagie et al.[12]	$O(g + b \log^* n)$	$O(1)$	-
Gagie et al.[13]	$O(r \log(n/r))$	$O(\log(\sigma)/w)$	$O(\log(n/r))$
Cording et al.[9]	$O((g + b)G)$	$O(1)$	-
OptBWTR	$O(r + b)$	$O(1)$	-

(iv) Decompression	Space (words)	Time
Lauther and Lukovszki [17]	$O(n(\log \log n + \log \sigma)/w)$	$O(n)$
Golynski et al.[14]	$O((n \log \sigma)/w)$	$O(n \log \log \sigma)$
Predecessor queries [4]	$O(r)$	$O(n \log \log_w(n/r))$
OptBWTR	$O(r)$	$O(n)$

(v) Prefix search	Space (words)	Time
Compact trie [20]	$(n \log \sigma)/w + O(\lvert D \rvert)$	$O(m + occ')$
Z-fast trie [2]	$(n \log \sigma)/w + O(\lvert D \rvert)$	expected $O(\lceil \frac{m \log(\sigma)}{w} \rceil + \log m + \log \log \sigma + occ')$
Packed c-trie [25]	$(n \log \sigma)/w + O(\lvert D \rvert)$	expected $O(\lceil \frac{m \log(\sigma)}{w} \rceil + \log \log n + occ')$
c-trie++ [26]	$(n \log \sigma)/w + O(\lvert D \rvert)$	expected $O(\lceil \frac{m \log(\sigma)}{w} \rceil + \log \log_\sigma w + occ')$
OptBWTR	$O(r' + \lvert D \rvert)$	$O(m + occ')$

2 Preliminaries

Let $\Sigma = \{1, 2, \ldots, \sigma\}$ be an ordered alphabet of size σ, T be a string of length n over Σ, and $|T|$ be the length of T. Let $T[i]$ be the i-th character of T (i.e., $T = T[1], T[2], \ldots, T[n]$) and $T[i..j]$ be the substring of T that begins at position i and ends at position j. For two strings, T and P, $T \prec P$ means that T is lexicographically smaller than P. Let ε be the empty string, i.e., $|\varepsilon| = 0$. We assume that (i) $\sigma = n^{O(1)}$ and (ii) the last character of string T is a special character \$ not occurring on substring $T[1..n-1]$ such that $\$ \prec c$ holds for any character $c \in \Sigma \setminus \{\$\}$. For two integers, b and e ($b \leq e$), *interval* $[b, e]$ is the set $\{b, b+1, \ldots, e\}$. $Occ(T, P)$ denotes all the occurrence positions of a string P in a string T, i.e., $Occ(T, P) = \{i \mid i \in [1, n-|P|+1] \text{ s.t. } P = T[i..(i+|P|-1)]\}$. A *count query* on a string T returns the number of occurrences of a given string P in T, i.e., $|Occ(T, P)|$. Similarly, a *locate query* on string T returns all the starting positions of P in T, i.e., $Occ(T, P)$.

A *rank* query $\mathsf{rank}(T, c, i)$ on a string T returns the number of occurrences of a character c in $T[1..i]$, i.e., $\mathsf{rank}(T, c, i) = |Occ(T[1..i], c)|$. A *select* query $\mathsf{select}(T, c, i)$ on a string T returns the i-th occurrence of c in T (i.e., it returns the smallest integer $j \geq 1$ such that $|Occ(T[1..j], c)| = i$) if T contains c; otherwise it returns -1. Assume that $T[b..e]$ contains a character c for an interval $[b, e] \subseteq [1, n]$. Let $\hat{b}$ and $\hat{e}$ be the first and last occurrences of a character c in $T[b..e]$ (i.e., $\hat{b} = \min\{i \mid i \in [b, e] \text{ s.t. } T[i] = c\}$ and $\hat{e} = \max\{i \mid i \in [b, e] \text{ s.t. } T[i] = c\}$). Then, we can compute $\hat{b}$ and $\hat{e}$ by the following lemma.

▶ **Lemma 1.** *The following statements hold: (i) $T[b..e]$ contains a character c if and only if $\mathsf{rank}(T, c, e) - \mathsf{rank}(T, c, b-1) \geq 1$ holds. (ii) $\hat{b} = \mathsf{select}(T, c, \mathsf{rank}(T, c, b-1) + 1)$ and $\hat{e} = \mathsf{select}(T, c, \mathsf{rank}(T, c, e))$ hold if $T[b..e]$ contains c.*

A suffix array (SA) of a string T is an integer array of size n such that $\mathsf{SA}[i]$ stores the starting position of the i-th suffix of T in lexicographical order. Formally, SA is a permutation of $[1, n]$ such that $T[\mathsf{SA}[1]..n] \prec \cdots \prec T[\mathsf{SA}[n]..n]$ holds. Each value in SA is called an *sa-value*.

The *suffix array interval* (*sa-interval*) of a string P is an interval $[b, e] \subseteq [1, n]$ such that $\mathsf{SA}[b..e]$ represents all the occurrence positions of P in string T, i.e., $Occ(T, P) = \{\mathsf{SA}[b], \mathsf{SA}[b+1], \ldots, \mathsf{SA}[e]\}$. The sa-interval of the empty string ε is defined as $[1, n]$.

LF is a function that returns the position with sa-value $\mathsf{SA}[i] - 1$ on SA (i.e., $\mathsf{SA}[\mathsf{LF}(i)] = \mathsf{SA}[i] - 1$) for a given integer $i \in [1, n]$ if $\mathsf{SA}[i] \neq 1$; otherwise, it returns the position with sa-value n (i.e., $\mathsf{SA}[\mathsf{LF}(i)] = n$). ϕ^{-1} [15] is a function that returns $\mathsf{SA}[i+1]$ for a given sa-value $\mathsf{SA}[i] \in [1, n]$ (i.e., $\phi^{-1}(\mathsf{SA}[i]) = \mathsf{SA}[i+1]$) if $i \neq n$; otherwise, it returns $\mathsf{SA}[1]$.

We will use base-2 logarithms throughout this paper unless indicated otherwise. Our computation model is a unit-cost word RAM with a machine word size of $w = \Theta(\log n)$ bits. We evaluate the space complexity in terms of the number of machine words. A bitwise evaluation of space complexity can be obtained with a $\log n$ multiplicative factor.

2.1 Rank-select data structure

We describe a set $\{c_1, c_2, \ldots, c_{\sigma'}\}$ and function γ for a string T. $c_1, c_2, \ldots, c_{\sigma'}$ are all the distinct characters in T, i.e., $\{c_1, c_2, \ldots, c_{\sigma'}\} = \{T[i] \mid i \in [1, |T|]\}$ ($c_1 < c_2 < \cdots < c_{\sigma'}$). The function γ returns the rank of a given character $c \in \Sigma$ in a string T; i.e., $\gamma(T, c) = j$ if there exists an integer j such that $c = c_j$ holds; otherwise $\gamma(T, c) = -1$.

A rank-select data structure $R(T)$ consists of three data structures R_{rank}, R_{select}, and R_{map}. R_{rank} is a *rank data structure* for solving a rank query on a string T in $O(\log \log_w \sigma)$ time and with $O(|T|)$ words of space [4]. R_{select} consists of σ' arrays $H_1, H_2, \ldots, H_{\sigma'}$. The size of H_j is $|Occ(T, c_j)|$ for each $j \in \{1, 2, \ldots, \sigma'\}$, and $H_j[i]$ stores $\mathsf{select}(T, c_j, i)$ for each $i \in [1, |Occ(T, c_j)|]$. R_{map} is a *deterministic dictionary* [24] storing the mapping function γ for T. The deterministic dictionary can compute $\gamma(T, c)$ for a given character c in constant time, and its space usage is $O(\sigma')$ words. The space usage of the rank-select data structure is $O(|T|)$ words in total, because $\sigma' \leq |T|$ holds. We can compute a given select query $\mathsf{select}(T, c, i)$ in two steps: (i) compute $j = \gamma(T, c)$; and (ii) return -1 if $j = -1$ or $|H_j| < i$; otherwise, return $H_j[i]$. Hence, the rank-select data structure can support rank and select queries on T in $O(\log \log_w \sigma)$ and $O(1)$ time, respectively.

2.2 BWT and run-length BWT (RLBWT)

The BWT [5] of a string T is a string L of length n built by permuting T as follows: (i) all n circular strings of T (i.e., $T[1..n]$, $T[2..n]T[1]$, $T[3..n]T[1..2]$, …, $T[n]T[2..n-1]$) are sorted in lexicographical order; (ii) $L[i]$ is the last character at the i-th circular string in

			Sorted circular strings		
i	SA	LF	F		L
1	15	10	$	baababaabaaba	b
2	7	11	a	abaabab$baaba	b
3	10	12	a	abab$baababaa	b
4	2	13	a	ababaabaabab$	b
5	13	14	a	b$baababaabaa	b
6	5	15	a	baabaabab$baa	b
7	8	2	a	baabab$baabab	a
8	11	3	a	bab$baababaab	a
9	3	4	a	babaabaabab$b	a
10	14	5	b	$baababaabaab	a
11	6	6	b	aabaabab$baab	a
12	9	7	b	aabab$baababa	a
13	1	1	b	aababaabaabab	$
14	12	8	b	ab$baababaaba	a
15	4	9	b	abaabaabab$ba	a

Figure 1 Table illustrating the BWT (L), SA, LF function, F, and the sorted circular strings of $T = baababaabaabab\$$.

the sorted order for $i \in [1, n]$. Similarly, F is a string of length n such that $F[i]$ is the first character at the i-th circular string in the sorted order. Formally, let $L[i] = T[\mathsf{SA}[\mathsf{LF}(i)]]$ and $F[i] = T[\mathsf{SA}[i]]$.

Let C be an array of size σ such that $C[c]$ is the number of occurrences of characters lexicographically smaller than $c \in \Sigma$ in string T i.e., $C[c] = |\{i \mid i \in [1, n] \text{ s.t. } T[i] \prec c\}|$. The BWT has the following property. For any integer $i \in [1, n]$, $\mathsf{LF}(i)$ is equal to the number of characters that are lexicographically smaller than the character $L[i]$ plus the rank of $L[i]$ on the BWT. Thus, $\mathsf{LF}(i) = C[c] + \mathsf{rank}(L, c, i)$ holds for $c = L[i]$. This is because $\mathsf{LF}(i) < \mathsf{LF}(j)$ if and only if either of the following conditions holds: (i) $L[i] \prec L[j]$ or (ii) $L[i] = L[j]$ and $i < j$ for two integers $1 \leq i < j \leq n$.

Let $[b, e]$ be the sa-interval of a string P and $[b', e']$ be the sa-interval of cP for a character c. Then, the following relation holds between $[b, e]$ and $[b', e']$ on the BWT L.

▶ **Lemma 2** (e.g., [10]). *Let $\hat{b}$ and $\hat{e}$ be the first and last occurrences of c in $L[b..e]$ (i.e., $\hat{b} = \min\{i \mid i \in [b, e] \text{ s.t. } L[i] = c\}$ and $\hat{e} = \max\{i \mid i \in [b, e] \text{ s.t. } L[i] = c\}$). Then, $b' = \mathsf{LF}(\hat{b})$, $e' = \mathsf{LF}(\hat{e})$, and $\mathsf{SA}[b'] = \mathsf{SA}[\hat{b}] - 1$ hold if P and cP are substrings of T.*

Figure 1 illustrates the BWT, SA, LF function, F, L and sorted circular strings of a string $T = baababaabaabab\$$. For example, let $P = ab$, $c = b$. Then $[b, e] = [5, 9]$, $[b', e'] = [14, 15]$, $\hat{b} = 5$, and $\hat{e} = 6$ (see also Figure 1). Moreover, $b' = \mathsf{LF}(\hat{b})$ and $e' = \mathsf{LF}(\hat{e})$ hold by Lemma 2.

The RLBWT of T is a BWT encoded by run-length encoding; i.e., it is a partition of L into r substrings $\mathsf{rlbwt}(L) = L_1, L_2, \ldots, L_r$ such that each substring L_i is a maximal repetition of the same character in L (i.e., $L_i[1] = L_i[2] = \cdots = L_i[|L_i|]$ and $L_{i-1}[1] \neq L_i[1] \neq L_{i+1}[1]$). Each L_i is called a *run*. Let ℓ_i be the starting position of the i-th run of BWT L, i.e., $\ell_1 = 1$, $\ell_i = \ell_{i-1} + |L_{i-1}|$ for $i \in [2, r]$. Let $\ell_{r+1} = n + 1$. The RLBWT is represented as r pairs $(L_1[1], \ell_1)$, $(L_2[1], \ell_2)$, …, $(L_r[1], \ell_r)$ using $2r$ words. For example, $\mathsf{rlbwt}(L) = bbbbbb, aaaaaa, \$, aa$ for BWT L illustrated in Figure 1. The RLBWT is represented as $(b, 1), (a, 7), (\$, 13)$, and $(a, 14)$.

Let δ be a permutation of $[1, r]$ satisfying $\mathsf{LF}(\ell_{\delta[1]}) < \mathsf{LF}(\ell_{\delta[2]}) < \cdots < \mathsf{LF}(\ell_{\delta[r]})$. The LF function has the following properties on RLBWT.

▶ **Lemma 3** (e.g., Lemma 2.1 in [16]). *The following two statements hold: (i) Let x be the integer satisfying $\ell_x \leq i < \ell_{x+1}$ for some $i \in [1, n]$. Then, $\mathsf{LF}(i) = \mathsf{LF}(\ell_x) + (i - \ell_x)$; (ii) $\mathsf{LF}(\ell_{\delta[1]}) = 1$ and $\mathsf{LF}(\ell_{\delta[i]}) = \mathsf{LF}(\ell_{\delta[i-1]}) + |L_{\delta[i-1]}|$ for all $i \in [2, r]$.*

Proof. (i) Let $y = (i - \ell_x)$ and $c = L[\ell_x + (i - \ell_x)]$. $\mathsf{LF}(\ell_x + y) = C[c] + \mathsf{rank}(L, c, \ell_x + y)$ holds by the BWT property. $\mathsf{rank}(L, c, \ell_x + y) = \mathsf{rank}(L, c, \ell_x) + y$ holds because the x-th run L_x is a repetition of the character c. Hence $\mathsf{LF}(\ell_x + y) = C[c] + \mathsf{rank}(L, c, \ell_x + y) = C[c] + \mathsf{rank}(L, c, \ell_x) + y = \mathsf{LF}(\ell_x) + y$ holds. By $i = \ell_x + y$, $\mathsf{LF}(i) = \mathsf{LF}(\ell_x) + (i - \ell_x)$ holds.

(ii) Clearly, $\mathsf{LF}(\ell_{\delta[1]}) = 1$. Next, $\mathsf{LF}(\ell_{\delta[i]}) = \mathsf{LF}(\ell_{\delta[i-1]}) + |L_{\delta[i-1]}|$ holds for any $i \in [2, r]$, because (a) the LF function maps the interval $[\ell_{\delta[i]}, \ell_{\delta[i]} + |L_{\delta[i]}| - 1]$ into the interval $[\mathsf{LF}(\ell_{\delta[i]}), \mathsf{LF}(\ell_{\delta[i]}) + |L_{\delta[i]}| - 1]$ by Lemma 3(i) for any $i \in [1, r]$, (b) LF is a bijection from $[1, n]$ to $[1, n]$, and (c) $\mathsf{LF}(\ell_{\delta[1]}) < \mathsf{LF}(\ell_{\delta[2]}) < \cdots < \mathsf{LF}(\ell_{\delta[r]})$ holds. ◀

The sequence $u_1, u_2, \ldots, u_{r+1}$ consists of sa-values such that (i) $\{u_1, u_2, \ldots, u_r\} = \{\mathsf{SA}[\ell_1 + |L_1| - 1], \mathsf{SA}[\ell_2 + |L_2| - 1], \ldots, \mathsf{SA}[\ell_r + |L_r| - 1]\}$, and (ii) $u_1 < u_2 < \cdots < u_r$. Let δ' be a permutation of $[1, r]$ satisfying $\phi^{-1}(u_{\delta'[1]}) < \phi^{-1}(u_{\delta'[2]}) < \cdots < \phi^{-1}(u_{\delta'[r]})$, and let $u_{r+1} = n + 1$. ϕ^{-1} has the following properties on RLBWT.

▶ **Lemma 4** (Lemma 3.5 in [13]). *The following three statements hold: (i) Let x be the integer satisfying $u_x \leq i < u_{x+1}$ for some integer $i \in [1, n]$. Then $\phi^{-1}(i) = \phi^{-1}(u_x) + (i - u_x)$; (ii) $\phi^{-1}(u_{\delta'[1]}) = 1$ and $\phi^{-1}(u_{\delta'[i]}) = \phi^{-1}(u_{\delta'[i-1]}) + d$ for all $i \in [2, r]$, where $d = u_{\delta'[i-1]+1} - u_{\delta'[i-1]}$; (iii) $u_1 = 1$.*

Proof. (i) Lemma 4(i) clearly holds for $i = u_x$. We show that Lemma 4(i) holds for $i \neq u_x$ (i.e., $i > u_x$). Let s_t be the position with sa-value $u_x + t$ for an integer $t \in [1, y]$ (i.e., $\mathsf{SA}[s_t] = u_x + t$), where $y = i - u_x$. s_t is not the ending position of a run (i.e., $(u_x + t) \notin \{u_1, u_2, \ldots, u_r\}$), and thus, two adjacent positions s_t and $s_t + 1$ are contained in an interval $[\ell_v, \ell_v + |L_v| - 1]$ on SA (i.e., $s_t, s_t + 1 \in [\ell_v, \ell_v + |L_v| - 1]$), which corresponds to the v-th run L_v of L. The LF function maps s_t into s_{t-1}, where s_0 is the position with sa-value u_x. LF also maps $s_t + 1$ into $s_{t-1} + 1$ by Lemma 3(i). The two mapping relationships established by LF produce y equalities $\phi^{-1}(\mathsf{SA}[s_1]) = \phi^{-1}(\mathsf{SA}[s_0]) + 1$, $\phi^{-1}(\mathsf{SA}[s_2]) = \phi^{-1}(\mathsf{SA}[s_1]) + 1$, …, $\phi^{-1}(\mathsf{SA}[s_y]) = \phi^{-1}(\mathsf{SA}[s_{y-1}]) + 1$. The equalities lead to $\phi^{-1}(\mathsf{SA}[s_y]) = \phi^{-1}(\mathsf{SA}[s_0]) + y$, which represents $\phi^{-1}(i) = \phi^{-1}(u_x) + (i - u_x)$ by $\mathsf{SA}[s_y] = i$, $\mathsf{SA}[s_0] = u_x$, and $y = i - u_x$.

(ii) Clearly, $\phi^{-1}(u_{\delta'[1]}) = 1$. $\phi^{-1}(u_{\delta'[i]}) = \phi^{-1}(u_{\delta'[i-1]}) + d$ holds for any $i \in [2, r]$, because (a) ϕ^{-1} maps the interval $[u_{\delta'[i]}, u_{\delta'[i]} + d - 1]$ into the interval $[\phi^{-1}(u_{\delta'[i]}), \phi^{-1}(u_{\delta'[i]}) + d - 1]$ by Lemma 4(i) for any $i \in [1, r]$, (b) ϕ^{-1} is a bijection from $[1, n]$ to $[1, n]$, and (c) $\phi^{-1}(u_{\delta'[1]}) < \phi^{-1}(u_{\delta'[2]}) < \cdots < \phi^{-1}(u_{\delta'[r]})$ holds.

(iii) Let p be the integer satisfying $L_p = \$$. Then there exists an integer q' such that $u_{q'}$ is the sa-value at position ℓ_p, because the length of L_p is 1. Hence, $u_1 = u_{q'} = 1$ holds. ◀

Here, we give an example of Lemma 3. In Figure 1, $(\ell_1, \ell_2, \ell_3, \ell_4) = (1, 7, 13, 14)$ and $(\mathsf{LF}(\ell_1), \mathsf{LF}(\ell_2), \mathsf{LF}(\ell_3), \mathsf{LF}(\ell_4)) = (10, 2, 1, 8)$. Hence, $\mathsf{LF}(3) = \mathsf{LF}(\ell_1) + (3 - \ell_1) = 12$ and $\mathsf{LF}(8) = \mathsf{LF}(\ell_2) + (8 - \ell_2) = 3$ hold by Lemma 3(i).

Next, we give an example of Lemma 4. In Figure 1, $(u_1, u_2, u_3, u_4) = (1, 4, 5, 9)$ and $(\phi^{-1}(u_1), \phi^{-1}(u_2), \phi^{-1}(u_3), \phi^{-1}(u_4)) = (12, 15, 8, 1)$. Hence $\phi^{-1}(3) = \phi^{-1}(u_1) + (3 - u_1) = 14$ and $\phi^{-1}(8) = \phi^{-1}(u_3) + (8 - u_3) = 11$ hold by Lemma 4(i).

3 Novel data structures for computing LF and ϕ^{-1} functions

In this section, we present two new data structures for computing LF and ϕ^{-1} functions in constant time with $O(r)$ words of space. Our key idea is to (i) divide the domains and ranges of two functions into at least r non-overlapping intervals on RLBWT and (ii) compute two

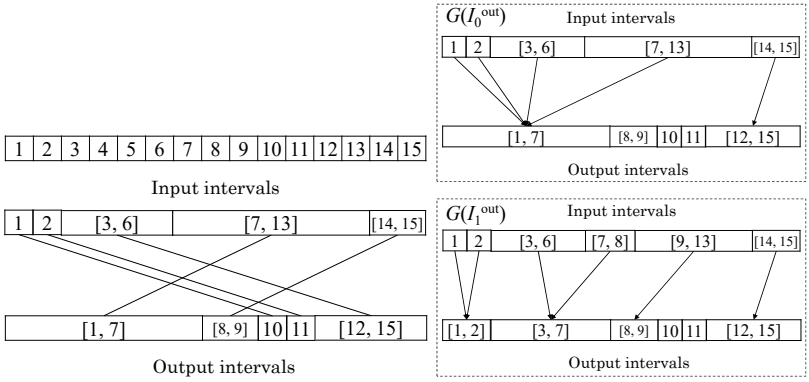


Figure 2 Left figure illustrates input and output intervals created by $I = (1, 10), (2, 11), (3, 12), (7, 1), (14, 8)$. The i-th input and output intervals are connected by a black line. Right figure illustrates two permutation graphs $G(I_0^{\text{out}})$ and $G(I_1^{\text{out}})$ for I.

functions for each domain and range by a linear search in constant time. First, we introduce a notion named *disjoint interval sequence* that is used for a function with non-overlapping intervals for its domain and range. Then, we present a *move query* for computing a function on each disjoint interval sequence and a novel data structure for efficiently computing move queries. Finally, we show that LF and ϕ^{-1} can be computed on two disjoint interval sequences using move queries.

3.1 Disjoint interval sequence and move query

Let $I = (p_1, q_1), (p_2, q_2), \ldots, (p_k, q_k)$ be a sequence of k pairs of integers. We introduce a permutation π of $[1, k]$ and sequence $d_1, d_2, \ldots, d_k$ for I. π satisfies $q_{\pi[1]} \leq q_{\pi[2]} \leq \cdots \leq q_{\pi[k]}$, and $d_i = p_{i+1} - p_i$ for $i \in [1, k]$, where $p_{k+1} = n + 1$. We call the sequence I a *disjoint interval sequence* if it satisfies the following three conditions: (i) $p_1 = 1 < p_2 < \cdots < p_k \leq n$ holds, (ii) $q_{\pi[1]} = 1$, and (iii) $q_{\pi[i]} = q_{\pi[i-1]} + d_{\pi[i-1]}$ holds for each $i \in [2, k]$.

We call the two intervals $[p_i, p_i + d_i - 1]$ and $[q_i, q_i + d_i - 1]$ the i-th *input and output intervals* of the disjoint interval sequence I, respectively, for each $i \in [1, k]$. The input intervals $[p_1, p_1 + d_1 - 1]$, $[p_2, p_2 + d_2 - 1]$, …, $[p_k, p_k + d_k - 1]$ do not overlap, i.e., $[p_i, p_i + d_i - 1] \cap [p_j, p_j + d_j - 1] = \emptyset$ holds for any pair of two distinct integers $i, j \in [1, k]$. Hence, the union of the input intervals is equal to the interval $[1, n]$, i.e., $\bigcup_{i=1}^{k} [p_i, p_i + d_i - 1] = [1, n]$. Similarly, the output intervals $[q_1, q_1 + d_1 - 1]$, $[q_2, q_2 + d_2 - 1]$, …, $[q_k, q_k + d_k - 1]$) do not overlap, and their union is equal to $[1, n]$.

A move query $\mathsf{Move}(I, i, x)$ returns a pair (i', x') on a disjoint interval sequence I for a position $i \in [1, n]$ and the index x of the input interval of I containing the position i (i.e., x is the integer satisfying $i \in [p_x, p_x + d_x - 1]$). Here, $i' = q_x + (i - p_x)$ and x' is the index of the input interval of I containing i'. We can represent a bijective function using a disjoint interval sequence and move query. Formally, let $f_I(i) = i'$ for an integer $i \in [1, n]$, where i' is the first value of the pair outputted by $\mathsf{Move}(I, i, x)$. f_I maps the j-th input interval into the j-th output interval (i.e., $f_I(i) = q_j + (i - p_j)$ for $i \in [p_j, p_j + d_j - 1]$). Hence, f_I is a bijective function from $[1, n]$ to $[1, n]$.

In Figure 2, the left figure illustrates the input and output intervals of the disjoint interval sequence $I = (1, 10), (2, 11), (3, 12), (7, 1), (14, 8)$, where $n = 15$. The input intervals created by I are $[1, 1], [2, 2], [3, 6], [7, 13]$, and $[14, 15]$. The output intervals created by I are $[10, 10], [11, 11], [12, 15], [1, 7]$, and $[8, 9]$. For example, $\mathsf{Move}(I, 3, 3) = (12, 4)$, $\mathsf{Move}(I, 5, 3) = (14, 5)$, and $\mathsf{Move}(I, 8, 4) = (2, 2)$.

3.2 Move data structure

In this section, we present a data structure called *move data structure* for computing move queries in constant time. To do so, we introduce three notions, i.e., the *permutation graph*, *split interval sequence*, and *balanced interval sequence*. A permutation graph $G(I)$ is a directed graph for a disjoint interval sequence I. The number of nodes in $G(I)$ is $2k$, and the nodes correspond one-by-one with the input and output intervals of I. Each input interval $[p_i, p_i + d_i - 1]$ has a single outgoing edge pointing to the output interval $[q_j, q_j + d_j - 1]$ containing p_i; i.e., j is the integer satisfying $p_i \in [q_j, q_j + d_j - 1]$. Hence, $G(I)$ has k edges. We say that I is *out-balanced* if every output interval has at most three incoming edges.

A split interval sequence I_t^{out} is a disjoint interval sequence for a disjoint interval sequence I and an integer $t \geq 0$. Let $I_0^{\mathsf{out}} = I$. For $t \geq 1$, we define I_t^{out} using I_{t-1}^{out} and two integers j, d if I_{t-1}^{out} is not out-balanced. Let (i) $I_{t-1}^{\mathsf{out}} = (p'_1, q'_1), (p'_2, q'_2), \ldots, (p'_{k'}, q'_{k'})$, (ii) j be the smallest integer such that the j-th output interval of I_{t-1}^{out} has at least four incoming edges in $G(I_{t-1}^{\mathsf{out}})$, and (iii) d be the largest integer satisfying $|[q_j, q_j + d - 1] \cap \{p_1, p_2, \ldots, p_{k'}\}| = 2$. Then, I_t^{out} is defined as (p'_1, q'_1), (p'_2, q'_2), …, (p'_{j-1}, q'_{j-1}), (p'_j, q'_j), $(p'_j + d, q'_j + d)$, …, $(p'_{k'}, q'_{k'})$. In other words, I_t^{out} is created by splitting the j-th pair (p'_j, q'_j) of I_{t-1}^{out} into two pairs (p'_j, q'_j) and $(p'_j + d, q'_j + d)$. Let $\tau \geq 0$ be the smallest integer such that I_τ^{out} is out-balanced.

In Figure 2, the right figure illustrates two permutation graphs $G(I_0^{\mathsf{out}})$ and $G(I_1^{\mathsf{out}})$, where I is the disjoint interval sequence illustrated in the left figure, i.e., $I = (1, 10)$, $(2, 11)$, $(3, 12)$, $(7, 1)$, $(14, 8)$. The fourth output interval $[1, 7]$ of I_0^{out} has four incoming edges, and the other output intervals have at most one incoming edge in $G(I_0^{\mathsf{out}})$. Hence, $I_1^{\mathsf{out}} = (1, 10)$, $(2, 11)$, $(3, 12)$, $(7, 1)$, $(9, 3)$, $(14, 8)$ holds by $j = 4$ and $d = 2$. I_1^{out} is out-balanced, and hence $\tau = 1$ holds.

The split interval sequence has the following four properties for each $t \in [0, \tau]$: (i) I_t^{out} consists of $k + t$ pairs. (ii) I_t^{out} consists of at least $2t$ pairs. (iii) Let $d'_i = p'_{i+1} - p'_i$ for $i \in [1, k']$ and $p'_{k'+1} = n + 1$. Both output intervals $[q'_j, q'_j + d - 1]$ and $[q'_j + d, q'_j + d'_j - 1]$ have at least two incoming edges in $G(I_t^{\mathsf{out}})$. (iv) Let f_I and f_I^t be the two bijective functions represented by I and I_t^{out}, respectively. Then, $f_I(i) = f_I^t(i)$ holds for $i \in [1, n]$. Formally, we obtain the second property from the following lemma.

▶ **Lemma 5.** *$|I_t^{\mathsf{out}}| \geq 2t$ holds for any $t \in [0, \tau]$.*

Proof. Let $\mathcal{Q}(I_{t-1}^{\mathsf{out}})$ be the set of the starting positions of input intervals in $G(I_{t-1}^{\mathsf{out}})$ (i.e., $\mathcal{Q}(I_{t-1}^{\mathsf{out}}) = \{p'_1, p'_2, \ldots, p'_{k'}\}$). Then $\mathcal{Q}(I_t^{\mathsf{out}}) = \mathcal{Q}(I_{t-1}^{\mathsf{out}}) \cup \{p'_j + d\}$ holds from the definition of I_t^{out}. Next, let $\mathsf{Edge}_2(I_{t-1}^{\mathsf{out}})$ be the set of output intervals such that each output interval has at least two incoming edges in $G(I_{t-1}^{\mathsf{out}})$, i.e., $\mathsf{Edge}_2(I_{t-1}^{\mathsf{out}}) = \{[q'_i, q'_i + d'_i - 1] \mid i \in [1, k'] \text{ s.t. } |[q'_i, q'_i + d'_i - 1] \cap \mathcal{Q}(I_{t-1}^{\mathsf{out}})| \geq 2\}$, where $d'_i = p'_{i+1} - p'_i$. $[q'_i, q'_i + d'_i - 1] \in \mathsf{Edge}_2(I_t^{\mathsf{out}})$ holds if $[q'_i, q'_i + d'_i - 1] \in \mathsf{Edge}_2(I_{t-1}^{\mathsf{out}})$ for any integer $i \in [1, k'] \setminus \{j\}$. This is because (i) $[q'_i, q'_i + d'_i - 1]$ is also an output interval of I_t^{out}, and (ii) $([q'_i, q'_i + d'_i - 1] \cap \mathcal{Q}(I_{t-1}^{\mathsf{out}})) \subseteq ([q'_i, q'_i + d'_i - 1] \cap \mathcal{Q}(I_t^{\mathsf{out}}))$ holds by $\mathcal{Q}(I_{t-1}^{\mathsf{out}}) \subseteq \mathcal{Q}(I_t^{\mathsf{out}})$. $[q'_j, q'_j + d - 1], [q'_j + d, q'_{j+1} - 1] \in \mathsf{Edge}_2(I_t^{\mathsf{out}})$ also holds by the third property of I_t^{out}. Hence, we obtain an inequality $|\mathsf{Edge}_2(I_t^{\mathsf{out}})| \geq |\mathsf{Edge}_2(I_{t-1}^{\mathsf{out}})| + 1$ for any integer $t \in [1, \tau]$. The inequality $|\mathsf{Edge}_2(I_t^{\mathsf{out}})| \geq |\mathsf{Edge}_2(I_{t-1}^{\mathsf{out}})| + 1$ guarantees that $|\mathsf{Edge}_2(I_t^{\mathsf{out}})| \geq t$ holds for any integer $t \in [0, \tau]$. The inequality $|\mathsf{Edge}_2(I_t^{\mathsf{out}})| \geq t$ indicates that I_t^{out} consists of at least $2t$ pairs, because each output interval in $\mathsf{Edge}_2(I_t^{\mathsf{out}})$ has at least two incoming edges from distinct input intervals. Hence, Lemma 5 holds. ◀

A balanced interval sequence $B(I)$ is defined as I_τ^{out} for a disjoint interval sequence I. We obtain the lemma below from the four properties of I_τ^{out}.

▶ **Lemma 6.** *Let f_I and $f_{B(I)}$ be the two bijective functions represented by I and $B(I)$, respectively for a disjoint interval sequence I of length k. The following three statements hold: (i) $|B(I)| \leq 2k$, (ii) $B(I)$ is out-balanced, and (iii) the two disjoint interval sequences I and $B(I)$ represent the same bijective function, i.e., $f_I(i) = f_{B(I)}(i)$ for $i \in [1, n]$.*

Proof. (i) We obtain an inequality $\tau \leq k$ from the first and second properties of I_t^{out}, because $k + t \geq 2t$ must hold for any $t \in [0, \tau]$. Hence, I_τ^{out} consists of at most $2k$ pairs; i.e., $|B(I)| \leq 2k$ holds. (ii) I_τ^{out} is out-balanced, and thus, $B(I)$ is out-balanced. (iii) $f_I(i) = f_I^0(i) = f_I^1(i) = \cdots = f_I^\tau(i) = f_{B(I)}(i)$, and thus, $f_I(i) = f_{B(I)}(i)$ for $i \in [1, n]$. ◀

The move data structure $F(I)$ is built on a balanced interval sequence $B(I) = (p_1, q_1)$, (p_2, q_2), …, $(p_{k'}, q_{k'})$ for a disjoint interval sequence I, and it supports move queries on $B(I)$. The move data structure consists of two arrays D_{pair} and D_{index} of size k'. $D_{\mathsf{pair}}[i]$ stores the i-th pair (p_i, q_i) of $B(I)$ for each $i \in [1, k']$. $D_{\mathsf{index}}[i]$ stores the index j of the input interval containing q_i. Hence, the space usage is $O(k')$ words in total.

Now let us describe an algorithm for solving a move query $\mathsf{Move}(B(I), i, x) = (i', x')$ on $B(I)$, where x and x' are the indexes of the two input intervals of $B(I)$ containing i and i', respectively, and $i' = q_x + (i - p_x)$. The algorithm consists of three steps. In the first step, the algorithm computes $i' = q_x + (i - p_x)$. In the second step, the algorithm finds the x'-th input interval by a linear search on the input intervals of $B(I)$. Let $b = D_{\mathsf{index}}[x]$. The linear search starts at the b-th input interval $[p_b, p_{b+1} - 1]$, reads the input intervals in the left-to-right order, and stops if the input interval containing position i' is found (i.e., the x'-th input interval). The linear search is always successful (i.e., $x' \geq b$), because $i' \geq q_x$ holds. In the third step, the algorithm returns the pair (i', x'). The running time of the algorithm is $O(x' - b + 1)$ in total.

The running time is computed as follows. Let i_{beg} and i_{end} be the indexes of the first and last input intervals that are connected to the x-th output interval in $G(B(I))$. The x-th output interval has at most three incoming edges, and hence, $i_{\mathsf{end}} - i_{\mathsf{beg}} + 1 \leq 3$ holds. Since b is the index of an input interval that overlaps the x-th output interval, $i_{\mathsf{beg}} - 1 \leq b \leq i_{\mathsf{end}}$. Similarly, $i_{\mathsf{beg}} - 1 \leq x' \leq i_{\mathsf{end}}$. Therefore, $x' - b \leq 3$ and we can solve the move query in constant time.

3.3 Computing LF and ϕ^{-1} functions using move data structures

Here, we show that we can compute the LF function using a move data structure. Recall that ℓ_i is the starting position of the i-th run on BWT L for $i \in [1, r]$, and δ is the permutation of $[1, r]$ introduced in Section 2.2. The sequence I_{LF} is defined as r pairs $(\ell_1, \mathsf{LF}(\ell_1))$, $(\ell_2, \mathsf{LF}(\ell_2))$, …, $(\ell_r, \mathsf{LF}(\ell_r))$. I_{LF} satisfies the three conditions of a disjoint interval sequence by Lemma 3, i.e., (i) $\ell_1 = 1 < \ell_2 < \cdots < \ell_r \leq n$, (ii) $\mathsf{LF}(\ell_{\delta[1]}) = 1$, and (iii) $\mathsf{LF}(\ell_{\delta[i]}) = \mathsf{LF}(\ell_{\delta[i-1]}) + |L_{\delta[i-1]}|$ holds for each $i \in [2, r]$. Hence I_{LF} is a disjoint interval sequence.

Let f_{LF} be the bijective function represented by the disjoint interval sequence I_{LF}. Then, $f_{\mathsf{LF}}(i) = \mathsf{LF}(\ell_x) + (i - \ell_x)$ holds, where x is the integer such that $\ell_x \leq i < \ell_{x+1}$ holds. On the other hand, we have $\mathsf{LF}(i) = \mathsf{LF}(\ell_x) + (i - \ell_x)$ by Lemma 3(i). Hence, f_{LF} and LF are the same function, i.e., $\mathsf{LF}(i) = f_{\mathsf{LF}}(i)$ for $i \in [1, n]$.

Let $F(I_{\mathsf{LF}})$ be the move data structure built on the balanced interval sequence $B(I_{\mathsf{LF}})$ for I_{LF}. By Lemma 6, the move data structure requires $O(r)$ words of space, and $\mathsf{LF}(i) = i'$ holds for a move query $\mathsf{Move}(B(I), i, x) = (i', x')$ on $B(I_{\mathsf{LF}})$. Hence, we have proven the following theorem.

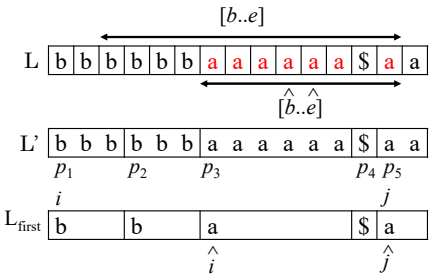


Figure 3 Example of modified toehold lemma.

▶ **Theorem 7.** *Let x and x' be the indexes of the two input intervals of $B(I_{\mathsf{LF}})$ containing an integer $i \in [1, n]$ and $\mathsf{LF}(i)$, respectively. We can compute $\mathsf{LF}(i)$ and x' in constant time by using $F(I_{\mathsf{LF}})$ and (i, x).*

Similarly, we can show that we can compute ϕ^{-1} by using a move data structure. A sequence I_{SA} consists of r pairs $(u_1, \phi^{-1}(u_1))$, $(u_2, \phi^{-1}(u_2))$, ..., $(u_r, \phi^{-1}(u_r))$, where $u_1, u_2, \ldots, u_r$ are the integers introduced in Section 2.2. I_{SA} has the following three properties: (i) $u_1 = 1 < u_2 < \cdots < u_r \leq n$ holds by Lemma 4(iii), (ii) $\phi^{-1}(u_{\delta'[1]}) = 1$ by Lemma 4(ii), and (iii) $\phi^{-1}(u_{\delta'[i]}) = \phi^{-1}(u_{\delta'[i-1]}) + (u_{\delta'[i-1]+1} - u_{\delta'[i-1]})$ holds by Lemma 4(ii) for each $i \in [2, r]$, where δ' is the permutation of $[1, r]$ introduced in Section 2.2. Hence, I_{SA} satisfies the three conditions of a disjoint interval sequence by Lemma 4.

Let f_{SA} be the bijective function represented by the disjoint interval sequence I_{SA}. Then $f_{\mathsf{SA}}(i) = \phi^{-1}(u_x) + (i - u_x)$ holds, where x is the integer such that $u_x \leq i < u_{x+1}$ holds. On the other hand, $\phi^{-1}(i) = \phi^{-1}(u_x) + (i - u_x)$ holds by Lemma 4(i). Hence f_{SA} and $\phi^{-1}(i)$ are the same function.

Let $F(I_{\mathsf{SA}})$ be the move data structure built on the balanced interval sequence $B(I_{\mathsf{SA}})$ for I_{SA}. Then, the result of a move query on $B(I_{\mathsf{SA}})$ contains $\phi^{-1}(i)$ for $i \in [1, n]$, and hence, we have proven (i) of the following theorem.

▶ **Theorem 8.** *Let x, x', $\hat{x}$ be the indexes of the three input intervals of $B(I_{\mathsf{SA}})$ containing an integer $i \in [1, n]$, $\phi^{-1}(i)$, and $i - 1$, respectively. Then, the following two statements hold: (i) We can compute $\phi^{-1}(i)$ and x' in constant time using data structure $F(I_{\mathsf{SA}})$ and the pair (i, x). (ii) We can compute the index $\hat{x}$ using $F(I_{\mathsf{SA}})$ and (i, x).*

Proof. (ii) Let $B(I_{\mathsf{SA}}) = (p_1, q_1), (p_2, q_2), \ldots, (p_{k'}, q_{k'})$. The x-th input interval is $[p_x, p_{x+1} - 1]$, which contains i. $\hat{x} = x$ holds if $p_x \neq i$; otherwise, $\hat{x} = x - 1$. We can verify $p_x \neq i$ holds in constant time by using $F(I_{\mathsf{SA}})$. ◀

Theorems 7 and 8 indicate that we can compute the position obtained by recursively applying LF and ϕ^{-1} to a position $i \in [1, n]$ t times in $O(t)$ time if we know the index of the input interval containing i. For example, let x, x', x'', and x''' be the indexes of the four input intervals of $B(I_{\mathsf{LF}})$ containing i, $\mathsf{LF}(i)$, $\mathsf{LF}(\mathsf{LF}(i))$, and $\mathsf{LF}(\mathsf{LF}(\mathsf{LF}(i)))$, respectively. $\mathsf{LF}(\mathsf{LF}(\mathsf{LF}(i)))$ can be computed by computing three move queries $\mathsf{Move}(B(I_{\mathsf{LF}}), i, x) = (\mathsf{LF}(i), x')$, $\mathsf{Move}(B(I_{\mathsf{LF}}), \mathsf{LF}(i), x') = (\mathsf{LF}(\mathsf{LF}(i)), x'')$, and $\mathsf{Move}(B(I_{\mathsf{LF}}), \mathsf{LF}(\mathsf{LF}(i)), x'') = (\mathsf{LF}(\mathsf{LF}(\mathsf{LF}(i))), x''')$.

4 New data structure for backward searches

Here, we present a modified version of the *backward search* [10, 1], which we call *backward search query for OptBWTR* (BSR query), for computing the sa-interval of cP for a given string P and character c. To define the BSR query, we will introduce a new tuple: a *balanced*

sa-interval of a string P is a 6-tuple $(b, e, \mathsf{SA}[b], i, j, v)$. Here, (i) $[b, e]$ is the sa-interval of P; (ii) i and j are the indexes of the two input intervals of $B(I_{\mathsf{LF}})$ containing b and e, respectively; (iii) v is the index of the input interval of $B(I_{\mathsf{SA}})$ containing $\mathsf{SA}[b]$. The balanced sa-interval of P is undefined if the sa-interval of P is $\emptyset$ (i.e., P is not a substring of T). The input of the BSR query is the balanced sa-interval $(b, e, \mathsf{SA}[b], i, j, v)$ of a string P and a character c. The output of the BSR query is the balanced sa-interval $(b', e', \mathsf{SA}[b'], i', j', v')$ of string cP if the sa-interval of cP is not the empty set; otherwise BSR outputs a mark $\perp$.

Now, we will present a data structure called the *BSR data structure*. The BSR data structure supports BSR queries in $O(\log \log_w \sigma)$ time. It consists of five data structures $F(I_{\mathsf{LF}})$, $F(I_{\mathsf{SA}})$, $R(L_{\mathsf{first}})$, SA^+, and $\mathsf{SA}^+_{\mathsf{index}}$. Here, $F(I_{\mathsf{LF}})$ and $F(I_{\mathsf{SA}})$ are the two move data structures introduced in Section 3.3. Let $B(I_{\mathsf{LF}}) = (p_1, q_1), (p_2, q_2), \ldots, (p_k, q_k)$. Then L_{first} is the string satisfying $L_{\mathsf{first}} = L[p_1], L[p_2], \ldots, L[p_k]$. $R(L_{\mathsf{first}})$ is a rank-select data structure built on L_{first}, which is defined in Section 2. $R(L_{\mathsf{first}})$ requires $O(|L_{\mathsf{first}}|)$ words of space, and it supports rank and select queries on L_{first} in $O(\log \log_w \sigma)$ and $O(1)$ time, respectively. SA^+ is an array of size k such that $\mathsf{SA}^+[x]$ stores the sa-value at the starting position of the x-th input interval of $B(I_{\mathsf{LF}})$ for each $x \in [1, k]$ (i.e., $\mathsf{SA}^+[x] = \mathsf{SA}[p_x]$). Let $B(I_{\mathsf{SA}}) = (p'_1, q'_1), (p'_2, q'_2), \ldots, (p'_{k'}, q'_{k'})$. $\mathsf{SA}^+_{\mathsf{index}}$ is an array of size k such that $\mathsf{SA}^+_{\mathsf{index}}[x]$ stores the index y of the input interval of $B(I_{\mathsf{SA}})$ containing the position $\mathsf{SA}^+[x]$ (i.e., y is the integer satisfying $\mathsf{SA}^+[x] \in [p'_y, p'_{y+1} - 1]$). The space usage of the five data structures is $O(|B(I_{\mathsf{LF}})| + |B(I_{\mathsf{SA}})|)$ words, and $|B(I_{\mathsf{LF}})|, |B(I_{\mathsf{SA}})| = O(r)$ holds by Lemma 6(i).

Next, we will present a key observation on BSR queries, which is based on the toehold lemma (see, e.g., [23, 13, 1]). Let L' be a sequence of k substrings $L[p_1..p_2 - 1], L[p_2..p_3 - 1], \ldots, L[p_k..p_{k+1} - 1]$ of BWT L, where $p_{k+1} = n + 1$. Then, L' has the following properties: (i) L' represents a partition of L. (ii) Each string of L' consists of a repetition of the same character. (iii) Each character $L_{\mathsf{first}}[t]$ corresponds to the first character of the t-th string of L'. (iv) The i-th and j-th strings of L' contain the b-th and e-th characters of BWT L, respectively. (v) Let $\hat{b}$ and $\hat{e}$ be the first and last occurrences of c in $L[b..e]$ (i.e., $\hat{b} = \min\{t \mid t \in [b, e] \text{ s.t. } L[t] = c\}$ and $\hat{e} = \max\{t \mid t \in [b, e] \text{ s.t. } L[t] = c\}$). Similarly, let $\hat{i}$ and $\hat{j}$ be the indexes of the two strings of L' containing the $\hat{b}$-th and $\hat{e}$-th characters of BWT L, respectively. Then $\hat{i}$ and $\hat{j}$ are equal to the first and last occurrences of c in $L_{\mathsf{first}}[i..j]$. We obtain the following relations among the four positions b, $\hat{b}$, e, and $\hat{e}$ by using the above five properties: (i) $\hat{b} = b$ if $L_{\mathsf{first}}[i] = c$; otherwise, $\hat{b} = p_{\hat{i}}$. (ii) Similarly, $\hat{e} = e$ if $L_{\mathsf{first}}[j] = c$; otherwise $\hat{b} = p_{\hat{j}+1} - 1$. We call these two relations the *modified toehold lemma*.

Let $\hat{v}$ be the index of the input interval of $B(I_{\mathsf{SA}})$ containing position $\mathsf{SA}[\hat{b}]$. $\hat{v} = v$ and $\mathsf{SA}[\hat{b}] = \mathsf{SA}[b]$ hold if $\hat{b} = b$; otherwise, $\hat{v} = \mathsf{SA}^+_{\mathsf{index}}[\hat{i}]$ and $\mathsf{SA}[\hat{b}] = \mathsf{SA}^+[\hat{i}]$ by the modified toehold lemma. We can compute the balanced sa-interval of cP by using $F(I_{\mathsf{LF}})$ and $F(I_{\mathsf{SA}})$ after computing the six integers $\hat{b}, \hat{e}, \hat{i}, \hat{j}, \hat{v}, \mathsf{SA}[\hat{b}]$, because $b' = \mathsf{LF}(\hat{b})$, $e' = \mathsf{LF}(\hat{e})$, and $\mathsf{SA}[b'] = \mathsf{SA}[\hat{b}] - 1$ hold by Lemma 2.

Figure 3 illustrates an example of the modified toehold lemma for a BWT $L = bbbbbb\ aaaaaa\$aa$. In this example, $c = a$ and $L' = bbb, bbb, aaaaaa, \$, aa$. (i) $k = 5$, (ii) $(p_1, p_2, p_3, p_4, p_5) = (1, 4, 7, 13, 14)$, (ii) $L_{\mathsf{first}} = bba\$a$, (iii) $(b, e) = (3, 14)$, (iv) $(\hat{b}, \hat{e}) = (7, 14)$, (v) $(i, j) = (1, 5)$, and (vi) $(\hat{i}, \hat{j}) = (3, 5)$. The i-th string of L' is not a repetition of the character c, and the $\hat{i}$-th string of L' contains the $\hat{b}$-th character of L. Hence $\hat{b} = p_{\hat{i}} = 7$ holds by the modified toehold lemma. Similarly, the j-th string of L' is a repetition of c, and hence $\hat{e} = e$ holds by the modified toehold lemma.

We solve a BSR query in four steps. In the first step, we verify whether $L_{\mathsf{first}}[i..j]$ contains character c by computing two rank queries $\mathsf{rank}(L_{\mathsf{first}}, c, j)$ and $\mathsf{rank}(L_{\mathsf{first}}, c, i)$. By Lemma 1(i), $L_{\mathsf{first}}[i..j]$ contains c if $\mathsf{rank}(L_{\mathsf{first}}, c, j) - \mathsf{rank}(L_{\mathsf{first}}, c, i) \geq 1$; otherwise, cP is not a substring

of T, and hence BSR outputs a mark $\perp$. In the second step, we compute two integers $\hat{i}$ and $\hat{j}$ using rank and select queries on the string L_{first}. $\hat{i} = \mathsf{select}(L_{\mathsf{first}}, c, \mathsf{rank}(L_{\mathsf{first}}, c, i-1)+1)$ and $\hat{j} = \mathsf{select}(L_{\mathsf{first}}, c, \mathsf{rank}(L_{\mathsf{first}}, c, j))$ hold by Lemma 1(ii). In the third step, we compute $\hat{b}$, $\hat{e}$, $\hat{v}$, and $\mathsf{SA}[\hat{b}]$ by the modified toehold lemma. In the fourth step, we compute the balanced sa-interval of cP by processing the six integers $\hat{b}, \hat{e}, \hat{i}, \hat{j}, \hat{v}, \mathsf{SA}[\hat{b}]$, i.e., we compute (i) the pair (b', i') using a move query on $B(I_{\mathsf{LF}})$ for the pair $(\hat{b}, \hat{i})$, (ii) the pair (e', j') using a move query on $B(I_{\mathsf{LF}})$ for the pair $(\hat{e}, \hat{j})$, and (iii) the pair $(\mathsf{SA}[v'], v')$ by Theorem 8(ii). The running time is $O(\log \log_w \sigma)$ in total.

5 OptBWTR

Here, we present OptBWTR, which supports optimal-time queries for polylogarithmic alphabets by leveraging data structures for computing LF and ϕ^{-1} functions. Let P be a string of length m in a count or locate query and $occ = |Occ(T, P)|$. The goal of this section is to prove the following theorem.

▶ **Theorem 9.** *OptBWTR requires $O(r)$ words, and it supports count and locate queries on a string T in $O(m \log \log_w \sigma)$ and $O(m \log \log_w \sigma + occ)$ time, respectively. We can construct OptBWTR in $O(n + r \log r)$ time and $O(r)$ words by processing the RLBWT of T.*

Proof. See the full version of this paper [22] for the proof of the construction time and working space in Theorem 9. ◀

OptBWTR consists of the five data structures composing the BSR data structure, i.e., $F(I_{\mathsf{LF}})$, $F(I_{\mathsf{SA}})$, $R(L_{\mathsf{first}})$, SA^+, and $\mathsf{SA}^+_{\mathsf{index}}$. First, we present an algorithm for a count query using OptBWTR that consists of two phases. In the first phase, the algorithm computes the balanced sa-interval of P by iterating BSR query m times. The input of the i-th BSR query is the $(m-i+1)$-th character of P (i.e., $P[m-i+1]$) and the balanced sa-interval of $P[m-i+2..m]$ for each $i \in [1, m]$. Here, $P[m+1..m]$ is defined as the empty string ε. The balanced sa-interval of ε is $(1, n, n, 1, |B(I_{\mathsf{LF}})|, |B(I_{\mathsf{SA}})|)$, because (i) the sa-interval of the empty string is $[1, n]$, and (ii) $\mathsf{SA}[1] = n$. The i-th BSR query outputs the balanced sa-interval of $P[m-i+1..m]$ if $P[m-i+1..m]$ is a substring of T; otherwise it outputs a mark $\perp$. If a BSR query outputs $\perp$, the pattern P does not occur in T. In this case, the algorithm stops and returns 0 as the solution for the count query. In the second phase, the algorithm returns the length of the sa-interval $[b, e]$ of P (i.e., $e-b+1$) as the solution for the count query, because $occ = e-b+1$ holds. The sa-interval of P is contained in the balanced sa-interval of P; hence, the running time is $O(m \log \log_w \sigma)$ in total.

Next, we present an algorithm for a locate query using OptBWTR. Assume that we already computed the balanced sa-interval of P by the algorithm for the count query. Let v_t be the index of the input interval of $B(I_{\mathsf{SA}})$ containing $\mathsf{SA}[b+t]$ for $t \in [0, e-b]$. Then $\mathsf{SA}[b+1..e]$ can be computed by computing $(e-b)$ move queries $\mathsf{Move}(B(I_{\mathsf{SA}}), \mathsf{SA}[b], v_0) = (\mathsf{SA}[b+1], v_1)$, $\mathsf{Move}(B(I_{\mathsf{SA}}), \mathsf{SA}[b+1], v_1) = (\mathsf{SA}[b+2], v_2)$, …, $\mathsf{Move}(B(I_{\mathsf{SA}}), \mathsf{SA}[e-1], v_{e-b}) = (\mathsf{SA}[e], v_{e-b+1})$ on $B(I_{\mathsf{SA}})$. The first sa-value $\mathsf{SA}[b]$ and the index v_0 are stored in the balanced sa-interval of P.

The algorithm for a locate query also consists of two phases. In the first phase, the algorithm computes the balanced sa-interval of P by iterating BSR query m times. In the second phase, it computes $(e-b)$ move queries $\mathsf{Move}(B(I_{\mathsf{SA}}), \mathsf{SA}[b], v_0)$, $\mathsf{Move}(B(I_{\mathsf{SA}}), \mathsf{SA}[b+1], v_1)$, …, $\mathsf{Move}(B(I_{\mathsf{SA}}), \mathsf{SA}[e-1], v_{e-b})$ by using the move data structure $F(I_{\mathsf{SA}})$, and outputs $\mathsf{SA}[b..e]$. Hence, we can solve a locate query in $O(m \log \log_w \sigma + occ)$ time.

6 Applications

In this section, we show that OptBWTR can support extract, decompression, and prefix search queries in optimal time.

Extract query. Let a string T of length n have b marked positions $i_1, i_2, \ldots, i_b \in [1, n]$. An extract query (also called the bookmarking problem) is to return substring $T[i_j..i_j + d - 1]$ for a given integer $j \in [1, b]$ and $d \in [1, n - i_j + 1]$.

We will use *FL function* to solve extract queries. FL is the inverse function of LF function, i.e., $\mathsf{FL}(\mathsf{LF}(i)) = i$ holds for $i \in [1, n]$. We will also use the function FL_x and integers $h_1, h_2, \ldots, h_b$. $\mathsf{FL}_x(i)$ returns the position obtained by recursively applying the FL function to a given integer i x times, i.e., $\mathsf{FL}_0(i) = i$ and $\mathsf{FL}_x(i) = \mathsf{FL}_{x-1}(\mathsf{FL}(i))$ for $x \geq 1$. h_j is the position with sa-value i_j on SA (i.e., $\mathsf{SA}[h_j] = i_j$). The FL function returns the position with the sa-value $y + 1$ on SA for a given position with sa-value y, and hence $T[i_j..i_j + d - 1] = F[\mathsf{FL}_0(h_j)], F[\mathsf{FL}_1(h_j)], \ldots, F[\mathsf{FL}_{d-1}(h_j)]$ holds for $j \in [1, b]$, where F is the string described in Section 2.2. We can construct a data structure of $O(r)$ words to compute FL function in constant time by modifying Theorem 7 and can solve an extract query in linear time by using the data structure. See the full version of this paper [22] for details of our data structure for solving extract queries.

▶ **Theorem 10.** *There exists a data structure of $O(r + b)$ words that solves the bookmarking problem for a string T and b positions $i_1, i_2, \ldots, i_b$ ($1 \leq i_1 < i_2 < \cdots < i_b \leq n$). This data structure supports an exact query in constant time per character. We can construct the data structure in $O(n)$ time and $O(r + b)$ words of space by processing the RLBWT and positions $i_1, i_2, \ldots, i_b$.*

Proof. See the full version of this paper [22]. ◀

Decompression of RLBWT. We apply Theorem 10 to $T[1..n]$ with marked position 1. Then, our data structure for extract queries can return the string T in $O(n)$ time (i.e., the data structure can recover T from the RLBWT of T in linear time to n). The $O(n)$ time decompression is the fastest among other decompression algorithms on compressed indexes in $O(r)$ words of space, as the following theorem shows.

▶ **Theorem 11.** *We can compute the characters of T in left-to-right order (i.e., $T[1], T[2], \ldots, T[n]$) in $O(n)$ time and $O(r)$ words of space by processing the RLBWT of string T.*

Prefix search. The prefix search for a set of strings $D = \{T_1, T_2, \ldots, T_d\}$ returns the indexes of the strings in D that include a given string P as their prefixes (i.e., $\{i \mid i \in [1, d]$ s.t. $T_i[1..|P|] = P\}$). We can construct a data structure supporting the prefix search by combining Theorem 10 with *compact trie* [20].

A compact trie for a set of strings D is a trie for D such that all unary paths are collapsed, and each node represents the string by concatenating labels on the path from the root to the node. For simplicity, we assume that the set D is *prefix-free*, i.e., T_i is not a prefix of $T_{i'}$ for any pair of two strings T_i and $T_{i'}$ in D. Each leaf in the compact trie represents a distinct string in D by the assumption. Let v be the node such that (i) P is a prefix of the string represented by the node and (ii) P is not a prefix of the string represented by its parent. Then, the leaves under v are the output of the prefix search query for P.

To find v, we decode the string on the path from the root to the node v in linear time using exact queries for the path. After we find v, we traverse the subtree rooted at v and output all the leaves in the subtree. This procedure runs in $O(|P| + occ')$ time, where occ' is the number of leaves under the lowest node. See the full version of this paper [22] for the details of our data structure for solving prefix search queries.

▶ **Theorem 12.** *Let r' be the number of runs in the RLBWT of a string T containing all the strings in $D = \{T_1, T_2, \ldots, T_d\}$. There exists a data structure that supports a prefix search on D in $O(|P| + occ')$ time and $O(r' + d)$ words of space for a string P. The data structure also returns the number of the strings in D that include P as their prefixes in $O(|P|)$ time.*

Proof. See the full version of this paper [22]. ◀

7 Conclusion

We presented OptBWTR, the first string index that can support count and locate queries on RLBWT in optimal time with $O(r)$ words of space for polylogarithmic alphabets. OptBWTR also supports extract queries and prefix searches on RLBWT in optimal time for any alphabet size. In addition, we presented the first decompression algorithm working in optimal time and $O(r)$ words of working space. This is the first optimal-time decompression algorithm working in $O(r)$ words of space.

We presented a new data structure of $O(r)$ words for computing LF and ϕ^{-1} functions in constant time by using a new data structure named move data structure, provided that we use an additional input. We also showed that the backward search works in optimal time for polylogarithmic alphabets with $O(r)$ words of space using the data structure. The two functions and the backward search are general and applicable to various queries on RLBWT.

The following problems remain open: Does there exist a string index of $O(r)$ words supporting locate queries in optimal time for any alphabet size? We assume $\sigma = O(\text{polylog}\, n)$ for supporting locate queries in optimal time with $O(r)$ words. As mentioned in Section 1, a faster version of r-index can support locate queries in optimal time with $O(r \log \log_w(\sigma + (n/r)))$ words. Thus, improving OptBWTR so that it can support locate queries in optimal time with $O(r)$ words for any alphabet size is an important future work. For this goal, one needs to solve a rank query on a string of length $\Theta(r)$ in constant time and $O(r)$ words of space. However, this seems impossible because any data structure of $O(r)$ words requires $\Omega(\log \log_w \sigma)$ time to compute a rank query on a string of length r [4]. Perhaps, we may be able to compute the sa-interval of a given pattern in $O(m)$ time and $O(r)$ words of space without using rank queries. After computing the sa-interval of the pattern, we can solve the locate query in optimal time by using our data structure for the ϕ^{-1} function.

References

1. Hideo Bannai, Travis Gagie, and Tomohiro I. Refining the r-index. *Theoretical Computer Science*, 812:96–108, 2020.
2. Djamal Belazzougui, Paolo Boldi, and Sebastiano Vigna. Dynamic z-fast tries. In *Proceedings of SPIRE*, pages 159–172, 2010.
3. Djamal Belazzougui and Gonzalo Navarro. Alphabet-independent compressed text indexing. *ACM Transactions on Algorithms*, 10:23:1–23:19, 2014.
4. Djamal Belazzougui and Gonzalo Navarro. Optimal lower and upper bounds for representing sequences. *ACM Transactions on Algorithms*, 11:31:1–31:21, 2015.
5. Michael Burrows and David J Wheeler. A block-sorting lossless data compression algorithm. *Technical report*, 1994.

6 Anders Roy Christiansen and Mikko Berggren Ettienne. Compressed indexing with signature grammars. In *Proceedings of LATIN*, pages 331–345, 2018.

7 Anders Roy Christiansen, Mikko Berggren Ettienne, Tomasz Kociumaka, Gonzalo Navarro, and Nicola Prezza. Optimal-time dictionary-compressed indexes. *ACM Transactions on Algorithms*, 17:8:1–8:39, 2021.

8 Francisco Claude and Gonzalo Navarro. Improved grammar-based compressed indexes. In *Proceedings of SPIRE*, pages 180–192, 2012.

9 Patrick Hagge Cording, Pawel Gawrychowski, and Oren Weimann. Bookmarks in grammar-compressed strings. In *Proceedings of SPIRE*, pages 153–159, 2016.

10 Paolo Ferragina and Giovanni Manzini. Indexing compressed text. *Journal of the ACM*, 52:552–581, 2005.

11 Paolo Ferragina, Giovanni Manzini, Veli Mäkinen, and Gonzalo Navarro. Compressed representations of sequences and full-text indexes. *ACM Transactions on Algorithms*, 3:20, 2007.

12 Travis Gagie, Pawel Gawrychowski, Juha Kärkkäinen, Yakov Nekrich, and Simon J. Puglisi. LZ77-based self-indexing with faster pattern matching. In *Proceedings of LATIN*, pages 731–742, 2014.

13 Travis Gagie, Gonzalo Navarro, and Nicola Prezza. Fully functional suffix trees and optimal text searching in BWT-runs bounded space. *Journal of the ACM*, 67, 2020.

14 Alexander Golynski, J. Ian Munro, and S. Srinivasa Rao. Rank/select operations on large alphabets: a tool for text indexing. In *Proceedings of SODA*, pages 368–373, 2006.

15 Juha Kärkkäinen, Giovanni Manzini, and Simon J. Puglisi. Permuted longest-common-prefix array. In *Proceedings of CPM*, pages 181–192, 2009.

16 Dominik Kempa. Optimal construction of compressed indexes for highly repetitive texts. In *Proceedings of SODA*, pages 1344–1357, 2019.

17 Ulrich Lauther and Tamás Lukovszki. Space efficient algorithms for the Burrows-Wheeler backtransformation. *Algorithmica*, 58:339–351, 2010.

18 Veli Mäkinen, Gonzalo Navarro, Jouni Sirén, and Niko Välimäki. Storage and retrieval of highly repetitive sequence collections. *Journal of Computational Biology*, 17:281–308, 2010.

19 Udi Manber and Eugene W. Myers. Suffix arrays: A new method for on-line string searches. *SIAM Journal on Computing*, 22:935–948, 1993.

20 Donald R. Morrison. PATRICIA – practical algorithm to retrieve information coded in alphanumeric. *Journal of the ACM*, 15:514–534, 1968.

21 Gonzalo Navarro and Nicola Prezza. Universal compressed text indexing. *Theoretical Computer Science*, 762:41–50, 2019.

22 Takaaki Nishimoto and Yasuo Tabei. Optimal-time queries on BWT-runs compressed indexes. *CoRR*, abs/2006.05104, 2021. `arXiv:2006.05104`.

23 Alberto Policriti and Nicola Prezza. LZ77 computation based on the run-length encoded BWT. *Algorithmica*, 80:1986–2011, 2018.

24 Milan Ruzic. Constructing efficient dictionaries in close to sorting time. In *Proceedings of ICALP*, pages 84–95, 2008.

25 Takuya Takagi, Shunsuke Inenaga, Kunihiko Sadakane, and Hiroki Arimura. Packed compact tries: A fast and efficient data structure for online string processing. *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, 100-A:1785–1793, 2017.

26 Kazuya Tsuruta, Dominik Köppl, Shunsuke Kanda, Yuto Nakashima, Shunsuke Inenaga, Hideo Bannai, and Masayuki Takeda. c-trie++: A dynamic trie tailored for fast prefix searches. In *Proceedings of DCC*, pages 243–252, 2020.

Application of the Level-2 Quantum Lasserre Hierarchy in Quantum Approximation Algorithms

Ojas Parekh
Sandia National Laboratories, Albuquerque, NM, USA

Kevin Thompson
Sandia National Laboratories, Albuquerque, NM, USA

Abstract

The Lasserre Hierarchy, [18, 19], is a set of semidefinite programs which yield increasingly tight bounds on optimal solutions to many NP-hard optimization problems. The hierarchy is parameterized by levels, with a higher level corresponding to a more accurate relaxation. High level programs have proven to be invaluable components of approximation algorithms for many NP-hard optimization problems [3, 7, 26]. There is a natural analogous quantum hierarchy [5, 8, 24], which is also parameterized by level and provides a relaxation of many (QMA-hard) quantum problems of interest [5, 6, 9]. In contrast to the classical case, however, there is only one approximation algorithm which makes use of higher levels of the hierarchy [5]. Here we provide the first ever use of the level-2 hierarchy in an approximation algorithm for a particular QMA-complete problem, so-called Quantum Max Cut [2, 9]. We obtain modest improvements on state-of-the-art approximation factors for this problem, as well as demonstrate that the level-2 hierarchy satisfies many physically-motivated constraints that the level-1 does not satisfy. Indeed, this observation is at the heart of our analysis and indicates that higher levels of the quantum Lasserre Hierarchy may be very useful tools in the design of approximation algorithms for QMA-complete problems.

2012 ACM Subject Classification Theory of computation → Approximation algorithms analysis; Theory of computation → Semidefinite programming; Theory of computation → Quantum complexity theory

Keywords and phrases Quantum Max Cut, Quantum Approximation Algorithms, Lasserre Hierarchy, Local Hamiltonian, Heisenberg model

Digital Object Identifier 10.4230/LIPIcs.ICALP.2021.102

Category Track A: Algorithms, Complexity and Games

Funding Sandia National Laboratories is a multimission laboratory managed and operated by National Technology and Engineering Solutions of Sandia, LLC., a wholly owned subsidiary of Honeywell International, Inc., for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-NA-0003525. This work was supported by the U.S. Department of Energy, Office of Science, Office of Advanced Scientific Computing Research, Accelerated Research in Quantum Computing and Quantum Algorithms Teams programs.

1 Introduction

The study of many body quantum systems, and their corresponding spectra is of utmost importance in many sub-fields of physics [4]. These systems generally have an exponentially large dimension, so a direct calculation is intractable. Indeed, determining the highest or lowest energy of a quantum state is the canonical QMA-hard problem [4, 17], so we should not expect to solve the problem even with access to a quantum computer. Hence, the study of algorithms which produce approximate solutions emerges as an interesting direction of study. These problems are made even more interesting by the fact that, in contrast to the classical case [29, 30], there are relatively few known rigorous approximation algorithms known.

Prior work. The 2-Local Hamiltonian problem has been a cornerstone of quantum complexity theory; however, it has been recently studied in the context of approximation algorithms [2, 5, 6, 9, 12, 22]. Many of these algorithms draw inspiration from the seminal Goemans-Williamson Max Cut approximation algorithm [10] or other appropriate classical counterparts [26]. For classical approximation algorithms, an effective meta-algorithm is to solve a linear or semidefinite program (SDP) which relaxes the (NP-hard) optimization problem, followed by a rounding procedure which seeks to turn the optimal SDP variable into a solution in the appropriate domain (binary, integral, etc.). The SDP provides a polynomial-time-computable bound on the optimization problem hence bounding the loss in objective allows one to bound the ratio of the objective obtained to the optimal solution (this quantity is called the approximation factor). In the quantum case, the SDP variable is polynomial size, and the goal is to produce a (classical description) of an exponentially large quantum state, again with quantifiable loss. Most such results use the same quantum generalization [5, 6, 9] of a semidefinite programming hierarchy discovered independently by several authors in the classical case [11, 18, 23]. Variations in the aforementioned results [6, 9, 12, 22] derive from differences in either the SDP used to relax the problem [12], changing the rounding algorithm [6, 9, 12, 22], or in some cases by slightly modifying the approximation algorithm and providing a better analysis for the formal proof of the approximation factor [22].

With only one exception, [2], these results all have a rounding step which produces a product state. Since there are upper bounds on the performance of product states [9], these results all have necessarily limited performance, and it is desireable to produce non-product states for a better objective. Another common thread in many of these works is the use of the level-1 instance of the quantum Lasserre Hierarchy. As we will demonstrate in Section 3, this is a relatively loose relaxation which does not satisfy important physical constraints that a consistent quantum state would satisfy. Hence, to get a better objective it is important to use a higher level of Lasserre, for a tighter bound on the optimal quantum state.

There are two works of particular interest in the current context: [2] and [5], which we comment on. We will first need to formally describe a specific 2-Local Hamiltonian problem, introduced as a quantum analog of Max Cut [9]. Note that here and throughout the paper, we will use the notation σ_i to mean the 2×2 matrix σ acting on i tensored with the $\mathbb{I} \in \mathbb{C}^{2\times 2}$ acting on each of the other qubits (the total number of qubits, n, will be clear from context when this notation is used). The formal definition of Quantum Max Cut, $QMC(G, w)$ is:

▶ **Definition 1** ($QMC(G, w)$)**.** *Given a graph $G = (V, E)$ with $|V| = n$, let $H \in \mathbb{C}^{2^n \times 2^n}$ such that:*

$$H = \sum_{ij \in E} w_{ij}(\mathbb{I} - X_i X_j - Y_i Y_j - Z_i Z_j)$$

Then, we define $QMC(G, w)$ to be the largest eigenvalue of H. Ideally, one also seeks to produce a (description) of a state achieving this value.

Gharibian and Parekh [9] introduced this problem as a maximization version of the well-known problem of finding ground states for the quantum Heisenberg model. They give a classical 0.498-approximation using product states, where a $\frac{1}{2}$-approximation is the best possible in the product state regime. Anshu, Gosset, and Morenz [2] present a classical rounding algorithm that outputs a description of an entangled state and are able to deliver a 0.531-approximation. To the best of our knowledge, this is the first approximation algorithm for a 2-Local Hamiltonian problem to move beyond product states. Likewise, the analysis in [2] differs from the analysis in the other related works. Instead of using SDPs to upper

bound the optimal quantum objective, [2] uses physical considerations for the particular kind of Hamiltonian they study [20]. The key technical component is an upper bound on $QMC(G, w)$ where G is a star graph. The rounding algorithm is also fundamentally different in that the output quantum state is produced from direct consideration of the Hamiltonian and its weights, rather than a solution to an SDP.

Another important work for understanding our contribution is that of Brandão and Harrow [5], since this paper makes use of higher levels of the quantum Lasserre Hierarchy. Essentially the relevant rounding algorithm from this paper proceeds in the same way as the classical counterpart by Barak, Raghavendra, and Steurer [26], where a set of subsystems is sampled and all other density matrices are sampled according to single qubit density matrices conditioned on this set. There are additional issues that arise in the analysis from the quantum-ness of the problem, but the rounding algorithm is semantically similar. Additionally, all of the results presented in [5] make strong non-local assumptions on the particular "topology" or structure of the instance.

Our contributions. In contrast to previous approaches, we make only local assumptions on the 2-Local terms, and apply the second level of the Lasserre Hierarchy in a radical new way which makes crucial use of "monogamy of entanglement" inequalities. Indeed, we believe that the methods we introduce constitute the most interesting contribution of this work.

We bridge the gap between [2] and more traditional SDP-based approximation algorithms by showing that the monogamy of entanglement bound derived in [2], based on a seminal result of Lieb and Mattis [20], is a consequence of the second level of a quantum analog [5,8,24] of the classical Lasserre Hierarchy [18,19]. To the best of our knowledge this is a first explicit example of such a connection. This establishes the second level of the quantum Lasserre Hierarchy as the source of the best upper bound for Quantum Max Cut that is amenable to analysis. We show that weaker versions of this SDP relaxation, including the first level, fail to yield the monogamy of entanglement bound. In addition we slightly improve upon the best-known approximation factor for QMC [2] through a simple rounding algorithm that uses an SDP solution to guide construction of an entangled solution. This is a significant departure from existing approximation algorithms for 2-Local Hamiltonian problems, requiring new connections between quantum SDP relaxations and the convex hull of matchings in a graph. Quantum Max Cut has emerged as a vehicle for advancement of approximation algorithms for 2-Local Hamiltonian problems, since it maintains the hardness and essence of more general problems while hiding technical details that hinder progress [2, 9, 22]. We expect that the insights we develop here for Quantum Max Cut may be generalized for other problems.

Our methods. As stated previously, our rounding algorithm begins by formulating and solving an appropriate SDP, which comes from the quantum generalization of the Lasserre Hierarchy. The SDP assigns a "value" for each edge, roughly corresponding to "how close" the parameters of the edge are to a singlet. An edge with *large* value has parameters nearly matching the singlet. Loosely speaking, if an edge has large value then the SDP "thinks" an optimal quantum solution is nearly a singlet along the edge. The rounding algorithm proceeds by picking a threshold and adding every edge with value over the threshold to the large edge set (denoted L in the paper). In a legitimate quantum state, the concept of monogamy of entanglement implies that we cannot have too many large edges attached to the same vertex. Since the SDP relaxation we use is relatively strong (Section 3), this implies the graph induced by the small edges must have low degree. Hence, if we find a maximum matching on this graph, and place a singlet (the state in Equation (1)) on each edge in the matching, we obtain a quantum state with performance approximately comparable to the SDP on this subgraph. For the remainder of the qubits we place the maximally mixed state.

Intuitively, this technique of thresholding the edges and then finding a matching has poor performance when all the edges have small values. However, in this case a product state gives a good approximation to the objective: if all the edges are small then the state does not align well with the singlet along the edges in the Hamiltonian, hence entanglement is not really needed to emulate the state. The rounding algorithm checks the value of both of these strategies (singlets on large edges vs. product state rounding) and takes whichever is better.

Future work. Our analysis is not optimal, and it is possible to obtain improvements. For example, we may consider stronger valid inequalities for Quantum Max Cut solutions arising from our relaxation, which we know exist through numerical experiments. How far can such an improvement be pushed? Can we significantly improve the approximation ratio for Quantum Max Cut beyond ≈ 0.53? Our analysis shows that the second level of the quantum Lasserre Hierarchy is exact for star graphs. Can similar results be achieved for more interesting classes of graphs?

Another important direction is the search for upper bounds on achievable approximation factors. For classical optimization problems there are many such bounds known [15, 16]. Most of these rely on a complexity theoretic conjecture referred to as the Unique Games Conjecture (UGC) [14], i.e. if UGC holds then we have the corresponding upper bound on the approximation factor. No analogous results are known for quantum optimization problems.

2 Preliminaries

We use standard quantum information and graph theory notation, highlighting a few specific definitions below.

For an integer $l \geq 1$, we let $[l] := \{1, \ldots, l\}$. For a set S, $\mathbb{R}^S$ refers to $\mathbb{R}^{|S|}$, where the dimensions of the Euclidean space are associated with the elements of S. We generally refer to the elements of a vector $x \in \mathbb{R}^S$ as x_l for $l \in S$; however, we will also refer to x_l as variables comprising a solution x in the context of semidefinite and linear programs.

Quantum information. The Pauli matrices take their usual definition:

$$\mathbb{I} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}, \quad X = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}, \quad Y = \begin{bmatrix} 0 & -i \\ i & 0 \end{bmatrix}, \text{ and } \quad Z = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}.$$

We follow the standard practice of using subscripts to indicate quantum subsystems among n qubits, and we use the notation σ_i to denote a Pauli matrix $\sigma \in \{X, Y, Z\}$ acting on qubit i, i.e. $\sigma_i := \mathbb{I} \otimes \mathbb{I} \otimes \ldots \otimes \sigma \otimes \ldots \otimes \mathbb{I} \in \mathbb{C}^{2^n \times 2^n}$, where the σ occurs at position i. The sets $\mathcal{S}(\mathcal{X})$ and $\mathcal{H}(\mathcal{X})$ refer to the symmetric and Hermitian matrices, respectively, acting on the (complex) Euclidean space $\mathcal{X}$.

Graph theory. We deal with only finite and simple graphs $G = (V, E)$, with vertex set V and edge set E. The notation $E(G)$ is the edge set of a graph G. We will refer to an edge e with endpoints $i, j \in V$ as $ij \in E$, or simply as $e \in E$ when endpoints are immaterial. We generally consider weighted graphs where a weight $w_e \geq 0$ is specified for each edge $e \in E$.

For a graph $G = (V, E)$, and a set of vertices $S \subseteq V$, we denote the *induced subgraph* on S, consisting of all edges in E with both endpoints in S, as $G[S]$. For a set of vertices and edges $S \subseteq V$ and $F \subseteq E$, respectively, the edge set $\delta_F(S)$ is defined as $\{ij \in F \mid |\{i, j\} \cap S| = 1\}$, and $E_F(S) := \{ij \in F \mid |\{i, j\} \cap S| = 2\}$. We drop the subscript F when $F = E$, and for a vertex $i \in V$, we abbreviate $\delta_F(\{i\})$ as $\delta_F(i)$.

A graph is *k-vertex connected* if it has at least k vertices and deleting any set of fewer than k vertices (and any incident edges) leaves a connected graph. A *matching* M is a set of edges such that no two distinct $e, f \in M$ share a common vertex. A *perfect matching* in G is a matching of size $\frac{|V|}{2}$.

2.1 Approximation Algorithm Overview

The formal rounding algorithm we propose is presented in Algorithm 1.

Algorithm 1 Approximation Algorithm for Quantum Max Cut.

1. Given as input a graph $G = (V, E)$ with weights $w = \{w_e \geq 0\}_{e \in E}$, solve $\text{Lasserre}_2(G, w)$ (Definition 4). Let the matrix M be an optimal solution.
2. For each $ij \in E$ calculate $v_{ij} := [M(X_iX_j, \mathbb{I}) + M(Y_iY_j, \mathbb{I}) + M(Z_iZ_j, \mathbb{I})]/3$, where $M(\Gamma, \Phi)$ refers to the (Γ, Φ) entry of the matrix M. Set $x_{ij} := -v_{ij}$.
3. Pick an integer $d \geq 1$, and define $L := \{e \in E \mid x_e > \alpha(d) := \frac{d+3}{3(d+1)}\}$. Find a maximum-weight matching F in the graph $G_L := (V, L)$ with respect to the weights $\{w_e\}_{e \in L}$. Let U be the vertices unmatched by F.
4. Define a quantum state:[1]

$$\rho_F := \prod_{ij \in F} \left(\frac{\mathbb{I} - X_iX_j - Y_iY_j - Z_iZ_j}{4} \right) \prod_{v \in U} \frac{\mathbb{I}_v}{2}. \tag{1}$$

5. Execute the randomized approximation algorithm for Quantum Max Cut from [9], yielding a product state ρ_{PS} from a Lasserre_1 solution.
6. Output the better of ρ_F and ρ_{PS}.

To understand the significance of the parameter d in Step 3, recall that we find a set of "large" edges L based on a threshold. The strength of Lasserre_2 implies that G_L has bounded degree. d is the degree upper bound we prove (Lemma 14) corresponding to threshold $\alpha(d) = (d+3)/(3(d+1))$. In particular, if $d = 1$ then no vertex has two adjacent edges and we may select all edges in L for our matching. The problem with this strategy, however, is that if all the edges have small values then the product state rounding algorithm (Step 5) has relatively poor performance. Hence, we obtain the result for $d = 2$. This allows us to get better performance for product state rounding but requires more work to show a maximum matching has good performance with respect to the SDP.

Analysis outline. The main theorem of this work (Theorem 2) proves the stated approximation factor of Algorithm 1. The proof of this theorem requires first demonstrating (in Section 3) several inequalities on the optimal solution of the second level of the quantum Lasserre Hierarhcy (demonted Lasserre_2). Roughly there are two sets of techniques we use to prove the inequalities we need. The first set (Section 3.2) involves using invariance of the the objective function under certain permutations of the SDP variable and Schur complements. The second set of bounds follows from sum-of-squares proof techniques Section 3.3.

Understanding the performance of the thresholding (Step 3 in the algorithm) involves showing that constraints satisfied by the SDP (Section 3) imply that the "large" edges L can be scaled by a not too small constant and brought into the convex hull of matchings

[1] Recall X_i is a tensor product of identity operators and a single X operator in the ith position. So, $((\mathbb{I} + X)/2) \otimes ((\mathbb{I} + X)/2)$ is expressed as $\prod_{i=1}^{2}(\mathbb{I} + X_i)/2$ rather than $\bigotimes_{i=1}^{2}(\mathbb{I} + X_i)/2$

(Theorem 16). This provides a lower bound on the performance of the state ρ_F, then we may appeal to [9] to lower bound the performance of ρ_{PS}. We will prove the main theorem first, using components proved subsequently. The reader is encouraged to come back to this proof after reading the document

▶ **Theorem 2** (Main Theorem). *Let $G = (V, E)$ be a graph and $\{w_e\}_{e\in E}$ be a set of weights with $w_e \geq 0$ for all $e \in E$. Let H be the QMC Hamiltonian in Definition 1, and let ρ be the density matrix output by Algorithm 1. Then,*

$$\frac{\mathbb{E}[\mathrm{Tr}(H\rho)]}{QMC(G,w)} \geq 0.533,$$

where the numerator is the expected objective value obtained by Algorithm 1

Proof. Let $d = 2$, let $\{x_e\}_{e\in E}$ be the values obtained from the SDP as in Item 2, let L be the set of edges found in Item 3, let $S := E - L$, and let $\{y_e^*\}_{e\in E}$ be such that $\{0, 1\} \ni y_e^* = 1$ if and only if edge e is chosen in the matching for ρ_F (see Lemma 17).

Define:

$$s := \frac{\sum_{e\in S} w_e(1+3x_e)}{\sum_{e\in S} w_e(1+3x_e) + \sum_{e\in L} w_e(1+3x_e)},$$

and note that $s \in [0, 1]$ since the comment below Lemma 12 implies that $(1 + 3x_e) \geq 0$ for $e \in E$. It holds that

$$\frac{\sum_{e\in S} w_e(1+3y_e^*) + \sum_{e\in L} w_e(1+3y_e^*)}{\sum_{e\in S} w_e(1+3x_e) + \sum_{e\in L} w_e(1+3x_e)} = \frac{\sum_{e\in S} w_e(1+3y_e^*)}{\sum_{e\in S} w_e(1+3x_e)}s + \frac{\sum_{e\in L} w_e(1+3y_e^*)}{\sum_{e\in L} w_e(1+3x_e)}(1-s)$$

Now we can apply Lemma 17,

$$\frac{\sum_{e\in S} w_e(1+3y_e^*)}{\sum_{e\in S} w_e(1+3x_e)}s + \frac{\sum_{e\in L} w_e(1+3y_e^*)}{\sum_{e\in L} w_e(1+3x_e)}(1-s) \geq \frac{3}{8}s + \frac{3}{4}(1-s).$$

A similar argument for ρ_{PS} using Lemma 18 yields:

$$\frac{\mathbb{E}[\mathrm{Tr}(H\rho_{PS})]}{\sum_{e\in S} w_e(1+3x_e) + \sum_{ij\in L} w_e(1+3x_e)} \geq 0.557931s + 0.498766(1-s)$$

A lower bound on the expected approximation factor is

$$\min_{s\in[0,1]} \max\left\{\frac{3}{8}s + \frac{3}{4}(1-s),\ 0.557931s + 0.498766(1-s)\right\},$$

which is calculated by the linear program,

$$\begin{aligned} &0.533 \leq \min r \\ &\quad s.t.\ \ \left\{\frac{3}{8}s + \frac{3}{4}(1-s) \leq r,\ 0.557931s + 0.498766(1-s) \leq r,\ 1 \geq s \geq 0\right\}. \end{aligned}$$

◀

3 The Level-2 Quantum Lasserre Hierarchy

3.1 Definitions

The classical or commutative Lasserre Hierarchy (and the dual Sum-of-Squares Hierarchy) is a set of semidefinite programs which relaxes the notion of a probability distribution to a pseudo-distribution [3]. A pseudo-distribution is an assignment of values to low order moments which

respects some, but not all, of the properties that a fully consistent probability distribution would satisfy. To understand this consider n binary random variables $(A_1, ..., A_n)$. We will be interested in expectations of polynomials in the A_i. For each monomial of degree $t \leq 2k$ in these variables, the level-k instance of the hierarchy assigns value: $v_k(A_{i_1}A_{i_2}\dots A_{i_t}) \in [0,1]$. The value v_k is meant to represent the expectation $\mathbb{E}_{\mathcal{D}}[A_{i_1}A_{i_2}\dots A_{i_t}]$ for a valid probability distribution $\mathcal{D}$, but it is also possible that it assigns values in such a way that it is impossible to have $v_k(A_{i_1}A_{i_2}\dots A_{i_t}) = \mathbb{E}_{\mathcal{D}}[A_{i_1}A_{i_2}\dots A_{i_t}]$ for any valid distribution $\mathcal{D}$. The level-k SDP assigns values so that polynomials of degree at most k behave as they should for a valid distribution. In particular the SDP assigns values to monomials in such a way that if one expanded $p(A_1, \dots, A_n)^2$ as a linear combination of monomials and applied v_k to the individual terms, the resulting value $v_k(p(A_1, \dots, A_n)^2) \geq 0$. Note that the expected behavior for random variables is the same: $\mathbb{E}_{\mathcal{D}}[p(A_1, \dots, A_n)^2] \geq 0$ for a distribution $\mathcal{D}$. The level k can be thought of as checking that the distribution looks valid from the perspective of low order polynomials.

A quantum analog of the Lasserre Hierarchy [5, 8, 24] is essentially the same except that it is checking the validity of low order polynomials in the Pauli matrices with respect to an overall quantum distribution (density matrix). The values we will assign are meant to represent values of $\mathrm{Tr}(\Gamma\rho)$ for Γ a "low-order" tensor product of Pauli matrices and ρ a valid density matrix. However, the relaxation will likely assign values $v(\Gamma)$ in such a way that it is impossible for $v(\Gamma) = \mathrm{Tr}(\Gamma\rho)$ to hold for any density matrix (and for all Γ)[2]. In this context, by "low-order monomial" we mean the following:

▶ **Definition 3** ($\mathcal{P}_n(k)$)**.** *Given k, n define $\mathcal{P}_n(k)$ as the set of Pauli operators of weight $\leq k$. Formally, $\Gamma \in \mathcal{P}_n(k)$ if Γ is a tensor product of n operators, each of which is in $\{\mathbb{I}, X, Y, Z\}$ such that at most k are not $\mathbb{I}$.*

Lasserre$_k$ will assign values to monomials (elements of $\mathcal{P}_n(2k)$) in such a way that if $p = \sum_{\Phi\in\mathcal{P}_n(k)} c_\Phi \Phi$, then $v(p^2) = \sum_{\Phi,\Phi'} c_\Phi c_{\Phi'} v(\Phi\Phi') \geq 0$. A value assignment which respects low order statistics is equivalent to a positive-semidefinite (PSD) constraint on a "moment matrix". To understand this imagine we had a PSD matrix M with rows and columns indexed by elements of $\mathcal{P}_n(k)$, and we assigned values so that $v(\Gamma) := M(\Phi, \Psi)$ if $\Phi\Psi = \Gamma$. Then, given some polynomial p,

$$v(p^2) = v\left(\left(\sum_{\Phi\in\mathcal{P}_n(k)} c_\Phi \Phi\right)^2\right) = \sum_{\substack{\Phi,\Phi' \\ \in\mathcal{P}_n(k)}} c_\Phi c_{\Phi'} v(\Phi\Phi') = \sum_{\substack{\Phi,\Phi' \\ \in\mathcal{P}_n(k)}} c_\Phi c_{\Phi'} M(\Phi,\Phi') = c^T M c,$$

where $c \in \mathbb{R}^{\mathcal{P}_n(k)}$ is the vector of monomial coefficients. Since M was assumed PSD we are guaranteed that the RHS is ≥ 0. Indeed, if we assign values based on a PSD matrix subject to appropriate constraints, we are guaranteed that Lasserre$_k$ will respect low degree polynomials:

[2] Indeed, if we were able to constrain the low order statistics to be *globally consistent* with some (physical) density matrix, then we could find the largest eigenvalue and solve a QMA-complete problem [21].

▶ **Definition 4** (Lasserre$_k(G,w)$)**.** *Given k, a graph $G = (V, E)$ on n vertices, as well as a vector of non-negative weights $\{w_e\}_{e\in E}$ let $\gamma := |\mathcal{P}_n(k)|$. For each $ij \in E$, define $C_{ij} \in \mathcal{S}(\mathbb{R}^{\gamma\times\gamma})$ where rows and columns are indexed by elements of $\mathcal{P}_n(k)$ such that*

$$C_{ij}(\sigma_i, \sigma_j) := -\frac{1}{2} \quad \text{for } \sigma \in \{X, Y, Z\},$$
$$C_{ij} := 0 \qquad \text{otherwise.}$$

Let $M \in \mathcal{S}(\mathbb{R}^{\gamma\times\gamma})$ be an SDP variable with rows and columns indexed by elements of $\mathcal{P}_n(k)$. We define Lasserre$_k(G,w)$ as the following SDP:

$$\max \sum_{ij\in E} w_{ij}\,(1 + \mathrm{Tr}(C_{ij}M)) \tag{2}$$

$$\begin{aligned}
s.t. \quad & M(\Gamma,\Gamma) = 1 && \forall\,\Gamma \in \mathcal{P}_n(k), && (3)\\
& M(\Gamma,\Phi) = 0 && \forall\,\Gamma,\Phi \in \mathcal{P}_n(k) \text{ s.t. } \Gamma\Phi \text{ is not Hermitian}, && (4)\\
& M(\Gamma,\Phi) = M(\Gamma',\Phi') && \forall\,\Gamma,\Phi,\Gamma',\Phi' \in \mathcal{P}_n(k) \text{ s.t. } \Gamma\Phi = \Gamma'\Phi', && (5)\\
& M(\Gamma,\Phi) = -M(\Gamma',\Phi') && \forall\,\Gamma,\Phi,\Gamma',\Phi' \in \mathcal{P}_n(k) \text{ s.t. } \Gamma\Phi = -\Gamma'\Phi', && (6)\\
& M \succeq 0, && && (7)\\
& M \in \mathcal{S}(\mathbb{R}^{\gamma\times\gamma}). && && (8)
\end{aligned}$$

We will denote Lasserre$_k(G)$ as the above problem with uniform weights (set all $w_{ij} = 1$). Note that we employ a real version of the Lasserre Hierarchy rather than the usual complex version. This still provides an upper bound on the optimal quantum state as shown below in Theorem 7.

Since in Lasserre$_2$ we have constraints $M(\sigma_i, \sigma_j) = M(\sigma_i\sigma_j, \mathbb{I})$, we could have equivalently defined the objective matrix using these moment matrix entries, i.e. taking $C_{ij}(\sigma_i\sigma_j, \mathbb{I}) \neq 0$.

▶ **Definition 5** (Lasserre$_k$ Edge Values)**.** *From a solution M to Lasserre$_k$(G,w) (Definition 4), we define edge values that are used by the rounding algorithm, Algorithm 1. Such values are defined for every pair of distinct vertices i, j, hence we assume, when referring to these values, that E is edge set of a complete graph, denoted K_n. We may set $w_e = 0$ for edges $e \in E$ that do not contribute to the objective value. We define:*

$$v_{ij} := \frac{M(X_iX_j, \mathbb{I}) + M(Y_iY_j, \mathbb{I}) + M(Z_iZ_j, \mathbb{I})}{3}, \text{ and } x_{ij} := -v_{ij},$$

for all $ij \in E := E(K_n)$. We say an edge is large *if $x_{ij} \approx 1$ (and $v_{ij} \approx -1$).*

We will also need to define a modified version of Lasserre$_1$. This is simply Lasserre$_1$ supplemented with positivity of 2-qubit marginals. This is a relaxation of intermediate strength between Lasserre$_1$ and Lasserre$_2$, so we have denoted is Lasserre$_{1.5}$. The additional marginal constraints are crucial for the analysis presented in [22], so a precise understanding of its strength is very interesting. We will define it only for unweighted graphs, since it will not be used in the context of the approximation algorithm.

▶ **Problem 6** (Lasserre$_{1.5}(G)$). *Given a graph $G = (V, E)$ on n vertices, for each $ij \in E$ let C_{ij} be as defined in Lasserre$_1$. Solve the following SDP:*

$$\max \sum_{ij \in E} (1 + \mathrm{Tr}(C_{ij} M)) \tag{9}$$

$$\begin{aligned}
s.t. \quad M(\Gamma, \Gamma) &= 1 && \forall \Gamma \in \mathcal{P}_n(1), && (10)\\
M(\Gamma, \Phi) &= 0 && \forall \Gamma, \Phi \in \mathcal{P}_n(1) \text{ s.t. } \Gamma\Phi \text{ is not Hermitian}, && (11)\\
M(\sigma_i, \eta_j) &= \mathrm{Tr}[\sigma \otimes \eta\ \rho_{ij}] && \forall ij \in E \text{ and } \sigma, \eta \in \{X, Y, Z\}, && (12)\\
M(\sigma_i, \mathbb{I}) &= \mathrm{Tr}[\sigma_i \otimes \mathbb{I}\ \rho_{ij}] && \forall ij \in E \text{ and } \sigma \in \{X, Y, Z\}, && (13)\\
M(\sigma_j, \mathbb{I}) &= \mathrm{Tr}[\mathbb{I} \otimes \sigma_j\ \rho_{ij}] && \forall ij \in E \text{ and } \sigma \in \{X, Y, Z\}, && (14)\\
\mathrm{Tr}[\rho_{ij}] &= 1 && \forall ij \in E, && (15)\\
\rho_{ij} &\succeq 0 && \forall ij \in E, && (16)\\
\rho_{ij} &\in \mathcal{H}(\mathbb{C}^{4\times 4}) && \forall ij \in E, && (17)\\
M &\succeq 0, && && (18)\\
M &\in \mathcal{S}(\mathbb{R}^{(3n+1)\times(3n+1)}). && && (19)
\end{aligned}$$

Note that the main difference between Lasserre$_{1.5}$ and Lasserre$_1$ is the presence of constraints Equation (12)-Equation (14). As stated previously, their intent is to force consistency of 2-local moment matrices by forcing them to correspond to physical 2-qubit density matrices. These relaxations are important because they relax quantum states, hence can be used as upper bounds on 2-Local Hamiltonian problems:

▶ **Theorem 7.** *For any constant k Lasserre$_k$ is an efficiently computable semidefinite program that provides an upper bound on $QMC(G, w)$.*

Proof. Except for $M \succeq 0$, the constraints and objective are affine on the entries of M, hence we do indeed have an SDP. Since M is of polynomial size (it has length $O(n^k)$ on one side), and there are polynomially many linear constraints ($O(n^{2k})$ many), the usual considerations show computational efficiency: All feasible M have bounded norm since moment matrices are constrained to be 1 along the diagonal, the identity matrix is feasible so strong duality holds, and there is a "ball" of operators around the identity which are feasible. Hence, the program can be solved to arbitrary additive precision in polynomial time via the ellipsoid or interior point methods (e.g., [28]).

Let $|\psi\rangle$ be an eigenvector corresponding to $\lambda_{max}(H)$ where H is the 2-Local Hamiltonian in QMC (Definition 1). Set $M(\Phi, \Gamma) = \mathrm{Tr}(\Phi\Gamma\rho)$. M is PSD since for a complex vector v, $v^\dagger M v = \mathrm{Tr}(S^2\rho)$ for S some polynomial as previously described. The remaining issue is that if $\Phi\Gamma$ is not Hermitian then the corresponding value of M is purely imaginary, so we may not be satisfying Equation (4). The solution is simply to set a new moment matrix M' as $M' = (M + M^*)/2$ where M^* is the same as M but with complex conjugate entries. Note that M' is PSD since M^* must also be PSD. For the objective, note that

$$\mathrm{Tr}\left(w_{ij}(\mathbb{I} - X_iX_j - Y_iY_j - Z_iZ_j)\,|\psi\rangle\,\langle\psi|\right) = w_{ij}(1 + \mathrm{Tr}(MC_{ij})).$$

Hence we have established that the optimal quantum state has the same energy as the objective for *some* feasible M. It follows that the optimal M has objective which upper bounds the optimal quantum solution. ◀

$$M(\sigma_i, \eta_k) = \begin{cases} 1 \text{ if } i = k \text{ and } \sigma = \eta \\ -1 \text{ if } i = 0 \text{ and } \sigma = \eta \\ 1 \text{ if } i \neq 0 \neq k, \text{ and } \sigma = \eta \\ 1 \text{ if } \sigma_i = \mathbb{I} = \sigma_k \\ 0 \text{ otherwise} \end{cases}$$

(a) Formal Description of Optimal Moment Matrix.

$$M = \begin{array}{c} \\ 0 \\ 1 \\ 2 \\ \vdots \\ n \\ \mathbb{I} \end{array} \begin{array}{c} \begin{array}{ccccccc} 0 & 1 & 2 & \dots & n & \mathbb{I} \end{array} \\ \begin{bmatrix} \mathbb{I} & -\mathbb{I} & -\mathbb{I} & \dots & -\mathbb{I} & \mathbf{0} \\ -\mathbb{I} & \mathbb{I} & \mathbb{I} & \dots & \mathbb{I} & \mathbf{0} \\ -\mathbb{I} & \mathbb{I} & \mathbb{I} & \dots & \mathbb{I} & \mathbf{0} \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ -\mathbb{I} & \mathbb{I} & \mathbb{I} & \dots & \mathbb{I} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \dots & \mathbf{0} & 1 \end{bmatrix} \end{array}$$

(b) Informal Description of Optimal Moment Matrix.

$$\begin{bmatrix} 1 & -1 & -1 & \dots & -1 \\ -1 & 1 & 1 & \dots & 1 \\ -1 & 1 & 1 & \dots & 1 \\ \vdots & \vdots & & \ddots & \vdots \\ -1 & 1 & 1 & \dots & 1 \end{bmatrix}$$

(c) One of the Diagonal Blocks of M.

$$\begin{bmatrix} 1 & 1 \dots & 1 \\ \vdots & \vdots & \vdots \\ 1 & 1 \dots & 1 \end{bmatrix} - \begin{bmatrix} -1 \\ -1 \\ \vdots \\ -1 \end{bmatrix} \begin{bmatrix} -1 & -1 & \dots & -1 \end{bmatrix} \succeq 0$$

(d) Schur Complement to complete the proof.

Figure 1 Matrices Needed For Proof of Theorem 9.

3.2 Relative Strength of Relaxations

An important contribution of this work is that the level-2 instances of the Lasserre Hierarchy satisfy important physical constraints which are not satisfied by the first level, even when the first level is further constrained with positive 2-qubit marginals (Lasserre$_{1.5}$). The physical property of interest can be thought of as a "monogamy of entanglement" with respect to specific partitions of the quantum state. Let $G = (V, E)$ be a graph on $n + 1$ vertices with vertex set $\{0, 1, ..., n\}$. Further, let the edge set be $E = \{(0, 1), (0, 2), ..., (0, n)\}$. This graph is easily visualized as n "leaves" connected to a central vertex 0. The Hamiltonian for $QMC(G)$, H, can be thought of as "testing" entanglement along the edges since it is testing overlap with respect to a maximally entangled state. If a state $|\psi\rangle$ had value $\langle\psi|H|\psi\rangle = 4n$, then $|\psi\rangle$ would appear to be maximally entangled along all the edges in E. Indeed, this is impossible, and the value of the maximum possible $\langle\psi|H|\psi\rangle$ (or the maximum eigenvalue of H) is an important result for the analysis presented in [2]:

▶ **Theorem 8** (Star Bound [2, 20]). *If G is a star graph with n leaves, $QMC(G) = 2(n + 1)$.*

This was proven by Anshu, Gosset, and Morenz [2] using a well-known monogamy of entanglement result for the Heisenberg model on complete bipartite interaction graphs by Lieb and Mattis [20].

The first observation we have is that the Lasserre$_1$ SDP violates this property in a maximal sense. By this, we mean that the optimal solution has objective $4n$, rather than $2(n + 1)$. Using the informal language we used to describe Lasserre$_k$, if we only tracked 1-local Pauli polynomials, we would think it is possible to have a state which is maximally entangled along many overlapping edges:

▶ **Theorem 9.** *For G a star graph on n vertices, Lasserre$_1(G)$ has optimal objective $4n$.*

Proof. We can demonstrate an optimal solution to $\text{Lasserre}_1(G)$ by showing that the solution which "picks up" all the edges is feasible. Since the moment matrix must be PSD, and since the diagonal blocks are forced to be $\mathbb{I}$, this is the maximum possible solution for any SDP variable and a solution with this objective value must be optimal. Define moment matrix M as in Figure 1a. We can describe this matrix pictorially by partitioning the rows and columns into blocks corresponding to individual qubits, i.e. block i would correspond to indices $\{X_i, Y_i, Z_i\}$, as well as a block corresponding to the $\mathbb{I}$ index. With these partitions, we can write M as in Figure 1b where $\mathbf{0}$ denotes the zero vector $[0, 0, 0]$. It is easy to see that M satisfies all the linear constraints on the matrix elements, the remaining task is to prove it is PSD. We may reshuffle the rows/columns to write M in block diagonal form as four blocks, where one of the blocks is 1 and three of the blocks have the form Figure 1c. Using the method of Schur complements [31], PSDness of this matrix above is equivalent to Figure 1d, which holds trivially. Hence M is block diagonal with PSD blocks so it must be PSD. ◀

One direction for fixing this problem is to note that many of the low-order statistics present in the optimal moment matrix (see Figure 1a) are non-sensical even for very small states. The submatrix corresponding to qubits 1 and 2, for instance, has the form:

$$\begin{array}{c} \\ X_1 \\ Y_1 \\ Z_1 \\ X_2 \\ Y_2 \\ Z_2 \end{array} \begin{array}{c} \begin{array}{cccccc} X_1 & Y_1 & Z_1 & X_2 & Y_2 & Z_2 \end{array} \\ \left[\begin{array}{cccccc} 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 \end{array}\right] \end{array}$$

which is impossible for any (reduced) two qubit density martrix ρ_{12} [13]. This can be remedied in the SDP by adding variables $\rho_{ij} \in \mathbb{C}^{4\times 4}$ for every pair of vertices i, j which force submatrices of the above form to conform to moment matrices of legitimate 2-qubit quantum states, as in [22] and as in in $\text{Lasserre}_{1.5}$.

Unfortunately, this relaxation is still not strong enough to enforce the star bound (Theorem 8):

▶ **Theorem 10.** *If G is a star graph on n vertices, then $\text{Lasserre}_{1.5}(G)$ has optimal objective:*

$$n + 3\sqrt{n(1 + (n-1)/3)}.$$

Proof. Let M be the optimal moment matrix. We may permute the leaves without changing the objective since the graph is unweighted, which corresponds to $M \to UMU^T$ for U some orthogonal matrix. Formally, if $f : [n] \to [n]$ is some permutation on the leaves, then define U as in Figure 2d for all $\sigma \in \{X, Y, Z\}$. Then, the set of marginal density matrix variables can be redefined using UMU^T. Similarly, we can permute $\{X, Y, Z\}$ within each block. We will assume that each $\{X_i, Y_i, Z_i\}$ is permuted in the same way so as to not change the objective. By taking convex combinations of the described permutations (setting M to be a convex combination of terms of the form UMU^T for U orthogonal matrices), we can then assume WLOG that M has been fully symeterized and hence it is invariant to such permutations. We can also assume that the entries $M(\sigma_i, \mathbb{I})$ are zero for $\sigma \in \{X, Y, Z\}$. To see this observe that these terms do not participate in the objective, and that if they are non-zero in the optimal M they can be set to zero without altering positivity. We have shown that M has the form Figure 2a where the rows/columns are

$$M = \begin{array}{c} \\ 0 \\ 1 \\ 2 \\ \vdots \\ n \\ \mathbb{I} \end{array} \begin{array}{c} \begin{array}{cccccc} 0 & 1 & 2 & \dots & n & \mathbb{I} \end{array} \\ \begin{bmatrix} \mathbb{I} & P_1 & P_1 & \dots & P_1 & \mathbf{0} \\ P_1^T & \mathbb{I} & P_2 & \dots & P_2 & \mathbf{0} \\ P_1^T & P_2 & \mathbb{I} & \dots & P_2 & \mathbf{0} \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ P_1^T & P_2 & \dots & P_2 & \mathbb{I} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \dots & \mathbf{0} & 1 \end{bmatrix} \end{array}$$

(a) Symmeterized Optimal Moment Matrix.

$$Q := \begin{bmatrix} 1 & \alpha & \alpha & \dots & \alpha \\ \alpha & 1 & \beta & \dots & \beta \\ \alpha & \beta & 1 & & \beta \\ \vdots & \vdots & & \ddots & \vdots \\ \alpha & \beta & \dots & \beta & 1 \end{bmatrix}$$

(b) One Block of M.

$$\begin{bmatrix} 1 & \beta & \dots & \beta \\ \beta & 1 & & \beta \\ \vdots & & \ddots & \vdots \\ \beta & \dots & \beta & 1 \end{bmatrix} - \begin{bmatrix} \alpha \\ \alpha \\ \alpha \\ \vdots \\ \alpha \end{bmatrix} \begin{bmatrix} \alpha & \alpha & \alpha & \dots & \alpha \end{bmatrix} \succeq 0$$

(c) Schur Complement for Proof.

$$U(\sigma_j, \sigma_i) = \begin{cases} 1 \text{ if } f(i) = j \text{ or } \sigma_i = \mathbb{I} = \sigma_j \\ 0 \text{ otherwise} \end{cases}$$

(d) Permutation Matrix for Symmeterization.

Figure 2 Matrices Needed For Proof of Theorem 10.

indexed according to $\{X_0, Y_0, Z_0, X_1, Y_1, Z_1, ..., X_n, Y_n, Z_n, \mathbb{I}\}$. Further, according to the symmeterization argument we can assume P_1 has the same entry on each diagonal, P_2 has the same entry on each diagonal and P_2 is symmetric. Consider the submatrix corresponding to just the X operators, and denote if Q as in Figure 2b where $\alpha = M(X_0, X_i)$ and $\beta = M(X_1, X_2)$. We know this matrix is PSD because it is a submatrix of a PSD matrix. Further, since the submatrix corresponding to qubits 1 and 2 must correspond to the (valid) density matrix ρ_{12}, by [13, 22] $-1 \leq \beta \leq 1/3$. Writing Q in block diagonal form $\begin{bmatrix} A & B \\ B^T & C \end{bmatrix}$ with A equal to the top left entry, and we can apply the method of Schur complements [31] to obtain Figure 2c.

We can take the inner product with the all ones vector to obtain the inequality $n(1+(n-1)\beta)-\alpha^2 n^2 \geq 0$. Observe that this is a concave (down) parabola, to so to get a uniform bound on α we must set β to the largest possible value. Hence we derive $\alpha \geq -\sqrt{(1+(n-1)/3)/n}$ with $\beta = 1/3$. The objective is $n + (3n\alpha)$, so far we have shown the optimal solution is at most $n + 3\sqrt{n(1+(n-1)/3)}$. To see the SDP achieves this upper bound note we can take $P_1 \propto \alpha\mathbb{I}$ and $P_2 \propto \beta\mathbb{I}$ for α and β saturating the upper bound. Just as in Theorem 9, M decomposes into a block diagonal form where every block is PSD. ◀

Lasserre$_2$, on the other hand, does satisfy the star bound, which will be an important fact for the analysis of our rounding algorithm.

▶ **Theorem 11.** *Let $G = (V, E)$ be a star graph with n leaves. If G is a subgraph of a larger graph G' on n' vertices, and if M is any feasible solution to Lasserre$_2(G')$, then the values $\{x_e\}_{e \in E}$ (as in Algorithm 1) satisfy $\sum_{e \in E}(1 + 3x_e) \leq 2(n+1)$. Additionally, Lasserre$_2(G)$ has optimal solution with objective $2(n+1)$.*

Proof. This proof is very similar to Theorem 10. Let M be the optimal solution to Lasserre$_2(G')$. We will once again permute the leaves. Consider some permutation of the leaves of G $f : [n] \to [n]$. Let $U \in \mathbb{C}^{2^{n'} \times 2^{n'}}$ be the (Clifford) unitary which satisfies $U\sigma_i U^\dagger = \sigma_j$ if $\sigma \in \{X, Y, Z\}$ and $f(i) = j$. Then, define the $|\mathcal{P}_{n'}(2)| \times |\mathcal{P}_{n'}(2)|$ permutation matrix W as:

$$W(\Gamma, \Phi) = \begin{cases} 1 \text{ if } U\Phi U^\dagger = \Gamma \\ 0 \text{ otherwise} \end{cases}$$

and define $M' = WMW^T$. Constraint Equation (5) holds because if $\Gamma\Phi = \Theta\Pi$ then $(U^\dagger \Gamma U)(U^\dagger \Phi U) = (U^\dagger \Theta U)(U^\dagger \Pi U)$ so

$$M'(\Gamma, \Phi) = M(U^\dagger \Gamma U, U^\dagger \Phi U) = M(U^\dagger \Theta U, U^\dagger \Pi U) = M'(\Theta, \Pi)$$

The other constraints are similar. Just as in Theorem 10, we can also permute $\{X, Y, Z\}$ in each block. Note that by permuting the leaves we have potentially reduced the objective value of M overall (including the edges outside of the star graph), however we have not changed the sum of the values of edges in the star: $\sum_{e \in E}(1 + 3x_e)$.

Now extract the submatrix corresponding to the 2-local terms along the edges corresponding to the G, as well as the identity. Denote this as Q. Formally, Q is the submatrix induced by the index set

$$\{X_0X_1, Y_0Y_1, Z_0Z_1, X_0X_2, ..., X_0X_n, Y_0Y_n, Z_0Z_n, \mathbb{I}\}. \tag{20}$$

By symmeterization, we may assume Q has the following block form[3]:

$\begin{matrix} 1 & -\alpha & -\alpha \\ -\alpha & 1 & -\alpha \\ -\alpha & -\alpha & 1 \end{matrix}$	$\beta\mathbb{I}$	$\cdots$	$\beta\mathbb{I}$	$\begin{matrix} \alpha \\ \alpha \\ \alpha \end{matrix}$
$\beta\mathbb{I}$	$\begin{matrix} 1 & -\alpha & -\alpha \\ -\alpha & 1 & -\alpha \\ -\alpha & -\alpha & 1 \end{matrix}$	$\ddots$	$\vdots$	$\begin{matrix} \alpha \\ \alpha \\ \alpha \end{matrix}$
$\vdots$	$\ddots$	$\ddots$	$\beta\mathbb{I}$	$\vdots$
$\beta\mathbb{I}$	$\cdots$	$\beta\mathbb{I}$	$\begin{matrix} 1 & -\alpha & -\alpha \\ -\alpha & 1 & -\alpha \\ -\alpha & -\alpha & 1 \end{matrix}$	$\begin{matrix} \alpha \\ \alpha \\ \alpha \end{matrix}$
$\begin{matrix} \alpha & \alpha & \alpha \end{matrix}$	$\begin{matrix} \alpha & \alpha & \alpha \end{matrix}$	$\begin{matrix} \alpha & \alpha & \alpha \end{matrix}$	$\begin{matrix} \alpha & \alpha & \alpha \end{matrix}$	1

(21)

Note that rows and columns are indexed as the same order as the set in Equation (20). By Lemma 12, $-1 \leq \beta \leq 1/3$.

Now observe that $Q \succeq 0$ since it is a submatrix of a PSD matrix. Consider this matrix in 2×2 block form $\begin{bmatrix} A & B \\ C & D \end{bmatrix}$ where $D = 1$ is the bottom right entry. Exactly as in previous proofs we can then use the method of Schur complement to derive a necessary condition for positivity:

$$-3n\alpha^2 - 2\alpha + (1 + (n-1)\beta) \geq 0.$$

[3] Note that off-diagonal elements of off-diagonal blocks are imaginary for a quantum state: $X_0X_j \cdot Y_0Y_k \propto iZ_0X_jY_k$. So, those entries correspond to non-Hermitain matrices and are set to zero by constraint Equation (4)

Observe that this is a concave down parabola in α, so to get a uniform bound we need to take β as large as possible. Setting $\beta = 1/3$ and solving for the zeros of the polynomial yields the bounds $-(n+2)/(3n) \leq \alpha \leq 1/3$. The lower bound yields the star bound for the sum of the edge values.

To show $\text{Lasserre}_2(G)$ has optimal solution meeting the star bound note from the previous analysis we have that the optimal objective is upper bounded by the star bound. Further, by Theorem 8 and the fact that we are considering a relaxation on the 2-Local Hamiltonian problem (Theorem 7) the optimal objective must be at least the star bound. The result follows. ◀

3.3 Valid Inequalities for Lasserre_2

We now turn our attention to deriving inequalities that any Lasserre_2 solution must satisfy. These will be used in the subsequent analysis of Algorithm 1. Let M be a feasible solution to $\text{Lasserre}_2(G, w)$. Consider a Cholesky decomposition $U^T U = M \succeq 0$; we will refer to the vectors obtained from the columns of U as $u(\Gamma) \in \mathbb{R}^l$, corresponding to $\Gamma \in \mathcal{P}_n(k)$ and the rows or columns of M. Thus we have

$$u(\Gamma)^T u(\Phi) = M(\Gamma, \Phi) \quad \forall \Gamma, \Phi \in \mathcal{P}_n(k), \tag{22}$$

and in particular each $u(\Gamma)$ is a unit vector by Equation (3). We will establish linear inequalities on the values v_e of Definition 5, as these are the input to the rounding portion of Algorithm 1. First we establish bounds on the v_e.

▶ **Lemma 12.** *For all $ij \in E$, $0 \leq 1 - M(X_iX_j, \mathbb{I}) - M(Y_iY_j, \mathbb{I}) - M(Z_iZ_j, \mathbb{I}) \leq 4$.*

Proof. We will use properties of the vectors $u(\Gamma)$ and M. In particular suppose $\sigma, \eta, \tau \in \{X, Y, Z\}$ are distinct Paulis, giving us:

$$\begin{aligned} u(\sigma_i\sigma_j)^T u(\eta_i\eta_j) &= M(\sigma_i\sigma_j, \eta_i\eta_j) && \text{[by Equation (22)]} \\ &= M(\sigma_i\eta_i\sigma_j\eta_j, \mathbb{I}) && \text{[by Equation (5)]} \\ &= -M(\tau_i\tau_j, \mathbb{I}) && \text{[by Equation (6)].} \end{aligned}$$

The above in conjunction with $u(\Gamma)^T u(\Gamma) = 1$ and $u(\sigma_i\sigma_j)^T u(\mathbb{I}) = M(\sigma_i\sigma_j, \mathbb{I})$ yields the lower bound we seek to prove:

$$\begin{aligned} 0 &\leq [u(\mathbb{I}) - u(X_iX_j) - u(Y_iY_j) - u(Z_iZ_j)]^T [u(\mathbb{I}) - u(X_iX_j) - u(Y_iY_j) - u(Z_iZ_j)] \\ &= 4[1 - M(X_iX_j, \mathbb{I}) - M(Y_iY_j, \mathbb{I}) - M(Z_iZ_j, \mathbb{I})]. \end{aligned}$$

For the upper bound, let the vector $z(1) := u(\mathbb{I}) - u(X_iX_j) + u(Y_iY_j) + u(Z_iZ_j)$. Analogously to above, we have

$$0 \leq \frac{1}{4} z(1)^T z(1) = 1 - M(X_iX_j, \mathbb{I}) + M(Y_iY_j, \mathbb{I}) + M(Z_iZ_j, \mathbb{I}). \tag{23}$$

If we let $z(2) := u(\mathbb{I}) + u(X_iX_j) - u(Y_iY_j) + u(Z_iZ_j)$ and $z(3) := u(\mathbb{I}) + u(X_iX_j) + u(Y_iY_j) - u(Z_iZ_j)$, then

$$0 \leq \sum_{l \in [3]} \frac{1}{4} z(l)^T z(l) = 3 + M(X_iX_j, \mathbb{I}) + M(Y_iY_j, \mathbb{I}) + M(Z_iZ_j, \mathbb{I}),$$

by the analogs of Equation (23) for $z(2)$ and $z(3)$. The above inequality is equivalent to the upper bound we seek to prove. ◀

The lemma implies that $-1 \leq v_e \leq \frac{1}{3}$ and $-\frac{1}{3} \leq x_e \leq 1$, for all $e \in E$. We next derive inequalities for odd cycles in G.

▶ **Lemma 13.** *For an odd-length cycle $C \subseteq E$, $\sum_{e \in C} v_e \geq 2 - |C|$.*

Proof. We take the same basic approach as the proof of Lemma 12, namely the inequality will follow from a positive combination of inequalities derived from $z^T z \geq 0$, for appropriately chosen vectors z. First we consider the case when $|C| = 3$; let C consist of $ij, ik, jk \in E$, and let $\sigma \in \{X, Y, Z\}$. Letting the vector $z(\sigma) := u(\mathbb{I}) + u(\sigma_i\sigma_j) + u(\sigma_i\sigma_k) + u(\sigma_j\sigma_k)$, we see:

$$0 \leq \frac{1}{4} z(\sigma)^T z(\sigma) = 1 + M(\sigma_i\sigma_j, \mathbb{I}) + M(\sigma_i\sigma_k, \mathbb{I}) + M(\sigma_j\sigma_k, \mathbb{I}),$$

since $u(\sigma_p\sigma_q)^T u(\sigma_q\sigma_r) = M(\sigma_p\sigma_q, \sigma_q\sigma_r) = M(\sigma_p\sigma_r, \mathbb{I})$, for $p, q, r \in V$. Averaging the above inequality over $\sigma \in X, Y, Z$ yields:

$$0 \leq \frac{1}{3} \sum_{\sigma \in \{X,Y,Z\}} \frac{1}{4} z(\sigma)^T z(\sigma) = 1 + v_{ij} + v_{ik} + v_{jk}, \tag{24}$$

establishing the lemma for $|C| = 3$. We establish another flavor of the triangle inequality above in order to extend the $|C| = 3$ bound to larger cycles. This time we let $z(\sigma) := u(\mathbb{I}) + u(\sigma_i\sigma_j) - u(\sigma_i\sigma_k) - u(\sigma_j\sigma_k)$, ultimately yielding:

$$0 \leq \frac{1}{3} \sum_{\sigma \in \{X,Y,Z\}} \frac{1}{4} z(\sigma)^T z(\sigma) = 1 + v_{ij} - v_{ik} - v_{jk}. \tag{25}$$

Let us derive the bound for $|C| = 5$. Suppose the vertices of C are in $[5]$. We sum three instances of the above inequalities: Equation (24) for triangles on $\{1, 2, 5\}$ and $\{2, 3, 4\}$, and $1 + v_{45} - v_{24} - v_{25} \geq 0$ for the triangle on $\{2, 4, 5\}$. The sum is $3 + v_{12} + v_{23} + v_{34} + v_{45} + v_{15} \geq 0$, as desired. More generally, for $|C| = 2k+1$ with $k > 2$, we may sum k instances of Equation (24) and $k - 1$ instances of Equation (25) to derive the desired inequality. These $2k - 1$ triangles represent a triangulation of the cycle C; chords introduced by the triangulation appear in exactly two triangles, and edges of C appear in exactly one triangle. The $k - 1$ instances of Equation (25) are used to cancel out variables on such chords. ◀

The inequalities of the above lemma are actually implied by level 2 of the classical Lasserre Hierarchy for the classical Max Cut problem. This is captured by restricting $\mathcal{P}_n(2)$ to only contain tensor products of at most two Pauli Z's and setting $v_{ij} := M(Z_i Z_j, \mathbb{I})$.

4 Analysis of Lasserre$_2$ Rounding

Our goal is to provide bounds on the quality of the rounded solutions produced by Algorithm 1, ρ_F and ρ_{PS}, relative to our Lasserre$_2$ relaxation. For each of these solutions, we consider both the contribution of the edges selected by Algorithm 1 to be in L as well as those in $S := E - L$.

4.1 Bounding the Quality of the Matching-Based Solution

Algorithm 1 leverages a matching on a graph obtained by keeping only edges with high-magnitude fractional SDP values. Here we show that the resulting graph has bounded degree and why this approach produces a matching of relatively large weight.

We consider the values x_e for $e \in E = E(K_n)$ (Definition 5) obtained from the Lasserre$_2(G, w)$ SDP (Definition 4), so that the SDP's objective value is $1 + 3x_e$ for each $e \in E$. Recall from Algorithm 1 that we threshold these variables so that $L = \{e \in E \mid x_e > \alpha(d) \geq 0\}$, where $\alpha(d) = \frac{d+3}{3(d+1)}$. The star bound allows us to bound the maximum degree in the resulting graph, $G_L = (V, L)$.

▶ **Lemma 14.** *The graph G_L, as defined above, has maximum degree at most d.*

Proof. Suppose a vertex $i \in V$ in has degree at least $d+1$ in G_L. Let $D \subseteq \delta(i)$ be a set of $d+1$ edges. We then have

$$\sum_{e \in D} (1 + 3x_e) > (d+1)(1 + 3\alpha(d)) = 2(d+2),$$

violating Theorem 11 for the star rooted at vertex i and containing the edges in D. ◀

Algorithm 1 finds a matching F^* in G_L that maximizes $\sum_{e \in F} w_e$ over matchings F in G_L. The algorithm obtains a quantum state ρ_{F^*} from F^* by putting a singlet, $\frac{1}{4}(\mathbb{I} - X_iX_j - Y_iY_j - Z_iZ_j)$, on each edge in F^* and a maximally mixed state, $\frac{1}{2}\mathbb{I}$, on each vertex unmatched by F^*. Since the objective is $w_{ij}(\mathbb{I} - X_iX_j - Y_iY_j - Z_iZ_j)$ for each edge $ij \in E$, this earns a weight of $4w_{ij}$ for every $ij \in F^*$ and a weight of w_{ij} for every $ij \in E - F^*$. If we define $y^*_e = 1$ for $e \in F^*$ and $y^*_e = 0$ otherwise, then we may express the weight earned by ρ_{F^*} on an edge $e \in E$ as: $4w_e y^*_e + w_e(1 - y^*_e) = w_e(1 + 3y^*_e)$. We would like to show that the total weight of F^* on the edges in L is approximately the weight earned by the SDP on L: $\sum_{e \in L} w_e(1 + 3x_e)$. The following lemma suggests a strategy for accomplishing this.

▶ **Lemma 15.** *If, for some $\beta \in [0, 1]$, the vector $\beta x \in \mathbb{R}^{|E|}$ is a convex combination of matchings, then $\sum_{e \in L} w_e(1 + 3y^*_e) \geq \sum_{e \in L} w_e(1 + 3\beta x_e)$.*

Proof. Write $\beta x = \sum_l \mu_l M_l$, where the latter is a convex combination of incidence vectors of matchings. We have, for the vector of weights w, $\beta w^T x = \sum_l \mu_l w^T M_l$, so there is some l' with $w^T M_{l'} \geq \beta w^T x$. Since y^* is the incidence vector of a maximum-weight matching with respect to w, we get $w^T y^* \geq w^T M_{l'} \geq \beta w^T x$, completing our proof. ◀

The hypothesis of the above lemma is equivalent to showing that βx is in the convex hull of matchings for our graph G, and the convex hull of matchings in a graph is well understood.

▶ **Theorem 16** (Pulleyblank and Edmonds [25]; see [27], Section 25.2)**.** *The convex hull of matchings in a graph $G = (V, E)$ is defined by the following linear inequalities:*

$$\sum_{e \in \delta(i)} y_e \leq 1 \qquad \forall i \in V, \tag{26}$$

$$\sum_{e \in E(S)} y_e \leq \frac{|S| - 1}{2} \qquad \forall S \in \mathcal{F}, \tag{27}$$

$$y_e \geq 0 \qquad \forall e \in E, \tag{28}$$

where $\mathcal{F} := \{S \subseteq V \mid |S| \geq 3$, and $G[S]$ is factor critical and 2-vertex connected$\}$. A graph H is called factor critical (or hypomatchable) if deleting any vertex in H leaves a graph containing a perfect matching (hence $|H|$ must be odd).

We obtain our main lemma by determining a relatively large $\beta \in [0, 1]$ for which $y_e = \beta x_e$ is a feasible solution for the inequalities above, when we take $d = 2$.

▶ **Lemma 17.** *Suppose v_e for $e \in E$ is a feasible solution to the SDP Definition 4, and set $x_e := -v_e$. Let $L := \{e \in E \mid x_e > \frac{5}{9} = \alpha(2)\}$ and $G_L := (V, L)$ be the graph consisting of the edges in L. If F^* is an maximum-weight matching in G_L with respect to the weights $w_e \geq 0$ for $e \in L$, then*

$$\frac{\sum_{e\in L} w_e(1+3y_e^*)}{\sum_{e\in L} w_e(1+3x_e)} > \frac{3}{4}, \tag{29}$$

where $\{0,1\} \ni y_e^* = 1$ if and only if $e \in F^*$. If $S := E - L$, then

$$\frac{\sum_{e\in S} w_e(1+3y_e^*)}{\sum_{e\in S} w_e(1+3x_e)} \geq \frac{3}{8}. \tag{30}$$

Proof. We begin by considering Equation (29) and first showing that the variables βx_e satisfy the inequalities of Theorem 16 for G_L with $\beta = \frac{9}{14}$. Then, Lemma 15 gives us

$$\frac{\sum_{e\in L} w_e(1+3y_e^*)}{\sum_{e\in L} w_e(1+3x_e)} \geq \frac{\sum_{e\in L} w_e(1+3\beta x_e)}{\sum_{e\in L} w_e(1+3x_e)}, \tag{31}$$

and we may focus our attention on bounding the latter, which only depends on the x_e.

Satisfying the inequalities of Theorem 16. Inequality (28) is satisfied since $e \in L$ implies $x_e \geq 0$. To see that the vector $\frac{9}{14}x$ is feasible for the inequality (26), first note that since we take $d = 2$, G_L has maximum degree at most 2 by Lemma 14. Inequality (26) is satisfied for vertices of degree 1 since SDP Definition 4 gives us $x_e \leq 1$ for all e. Now another application of the star bound (Theorem 11) to a degree-2 vertex, i in G_L with neighbors j and k, gives us that

$$(1+3x_{ij}) + (1+3x_{ik}) \leq 6 \;\Rightarrow\; x_{ij} + x_{ik} \leq \frac{4}{3}, \tag{32}$$

hence $\frac{9}{14}x_{ij} + \frac{9}{14}x_{ik} \leq \frac{3}{4}x_{ij} + \frac{3}{4}x_{ik} \leq 1$.

Next we will show that Inequality (27) is satisfied. For these inequalities, we may assume that the induced subgraph on $S \in \mathcal{F}$ in G_L, $G_L[S]$, is an odd cycle. The set $\mathcal{F}$ contains only odd-sized sets that are (2-vertex) connected, and G_L has degree at most 2; hence, $G_L[S]$ must be a path or a cycle. The graph $G_L[S]$ cannot be a path since it must be factor critical, and removing a penultimate vertex in a path leaves a graph with no perfect matching.

Pick some $S \in \mathcal{F}$, and sum the inequalities of (26) over $i \in S$. This yields

$$\sum_{e\in\delta_L(S)} y_e + \sum_{e\in E_L(S)} 2y_e \leq |S| \;\Rightarrow\; \sum_{e\in E_L(S)} y_e \leq \frac{|S|}{2},$$

since $y_e \geq 0$ by Inequality (28). This shows that any vector y that satisfies Inequality (26) gives a RHS of $\frac{|S|}{2}$ instead of the desired value, $\frac{|S|-1}{2}$ for Inequality (27). To make such a vector feasible for Inequality (27), we must scale it by $\max_{k\geq 1} \frac{2k}{2k+1} = \frac{2}{3}$. In our case, $\frac{3}{4}x$ satisfies Inequality (26), hence $\frac{2}{3}\cdot\frac{3}{4}x = \frac{1}{2}x$ is feasible for the inequalities of Theorem 16. However, we can do better by considering additional inequalities satisfied by the x_e that are implied by our Lasserre$_2$ SDP relaxation. Lemma 13 gives us that $x_{ij} + x_{ik} + x_{jk} \leq 1$ for $ij, ik, jk \in E$, hence the inequalities of (27) for $|S| = 3$ are satisfied by the vector x (in fact, since $x_e > \frac{5}{9}$ for all $e \in L$, G_L contains no triangles).

For any cycle $C \subseteq L$ on 5 vertices, Lemma 13 yields $\sum_{e\in C} x_e \leq 3$, so that $\sum_{e\in C} \frac{3}{4}x_e \leq \frac{9}{4}$. Hence, $\frac{3}{4}x$ must be scaled by an additional factor of $\frac{8}{9}$ in order satisfy the inequalities of (27) for $|S| = 5$. For $|S| \geq 7$, an additional factor of $\max_{k\geq 3} \frac{2k}{2k+1} = \frac{6}{7}$ suffices. Thus $\frac{6}{7}\cdot\frac{3}{4}x = \frac{9}{14}x$ is a feasible solution for the inequalities of *Theorem* 16, and it is consequently a convex combination of incidence vectors of matchings in G_L.

Establishing Equation 29. For the RHS of Equation (31), we have

$$\frac{\sum_{e\in L} w_e(1+3\beta x_e)}{\sum_{e\in L} w_e(1+3x_e)} \geq \min_{\{e\in L|w_e>0\}} \frac{w_e(1+3\beta x_e)}{w_e(1+3x_e)}, \tag{33}$$

since $1+3x_e > 0$ for all $e \in L$, and the LHS above is a convex combination of the ratios $(1+3\beta x_e)/(1+3x_e)$ for $e \in L$ with $w_e > 0$. This reduces our task to bounding $(1+3\beta x_e)/(1+3x_e)$, for a worst-case value of x_e achieving the minimum above. If e is an isolated edge in G_L (i.e. it is not incident to any other edges in L), then we may assume e is in the maximum-weight matching F^* without loss of generality, since we can apply the arguments of this section to each connected component of G_L. For such an edge e, we may simply take $\beta = 1$, yielding $(1+3\beta x_e)/(1+3x_e) = 1$. If e is not isolated in G_L, then it is incident to another edge f at a vertex of degree 2 in G_L. Since $x_f > \frac{5}{9}$, by Equation (32) we see that $x_e \leq \frac{4}{3} - x_f < \frac{7}{9}$. We consequently have, for $\beta = \frac{9}{14}$:

$$\min_{\{e\in L|w_e>0\}} \frac{w_e(1+3\beta x_e)}{w_e(1+3x_e)} \geq \min_{x_e\in(\frac{5}{9},\frac{7}{9})} \frac{1+3\beta x_e}{1+3x_e} = \beta + \min_{x_e\in(\frac{5}{9},\frac{7}{9})} \frac{1-\beta}{1+3x_e} > \frac{3}{10} + \frac{7}{10}\beta = \frac{3}{4},$$

demonstrating Equation (29).

Establishing Equation 30. We now turn our attention to the edges in $S = E - L$. Since F^* includes no edges in S, we have $y^*_e = 0$ for $e \in S$. By the definition of S, $x_e \leq \frac{5}{9}$ for $e \in S$. These facts yield Equation (30). ◀

Finding a maximum-weight matching in G_L. We note that since each vertex in G_L has degree at most 2 when $d = 2$, each connected component of G_L is a path or cycle. In this case a maximum-weight matching may be found in linear time by a dynamic programming algorithm.

4.2 Bounding the Quality of the Product State Solution

We have established performance bounds on the matching part of the rounding algorithm. The only remaining piece is a performance bound on the product state solution produced by the rounding algorithm, ρ_{PS}.

▶ **Lemma 18.** *Suppose v_e for $e \in E$ are values derived from the optimal solution to Lasserre$_2$, and set $x_e := -v_e$. Let $L := \{e \in E \mid x_e > \frac{5}{9} = \alpha(2)\}$ and let $S := E - L$. Then, with respect to the weights $w_e \geq 0$, the approximation algorithm from [9] produces a random product state ρ satisfying:*

$$\frac{\sum_{ij\in L} w_{ij}\mathbb{E}[\mathrm{Tr}((\mathbb{I} - X_iX_j - Y_iY_j - Z_iZ_j)\rho)]}{\sum_{ij\in L} w_{ij}(1+3x_{ij})} \geq 0.498766, \tag{34}$$

and

$$\frac{\sum_{ij\in S} w_{ij}\mathbb{E}[\mathrm{Tr}((\mathbb{I} - X_iX_j - Y_iY_j - Z_iZ_j)\rho)]}{\sum_{ij\in S} w_{ij}(1+3x_{ij})} \geq 0.557931. \tag{35}$$

Proof. Let M be the optimal solution to Lasserre$_2(G, w)$ produced by Algorithm 1. The product state approximation algorithm of [9] relies on a feasible solution to Lasserre$_1(G, w)$, which we may obtain from M. In particular the [9] algorithm takes as input the vectors $u(\Phi)$, from Equation (22), for $\Phi \in \mathcal{P}_n(1)$ and rounds them to a product state solution.

Let Γ and ${}_2F_1$ be the Gamma and Hyergeometric functions as they are normally defined [1]. The analysis of the [9] algorithm considers a worst-case edge $ij \in E$ and depends on the value $v_{ij} = -x_{ij}$. The worst-case approximation ratio is determined by the quantity

$$\min_{v_{ij} \in [-1, \frac{1}{3}]} \frac{1 - F(v_{ij})}{1 - 3v_{ij}}, \text{ where } F(t) = \frac{2}{3} \left(\frac{\Gamma(2)}{\Gamma(3/2)} \right)^2 {}_2F_1 \begin{bmatrix} 1/2 & 1/2 \\ 5/2 & \end{bmatrix} ; t^2 \Bigg]. \tag{36}$$

For more details see the paragraph above Section 4.1 in [9], where t in that paper is equal to v_{ij} in our terminology. The first inequality, Equation (34) is immediate because it is the worst case approximation factor for their algorithm.

The worst-case value of v_{ij} in Equation (36) is close to -1. We take advantage of the fact that $v_{ij} \geq -\alpha(2)$ for $ij \in S$, avoiding the worst case. In particular we get a ratio of:

$$\min_{v_{ij} \in [-\frac{5}{9}, \frac{1}{3}]} \frac{1 - F(v_{ij})}{1 - 3v_{ij}} = \frac{3}{8} (1 - F(5/9)) \geq 0.557931. \qquad \blacktriangleleft$$

References

1. Milton Abramowitz and Irene A Stegun. *Handbook of mathematical functions with formulas, graphs, and mathematical tables*, volume 55. US Government printing office, 1948.
2. Anurag Anshu, David Gosset, and Karen Morenz. Beyond Product State Approximations for a Quantum Analogue of Max Cut. In Steven T. Flammia, editor, *15th Conference on the Theory of Quantum Computation, Communication and Cryptography (TQC 2020)*, volume 158 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 7:1–7:15, Dagstuhl, Germany, 2020. Schloss Dagstuhl–Leibniz-Zentrum für Informatik. ISSN: 1868-8969.
3. Boaz Barak, Fernando G.S.L. Brandão, Aram W. Harrow, Jonathan Kelner, David Steurer, and Yuan Zhou. Hypercontractivity, sum-of-squares proofs, and their applications. In *Proceedings of the forty-fourth annual ACM symposium on Theory of computing*, pages 307–326, 2012.
4. Adam D. Bookatz. QMA-complete problems. *Quantum Info. Comput.*, 14(5 & 6):361–383, 2014.
5. Fernando G.S.L. Brandão and Aram W. Harrow. Product-state approximations to quantum ground states. In *Proceedings of the Forty-Fifth Annual ACM Symposium on Theory of Computing*, STOC '13, page 871–880, New York, NY, USA, 2013. Association for Computing Machinery.
6. Sergey Bravyi, David Gosset, Robert König, and Kristan Temme. Approximation algorithms for quantum many-body problems. *Journal of Mathematical Physics*, 60(3):032203, 2019.
7. Eden Chlamtáč and Gyanit Singh. Improved approximation guarantees through higher levels of SDP hierarchies. In *Approximation, Randomization and Combinatorial Optimization. Algorithms and Techniques*, pages 49–62. Springer, 2008.
8. Andrew C Doherty, Yeong-Cherng Liang, Ben Toner, and Stephanie Wehner. The quantum moment problem and bounds on entangled multi-prover games. In *2008 23rd Annual IEEE Conference on Computational Complexity*, pages 199–210. IEEE, 2008.
9. Sevag Gharibian and Ojas Parekh. Almost optimal classical approximation algorithms for a quantum generalization of max-cut. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques (APPROX/RANDOM 2019)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2019.
10. Michel X Goemans and David P Williamson. Improved approximation algorithms for maximum cut and satisfiability problems using semidefinite programming. *Journal of the ACM (JACM)*, 42(6):1115–1145, 1995.
11. Dima Grigoriev. Complexity of positivstellensatz proofs for the knapsack. *computational complexity*, 10(2):139–154, 2001.

12 Sean Hallgren, Eunou Lee, and Ojas Parekh. An approximation algorithm for the MAX-2-local Hamiltonian problem. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques (APPROX/RANDOM 2020)*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2020.

13 Ryszard Horodecki and Michał Horodecki. Information-theoretic aspects of inseparability of mixed states. *Physical Review A*, 54(3):1838, 1996.

14 Subhash Khot. On the power of unique 2-prover 1-round games. In *Proceedings of the thiry-fourth annual ACM symposium on Theory of computing*, pages 767–775, 2002.

15 Subhash Khot, Guy Kindler, Elchanan Mossel, and Ryan O'Donnell. Optimal inapproximability results for MAX-CUT and other 2-variable CSPs? *SIAM Journal on Computing*, 37(1):319–357, 2007.

16 Subhash Khot and Nisheeth K Vishnoi. On the unique games conjecture. In *FOCS*, volume 5, page 3. Citeseer, 2005.

17 Alexei Yu Kitaev, Alexander Shen, Mikhail N Vyalyi, and Mikhail N Vyalyi. *Classical and quantum computation*. Number 47 in Graduate studies in mathematics. American Mathematical Soc., 2002.

18 Jean B Lasserre. An explicit exact SDP relaxation for nonlinear 0-1 programs. In *International Conference on Integer Programming and Combinatorial Optimization*, pages 293–303. Springer, 2001.

19 Jean B Lasserre. Global optimization with polynomials and the problem of moments. *SIAM Journal on optimization*, 11(3):796–817, 2001.

20 Elliott Lieb and Daniel Mattis. Ordering energy levels of interacting spin systems. *Journal of Mathematical Physics*, 3(4):749–751, 1962.

21 Yi-Kai Liu. Consistency of local density matrices is QMA-complete. In *Approximation, randomization, and combinatorial optimization. algorithms and techniques*, pages 438–449. Springer, 2006.

22 Ojas Parekh and Kevin Thompson. Beating random assignment for approximating quantum 2-local Hamiltonian problems. *arXiv preprint*, 2020. `arXiv:2012.12347`.

23 Pablo A Parrilo. *Structured semidefinite programs and semialgebraic geometry methods in robustness and optimization*. PhD thesis, California Institute of Technology, 2000.

24 Stefano Pironio, Miguel Navascués, and Antonio Acìn. Convergent relaxations of polynomial optimization problems with noncommuting variables. *SIAM Journal on Optimization*, 20(5):2157–2180, 2010.

25 William Pulleyblank and Jack Edmonds. Facets of 1-matching polyhedra. In Claude Berge and Dijen Ray-Chaudhuri, editors, *Hypergraph Seminar*, pages 214–242, Berlin, Heidelberg, 1974. Springer Berlin Heidelberg.

26 Prasad Raghavendra and Ning Tan. Approximating CSPs with global cardinality constraints using SDP hierarchies. In *Proceedings of the twenty-third annual ACM-SIAM symposium on Discrete Algorithms*, pages 373–387. SIAM, 2012.

27 Alexander Schrijver. *Combinatorial optimization: polyhedra and efficiency*, volume 24. Springer Science & Business Media, 2003.

28 Lieven Vandenberghe and Stephen Boyd. Semidefinite programming. *SIAM Review*, 38(1):49–95, 1996. Publisher: Society for Industrial and Applied Mathematics.

29 Vijay V Vazirani. *Approximation algorithms*. Springer Science & Business Media, 2013.

30 David P Williamson and David B Shmoys. *The design of approximation algorithms*. Cambridge University Press, 2011.

31 Fuzhen Zhang. *The Schur complement and its applications*, volume 4. Springer Science & Business Media, 2006.

Matching on the Line Admits No $o(\sqrt{\log n})$-Competitive Algorithm

Enoch Peserico
Università degli Studi di Padova, Italy

Michele Scquizzato ✉
Università degli Studi di Padova, Italy

Abstract

We present a simple proof that the competitive ratio of any randomized online matching algorithm for the line exceeds $\sqrt{\log_2(n+1)}/15$ for all $n = 2^i - 1 : i \in \mathbb{N}$, settling a 25-year-old open question.

2012 ACM Subject Classification Theory of computation → Online algorithms

Keywords and phrases Metric matching, online algorithms, competitive analysis

Digital Object Identifier 10.4230/LIPIcs.ICALP.2021.103

Category Track A: Algorithms, Complexity and Games

Related Version *Extended Version*: https://arxiv.org/abs/2012.15593

Funding *Michele Scquizzato*: supported, in part, by Univ. Padova grant BIRD197859/19.

Acknowledgements We are indebted to Kirk Pruhs and the anonymous reviewers for their constructive criticism and insightful observations.

1 Online matching, on the line

In *online metric matching* [7, 9] n points of a metric space are designated as *servers*. One by one n *requests* arrive at arbitrary points of the space; upon arrival each must be matched to a yet unmatched server, at a cost equal to their distance. Matchings should minimize the ratio between the total cost and the *offline* cost attainable if all requests were known beforehand. A matching algorithm is $c(n)$-*competitive* if it keeps this ratio no higher than $c(n)$ for all possible placements of servers and requests.

It is widely acknowledged [1, 10, 14] that the line is the most interesting metric space for the problem. Matching on the line models many scenarios, like a shop that must rent to customers skis of approximately their height, where a stream of requests must be serviced with minimally mismatched items from a known store. Despite matching being specifically studied on the line since at least 1996 [8], no tight competitiveness bounds are known.

As for upper bounds, the line is a doubling space and thus admits an $O(\log n)$-competitive randomized algorithm [5]; a sequence of recent developments [1, 12, 13] yielded the same ratio without randomization. Better bounds have been obtained only by algorithms with additional power, such as that to re-assign past requests [6, 11] or predict future ones [2].

As for lower bounds, the competitive ratio is at least 4.591 for randomized algorithms and 9 for deterministic ones since the *cow-path* problem is a special case of matching on the line [8]. These bounds were conjectured tight [8] until a complex adversarial strategy yielded a lower bound of 9.001 for deterministic algorithms [4]. Beyond some $\Omega(\log n)$ bounds for restricted classes of algorithms [3, 10, 12], there has been no further progress on the lower-bound side before this work.

2 An $\Omega(\sqrt{\log n})$-competitiveness bound

We prove a simple $\Omega(\sqrt{\log n})$ lower bound on the competitive ratio of randomized online matching algorithms for the line.

For any $n = 2^i - 1$ with $i \in \mathbb{N}$ consider the $[0, n{+}1]$ interval; for each positive integer $j \leq n$ place a server at point j, and place n requests over $\log_2(n{+}1)$ rounds as follows. On the r^{th} round (for $1 \leq r \leq \log_2(n{+}1)$) partition the interval into $(n{+}1)/2^r$ subintervals of length 2^r, choose within each uniformly and independently at random an *origin* point, and place a request on the closest integer multiple of 2^{-n} breaking ties arbitrarily. "Discretizing" requests instead of directly using the corresponding origins prevents some technical difficulties – see our remark at the end.

We prove in Lemma 1 that the expected distance between the ℓ^{th} leftmost server and the ℓ^{th} leftmost origin is $O(\sqrt{\log n})$, so servers and requests can be matched with an expected offline cost $O(n\sqrt{\log n})$. Conversely, we prove in Lemma 2 that *any* online matching algorithm ALG incurs an expected $\Omega(n)$ cost in any given round, for a total cost $\Omega(n \log n)$. The two results can be combined to prove that on *some* request sequence ALG incurs $\Omega(\sqrt{\log n})$ times the offline cost.

▶ **Lemma 1.** *The expected distance between the ℓ^{th} leftmost origin and the ℓ^{th} leftmost server is at most $\sqrt{\log_2(n{+}1)} + 3$.*

Proof. Let S_ℓ be ℓ^{th} leftmost server and g_ℓ be the number of origins to its left. Note that if g_ℓ equals respectively ℓ or $\ell - 1$, the ℓ^{th} origin is the first immediately to the left, or to the right of S_ℓ; and since the first round placed one origin in every subinterval of size 2, such an origin is within distance 3 of S_ℓ. By the same token, denoting by δ_ℓ the quantity $|g_\ell - (\ell - \frac{\ell}{n+1})|$, the ℓ^{th} leftmost origin is within distance $2\delta_\ell + 3$ of S_ℓ. Note that δ_ℓ is the absolute deviation from the mean of r_ℓ, since r_ℓ is the sum of n independent indicator random variables each denoting whether a given origin was placed to the left of S_ℓ, with total expectation $\frac{n}{n+1}\ell = \ell - \frac{\ell}{n+1}$ (by construction, the expected density of origins is constant throughout the main interval). At most one such variable in a given round has variance greater than 0, albeit obviously at most 1/4: that corresponding to the origin placed in a subinterval holding S_ℓ strictly in its interior. Adding the individual variances we obtain that the variance of r_ℓ, i.e. the expectation of δ_ℓ^2, is at most $\log_2(n{+}1)/4$; and since by Jensen's inequality $E[\delta_\ell] \leq E[\delta_\ell^2]^{\frac{1}{2}}$, the expected distance between S_ℓ and the ℓ^{th} leftmost origin is at most $\sqrt{\log_2(n{+}1)} + 3$. ◀

▶ **Lemma 2.** *Any randomized online matching algorithm incurs an expected cost greater than $(n{+}1)/12$ in each round.*

Proof. Consider an origin placed uniformly at random in a subinterval of size 2^r during the r^{th} round. Assume m unmatched servers in the interior points of that subinterval divide it into $m + 1$ segments of (integer) length $d_0, \ldots, d_m$. Then the probability the corresponding *request* falls within a segment of length d is $d/2^r$, in which case the expected distance of the *request* from the segment's closer endpoint is $d/4$. Adding over all the s_r segments in all the round's subintervals, applying Jensen's inequality, and noting that s_r does not exceed the number of subintervals (i.e. $(n{+}1)/2^r$) plus the total number of unmatched servers (i.e. $(n{+}1)/2^{r-1} - 1$), the expected cost to service all requests in the round is at least:

$$\sum_{h=1}^{s_r} \frac{d_h}{4} \cdot \frac{d_h}{2^r} \geq \frac{1}{4 \cdot 2^r} s_r \left(\frac{n{+}1}{s_r}\right)^2 > \frac{(n{+}1)^2}{4 \cdot 2^r} \cdot \frac{2^r}{3(n{+}1)} = \frac{n{+}1}{12}.$$ ◀

We can then easily prove the following:

▶ **Theorem.** *The competitive ratio of any randomized online matching algorithm for the line exceeds* $\sqrt{\log_2(n+1)}/15$ *for all* $n = 2^i - 1 : i \in \mathbb{N}$.

Proof. Let $C_A(\sigma)$ be the expected cost incurred by a randomized online matching algorithm ALG on a request sequence σ, and $C_O(\sigma)$ the offline cost; and let p_σ be the probability of generating σ through the origin-request process described earlier. Since $\forall a_i, b_i > 0$ we have that $(\sum_i a_i)/(\sum_i b_i)$ is a convex linear combination of the individual ratios a_i/b_i, focusing on the case $\sqrt{\log_2(n+1)}/15 \geq 1$ for which $\sqrt{\log_2(n+1)} + 3 + 2^{-n} < (5/4)\sqrt{\log_2(n+1)}$:

$$\max_{\sigma:p_\sigma\neq 0} \frac{C_A(\sigma)}{C_O(\sigma)} \geq \frac{\sum_{\sigma:p_\sigma\neq 0} C_A(\sigma)p_\sigma}{\sum_{\sigma:p_\sigma\neq 0} C_O(\sigma)p_\sigma} > \frac{(n+1)\log_2(n+1)/12}{n(\sqrt{\log_2(n+1)} + 3 + 2^{-n})} > \frac{\sqrt{\log_2(n+1)}}{15}. \qquad \blacktriangleleft$$

▶ Remark. Without discretized requests the term $\sum_{\sigma:p_\sigma\neq 0} C_A(\sigma)p_\sigma$ in the theorem's proof would have been an integral, potentially ill-defined (for example, if ALG serviced requests for rational points in an interval with one server and for irrational points with another).

References

1. Antonios Antoniadis, Neal Barcelo, Michael Nugent, Kirk Pruhs, and Michele Scquizzato. A $o(n)$-competitive deterministic algorithm for online matching on a line. *Algorithmica*, 81(7):2917–2933, 2019.
2. Antonios Antoniadis, Christian Coester, Marek Eliás, Adam Polak, and Bertrand Simon. Online metric algorithms with untrusted predictions. In *Proceedings of the 37th ICML*, pages 345–355, 2020.
3. Antonios Antoniadis, Carsten Fischer, and Andreas Tönnis. A collection of lower bounds for online matching on the line. In *Proceedings of the 13th LATIN*, pages 52–65, 2018.
4. Bernhard Fuchs, Winfried Hochstättler, and Walter Kern. Online matching on a line. *Theor. Comput. Sci.*, 332(1-3):251–264, 2005.
5. Anupam Gupta and Kevin Lewi. The online metric matching problem for doubling metrics. In *Proceedings of the 39th ICALP*, pages 424–435, 2012.
6. Varun Gupta, Ravishankar Krishnaswamy, and Sai Sandeep. Permutation strikes back: The power of recourse in online metric matching. In *Proceedings of the 23rd APPROX*, pages 40:1–40:20, 2020.
7. Bala Kalyanasundaram and Kirk Pruhs. Online weighted matching. *J. Algorithms*, 14(3):478–488, 1993.
8. Bala Kalyanasundaram and Kirk Pruhs. Online network optimization problems. In *Online Algorithms: The State of the Art*, pages 268–280. Springer-Verlag, 1998. From the Dagstuhl Seminar on Online Algorithms, 1996.
9. Samir Khuller, Stephen G. Mitchell, and Vijay V. Vazirani. On-line algorithms for weighted bipartite matching and stable marriages. *Theor. Comput. Sci.*, 127(2):255–267, 1994.
10. Elias Koutsoupias and Akash Nanavati. The online matching problem on a line. In *Proceedings of the 1st WAOA*, pages 179–191, 2003.
11. Nicole Megow and Lukas Nölke. Online minimum cost matching with recourse on the line. In *Proceedings of the 23rd APPROX*, pages 37:1–37:16, 2020.
12. Krati Nayyar and Sharath Raghvendra. An input sensitive online algorithm for the metric bipartite matching problem. In *Proceedings of the 58th IEEE FOCS*, pages 505–515, 2017.
13. Sharath Raghvendra. Optimal analysis of an online algorithm for the bipartite matching problem on a line. In *Proceedings of the 34th SoCG*, pages 67:1–67:14, 2018.
14. Rob van Stee. SIGACT news online algorithms column 27: Online matching on the line, part 1. *SIGACT News*, 47(1):99–110, 2016.

Non-Mergeable Sketching for Cardinality Estimation

Seth Pettie
University of Michigan, Ann Arbor, MI, USA

Dingyu Wang
University of Michigan, Ann Arbor, MI, USA

Longhui Yin
Tsinghua University, Beijing, China

Abstract

Cardinality estimation is perhaps the simplest non-trivial statistical problem that can be solved via sketching. Industrially-deployed sketches like HyperLogLog, MinHash, and PCSA are *mergeable*, which means that large data sets can be sketched in a distributed environment, and then merged into a single sketch of the whole data set. In the last decade a variety of sketches have been developed that are *non-mergeable*, but attractive for other reasons. They are *simpler*, their cardinality estimates are *strictly unbiased*, and they have substantially *lower variance*.

We evaluate sketching schemes on a reasonably level playing field, in terms of their *memory-variance product* (MVP). E.g., a sketch that occupies $5m$ bits and whose relative variance is $2/m$ (standard error $\sqrt{2/m}$) has an MVP of 10. Our contributions are as follows.

- Cohen [14] and Ting [35] independently discovered what we call the *Martingale transform* for converting a mergeable sketch into a non-mergeable sketch. We present a simpler way to analyze the limiting MVP of Martingale-type sketches.
- Pettie and Wang proved that the Fishmonger sketch [31] has the best MVP, $H_0/I_0 \approx 1.98$, among a class of mergeable sketches called "linearizable" sketches. (H_0 and I_0 are precisely defined constants.) We prove that the Martingale transform is optimal in the non-mergeable world, and that Martingale Fishmonger in particular is optimal among linearizable sketches, with an MVP of $H_0/2 \approx 1.63$. E.g., this is circumstantial evidence that to achieve 1% standard error, we cannot do better than a 2 kilobyte sketch.
- Martingale Fishmonger is neither simple nor practical. We develop a new mergeable sketch called Curtain that strikes a nice balance between simplicity and efficiency, and prove that Martingale Curtain has limiting MVP ≈ 2.31. It can be updated with $O(1)$ memory accesses and it has lower empirical variance than Martingale LogLog, a practical non-mergeable version of HyperLogLog.

2012 ACM Subject Classification Theory of computation → Sketching and sampling

Keywords and phrases Cardinality Estimation, Sketching

Digital Object Identifier 10.4230/LIPIcs.ICALP.2021.104

Category Track A: Algorithms, Complexity and Games

Related Version *Full Version*: https://arxiv.org/abs/2008.08739

Funding This work was supported by NSF grants CCF-1637546 and CCF-1815316.

1 Introduction

Cardinality estimation[1] is a fundamental problem in streaming and sketching with diverse applications in databases [12, 21], network monitoring [5, 8, 39, 11], nearest neighbor search [33], caching [37], and genomics [30, 17, 38, 2]. In the ***sequential*** setting of this problem, we receive the elements of a multiset $\mathcal{A} = \{a_1, a_2, \ldots, a_N\}$ one at a time. We

[1] (aka F_0 estimation or *Distinct Elements*)

maintain a small *sketch* S of the elements seen so far, such that the true cardinality $\lambda = |\mathcal{A}|$ is estimated by some $\hat{\lambda}(S)$. The ***distributed*** setting is similar, except that $\mathcal{A}$ is partitioned arbitrarily among several machines, the shares being sketched separately and combined into a sketch of $\mathcal{A}$. Only ***mergeable*** sketches are deployed in distributed settings; see Definition 2 below.

▶ **Definition 1.** *In the* RANDOM ORACLE MODEL $\mathcal{A} \subseteq [U]$ *and we have oracle access to a uniformly random permutation* $h : [U] \to [U]$ *(or a uniformly random hash function* $h : [U] \to [0, 1]$*). In the* STANDARD MODEL *we can generate random bits as necessary, but must explicitly store any hash functions in the sketch.*

▶ **Definition 2.** *Suppose* $\mathcal{A}^{(1)}, \mathcal{A}^{(2)}$ *are multisets such that* $\mathcal{A} = \mathcal{A}^{(1)} \cup \mathcal{A}^{(2)}$*. A sketching scheme is* **mergeable** *if, whenever,* $\mathcal{A}^{(1)}, \mathcal{A}^{(2)}$ *are sketched as* $S^{(1)}, S^{(2)}$ *(using the same* RANDOM ORACLE h *or the same source of random bits in the* STANDARD MODEL*), the sketch* S *of* $\mathcal{A}$ *can be computed from* $S^{(1)}, S^{(2)}$ *alone.*

STANDARD MODEL sketches [1, 3, 4, 6, 22, 27] usually make an (ϵ, δ)-guarantee, i.e.,

$$\Pr\left(\hat{\lambda} \notin [(1-\epsilon)\lambda, (1+\epsilon)\lambda]\right) < \delta.$$

The state-of-the-art STANDARD MODEL sketch [6, 27] uses $O(\epsilon^{-2} \log \delta^{-1} + \log U)$ bits, which is optimal at this level of specificity, as it meets the space lower bounds of $\Omega(\log U)$, $\Omega(\epsilon^{-2})$ (when $\delta = \Theta(1)$), and $\Omega(\epsilon^{-2} \log \delta^{-1})$ [1, 25, 26]. However, the leading constants hidden by [6, 27] are quite large.

In the RANDOM ORACLE MODEL the cardinality estimate $\hat{\lambda}$ typically has negligible bias, and errors are expressed in terms of the *relative variance* $\lambda^{-2} \cdot \mathrm{Var}(\hat{\lambda} \mid \lambda)$ or *relative standard deviation* $\lambda^{-1}\sqrt{\mathrm{Var}(\hat{\lambda} \mid \lambda)}$, also called the *standard error*. Sketches that use $\Omega(m)$ bits typically have relative variances of $O(1/m)$. Thus, the most natural way to measure the quality of the *sketching scheme* itself is to look at its limiting *memory-variance product* (MVP), i.e., the product of its memory and variance as $m \to \infty$.

Until about a decade ago, all STANDARD/RANDOM ORACLE sketches were mergeable, and suitable to both distributed and sequential applications. For reasons that are not clear to us, the idea of ***non-mergeable*** sketching was discovered independently by multiple groups [10, 24, 14, 35] at about the same time, and quite *late* in the 40-year history of cardinality estimation. Chen, Cao, Shepp, and Nguyen [10] invented the S-Bitmap in 2011, followed by Helmi, Lumbroso, Martínez, and Viola's[24] Recordinality in 2012. In 2014 Cohen [14] and Ting [35] independently invented what we call the *Martingale transform*, which is a simple, mechanical way to transform any mergeable sketch into a (better) non-mergeable sketch.[2]

In a companion paper [31], we analyzed the MVPs of *mergeable* sketches under the assumption that the sketch was compressed to its entropy bound. Fishmonger (an entropy compressed variant of PCSA with a different estimator function) was shown to have MVP $= H_0/I_0 \approx 1.98$, where

$$H_0 = (\ln 2)^{-1} + \sum_{k=1}^{\infty} k^{-1} \log_2(1 + 1/k) \qquad \text{and} \quad I_0 = \zeta(2) = \pi^2/6.$$

[2] Cohen [14] called these *Historical Inverse Probability* (HIP) sketches and Ting [35] applied the prefix *Streaming* to emphasize that they can be used in the single-stream setting, not the distributed setting.

Furthermore, H_0/I_0 was shown to be the minimum MVP among *linearizable* sketches, a subset of *mergeable* sketches that includes all the popular sketches (HyperLogLog, PCSA, MinHash, etc.).

Our aim in *this* paper is to build a useful framework for designing and analyzing *non-mergeable* sketching schemes, and, following [31], to develop a theory of space-variance optimality in the non-mergeable world. We work in the RANDOM ORACLE MODEL. Our results are as follows.

- Although the *Martingale transform* itself is simple, analyzing the variance of these sketches is not. For example, Cohen [14] and Ting [35] estimated the standard error of Martingale LogLog to be about $\approx \sqrt{3/(4m)} \approx 0.866/\sqrt{m}$ and about $\approx 1/(2\alpha_m m)$, respectively, where the latter tends to $\sqrt{\ln 2/m} \approx 0.8326/\sqrt{m}$ as $m \to \infty$.[3] We give a general method for determining the limiting relative variance of Martingale sketches that is strongly influenced by Ting's perspective.
- What is the most efficient (smallest MVP) non-mergeable sketch for cardinality estimation? The best Martingale sketches perform better than the *ad hoc* non-mergeable S-Bitmap and Recordinality, but perhaps there is a completely different, better way to systematically build non-mergeable sketches. We prove that up to some natural assumptions[4] the best non-mergeable sketch is a Martingale X sketch, for some X. Furthermore, we prove that Martingale Fishmonger, having MVP of $H_0/2 \approx 1.63$, is optimal among all Martingale X sketches, where X is *linearizable*. This provides some circumstantial evidence that Martingale Fishmonger is optimal, and that if we want, say, 1% standard error, we need to use a $H_0/2 \cdot (0.01)^{-2}$-bit sketch, ≈ 2 kilobytes.
- Martingale Fishmonger has an attractive MVP, but it is slow and cumbersome to implement. We propose a new mergeable sketch called Curtain that is "naturally" space efficient and easy to update in $O(1)$ memory accesses, and prove that Martingale Curtain has a limiting MVP ≈ 2.31.

1.1 Prior Work: Mergeable Sketches

Let S_i be the state of the sketch after processing $(a_1, \ldots, a_i)$.

The state of the PCSA sketch [20] is a 2D matrix $S \in \{0,1\}^{m \times \log U}$ and the hash function $h : [U] \to [m] \times \mathbb{Z}^+$ produces two indices: $h(a) = (j,k)$ with probability $m^{-1}2^{-k}$. $S_i(j,k) = 1$ iff $\exists i' \in [i].h(a_{i'}) = (j,k)$. Flajolet and Martin [20] proved that a certain estimator has standard error $0.78/\sqrt{m}$, making the MVP around $(0.78)^2 \log U \approx 0.6 \log U$.

Durand and Flajolet's LogLog sketch [16] consists of m counters. It interprets h exactly as in PCSA, and sets $S_i(j) = k$ iff k is maximum such that $\exists i' \in [i].h(a_{i'}) = (j,k)$. Durand and Flajolet's estimator is of the form $\hat{\lambda}(S) \propto m 2^{m^{-1}\sum_j S(j)}$ and has standard error $\approx 1.3/\sqrt{m}$. Flajolet, Fusy, Gandouet, and Meunier's HyperLogLog [19] is the same sketch but with the estimator $\hat{\lambda}(S) \propto m^2(\sum_j 2^{-S(j)})^{-1}$. They proved that it has standard error tending to $\approx 1.04/\sqrt{m}$. As the space is $m \log\log U$ bits, the MVP is $\approx 1.08 \log\log U$.

The MinCount sketch (aka MinHash or Bottom-m [13, 15, 7]) stores the smallest m hash values, which we assume requires $\log U$ bits each. Using an appropriate estimator [23, 9, 29], the standard error is $1/\sqrt{m}$ and MVP $= \log U$.

---

[3] Here $\alpha_m = \left(m \int_0^\infty \left(\log_2\left(\frac{2+u}{1+u}\right)\right)^m du\right)^{-1}$ is the coefficient of Flajolet et al.'s HyperLogLog estimator.

[4] (the sketch is insensitive to duplicates, and the estimator is unbiased)

It is straightforward to see that the entropy of PCSA and LogLog are both $\Theta(m)$. Scheuermann and Mauve [34] experimented with entropy compressed versions of PCSA and HyperLogLog and found PCSA to be slightly superior. Rather than use the given estimators of [20, 19, 16], Lang [28] used Maximum Likelihood-type Estimators and found entropy-compressed PCSA to be significantly better than entropy-compressed LogLog (with MLE estimators). Pettie and Wang [31] defined the Fisher-Shannon (Fish)[5] number of a sketch as the ratio of its Shannon entropy (controlling its entropy-compressed size) to its Fisher information (controlling the variance of a statistically efficient estimator), and proved that the Fish-number of any base-q PCSA is H_0/I_0, and that the Fish-number of base-q LogLog is worse, but tends to H_0/I_0 in the limit as $q \to \infty$. (The constants H_0, I_0 were defined earlier.)

Table 1 A selection of results on composable sketches (top) and non-composable Martingale sketches (bottom) in terms of their limiting memory-variance product (MVP). Logarithms are base 2.

Mergeable Sketch		Limiting MVP	Notes
PCSA	[20]	$.6 \log U \approx 38.9$	For $U = 2^{64}$
LogLog	[16]	$1.69 \log\log U \approx 10.11$	For $U = 2^{64}$
MinCount	[23, 9, 29]	$\log U = 64$	For $U = 2^{64}$
HyperLogLog	[19]	$1.08 \log\log U \approx 6.48$	For $U = 2^{64}$
Fishmonger	[31]	$H_0/I_0 \approx 1.98$	
Non-Mergeable Sketch			
S-Bitmap	[10]	$O(\log^2(U/m))$	
Recordinality	[24]	$O(\log(\lambda/m) \log U)$	
Martingale PCSA	**new**	$0.35 \log U \approx 22.4$	For $U = 2^{64}$
Martingale LogLog	[14, 35]	$0.69 \log\log U \approx 4.16$	For $U = 2^{64}$
Martingale MinCount	[14, 35]	$0.5 \log U = 32$	For $U = 2^{64}$
Martingale Fishmonger	**new**	$H_0/2 \approx 1.63$	$H_0 = (\ln 2)^{-1} + \sum_{k\geq 1} \frac{\log_2(1+1/k)}{k}$
Martingale Curtain	**new**	≈ 2.31	Theorem 4 with $(q, a, h) = (2.91, 2, 1)$
Non-Mergeable Lower Bound			
Martingale X	**new**	$\geq H_0/2$	X is a *linearizable* sketch

1.2 Prior Work: Non-Mergeable Sketches

Chen, Cao, Shepp, and Nguyen's S-Bitmap [10] consists of a bit string $S \in \{0,1\}^m$ and m known constants $0 \leq \tau_0 < \tau_1 < \cdots < \tau_{m-1} < 1$. It interprets $h(a) = (j, \rho) \in [m] \times [0,1]$ as an index j and real ρ and when processing a, sets $S(j) \leftarrow 1$ iff $\rho > \tau_{\text{HammingWeight}(S)}$. One may confirm that S is insensitive to duplicates in the stream $\mathcal{A}$, but its state depends on the *order* in which $\mathcal{A}$ is scanned. By setting the τ-thresholds and estimator properly, the standard error is $\approx \ln(eU/m)/(2\sqrt{m})$ and $\text{MVP} = O(\log^2(U/m))$.

Recordinality [24] is based on MinCount; it stores (S, cnt), where S is the m smallest hash values encountered and cnt is the *number of times* that S has changed. The estimator looks only at cnt, not S, and has standard error $\approx \sqrt{\ln(\lambda/em)/m}$ and $\text{MVP} = O(\log(\lambda/m) \log U)$.

[5] Fish is essentially the same as MVP, under the assumption that the sketch state is compressed to its entropy.

Cohen [14] and Ting [35] independently described how to turn any sketch into a non-mergeable sketch using what we call the Martingale transform. Let S_i be the state of the original sketch after seeing $(a_1, \ldots, a_i)$ and $P_{i+1} = \Pr(S_{i+1} \neq S_i \mid S_i, a_{i+1} \notin \{a_1, \ldots, a_i\})$ be the probability that it changes state upon seeing a *new* element a_{i+1}.[6] The state of the Martingale sketch is $(S_i, \hat{\lambda}_i)$. Upon processing a_{i+1} it becomes $(S_{i+1}, \hat{\lambda}_{i+1})$, where

$$\hat{\lambda}_{i+1} = \hat{\lambda}_i + P_{i+1}^{-1} \cdot \left[\!\left[S_{i+1} \neq S_i \right]\!\right].$$

Here $\left[\!\left[\mathcal{E}\right]\!\right]$ is the indicator variable for the event $\mathcal{E}$. We assume the original sketch is insensitive to duplicates, so

$$\mathbb{E}(\hat{\lambda}_{i+1}) = \begin{cases} \hat{\lambda}_i & \text{when } a_{i+1} \in \{a_1, \ldots, a_i\} \text{ (and hence } S_{i+1} = S_i) \\ \hat{\lambda}_i + 1 & \text{when } a_{i+1} \notin \{a_1, \ldots, a_i\}. \end{cases}$$

Thus, with $\hat{\lambda}_0 = \lambda_0 = 0$, $\hat{\lambda}_i$ is an unbiased estimator of the true cardinality $\lambda_i = |\{a_1, \ldots, a_i\}|$ and $(\hat{\lambda}_i - \lambda_i)_i$ is a *martingale*. The Martingale-transformed sketch requires the same space, plus just $\log U$ bits to store the estimate $\hat{\lambda}$.

Cohen and Ting [14, 35] both proved that Martingale MinCount has standard error $1/(2\sqrt{m})$ and MVP $= (\log U)/2$. They gave different estimates for the standard error of Martingale LogLog. Ting's estimate is quite accurate, and tends to $\sqrt{\ln 2/m}$ as $m \to \infty$, giving it an MVP $= \ln 2 \log\log U \approx 0.69 \log\log U$.

▶ Remark 3. We call Martingale sketches *non-mergeable* because, in a distributed environment, there is no obvious way to merge the cardinality estimates $(\hat{\lambda})$. On the other hand, Ting [36] has shown that if $(S^A, \hat{\lambda}^A)$ and $(S^B, \hat{\lambda}^B)$ are Martingale MinCount sketches obtained by sequentially processing A and B, that $\hat{\lambda}^A, \hat{\lambda}^B$ carry useful information for estimating $|A \cup B|$ and $|A \cap B|$ beyond that contained in S^A, S^B.

1.3 The Dartboard Model

The ***dartboard model*** [31] is useful for describing cardinality sketches with a single, uniform language. The dartboard model is essentially the same as Ting's [35] *area cutting* process, but with a specific, discrete cell partition and state space fixed in advance.

The *dartboard* is the unit square $[0,1]^2$, partitioned into a set $\mathcal{C} = \{c_0, \ldots, c_{|\mathcal{C}|-1}\}$ of *cells* of various sizes. Every cell may be either *occupied* or *unoccupied*; the *state* is the set of occupied cells and the state space some $\mathcal{S} \subseteq 2^{\mathcal{C}}$.

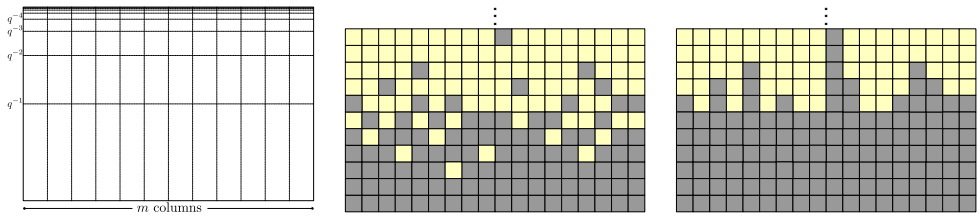
We process a stream of elements one by one; when a *new* element is encountered we throw a *dart* uniformly at random at the dartboard and update the state in response. The relationship between the state and the dart distribution satisfies two rules:

(R1) Every cell with at least one dart is occupied; occupied cells may contain no darts.

(R2) If a dart lands in an occupied cell, the state does not change.

As a consequence of (R1) and (R2), if a dart lands in an empty cell the state *must* change, and occupied cells may never become unoccupied. Dart throwing is merely an intuitive way of visualizing the hash function. Base-q PCSA and LogLog use the same cell partition but with different state spaces; see Figure 1.

[6] These probabilities are over the choice of $h(a_{i+1})$, which, in the RANDOM ORACLE MODEL, is independent of all other hash values.



(a) (b) (c)

Figure 1 The unit square is partitioned into m columns. Each column is partitioned into cells. Cell j covers the vertical interval $[q^{-(j+1)}, q^{-j})$. (b) The state of a PCSA sketch records precisely which cells contain a dart (gray); all others are empty (yellow). (c) The state of the corresponding LogLog sketch.

It was observed [31] that the dartboard model includes all mergeable sketches, and some non-mergeable ones like S-Bitmap. Recordinality and the Martingale sketches obey rules (R1),(R2) but are not strictly dartboard sketches as they maintain some small state information (cnt or $\hat{\lambda}$) outside of the set of occupied cells. Nonetheless, it is useful to speak of the *dartboard part* of their state information.

1.4 Linearizable Sketches

The lower bound of [31] applies to *linearizable* sketches, a subset of mergeable sketches. A sketch is called linearizable if it is possible to encode the occupied/unoccupied status of its cells in some fixed linear order $(c_0, \ldots, c_{\mathcal{C}-1})$, so whether c_i is occupied only depends on the status of $c_0, \ldots, c_{i-1}$ and whether c_i has been hit by a dart. (Thus, it is independent of $c_{i+1}, \ldots, c_{\mathcal{C}-1}$.) Specifically, let Y_i, Z_i be the indicators for whether c_i is occupied, and has been hit by a dart, respectively, and $\mathbf{Y}_i = (Y_0, \ldots, Y_i)$. The state of the sketch is $\mathbf{Y}_{\mathcal{C}-1}$; it is called linearizable if there is some monotone function $\phi : \{0,1\}^* \to \{0,1\}$ such that

$$Y_i = Z_i \vee \phi(\mathbf{Y}_{i-1}).$$

I.e., if $\phi(\mathbf{Y}_{i-1}) = 1$, c_i is forced to be occupied and the state is forever independent of Z_i.

PCSA-type sketches [20, 18] are linearizable, as are (Hyper)LogLog [19, 16], and all MinCount, MinHash, and Bottom-m type sketches [13, 7, 23, 9, 29]. It is very easy to engineer non-linearizable sketches; see [31]. The open problem is whether this is ever a *good idea* in terms of memory-variance performance.

1.5 Organization

In Section 2 we introduce the Curtain sketch, which is a linearizable (hence mergeable) sketch in the dartboard model. In Section 3 we prove some general theorems on the bias and asymptotic relative variance of Martingale-type sketches, and in Section 4 we apply this framework to bound the limiting MVP of Martingale PCSA, Martingale Fishmonger, and Martingale Curtain.

In Section 5 we prove some results on the optimality of the Martingale transform itself, and that Martingale Fishmonger has the lowest variance among those based on linearizable sketches.

Section 6 presents some experimental findings that demonstrate that the conclusions drawn from the asymptotic analysis of Martingale sketches are extremely accurate in the pre-asymptotic regime as well, and that Martingale Curtain has lower variance than Martingale LogLog.

All the missing proofs can be found in the full version [32].

2 The Curtain Sketch

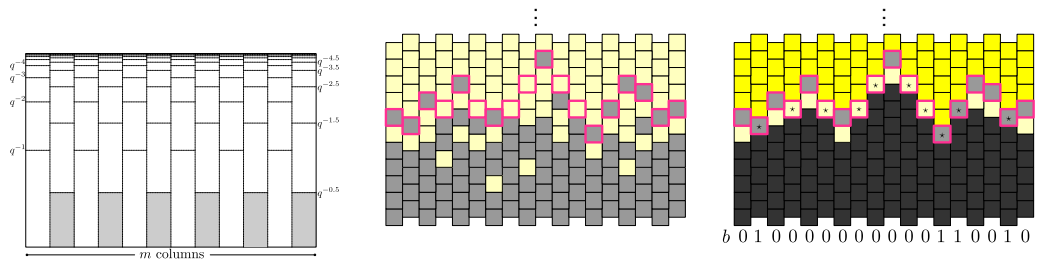
Design Philosophy

Our goal is to strike a nice balance between the simplicity and time-efficiency of (Hyper)LogLog, and the superior information-theoretic efficiency of PCSA, which can only be fully realized under extreme (and time-inefficient) compression to its entropy bound [31, 28]. Informally, if we are dedicating at least 1 bit to encode the status of a cell, the *best* cells to encode have mass $\Theta(\lambda^{-1})$ and we should design a sketch that maximizes the number of such cells encoded.

We assume the dartboard is partitioned into m columns; define $\mathrm{Cell}(j,i)$ to be the cell in column i covering the vertical interval $[q^{-(j+1)}, q^{-j})$. In a PCSA sketch, the occupied cells are precisely those with at least one dart. In LogLog, the occupied cells in each column are contiguous, extending to the highest cell containing a dart. In Figure 1, cells are drawn with uniform sizes for clarity.

Consider the vector $v = (g_0, g_1, \ldots, g_{m-1})$ where $\mathrm{Cell}(g_i, i)$ is the highest occupied cell in LogLog/PCSA. The *curtain* of v w.r.t. allowable offsets $\mathcal{O}$ is a vector $v_{\mathrm{curt}} = (\hat{g}_0, \hat{g}_1, \ldots, \hat{g}_{m-1})$ such that (i) $\forall i \in [1, m-1].\ \hat{g}_i - \hat{g}_{i-1} \in \mathcal{O}$, and (ii) v_{curt} is the minimal such vector dominating v, i.e., $\forall i.\ \hat{g}_i \geq g_i$. Although we have described v_{curt} as a function of v, it is clearly possible to maintain v_{curt} as darts are thrown, without knowing v.

We have an interest in $|\mathcal{O}|$ being a power of 2 so that curtain vectors may be encoded efficiently, as a series of offsets. On the other hand, it is most efficient if $\mathcal{O}$ is symmetric around zero. For these reasons, we use a base-q "sawtooth" cell partition of the dartboard; see Figure 2. Henceforth $\mathrm{Cell}(j,i)$ is defined as usual, except j is an integer when i is even and a half-integer when i is odd. Then the allowable offsets are $\mathcal{O}_a = \{-(a-1/2), -(a-3/2), \ldots, -1/2, 1/2, \ldots, a-3/2, a-1/2\}$, for some a that is a power of 2.



(a) (b) (c)

Figure 2 (a) The base-q "sawtooth" cell partition. (b) and (c) depict a Curtain sketch w.r.t. $\mathcal{O} = \{-3/2, -1/2, 1/2, 3/2\}$ and $h = 1$. (b) Gray cells contain at least one dart; light yellow cells contain none. The curtain $v_{\mathrm{curt}} = (\hat{g}_i)$ is highlighted with a pink boundary. (c) Columns that are in *tension* have a $\star$ in their curtain cell. All *dark* gray cells are occupied and all *dark* yellow cells are free according to Rule 3. All other cells are occupied/free (light gray, light yellow) according to Rules 1 and 2.

Let $\mathrm{Cell}(g_i, i)$ be the highest cell containing a dart in column i in the *sawtooth* cell partition and $v_{\mathrm{curt}} = (\hat{g}_i)$ be the curtain vector of $v = (g_i)$ w.r.t. offsets $\mathcal{O} = \mathcal{O}_a$. We say column i is *in tension* if $(\cdots, \hat{g}_{i-1}, \hat{g}_i - 1, \hat{g}_{i+1}, \cdots)$ is not a valid curtain, i.e., if $\hat{g}_i - \hat{g}_{i-1} = \min(\mathcal{O})$ or $\hat{g}_{i+1} - \hat{g}_i = \max(\mathcal{O})$. In particular, if column i is *not* in tension, then $\mathrm{Cell}(\hat{g}_i, i)$ must contain at least one dart, for if it contained no darts the curtain would be dropped to $\hat{g}_i - 1$ at column i. However, if column i *is* in tension, then $\mathrm{Cell}(\hat{g}_i, i)$ might not contain a dart.

The Curtain sketch encodes $v_{\mathrm{curt}} = (\hat{g}_i)$ w.r.t. the base-q sawtooth cell partition and offsets $\mathcal{O}_a$, and a bit-array $b = \{0,1\}^{h \times m}$. This sketch designates each cell *occupied* or *free* as follows.

Rule 1. If column i is not in tension then $\mathrm{Cell}(\hat{g}_i, i)$ is occupied, and $b(\cdot, i)$ encodes the status of the h cells below the curtain, i.e., $\mathrm{Cell}(\hat{g}_i - (j+1), i)$ is occupied iff $b(j,i) = 1$, $j \in \{0, \ldots, h-1\}$.

Rule 2. If column i is in tension, then $\mathrm{Cell}(\hat{g}_i - j, i)$ is occupied iff $b(j,i) = 1$, $j \in \{0, \ldots, h-1\}$.

Rule 3. Every cell above the curtain is free ($\mathrm{Cell}(\hat{g}_i + j, i)$, when $j \geq 1$) and all remaining cells are occupied.

Figure 2 gives an example of a Curtain sketch, with $\mathcal{O} = \{-3/2, -1/2, 1/2, 3/2\}$ and $h = 1$. (The base q of the cell partition is unspecified in this example.)

▶ **Theorem 4.** *Consider the Martingale Curtain sketch with parameters q, a, h (base q, $\mathcal{O}_a = \{-(a-1/2), \ldots, a-1/2\}$, and $b \in \{0,1\}^{h \times m}$), and let $\hat{\lambda}$ be its estimate of the true cardinality λ.*

1. *$\hat{\lambda}$ is an unbiased estimate of λ.*
2. *The relative variance of $\hat{\lambda}$ is:*

$$\frac{1}{\lambda^2}\operatorname{Var}(\hat{\lambda} \mid \lambda) = \frac{(1 + o_{\lambda/m}(1) + o_m(1))q \ln q}{2m(q-1)} \left(\frac{q-1}{q} + \frac{2}{q^h(q^{a-1/2}-1)} + \frac{1}{q^{h+1}} \right),$$

As a result, the limiting MVP of Martingale Curtain is

$$\mathit{MVP} = (\log_2(2a) + h) \times \frac{q \ln q}{2(q-1)} \left(\frac{q-1}{q} + \frac{2}{q^h(q^{a-1/2}-1)} + \frac{1}{q^{h+1}} \right).$$

Proof. Follows from Theorems 11 and 17. ◀

Here $o_{\lambda/m}(1)$ and $o_m(1)$ are terms that go to zero as m and λ/m get large. Recall that for practical reasons we want to parameterize Theorem 4 with a a power of 2 and h an integer, but it is realistic to set $q > 1$ to be any real. Given these constraints, the optimal setting is $q = 2.91$, $a = 2$, and $h = 1$, exactly as in the example in Figure 2. This uses $\log\log U + 3(m-1)$ bits to store the sketch proper, $\log U$ bits[7] to store $\hat{\lambda}$, and achieves a limiting MVP ≈ 2.31. In other words, to achieve a standard error $1/\sqrt{b}$, we need about $2.31b$ bits.

Implementation Considerations

We encode a curtain $(\hat{g}_0, \hat{g}_1, \ldots, \hat{g}_{m-1})$ as $\hat{g}_0$ and an offset vector $(o_1, o_2, \ldots, o_{m-1})$, $o_i = \hat{g}_i - \hat{g}_{i-1}$, where $\hat{g}_0$ takes $\log_2 \log_q U \leq 6$ bits and o_i takes $\log_2 |\mathcal{O}| = \log_2(2a)$ bits. Clearly, to evaluate $\hat{g}_i$ we need to compute the prefix sum $\hat{g}_0 + \sum_{i' \leq i} o_{i'}$.

[7] It is fine to store an approximation $\tilde{\lambda}$ of $\hat{\lambda}$ with $O(\log m)$ bits of precision.

▶ **Lemma 5.** *Let $(x_0, \ldots, x_{\ell-1})$ be a vector of t-bit unsigned integers packed into $\lceil t\ell/w \rceil$ words, where each word has $w = \Omega(\log(t\ell))$ bits. The prefix sum $\sum_{j\in[0,i]} x_j$ can be evaluated in $O(t\ell/w + \log w)$ time.*

Proof. W.l.o.g. we can assume $i = \ell - 1$, so the task is to sum the entire list. In $O(\lceil (t\ell)/w \rceil)$ time we can halve the number of summands, by masking out the odd and even summands and adding these vectors together. After halving twice in this way, we have a vector of $\ell/4$ $(t+2)$-bit integers, each allocated $4t$ bits. At this point we can halve the number of words by adding the $(2i+1)$th word to the $2i$th word. Thus, if $T_w(\ell, t)$ is the time needed to solve this problem, $T_w(\ell, t) = T_w(\ell/8, 4t) + O(\lceil (t\ell)/w \rceil)$, which is $O((t\ell)/w + \log w)$. ◀

In our context $t = \log_2(2a) = 2$, so even if m is a medium-size constant, say at most 256 or 512, we only have to do prefix sums over 8 or 16 consecutive 64-bit words. If m is much larger then it would be prudent to partition the dartboard into m/c independent curtains, each with $c = 256$ or 512 columns. This keeps the update time independent of m and increases the space overhead negligibly.

We began this section by highlighting the design philosophy, which emphasizes conceptual simplicity and efficiency. Our encoding uses fixed-length codes for the offsets, and can be decoded very efficiently by exploiting bit-wise operations and word-level parallelism. That said, we are mainly interested in analyzing the *theoretical* performance of sketches, and will not attempt an exhaustive experimental evaluation in this work.

3 Foundations of the Martingale Transform

In this section we present a simple framework for analyzing the limiting variance of Martingale sketches, which is strongly influenced by Ting's [35] work. Theorem 7 gives simple unbiased estimators for the cardinality and the variance of the the cardinality estimator. The upshot of Theorem 7 is that to analyze the variance of the estimator, we only need to bound $\mathbb{E}(P_k^{-1})$, where P_k is the probability the kth distinct element changes the sketch. Theorem 11 further shows that for sketches composed of m subsketches (like Curtain, HyperLogLog, and PCSA), the limiting variance tends to $\frac{1}{2\kappa m}$, where κ is a constant that depends on the sketch scheme. Section 4 analyzes the constant κ for each of PCSA, LogLog, and Curtain. Using results of [31] on the entropy of PCSA we can calculate the limiting MVP of PCSA, LogLog, Curtain, and Fishmonger.

3.1 Martingale Estimators and Retrospective Variance

Consider an arbitrary sketch with state space $\mathcal{S}$. We assume the sketch state does not change upon seeing duplicated elements, hence it suffices to consider streams of *distinct* elements. We model the evolution of the sketch as a Markov chain $(S_k)_{k\geq 0} \in \mathcal{S}^*$, where S_k is the state after seeing k *distinct* elements. Define $P_k = \Pr(S_k \neq S_{k-1} \mid S_{k-1})$ to be the *state changing probability*, which depends only on S_{k-1}. In the dartboard terminology P_k is the total size of all unoccupied cells in S_{k-1}.

▶ **Definition 6.** *Let* $[\![\mathcal{E}]\!]$ *be the indicator variable for event* $\mathcal{E}$*. For any* $\lambda \geq 0$*, define:*

$$E_\lambda = \sum_{k=1}^{\lambda} [\![S_k \neq S_{k-1}]\!] \cdot \frac{1}{P_k}, \qquad \textit{the martingale estimator,}$$

$$\textit{and } V_\lambda = \sum_{k=1}^{\lambda} [\![S_k \neq S_{k-1}]\!] \cdot \frac{1-P_k}{P_k^2}, \qquad \textit{the "retrospective" variance.}$$

Note that $E_0 = V_0 = 0$*.*

The Martingale transform of this sketch stores $\hat{\lambda} = E_\lambda$ in one machine word and returns it as a cardinality estimate. It can also store V_λ in one machine word as well. Theorem 7 shows[8] that the retrospective variance V_λ is a good running estimate of the empirical squared error $(E_\lambda - \lambda)^2$.

▶ **Theorem 7.** *The martingale estimator* E_λ *is an unbiased estimator of* λ *and the retrospective variance* V_λ *is an unbiased estimator of* $\mathrm{Var}(E_\lambda)$*. Specifically, we have,*

$$\mathbb{E}(E_\lambda) = \lambda, \textit{ and } \mathrm{Var}(E_\lambda) = \mathbb{E}(V_\lambda) = \sum_{k=1}^{\lambda} \mathbb{E}\left(\frac{1}{P_k}\right) - \lambda.$$

▶ Remark 8. Theorem 7 contradicts Ting's claim [35], that V_λ is unbiased *only at "jump" times*, i.e., those λ for which $S_\lambda \neq S_{\lambda-1}$, and therefore inadequate to estimate the variance. In order to correct for this, Ting introduced a Bayesian method for estimating the time that has passed since the last jump time. The reason for thinking that jump times are different is actually quite natural. Suppose we record the list of *distinct* states $s_0, \ldots, s_k$ encountered while inserting λ elements, λ being unknown, and let p_i be the probability of changing from s_i to some other state. The amount of time spent in state s_i is a geometric random variable with mean p_i^{-1} and variance $(1-p_i)/p_i^2$. Furthermore, these waiting times are independent. Thus, $\sum_{i\in[0,k)} p_i^{-1}$ and $\sum_{i\in[0,k)}(1-p_i^{-1})/p_i^2$ are unbiased estimates of the cardinality λ' and squared error *upon entering state* s_k. These exactly correspond to E_λ and V_λ, but they *should* be biased since they do not take into account the $\lambda - \lambda'$ elements that had no effect on s_k. As Theorem 7 shows, this is a mathematical optical illusion. The history is a random variable, and although the last $\lambda - \lambda'$ elements did not change the state, *they could have*, which would have altered the observed history $s_0, \ldots, s_k$ and hence the estimates E_λ and V_λ.

3.2 Asymptotic Relative Variance

3.2.1 The ARV Factor

We consider classes of sketches composed of m *subsketches*, which controls the size and variance. In LogLog, PCSA, and Curtain these subsketches are the m columns. When considering a sketch with m subsketches, instead of using λ as the total number of insertions, we always use λ to denote the number of insertions *per subsketch* and therefore the total number of insertions is λm. We care about the *asymptotic relative variance* (ARV) as m and λ both go to infinity (defined below). A reasonable sketch should have relative variance $O(1/m)$. Informally, the ARV factor is just the leading constant of this expression.

[8] The proof can be found in the full version [32].

▶ **Definition 9** (ARV factor). *Consider a class of sketches whose size is parameterized by m. For any $k \geq 0$, define $P_{m,k}$ to be the probability the sketch changes state upon the kth insertion and $E_{m,k}$ the martingale estimator. The* ARV factor *of this class of sketches is defined as*

$$\lim_{\lambda\to\infty} \lim_{m\to\infty} m \cdot \frac{\mathrm{Var}(E_{m,\lambda m})}{(\lambda m)^2}. \tag{1}$$

3.2.2 Scale-Invariance and the Constant κ

Few sketches have *strictly* well-defined ARV factors. In Martingale LogLog, for example, the quantity $\left(\lim_{m\to\infty} m\frac{\mathrm{Var}(E_{m,\lambda m})}{(\lambda m)^2}\right)$ is not constant, but periodic in $\log_2 \lambda$; it does not converge as $\lambda \to \infty$. We explain how to fix this issue using *smoothing* in Section 3.2.3. *Scale-invariant* sketches must have well-defined ARV factors.

▶ **Definition 10** (scale-invariance and constant κ). *A combined sketch is* scale-invariant *if*

1. *For any λ, there exists a constant κ_λ such that $\lambda \cdot P_{m,\lambda m}$ converges to κ_λ almost surely as $m \to \infty$.*
2. *The limit of κ_λ as $\lambda \to \infty$ exists, and $\kappa \stackrel{\text{def}}{=} \lim_{\lambda\to\infty} \kappa_\lambda$.*

The constant of a sketch A is denoted as κ_A, where the subscript A is often dropped when the context is clear.

The next theorem proves that under mild regularity conditions, all scale-invariant sketches have well defined ARV factors and there is a direct relation between the ARV factor and the constant κ.

▶ **Theorem 11** (ARV factor of a scale-invariant sketch). *Consider a sketching scheme satisfying the following properties.*

1. *It is scale-invariant with constant κ.*
2. *For any $\lambda > 0$, the limit operator and the expectation operator of $\{\frac{1}{P_{m,\lambda m}}\}_m$ can be interchanged.*

Then the ARV factor of the sketch exists and equals $\frac{1}{2\kappa}$.

The constant κ together with Theorem 11 is useful in that it gives a simple and systematic way to evaluate the asymptotic performance of a well behaved (scale-invariant) sketch scheme.

MinCount [23, 9, 29] is an example of a scale-invariant sketch. The function $h(a) = (i, v) \in [m] \times [0, 1]$ is interpreted as a pair containing a bucket index and a real hash value. A (k, m)-MinCount sketch stores the smallest k hash values in each bucket.

▶ **Theorem 12.** *(k, m)-MinCount is scale-invariant and $\kappa_{(k,m)\text{-MinCount}} = k$.*

Proof. When a total of λm elements are inserted to the combined sketch, each subsketch receives $(1 + o(1))\lambda$ elements as $\lambda \to \infty$. Since we only care the asymptotic behavior, we assume for simplicity that each subsketch receives exactly λ elements.

Let $P_\lambda^{(i)}$ be the probability that the sketch of the ith bucket changes after the λth element is thrown into the ith bucket. Then by definition, we have

$$P_{m,\lambda m} = \frac{\sum_{i=1}^m P_\lambda^{(i)}}{m}.$$

Since all the subsketches are i.i.d., by the law of large numbers, $\lambda \cdot P_{m,\lambda} \to \lambda \cdot \mathbb{E}\left(P_\lambda^{(1)}\right)$ almost surely as $m \to \infty$.

Let X be the kth smallest hash value among λ uniformly random numbers in $[0,1]$, which distributes identically with $P_\lambda^{(1)}$. By standard order statistics, X is a Beta random variable $\mathrm{Beta}(k, \lambda - 1 + k)$ which has mean $\frac{k}{\lambda+1}$. Thus $\kappa_\lambda = \lambda \cdot \mathbb{E}(X) = \frac{k\lambda}{\lambda+1}$. We conclude that

$$\kappa = \lim_{\lambda\to\infty} \kappa_\lambda = \lim_{\lambda\to\infty} \frac{k\lambda}{\lambda+1} = k. \qquad \blacktriangleleft$$

Applying Theorem 11 to (k,m)-MinCount, we see its ARV is $\frac{1}{2km}$,[9] matching Cohen [14] and Ting [35]. Technically its MVP is unbounded since hash values were real numbers, but any realistic implementation would store them to $\log U$ bits of precision, for a total of $km \log U$ bits. Hence we regard its MVP to be $\frac{1}{2} \cdot \log_2 U$.

3.2.3 Smoothing Discrete Sketches

Sketches that partition the dartboard in some exponential fashion with base q (like LogLog, PCSA, and Curtain) have the property that their estimates and variance are periodic in $\log_q \lambda$. Pettie and Wang [31] proposed a simple method to *smooth* these sketches and make them truly scale-invariant as $m \to \infty$.

We assume that the dartboard is partitioned into m columns. The base-q *smoothing* operation uses an *offset vector* $\vec{r} = (r_0, \ldots, r_{m-1})$. We scale down all the cells in column i by the factor q^{-r_i}, then add a dummy cell spanning $[q^{-r_i}, 1)$ which is always occupied. (Phrased algorithmically, if a dart is destined for column i, we filter it out with probability $1 - q^{-r_i}$ and insert it into the sketch with probability q^{-r_i}.) When analyzing variants of (Hyper)LogLog and PCSA, we use the uniform offset vector $(0, 1/m, 2/m, \ldots, (m-1)/m)$. The Curtain sketch can be viewed as having a built-in offset vector of $(0, 1/2, 0, 1/2, 0, 1/2, \ldots)$ which effects the "sawtooth" cell partition. To smooth it, we use the offset vector[10]

$$(0,\ 1/2,\ 1/m,\ 1/2+1/m,\ 2/m,\ 1/2+2/m,\ \ldots,\ 1/2-1/m,\ 1-1/m).$$

As $m \to \infty$, $\vec{r}$ becomes uniformly dense in $[0,1]$.

The smoothing technique makes the empirical estimation more scale-invariant (see [31, Figs. 1& 2]) but also makes the sketch theoretically scale-invariant according to Definition 10. Thus, in the analysis, we will always assume the sketches are smoothed. However, in practice it is probably not necessary to do smoothing if $q < 3$.

In the next section, we will prove that *smoothed* q-LL, q-PCSA, and Curtain are all scale-invariant.

4 Analysis of Dartboard Based Sketches

Consider a dartboard cell that covers the vertical interval $[q^{-(t+1)}, q^{-t})$. We define the *height* of the cell to be t. In a smoothed cell partition, no two cells have the same height and all heights are of the form $t = j/m$, for some integer j. Thus, we may refer to it unambiguously as *cell* t. Note that cell t is an $m^{-1} \times \frac{1}{q^t}\frac{q-1}{q}$ rectangle.

[9] For simplicity, we assume the second condition of Theorem 4 holds for all the sketches analyzed in this paper.

[10] In [31], the smoothing was implemented via *random* offsetting, instead of the *uniform* offsetting. In Curtain we need to use uniform offsetting so that the offset values of columns are similar to their neighbors.

4.1 Poissonized Dartboard

Since we care about the asymptotic case where $\lambda \to \infty$, we model the process of "throwing darts" by a Poisson point process on the dart board (similar to the "poissonization" in the analysis of HyperLogLog [19]). Specifically, after throwing λm darts (events) to the dartboard, we assume the number of darts in cell t is a Poisson random variable with mean $\lambda \frac{1}{q^t}\frac{q-1}{q}$ and the number of darts in different cells are independent. For the poissonized dartboard, the range of height of cells naturally extend to the whole set of real numbers, instead of just having cells with positive height.

For any $t \in \mathbb{R}$, let $Y_{t,\lambda}$ be the indicator whether cell t contains at least one dart. Note that the probability that a Poisson random variable with mean λ' is zero is $e^{-\lambda'}$. Thus we have,

$$\Pr(Y_{t,\lambda} = 0) = e^{-\frac{\lambda}{q^t}\frac{q-1}{q}}.$$

Here, we note some simple identities for integrals that we will use frequently in the analysis.

▶ **Lemma 13.** *For any $q > 1$, we have*

$$\int \frac{1}{q^t} e^{-\frac{1}{q^t}}\, dt = \frac{1}{\ln q} e^{-\frac{1}{q^t}} + C.$$

Furthermore, let c_0, c_1 be any positive numbers, we have

$$\int_{-\infty}^{\infty} \frac{c_0}{q^t} e^{-\frac{c_1}{q^t}}\, dt = \frac{c_0}{c_1}\frac{1}{\ln q}.$$

4.2 The Constant κ

Let $Z_{t,\lambda}$ be the indicator of whether the cell t is *free*. Unlike $Y_{t,\lambda}$, $Z_{t,\lambda}$ depends on which sketching algorithm we are analyzing. Since the state changing probability is equal to the sum of the area of free cells, we have

$$P_{m,\lambda m} = \sum_{j=0}^{\infty} \frac{1}{m}\left(\frac{1}{q^{j/m}} - \frac{1}{q^{j/m+1}}\right) Z_{j/m,\lambda}. \tag{2}$$

If $P_{m,\lambda m}$ converges to κ_λ/λ almost surely as $m \to \infty$, then $\mathbb{E}(P_{m,\lambda m})$ also converges to κ_λ/λ as $m \to \infty$. Thus we have, from (2),

$$\begin{aligned}\kappa_\lambda/\lambda &= \lim_{m\to\infty} \mathbb{E}(P_{m,\lambda m}) = \lim_{m\to\infty} \sum_{j=0}^{\infty} \frac{1}{m}\left(\frac{1}{q^{j/m}} - \frac{1}{q^{j/m+1}}\right)\mathbb{E}(Z_{j/m,\lambda}) \\ &= \int_0^{\infty}\left(\frac{1}{q^t} - \frac{1}{q^{t+1}}\right)\mathbb{E}(Z_{t,\lambda})dt \approx \int_{-\infty}^{\infty}\left(\frac{1}{q^t} - \frac{1}{q^{t+1}}\right)\mathbb{E}(Z_{t,\lambda})dt, \end{aligned} \tag{3}$$

where we can extend the integration range to negative infinity without affecting the limit of κ_λ as $\lambda \to \infty$.[11] We conclude that

$$\kappa = \lim_{\lambda\to\infty} \kappa_\lambda = \lim_{\lambda\to\infty} \lambda \int_{-\infty}^{\infty}\left(\frac{1}{q^t} - \frac{1}{q^{t+1}}\right)\mathbb{E}(Z_{t,\lambda})dt. \tag{4}$$

The formula (4) is novel in the sense that, in order to evaluate κ, we now only need to understand the probability that $Z_{t,\lambda}$ is 1 for fixed t and λ.[12]

[11] See [32].

[12] Technically, to apply formula (4) one needs to first prove that the state changing probability $P_{m,\lambda m}$ converges almost surely to some constant κ_λ/λ for any λ, which is a mild regularity condition for any reasonable sketch. Thus in this paper we will assume the sketches in the analysis all satisfy this regularity condition and claim that a sketch is scale-invariant if formula (4) converges.

4.3 Analysis of Smoothed q-PCSA and q-LL

The sketches q-PCSA and q-LL are the natural smoothed base-q generalizations of PCSA [20] and LogLog [16].

▶ **Theorem 14.** *q-PCSA and q-LL are scale-invariant. In particular, we have,*

$$\kappa_{q\text{-PCSA}} = \frac{1}{\ln q}, \text{ and } \kappa_{q\text{-LL}} = \frac{1}{\ln q}\frac{q-1}{q}.$$

Proof. For q-LL, cell t is free iff both itself and all the cells above it in its column contain no darts. Thus we have

$$\mathbb{E}(Z_{t,\lambda}) = \prod_{i=0}^{\infty} \Pr(Y_{t+i,\lambda} = 0) = \prod_{i=0}^{\infty} e^{-\frac{\lambda}{q^{t+i}}\frac{q-1}{q}} = e^{-\frac{\lambda}{q^t}}.$$

Insert it to formula (4) and we get

$$\kappa_{q\text{-LL}} = \lim_{\lambda\to\infty} \lambda \int_{-\infty}^{\infty} \left(\frac{1}{q^t} - \frac{1}{q^{t+1}}\right) e^{-\frac{\lambda}{q^t}} dt = \frac{1}{\ln q}\frac{q-1}{q}.$$

For q-PCSA, cell t is free iff it has no dart. Thus $Z_{t,\lambda} = 1 - Y_{t,\lambda}$ and by formula (4) we have

$$\kappa_{q\text{-PCSA}} = \lim_{\lambda\to\infty} \lambda \int_{-\infty}^{\infty} \left(\frac{1}{q^t} - \frac{1}{q^{t+1}}\right) e^{-\frac{\lambda}{q^t}\frac{q-1}{q}} dt = \frac{1}{\ln q}. \quad ◀$$

The Fishmonger [31] sketch is based on a smoothed, entropy compressed version of base-e PCSA. The memory footprint of Fishmonger approaches its entropy as $m \to \infty$, which was calculated to be mH_0 [31, Lemma 4]. From Theorem 14, we know $\kappa_{e\text{-PCSA}} = 1$.

▶ **Corollary 15.** *Fishmonger has limiting MVP $H_0/2 \approx 1.63$.*

Proof. By Theorem 11, limiting MVP equals $mH_0 \cdot \frac{1}{2m} = \frac{H_0}{2}$. ◀

4.4 Asymptotic Local View

For any t and λ, since we want to evaluate $Z_{t,\lambda}$, whose value may depend on its "neighbors" on the dartboard, we need to understand the configurations of the cells near cell t. Since we consider the case where m goes to infinity, we may ignore the effect of smoothing to the cells in the immediate vicinity of cell t.

After taking these asymptotic approximations, we can index the cells near cell t as follows.

▶ **Definition 16** (neighbors of cell t)**.** *Fix a cell t. Let $i \in \mathbb{Z}$ and $c \in \mathbb{R}$. The (i, c)-neighbor of cell t is a cell whose column index differs by i (negative i means to the left, positive to the right) and has height $t + c$, it covers the vertical interval $[q^{-(t+c+1)}, q^{-(t+c)})$. In the sawtooth partition, c is an integer when i is even and a half-integer when i is odd. (Note that we are locally ignoring the effect of smoothing.)*

Once cell t is fixed, define $W(i, c)$ to be the indicator for whether the (i, c)-neighbor of cell t has at least one dart in it. Thus, for fixed t, λ, we have

$$\Pr(W(i,c) = 0) = \Pr(Y_{t+c,\lambda} = 0) = e^{-\frac{\lambda}{q^{t+c}}\frac{q-1}{q}}.$$

In the asymptotic local view, we lose the property that a cell can be uniquely identified by its height, hence the need to refer to nearby cells by their position *relative* to cell t.

4.5 Analysis of Curtain

We first briefly state some properties of curtain. For any $a \geq 1$, recall that $\mathcal{O}_a = \{-(a-1/2), -(a-3/2), \ldots, -1/2, 1/2, \ldots, a-3/2, a-1/2\}$. It is easy to see that for any vector $v = (g_0, g_1, \ldots, g_{m-1})$, $v_{\text{curt}} = (\hat{g}_i)$ can be expressed as

$$\hat{g}_i = \max_{j \in [0,m-1]} \{g_j - |i-j|(a-1/2)\}.$$

For each i, we define the *tension point* τ_i to be the lowest allowable value of $\hat{g}_i$, given the context of its neighboring columns.

$$\tau_i = \max_{j \in [0,m-1] \setminus \{i\}} \{g_j - |i-j|(a-1/2)\},$$

and thus we have $\hat{g}_i = \max(g_i, \tau_i)$. We see that the column i is *in tension* iff $g_i \leq \tau_i$, that is, $\hat{g}_i = \tau_i$.

▶ **Theorem 17.** *Curtain is scale-invariant with*

$$\kappa_{\textsf{Curtain}} = \frac{1}{\ln q} \frac{q-1}{q} \frac{1}{\frac{q-1}{q} + \frac{2}{q^h(q^{a-1/2}-1)} + \frac{1}{q^{h+1}}}.$$

Proof. Fix cell t and λ. Define $W_1(k)$ to be the height of the highest cell containing darts in the column k away from t's column. I.e., define $\iota = \left[\!\left[k \text{ is odd}\right]\!\right]/2$ to be $1/2$ if k is odd and zero if k is even, and $W_1(k) \stackrel{\text{def}}{=} \max\{t+i+\iota \mid i \in \mathbb{Z} \text{ and } W(k, i+\iota) = 1\}$.

We have for any $i \in \mathbb{Z}$,

$$\Pr(W_1(k) \leq t+i+\iota) = \prod_{j=1}^{\infty} \Pr(W(k, i+j+\iota) = 0) = e^{-\frac{\lambda}{q^{t+1+i+\iota}}}.$$

Let T_1 be the tension point of the column of cell t, which equals $\max_{j \in \mathbb{Z}\setminus\{0\}} \{W_1(j) - |j|(a-1/2)\}$. We have for any $i \in \mathbb{Z}$,

$$\begin{aligned}
\Pr(T_1 \leq i+t) &= \Pr\left(\max_{j \in \mathbb{Z}\setminus\{0\}} \{W_1(j) - |j|(a-1/2)\} \leq i+t\right) \\
&= \prod_{j \in \mathbb{Z}\setminus\{0\}} \Pr(W_1(j) - |j|(a-1/2) \leq i+t) \\
&= \left(\prod_{j=1}^{\infty} e^{-\lambda \frac{1}{q^{t+i+1+j(a-1/2)}}}\right)^2 = e^{-\lambda \frac{2}{q^{t+i+1}} \frac{1}{q^{a-1/2}-1}}.
\end{aligned}$$

From the rules of Curtain, we know that a cell is free iff it contains no dart, it is at most $h-1$ below its column's tension point, and at most h below the highest cell in its column containing darts. Thus,

$$Z_{t,\lambda} = \left[\!\left[Y_{t,\lambda} = 0\right]\!\right] \cdot \left[\!\left[t \geq T_1 - (h-1)\right]\!\right] \cdot \left[\!\left[t \geq W_1(0) - h\right]\!\right],$$

Note that T_1 is independent from $Y_{t,\lambda}$ and $W_1(0)$. In addition, $Y_{t,\lambda}$ is also independent from $\left[\!\left[t \geq W_1(0) - h\right]\!\right]$, since the latter only depends on $Y_{t',\lambda}$ with $t' \geq h+t+1$. Thus, we have

$$\begin{aligned}
\mathbb{E}(Z_{t,\lambda}) &= \Pr(Y_{t,\lambda} = 0) \cdot \Pr(T_1 \leq t+h-1) \cdot \Pr(W_1(0) \leq t+h) \\
&= e^{-\frac{\lambda}{q^t}\frac{q-1}{q}} e^{-\lambda \frac{2}{q^{t+h}} \frac{1}{q^{a-1/2}-1}} e^{-\frac{\lambda}{q^{t+h+1}}} \\
&= \exp\left(-\frac{\lambda}{q^t}\left(\frac{q-1}{q} + \frac{2}{q^h(q^{a-1/2}-1)} + \frac{1}{q^{h+1}}\right)\right).
\end{aligned}$$

Thus by formula (4), we have

$$\kappa_{\mathsf{Curtain}} = \lim_{\lambda\to\infty} \lambda \int_{-\infty}^{\infty} \left(\frac{1}{q^t} - \frac{1}{q^{t+1}}\right) \exp\left(-\frac{\lambda}{q^t}\left(\frac{q-1}{q} + \frac{2}{q^h(q^{a-1/2}-1)} + \frac{1}{q^{h+1}}\right)\right) dt$$
$$= \frac{1}{\ln q}\frac{q-1}{q}\frac{1}{\frac{q-1}{q} + \frac{2}{q^h(q^{a-1/2}-1)} + \frac{1}{q^{h+1}}}. \qquad \blacktriangleleft$$

5 Optimality of Martingale Fishmonger

Martingale sketches have several attractive properties, e.g., being strictly unbiased and insensitive to duplicate elements in the data stream. In Section 5.1 we argue that any sketch that satisfies these natural assumptions can be systematically transform into a Martingale X sketch with equal or lesser variance, where X is a dartboard sketch. In other words, the Martingale *transform* is optimal.

In Section 5.2 we prove that within the class of *linearizable* dartboard sketches, Martingale Fishmonger is optimal. The class of linearizable sketches is broad and includes state-of-the-art sketches, which lends strong *circumstantial* evidence that the memory-variance product of Martingale Fishmonger cannot be improved.

5.1 Optimality of the Martingale Transform

Consider a non-mergeable sketch processing a stream $\mathcal{A} = (a_1, a_2, \ldots)$. Let S_i be its state after seeing $(a_1, \ldots, a_i)$, $\lambda_i = |\{a_1, \ldots, a_i\}|$, and $\hat{\lambda}(S_i)$ be the estimate of cardinality λ_i when in state S_i. We make the following natural assumptions.

Randomness. The random oracle h is the only source of randomness. In particular, S_i is a function of $(h(a_1), h(a_2), \ldots, h(a_i))$.

Duplicates. If $a_i \in \{a_1, \ldots, a_{i-1}\}$, $S_i = S_{i-1}$, i.e., duplicates do not trigger state transitions.

Unbiasedness. Suppose one examines the data structure at time i and sees $S_i = \mathbf{s}_i$ and then examines it at time j. Then $\hat{\lambda}(S_j) - \hat{\lambda}(\mathbf{s}_i)$ is an unbiased estimate of $\lambda_j - \lambda_i$.

In the full version [32], we show that as a consequence of the *Randomness*, *Duplicates*, and *Unbiased* assumptions, the Martingale estimator has minimum variance.

▶ Remark 18. We should note that under some circumstances it is possible to achieve smaller variance by violating the *duplicates* and *unbiasedness* assumptions. For example, suppose the sketch state after seeing i elements were $(\hat{\lambda}_i, S_i, i)$. If the stream is duplicate-heavy, "i" carries no useful information, but if nearly all elements are distinct, i is also a good cardinality estimate. Since $\lambda_i \le i$, the cardinality estimate $\min\{\hat{\lambda}_i, i\}$ is never worse than $\hat{\lambda}_i$ alone, but when $\lambda_i \approx i$, it is biased and has a constant factor lower variance.

5.2 Optimality of Martingale Fishmonger

Given an abstract linearizable sketching scheme X, its space is minimized by compressing it to its entropy. On the other hand, by Theorem 11 the variance of Martingale X is controlled by the normalized expected probability of changing state: $2\lambda \cdot \mathbb{E}(P_\lambda)$. Theorem 19 lower bounds the ratio of these two quantities for any sketch that behaves well over a sufficiently large interval of cardinalities $\lambda \in [e^a, e^b]$. The proof technique is very similar to [31], as is the take-away message (that X=Fishmonger is optimal up to some assumptions). However, the two proofs are mathematically distinct as [31] focuses on Fisher information while Theorem 19 focuses on the *probability of state change*.

▶ **Theorem 19.** *Fix reals $a < b$ with $d = b - a > 1$. Let $\bar{H}, \bar{R} > 0$. For any linearizable sketch, let $H(\lambda)$ be the entropy of its state and P_λ be the probability of state change*[13] *at cardinality λ satisfies that*

1. *for all $\lambda > 0$, $H(\lambda) \le \bar{H}$, and*
2. *for all $\lambda \in [e^a, e^b]$, $2\lambda \mathbb{E}(P_\lambda) \ge \bar{R}$, then*

$$\frac{\bar{H}}{\bar{R}} \ge \frac{H_0}{2} \frac{1 - \max(8d^{-1/4}, 5e^{-d/2})}{1 + \frac{(344+4\sqrt{d})}{d} \frac{H_0}{I_0} \left(1 - \max(8d^{-1/4}, 5e^{-d/2})\right)} = \frac{H_0}{2}(1 - o_d(1)).$$

▶ **Corollary 20.** *The MVP of any linearizable and scale-invariant sketch is at least $\frac{H_0}{2}$.*

6 Experimental Validation

Throughout the paper we have maintained a possibly unhealthy devotion to asymptotic analysis, taking $m \to \infty$ whenever it was convenient. In practice m will be a constant, and possibly a smallish constant. How do the sketches perform in the pre-asymptotic region?

In turns out that the theoretical analysis predicts the performance of Martingale sketches pretty well, even whem m is small. In the experiment of Figure 3, we fixed the sketch size at a tiny 128 bits. Therefore HyperLogLog uses $m_1 = \lfloor 128/6 \rfloor = 21$ counters. The Martingale LogLog and Martingale Curtain sketches encode the martingale estimator with a floating point *approximation* of $\hat{\lambda}$ in 14 bits, with a 6-bit exponent and 8-bit mantissa. Thus, Martingale LogLog uses $m_2 = (128 - 14)/6 = 19$ counters, and Martingale Curtain uses $m_3 = 37$.[14]

For larger sketch sizes, the distribution of $\hat{\lambda}/\lambda$ is more symmetric, and closer to the predicted performance. Figure 4 gives the empirical distribution of $\hat{\lambda}/\lambda$ over 100,000 runs when $\lambda = 10^6$ and the sketch size is fixed at 1,200 bits. Here MartingaleCurtain uses $m = 400$, and both Martingale LogLog and HyperLogLog use $m = 200$. The experimental and predicted relative variances and standard errors are given in Table 2.

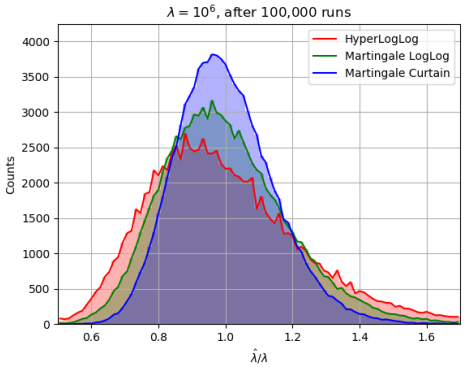


Figure 3 The sketch size is fixed at 128 bits.

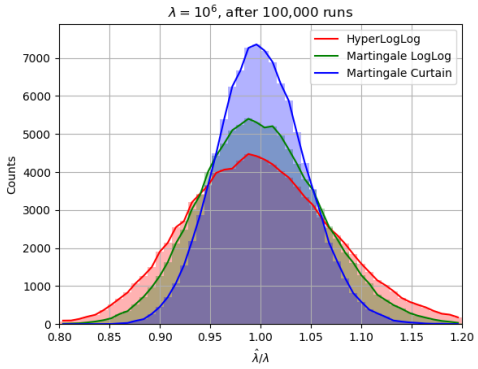


Figure 4 The sketch size is fixed at 1200 bits.

[13] The probability of state change P_λ is itself a random variable.
[14] It uses the optimal parameterization $(q, a, h) = (2.91, 2, 1)$ of Theorem 4.

Table 2 The relative variance is $\frac{1}{\lambda^2}\operatorname{Var}(\hat{\lambda} \mid \lambda)$ and standard error is $\frac{1}{\lambda}\sqrt{\operatorname{Var}(\hat{\lambda} \mid \lambda)}$. The predictions for Martingale LogLog and Martingale Curtain use Theorems 11, 14, and 17. The predictions for HyperLogLog are from Flajolet et al. [19, p. 139].

Sketch	Using 128 bits				Using 1200 bits			
	Experiment		Prediction		Experiment		Prediction	
	Var	StdErr	Var	StdErr	Var	StdErr	Var	StdErr
HyperLogLog	0.0573	23.94%	0.0549	23.44%	0.00541	7.36%	0.00539	7.35%
Martingale LogLog	0.0348	18.65%	0.0365	19.10%	0.00350	5.91%	0.00347	5.89%
Martingale Curtain	0.0211	14.54%	0.0208	14.43%	0.00189	4.35%	0.00193	4.39%

7 Conclusion

The Martingale transform is attractive due to its simplicity and low variance, but it results in *non*-mergeable sketches. We proved that under natural assumptions,[15] it generates optimal estimators automatically, allowing one to design structurally more complicated sketches, without having to worry about designing or analyzing *ad hoc* estimators. We proposed the Curtain sketch, in which each subsketch only needs a constant number of bits of memory, for *arbitrarily large* cardinality U.[16]

The analytic framework of Theorems 7 and 11 simplifies Cohen [14] and Ting [35], and gives a user-friendly formula for the asymptotic relative variance (ARV) of the Martingale estimator, as a function of the sketch's constant κ. We applied this framework to Martingale Curtain as well as the Martingale version of the classic sketches (MinCount, HLL and PCSA).

Assuming perfect compression, one gets the *memory-variance product* (MVP) of an sketch by multiplying its entropy and ARV. It is proved that for linearizable sketches, Fishmonger is optimal for mergeable sketches [31] (limiting MVP $= H_0/I_0 \approx 1.98$). In this paper we proved that in the sequential (non-mergeable) setting, if we restrict our attention to linearizable sketches, that Martingale Fishmonger is optimal, with limiting MVP $= H_0/2 \approx 1.63$ (Section 5.2). We conjecture that these two lower bounds hold for general, possibly *non*-linearizable sketches.

References

1 Noga Alon, Phillip B. Gibbons, Yossi Matias, and Mario Szegedy. Tracking join and self-join sizes in limited storage. In *Proceedings 18th ACM Symposium on Principles of Database Systems (PODS)*, pages 10–20, 1999. doi:10.1145/303976.303978.

2 Daniel N Baker and Ben Langmead. Dashing: Fast and accurate genomic distances with hyperloglog. *bioRxiv*, 2019. doi:10.1101/501726.

3 Ziv Bar-Yossef, T. S. Jayram, Ravi Kumar, D. Sivakumar, and Luca Trevisan. Counting distinct elements in a data stream. In *Proceedings 6th International Workshop on Randomization and Approximation Techniques (RANDOM)*, volume 2483 of *Lecture Notes in Computer Science*, pages 1–10, 2002. doi:10.1007/3-540-45726-7_1.

4 Ziv Bar-Yossef, Ravi Kumar, and D. Sivakumar. Reductions in streaming algorithms, with an application to counting triangles in graphs. In *Proceedings 13th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 623–632, 2002.

[15] insensitivity to duplicates, and unbiasedness

[16] Note that an $O(\log \log U)$-bit offset register is needed for the whole sketch.

5 Ran Ben-Basat, Gil Einziger, Shir Landau Feibish, Jalil Moraney, and Danny Raz. Network-wide routing-oblivious heavy hitters. In *Proceedings of the 2018 Symposium on Architectures for Networking and Communications Systems (ANCS)*, pages 66–73, 2018. doi:10.1145/3230718.3230729.

6 Jarosław Błasiok. Optimal streaming and tracking distinct elements with high probability. *ACM Trans. Algorithms*, 16(1):3:1–3:28, 2020. doi:10.1145/3309193.

7 Andrei Z. Broder. On the resemblance and containment of documents. In *Proceedings of Compression and Complexity of SEQUENCES*, pages 21–29, 1997. doi:10.1109/SEQUEN.1997.666900.

8 Thilina Buddhika, Matthew Malensek, Sangmi Lee Pallickara, and Shrideep Pallickara. Synopsis: A distributed sketch over voluminous spatiotemporal observational streams. *IEEE Trans. Knowl. Data Eng.*, 29(11):2552–2566, 2017. doi:10.1109/TKDE.2017.2734661.

9 Philippe Chassaing and Lucas Gerin. Efficient estimation of the cardinality of large data sets. In *Proceedings of the 4th Colloquium on Mathematics and Computer Science Algorithms, Trees, Combinatorics and Probabilities*, 2006.

10 Aiyou Chen, Jin Cao, Larry Shepp, and Tuan Nguyen. Distinct counting with a self-learning bitmap. *Journal of the American Statistical Association*, 106(495):879–890, 2011. doi:10.1198/jasa.2011.ap10217.

11 Min Chen, Shigang Chen, and Zhiping Cai. Counter tree: A scalable counter architecture for per-flow traffic measurement. *IEEE/ACM Trans. Netw.*, 25(2):1249–1262, 2017. doi:10.1109/TNET.2016.2621159.

12 Pern Hui Chia, Damien Desfontaines, Irippuge Milinda Perera, Daniel Simmons-Marengo, Chao Li, Wei-Yen Day, Qiushi Wang, and Miguel Guevara. KHyperLogLog: Estimating reidentifiability and joinability of large data at scale. In *Proceedings of the 2019 IEEE Symposium on Security and Privacy*, pages 350–364, 2019. doi:10.1109/SP.2019.00046.

13 Edith Cohen. Size-estimation framework with applications to transitive closure and reachability. *J. Comput. Syst. Sci.*, 55(3):441–453, 1997. doi:10.1006/jcss.1997.1534.

14 Edith Cohen. All-distances sketches, revisited: HIP estimators for massive graphs analysis. *IEEE Trans. Knowl. Data Eng.*, 27(9):2320–2334, 2015. doi:10.1109/TKDE.2015.2411606.

15 Edith Cohen and Haim Kaplan. Tighter estimation using bottom k sketches. *Proc. VLDB Endow.*, 1(1):213–224, 2008. doi:10.14778/1453856.1453884.

16 Marianne Durand and Philippe Flajolet. Loglog counting of large cardinalities. In *Proceedings 11th Annual European Symposium on Algorithms (ESA)*, volume 2832 of *Lecture Notes in Computer Science*, pages 605–617. Springer, 2003. doi:10.1007/978-3-540-39658-1_55.

17 R. A. Leo Elworth, Qi Wang, Pavan K. Kota, C. J. Barberan, Benjamin Coleman, Advait Balaji, Gaurav Gupta, Richard G. Baraniuk, Anshumali Shrivastava, and Todd J. Treangen. To petabytes and beyond: recent advances in probabilistic and signal processing algorithms and their application to metagenomics. *Nucleic Acids Research*, 48(10):5217–5234, 2020. doi:10.1093/nar/gkaa265.

18 Cristian Estan, George Varghese, and Michael E. Fisk. Bitmap algorithms for counting active flows on high-speed links. *IEEE/ACM Trans. Netw.*, 14(5):925–937, 2006. doi:10.1145/1217709.

19 Philippe Flajolet, Éric Fusy, Olivier Gandouet, and Frédéric Meunier. HyperLogLog: the analysis of a near-optimal cardinality estimation algorithm. In *Proceedings of the 18th International Meeting on Probabilistic, Combinatorial, and Asymptotic Methods for the Analysis of Algorithms (AofA)*, 2007.

20 Philippe Flajolet and G. Nigel Martin. Probabilistic counting algorithms for data base applications. *J. Comput. Syst. Sci.*, 31(2):182–209, 1985. doi:10.1016/0022-0000(85)90041-8.

21 Michael J. Freitag and Thomas Neumann. Every row counts: Combining sketches and sampling for accurate group-by result estimates. In *Proceedings of the 9th Biennial Conference on Innovative Data Systems Research (CIDR)*, 2019.

22 Phillip B. Gibbons and Srikanta Tirthapura. Estimating simple functions on the union of data streams. In *Proceedings 13th Annual ACM Symposium on Parallel Algorithms and Architectures (SPAA)*, pages 281–291, 2001. doi:10.1145/378580.378687.

23 Frédéric Giroire. Order statistics and estimating cardinalities of massive data sets. *Discret. Appl. Math.*, 157(2):406–427, 2009. doi:10.1016/j.dam.2008.06.020.

24 Ahmed Helmi, Jérémie Lumbroso, Conrado Martínez, and Alfredo Viola. Data Streams as Random Permutations: the Distinct Element Problem. In *Proceedings of the 23rd International Meeting on Probabilistic, Combinatorial, and Asymptotic Methods for the Analysis of Algorithms (AofA)*, pages 323–338, 2012.

25 Piotr Indyk and David P. Woodruff. Tight lower bounds for the distinct elements problem. In *Proceedings 44th IEEE Symposium on Foundations of Computer Science (FOCS), October 2003, Cambridge, MA, USA, Proceedings*, pages 283–288, 2003. doi:10.1109/SFCS.2003.1238202.

26 T. S. Jayram and David P. Woodruff. Optimal bounds for Johnson-Lindenstrauss transforms and streaming problems with subconstant error. *ACM Trans. Algorithms*, 9(3):26:1–26:17, 2013. doi:10.1145/2483699.2483706.

27 Daniel M. Kane, Jelani Nelson, and David P. Woodruff. An optimal algorithm for the distinct elements problem. In *Proceedings 29th ACM Symposium on Principles of Database Systems (PODS)*, pages 41–52, 2010. doi:10.1145/1807085.1807094.

28 Kevin J. Lang. Back to the future: an even more nearly optimal cardinality estimation algorithm. *CoRR*, abs/1708.06839, 2017. arXiv:1708.06839.

29 Jérémie Lumbroso. An optimal cardinality estimation algorithm based on order statistics and its full analysis. In *Proceedings of the 21st International Meeting on Probabilistic, Combinatorial, and Asymptotic Methods in the Analysis of Algorithms (AofA)*, pages 489–504, 2010.

30 Guillaume Marçais, Brad Solomon, Rob Patro, and Carl Kingsford. Sketching and sublinear data structures in genomics. *Annual Review of Biomedical Data Science*, 2(1):93–118, 2019. doi:10.1146/annurev-biodatasci-072018-021156.

31 Seth Pettie and Dingyu Wang. Information theoretic limits of cardinality estimation: Fisher meets Shannon. In *Proceedings 53rd ACM Symposium on Theory of Computing (STOC)*, 2021.

32 Seth Pettie, Dingyu Wang, and Longhui Yin. Non-mergeable sketching for cardinality estimation, 2021. arXiv:2008.08739.

33 Ninh Pham. Hybrid LSH: faster near neighbors reporting in high-dimensional space. In *Proceedings of the 20th International Conference on Extending Database Technology (EDBT)*, pages 454–457, 2017. doi:10.5441/002/edbt.2017.43.

34 Björn Scheuermann and Martin Mauve. Near-optimal compression of probabilistic counting sketches for networking applications. In *Proceedings of the 4th International Workshop on Foundations of Mobile Computing (DIALM-POMC)*, 2007.

35 Daniel Ting. Streamed approximate counting of distinct elements: beating optimal batch methods. In *Proceedings 20th ACM Conference on Knowledge Discovery and Data Mining (KDD)*, pages 442–451, 2014. doi:10.1145/2623330.2623669.

36 Daniel Ting. Towards optimal cardinality estimation of unions and intersections with sketches. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*, pages 1195–1204. ACM, 2016. doi:10.1145/2939672.2939772.

37 Jake Wires, Stephen Ingram, Zachary Drudi, Nicholas J. A. Harvey, and Andrew Warfield. Characterizing storage workloads with counter stacks. In *Proceedings of the 11th USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, pages 335–349, 2014.

38 Derrick E. Wood, Jennifer Lu, and Ben Langmead. Improved metagenomic analysis with Kraken 2. *bioRxiv*, 2019. doi:10.1101/762302.

39 Qingjun Xiao, Shigang Chen, You Zhou, Min Chen, Junzhou Luo, Tengli Li, and Yibei Ling. Cardinality estimation for elephant flows: A compact solution based on virtual register sharing. *IEEE/ACM Trans. Netw.*, 25(6):3738–3752, 2017. doi:10.1109/TNET.2017.2753842.

The Structure of Minimum Vertex Cuts

Seth Pettie
University of Michigan, Ann Arbor, MI, USA

Longhui Yin
Tsinghua University, Beijing, China

Abstract

In this paper we continue a long line of work on representing the *cut structure* of graphs. We classify the types of minimum *vertex* cuts, and the possible relationships between multiple minimum vertex cuts.

As a consequence of these investigations, we exhibit a simple $O(\kappa n)$-space data structure that can quickly answer pairwise $(\kappa+1)$-connectivity queries in a κ-connected graph. We also show how to compute the "closest" κ-cut to every vertex in near linear $\tilde{O}(m + \text{poly}(\kappa)n)$ time.

2012 ACM Subject Classification Mathematics of computing → Paths and connectivity problems; Theory of computation → Data structures design and analysis

Keywords and phrases Graph theory, vertex connectivity, data structures

Digital Object Identifier 10.4230/LIPIcs.ICALP.2021.105

Category Track A: Algorithms, Complexity and Games

Related Version *Full Version*: `https://arxiv.org/abs/2102.06805`

Funding *Seth Pettie*: This work was supported by NSF grants CCF-1637546 and CCF-1815316. *Longhui Yin*: Supported by a grant from Tsinghua University.

1 Introduction

One of the strong themes running through graph theory is to understand the *cut structure* of graphs and to apply these structural theorems to solve algorithmic and data structural problems. Consider the following exemplars of this line of work:

Gomory-Hu Tree. Gomory and Hu (1961) [31] proved that any weighted, undirected graph $G = (V, E)$ can be replaced by a weighted, undirected tree $T = (V, E_T)$ such that for every $s, t \in V$, the minimum s-t cut partition in T (removing a single edge, partitioning V into two sets) corresponds to a minimum s-t cut partition in G. These are sometimes called *cut-equivalent trees* [1].

Cactus Representations. Dinitz, Karzanov, and Lomonosov (1976) [13] proved that all the *global* minimum edge-cuts of any weighted, undirected graph $G = (V, E)$ could be succinctly encoded as an (unweighted) *cactus graph*. A cactus is a connected multigraph in which every edge participates in exactly one cycle. It was proved that there exists a cactus $C = (V_C, E_C)$ and an embedding $\phi : V \to V_C$ such that the minimum edge-cuts in C (2 edges in a common cycle) are in 1-1 correspondence with the minimum edge-cuts of G. A corollary of this theorem is that there are at most $\binom{n}{2}$ minimum edge-cuts.

Picard-Queyrenne Representation. In a *directed* s-t flow network there can be exponentially many min s-t cuts. Picard and Queyrenne (1980) [56] proved that the family $\mathcal{S} = \{S \mid (S, \overline{S})$ is a min s-$t\}$ corresponds 1-1 with the downward-closed sets of a partial order, and is therefore closed under union and intersection.

Block Trees, SPQR Trees, and Beyond. Whitney (1932) [61, 62] proved that the cut vertices (articulation points) of an undirected graph $G = (V, E)$ partition E into single edges and 2-edge connected components (blocks). This yields the *block tree* representation. Di

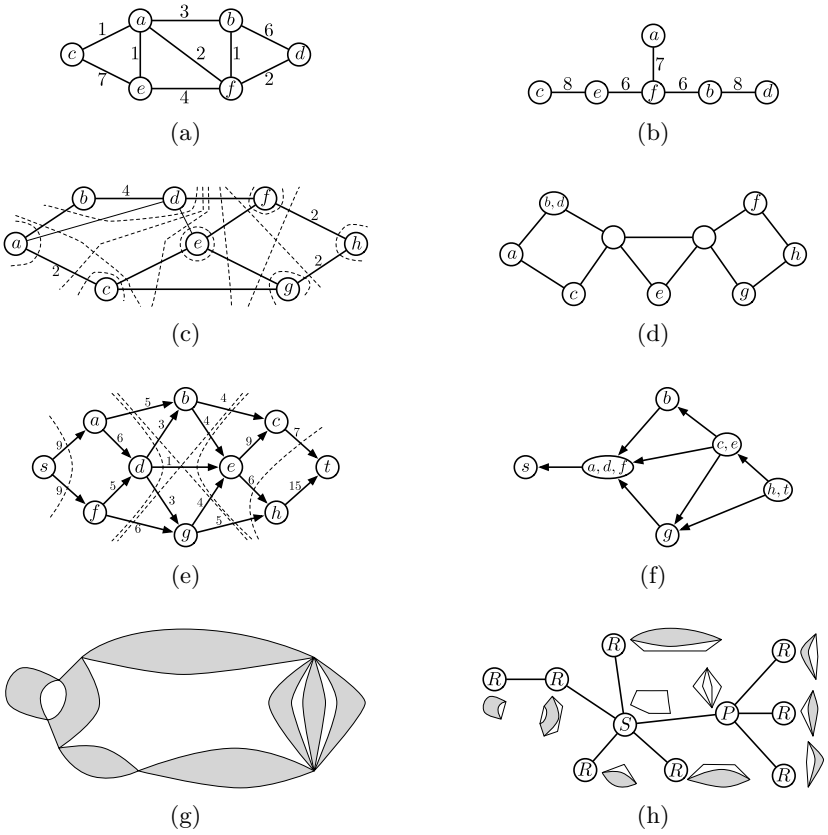


Figure 1 (a) A weighted undirected graph; (b) Its Gomory-Hu (cut-equivalent) tree [31]. (c) A weighted undirected graph (unmarked edges have unit weight); (d) the Cactus representation [13] of its minimum edge cuts. (e) A directed s-t flow network; (f) A dag whose downward-closed sets (that include s but not t) correspond to min s-t cuts (Picard-Queyrenne [56]). (g) An abstract representation of a 2-connected graph; (h) The representation of its 3-connected components as an SPQR tree (Di Battista-Tamassia [4]).

Battista and Tamassia (1989) [5, 4] formally defined the *SPQR tree*, which succinctly encodes all 2-vertex cuts in a biconnected graph, and Kanevsky, Tamassia, Di Battista, and Chen [42] extended this structure to represent 3-vertex cuts in a triconnected graph.[1]

It is natural to ask how, and to what extent, these structures can be extended and generalized. Gusfield and Naor [34] combined the Gomory-Hu tree and the Picard-Queyrenne representation for edge-connetivity. They [33] also described an analogue of Gomory-Hu trees (cut-equivalent trees) for vertex connectivity, i.e., a tree that compactly represents a minimum s-t *vertex* cut for every $s, t \in V$. It used a result of Schnorr [57] on an analogue of Gomory-Hu trees for "roundtrip" flow-values in directed networks. These claims were refuted by Benczur [6], who illustrated that Schnorr's and Gusfield and Naor's proofs were incorrect and could not be rectified. In particular, there are no *cut-equivalent* trees for s-t vertex connectivity and directed s-t cuts. Benczur [6, p. 505-506] suggested a way to construct a *flow-equivalent* tree for vertex connectivity (vertex capacitated s-t flows) using a result of

[1] Many of the structural insights behind [4, 42] were latent in prior work. See, for example. Mac Lane [48], Tutte [59, 60] (1961-6), Hopcroft and Tarjan [36], and Cunningham and Edmonds [12].

Cheng and Hu [9]. This, too, turned out to be incorrect. Hassin and Levin [35] proved that a graph can have $\Omega(n^2)$ s-t vertex-capacitated cut values, which cannot be captured by a flow-equivalent tree. We take these episodes as a reminder that having published proofs (*even incorrect ones*) is essential for facilitating self-correction in science.

The inspiration for this paper is an extended abstract of Cohen, Di Battista, Kanevsky, and Tamassia [11] from STOC 1993. Their goal was to find a cactus-analogue for global minimum vertex cuts, or from a different perspective, to extend SPQR trees [4] and [42] from $\kappa \in \{2, 3\}$ vertex cuts to arbitrarily large κ. As an application of their ideas, they described a data structure for κ-connected graphs occupying space $O(\kappa^3 n)$ that, given u, v, decided whether u, v are separated by a κ-cut or $(\kappa + 1)$-connected. There are no suspect claims in [11]. On the other hand, the paper is 7 pages and leaves many of its central claims unproven.[2] We believe that understanding the *structure* of minimum vertex cuts is a fundamental problem in graph theory, and deserving of a complete, formal treatment.

In this paper we investigate the structure of the set of all minimum vertex cuts and classify the relationships between different minimum vertex cuts. Our work reveals some structural features of minimum κ-cuts not evident in Cohen, Di Battista, Kanevsky, and Tamassia [11], and ultimately allows us to develop a simpler data structure to answer pairwise κ-cut queries in a κ-connected graph. It occupies (optimal) $O(\kappa n)$ space and can be constructed in randomized $\tilde{O}(m + \text{poly}(\kappa)n)$ time, in contrast to [11], which occupies $O(\kappa^3 n)$ space and is constructed in $\exp(\kappa)n^5$ time.[3]

1.1 Related Work

Dinitz and Vainshtein [17, 18] combined elements of the cactus [13] and Picard-Queyrenne [56] representations, which they called the *connectivity carcass*. Given an undirected, unweighted $G = (V, E)$ and $S \subseteq V$ of terminals, λ_S is the size of the minimum edge-cut that separates S. The carcass represents *all* size-λ_S separating cuts in $O(\min\{m, \lambda_S n\})$ space and answers various cut queries in $O(1)$ time.[4]

Benczur and Goemans [7] generalized the cactus representation [13] in a different direction, by giving a compact representation of all cuts that are within a factor 6/5 of the global minimum edge-cut.

Dinitz and Nutov [14] generalized the cactus representation [13] in another direction, by giving an $O(n)$-space representation of all λ and $\lambda + 1$ edge cuts, where λ is the edge-connectivity of the undirected, unweighted graph. Another feature of representations in [17, 18, 14] worth to mention is that they answer connectivity queries with supporting edge insertions in the graph. Unpublished manuscripts [15, 16] give detailed treatments of the λ odd and λ even cases separately.

Georgiadis et al. [30, 22, 28, 29] investigated various notions of 1- and 2-edge and vertex connectivity in *directed* graphs, and the compact representation of edge/vertex cuts.

Gabow [25] provided a $O(m \log n^2/m)$ data structure for all mincuts of a directed graph by drawing a correspondence between cuts and intersecting set families.

---

[2] The full version of this paper was never written (personal communication with R. Tamassia, 2011, and R. Di Battista, 2016).

[3] The algorithm enumerates *all* minimum κ-cuts, which can be as large as $\Omega(2^\kappa(n/\kappa)^2)$; modern vertex connectivity algorithms [23, 27, 26] may reduce the exponent of n in the running time.

[4] The carcass was introduced in extended abstracts [17, 18] and the (simpler) case of odd λ_S was analyzed in detail in a journal article [19]. We are not aware of a full treatment of the case when λ_S is even.

Granot and Hassin [32] generalized the Gomory-Hu tree into node- and arc-capacitated case and gave an algorithm for finding a cut-tree over a set of terminals K by solving $|K|-1$ minimum-cut problems.

Sparsification

One general way to compactly represent connectivity information is to produce a *sparse* graph with the same cut structure. Nagamochi and Ibaraki [50] proved that every unweighted, undirected graph $G = (V, E)$ contains a subgraph $H = (V, E_H)$ with $|E_H| < (k+1)n$ (arboricity $k+1$) such that H is computable in $O(m)$ time and contains exactly the same k'-vertex cuts and k'-edge cuts as G, for all $k' \in \{1, \ldots, k\}$. Benczur and Karger [8] proved that for any capacitated, undirected graph $G = (V, E)$, there is another capacitated graph $H = (V, E_H)$ with $|E_H| = O(\epsilon^{-2} n \log n)$ such that the capacity of *every* cut in G is preserved in H up to a $(1 \pm \epsilon)$-factor. This bound was later improved to $O(\epsilon^{-2} n)$ by Batson, Spielman, and Srivastava [3], which is optimal.

In directed graphs, Baswana, Choudhary, and Roditty [2] considered the problem of finding a sparse subgraph that preserves reachability from a single source, even if d vertices are deleted. They proved that $\Theta(2^d n)$ edges are necessary and sufficient for $d \in [1, \log n]$.

d-Failure Connectivity

An undirected graph can be compactly represented such that connectivity queries can be answered after the deletion of any d vertices/edges (where d could be much larger than the underlying connectivity of the graph). Improving on [54, 43, 20], Duan and Pettie [21] proved that d vertex failures could be processed in $\tilde{O}(d^2)$ time such that connectivity queries are answered in $O(d)$ time, and d edge failures could be processed in $O(d \log d \log \log n)$ time such that connectivity queries are answered in $O(\log \log n)$ time. The size of the [21] structure is $\tilde{O}(m)$ for vertex failures and $\tilde{O}(n)$ for edge failures. Choudhary [10] gave an optimal $O(n)$-space data structure that could answer directed reachability queries after $d \in \{1, 2\}$ vertex or edge failures.

Labeling Schemes

Benczur's refutation [6] of [57, 33] shows that all pairwise vertex connectivities cannot be captured in a *tree* structure, but it does not preclude other representations of this information. Hsu and Lu [37] designed a $O(k \log n)$-bit labeling scheme to determine whether $\kappa(u, v) \geq k$, given just the labels of u and v. This improved [45] and matched an $\Omega(k \log n)$-bit lower bound of Katz, Katz, Korman, and Peleg [44]. By applying it to all $k \in \{1, \ldots, \overline{\kappa}\}$, the Hsu-Lu labeling has size $O(\overline{\kappa}^2 \log n)$ and reports $\min\{\kappa(u, v), \overline{\kappa}\}$. Using a different approach, Izsak and Nutov [38] gave a $O(\overline{\kappa} \log^3 n)$-bit labeling scheme for computing $\min\{\kappa(u, v), \overline{\kappa}\}$. The schemes [37, 45, 44, 38] have large polynomial construction times, and cannot report a u-v cut of size $\kappa(u, v)$

Vertex Connectivity Algorithms

In optimal linear time we can decide whether the connectivity of a graph is $\kappa = 1, \kappa = 2$, or $\kappa \geq 3$ [58, 36]. For larger κ, the state-of-the-art in vertex connectivity has been improved substantially in the last few years. Forster, Nanongkai, Yang, Saranurak, and Yingchareonthawornchai [23] gave a *Monte Carlo* algorithm for computing the vertex connectivity κ of an undirected graph in $\tilde{O}(m + n\kappa^3)$ time, w.h.p.[5] A new result of Li, Nanongkai,

[5] The algorithm does not produce a witness, and hence may err with small probability.

Panigrahi, Saranurak, and Yingchareonthawornchai [46] gave a randomized algorithm running in $O(m^{4/3+o(1)})$ time. The best deterministic algorithm, due to Gao, Li, Nanongkai, Peng, Saranurak, and Yingchareonthawornchai [27], computes the connectivity $\kappa < n^{1/8}$ in $O((m+n^{7/4}\kappa^{O(\kappa)})n^{o(1)})$ time or $O((m+n^{19/20}\kappa^{5/2})n^{o(1)})$ time. For $\kappa > n^{1/8}$, Gabow's algorithm [26] runs in $O(\kappa n^2 + \kappa^2 n \cdot \min\{n^{3/4}, \kappa^{3/2}\})$ time.

The connectivity augmentation problem is to *improve* the global vertex connectivity or specific pairwise connectivities by adding few edges. See, for example, Frank and Jordán [24], Jordán [40], Jackson and Jordán [39], and Nutov [52] for positive results, and Nutov [51] for a hardness of approximation result.

1.2 Organization

In Section 2 we review basic definitions and lemmas regarding vertex cuts. Section 3 gives the basic classification theorem for minimum vertex cuts, and lists some useful corollaries. In short, every pair of cuts have *laminar*, *wheel*, *crossing matching*, or *small* relation. Sections 3.1–3.4 analyze these four categories in more detail. Section 4 exhibits a new $O(\kappa n)$-space[6] data structure that, given two vertices, answers $(\kappa+1)$-connectivity queries in $O(1)$ time, and produces a separating κ-cut (if one exists) in $O(\kappa)$ time. We conclude with some remarks and open problems in Section 5. All missing proofs appear in the full version of the paper [55].

2 Preliminaries

The input is a simple, connected, undirected graph $G=(V,E)$ with $n=|V|$ and $m=|E|$.

Let the subgraph of G induced by A be denoted $G|_A$. We call $U \subset V$ a *cut* if the graph $G|_{V\setminus U}$ is disconnected. A *side* of the cut U is a connected component of $G|_{V\setminus U}$. If P is a side of U and $A \subseteq P$, we say A is *within a side of* U, and let $\mathrm{Side}_U(A) = P$ denote the side containing A. A *region* of a cut U is a side, or the union of several sides of U. Denote $\mathrm{Region}_U(A)$ as the region containing the sides of U that intersects with A. [7] We say a cut *disconnects* or *separates* A and B if they are in distinct sides of U. In particular, if $B = V\setminus(A\cup U)$, we say U *disconnects* or *separates* B *from the rest of the graph*.

A path $\pi = v_1 v_2 \cdots v_l$ is *from* A *to* B, if $v_1 \in A$ and $v_l \in B$. Two paths π, π' from v_1 to v_l are *internally* vertex disjoint if they have no common vertices, except for v_1, v_l. We say U *blocks* π if $U \cap \{v_2, \ldots, v_{l-1}\} \neq \emptyset$.

A k-*cut* is a cut of size k. Define $\kappa(u,v)$ to be the minimum k such that there exists a k-cut separating u and v, where $\{u,v\} \neq E(G)$. Define $\kappa = \kappa(G)$ to be the minimum of $\kappa(u,v)$ over all pairs $\{u,v\} \in \binom{V(G)}{2}\setminus E(G)$. We say G is k-*connected* if $\kappa(G) \geq k$.

In this paper we assume that $\kappa < n/4$ and consider the set of *all* (minimum) κ-cuts.

▶ Remark 1. There is some flexibility in defining the corner cases. Some authors leave $\kappa(u,v)$ undefined when $\{u,v\} \in E(G)$ or define it to be $n-1$. Other authors define connectivity as the maximal number of vertex-disjoint paths. In [11] a k-cut is defined to be a mixed set of edges and vertices whose removal disconnects the graph. Under this definition, when $\{u,v\} \in E(G)$, $\kappa(u,v) = k$ if removing $k-1$ vertices and $\{u,v\}$ disconnects u and v. The last two definitions are equivalent and are compatible with Menger's theorem.

▶ **Theorem 2** (Menger [49]). *Let $G=(V,E)$ be an undirected graph and $\{u,v\}$ a pair* not *in E. Let $U \subset V$ be a minimum size cut disconnecting u and v and Π be a maximum size set of internally vertex disjoint paths from u to v. Then $\kappa(u,v) = |U| = |\Pi|$.*

[6] Formally speaking, this is $O(\kappa n)$ words of space, where a word store the index of a vertex, and takes up $O(\log n)$ bits of space.

[7] Note when A is a singleton set $\{u\}$, $\mathrm{Region}_U(A) = \mathrm{Side}_U(A)$.

The following categories make sense when applied to *non*-minimal vertex cuts, but we are only interested in applying them to *minimum* vertex cuts. Henceforth *cut* usually means *minimum cut*.

Laminar Cuts

Let U be a cut and P be a side of U. If W is a cut and $W \subset U \cup P$, we say W is a *laminar cut* of U in side P.[8]

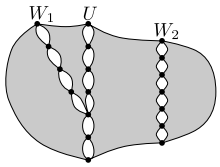


Figure 2 A 7-cut U with two sides, and two 7-cuts W_1, W_2 that are laminar w.r.t. U.

Small Cuts

Informally, when a side of a cut is tiny we call the cut *small*. We define three levels of small cuts. Let U be a cut with sides $A_1, A_2, \ldots, A_a$. We say that

1° U is (I, t)-*small* if there exists an index $i^\sharp$ such that $\sum_{i \neq i^\sharp} |A_i| \leq t$. $A_{i^\sharp}$ is called the *large side* of U and the others the *small sides* of U.

2° U is (II, t)-*small* if there exists $i^\sharp$ such that for every $i \neq i^\sharp$, $|A_i| \leq t$.

3° U is (III, t)-*small*, if there exists $i^\sharp$ such that $|A_{i^\sharp}| \leq t$. In this case $A_{i^\sharp}$ is the *small side* of U.

Note that for any t, I-small cuts are II-small, and II-small cuts are III-small. We typically apply this definition with $t = \kappa$, $t = \Theta(\kappa)$, or $t = \lceil \frac{n-\kappa}{2} \rceil$.

Wheel Cuts

Suppose V can be partitioned into a series of disjoint sets T, $\{C_i\}, \{S_i\}$ ($1 \leq i \leq w, w \geq 4$, subscripts are taken module w), such that the $\{C_i\}$ and $\{S_i\}$ are nonempty (T may be empty), and $C_i \cup T \cup C_{i+2}$ disconnects $S_i \cup C_i \cup S_{i+1}$ from the rest of the graph. We say $(T; C_1, C_2, \ldots, C_w)$ forms a *w-wheel* with *sectors* $S_1, S_2, \ldots, S_w$. We call T the *center* of the wheel, $\{C_i\}$ the *spokes* of the wheel, and $C(i,j) = C_i \cup T \cup C_j$ the *cuts* of the wheel. Define $D(i,j) = S_i \cup C_{i+1} \cup \cdots \cup C_{j-1} \cup S_{j-1}$.

Recall that we are only interested in wheels whose cuts are minimum κ-cuts. The cut of the wheels discussed in this paper are all κ-cuts. It is proved in Lemma 10 that, if $(T; C_1, C_2, \ldots, C_w)$ forms a wheel, then for every i, j such that $j - i \notin \{1, w-1\}$, $C(i,j)$ is a κ-cut with exactly two sides, namely $D(i,j)$ and $D(j,i)$. Note that a w-wheel $(T; C_1, C_2, \ldots, C_w)$ contains x-wheels, $x \in [4, w-1]$. Specifically, for *any* subset $\{i_1, i_2, \ldots, i_x\} \subseteq \{1, 2, \ldots, w\}$ with $x \geq 4$, $(T; C_{i_1}, C_{i_2}, \ldots, C_{i_x})$ forms an x-wheel called a *subwheel* of the original. If a wheel is not a subwheel of any other wheel, it is a *maximal wheel*. If there exists an index $i^\sharp$ such that, $\sum_{i \neq i^\sharp} |S_i| \leq \kappa$, then we say this is a *small wheel*.[9]

[8] These are sometimes called *parellel cuts*.

[9] For a small wheel, all its cuts $C(i,j)$ are (II, $O(\kappa^2)$)-small.

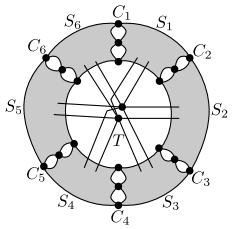


Figure 3 A 6-wheel of 8-cuts with a center of size $|T| = 2$.

Note that wheel cuts resemble the "cycle cuts" of a cactus, or the set of 2−cuts of a cycle.

Matching Cuts and Crossing Matching Cuts

Let U be a cut, A be a side of U, and $P \subseteq U$ be a subset of the cut. We call a cut W a *matching cut of U in side A w.r.t. P* if (i) $U \backslash P \subseteq W \subseteq U \cup A$, (ii) $A \backslash W \neq \emptyset$, and (iii) W disconnects $P \cup (V \backslash (U \cup A))$ from $A \backslash W$. The set $\text{Match}_{U;A}(P) \stackrel{\text{def}}{=} W \backslash U$ is the neighborhood of P restricted to A. Note that a matching cut is a type of laminar cut.

Now suppose U is a cut with exactly two sides A and B, and let $P \subseteq U$ be a non-empty subset of U. We call W a *crossing matching cut of U in side A w.r.t. P* if (i) $W \cap B \neq \emptyset$, (ii) $(U \backslash P) \cup (W \cap A)$ is a matching cut of U in side A w.r.t. P,

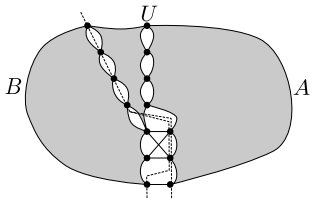


Figure 4 A cut U (drawn vertically) with two sides A and B. Dotted lines indicate two crossing matching cuts w.r.t. P_1 (bottom 3 vertices of U) and P_2 (top 2 vertices of P_1).

One could view U and a crossing matching cut W as a degenerate 4-wheel, in which one sector $S_1 = \emptyset$ is empty. Such cuts should *not* be regarded as wheels, as they do not possess key properties of wheels, e.g., that when U and W are (minimum) κ-cuts, that $|C_1| = \cdots = |C_4| = \frac{\kappa - |T|}{2}$, because $C_1 \cup T \cup C_2$ is not a cut.

Lemmas 3 and 4 are used throughout the paper. Recall here $\kappa = \kappa(G)$ is the vertex connectivity of G.

▶ **Lemma 3.** *Suppose U is a κ-cut and P a side of U. For every $p \in P$ and $u \in U$, there exists a path from p to u that is not blocked by $V \backslash P$.*

▶ **Lemma 4.** *Suppose U and W are two cuts, P is disconnected by U from the rest of the graph G and Q is disconnected by W from the rest of the graph G. Then we have the following two rules:*

- *(Intersection Rule) If $P \cap Q \neq \emptyset$, then $P \cap Q$ is disconnected by $(U \cap Q) \cup (U \cap W) \cup (W \cap P)$ from the rest of the graph G;*
- *(Union Rule) If $V \backslash (U \cup P \cup W \cup Q) \neq \emptyset$, then $P \cup Q$ is disconnected by $(U \backslash Q) \cup (W \backslash P)$ from the rest of the graph G.*

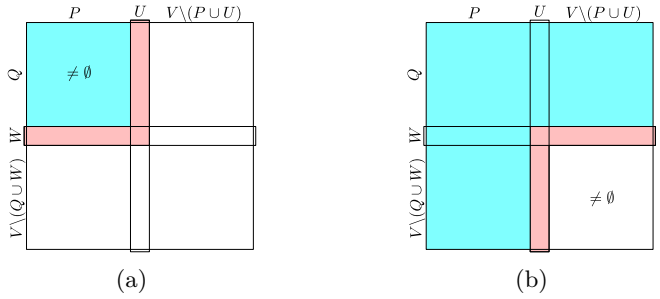


Figure 5 (a) Intersection rule; (b) Union rule.

3 The Classification of Minimum Vertex Cuts

The main *binary* structural theorem for vertex connectivity is, informally, that every two minimum vertex cuts have a relationship that is *Laminar*, *Wheel*, *Crossing Matching*, or *Small*; cf. [11]. Moreover, any strict subset of this list would be inadequate to capture all possible relationships between two vertex cuts.[10]

▶ **Theorem 5.** *Fix a minimum κ-cut U with sides $A_1, A_2, \ldots, A_a$, $a \geq 2$, and let W be any other κ-cut with sides $B_1, B_2, \ldots, B_b$, $b \geq 2$. Denote $T = U \cap W$, $W_i = W \cap A_i$ and $U_j = U \cap B_j$. Then W may be classified w.r.t. U as follows:*

Laminar type. *W is a laminar cut of U, and in particular, there exists indices i^* and j^* such that $B_{j^*} \backslash A_{i^*} = (U \backslash W) \cup (\cup_{i \neq i^*} A_i)$ and $A_{i^*} \backslash B_{j^*} = (W \backslash U) \cup (\cup_{j \neq j^*} B_j)$.*

Wheel type. *$a = b = 2$, and $(T; U_1, W_1, U_2, W_2)$ forms a 4-wheel with sectors $A_1 \cap B_1$, $A_1 \cap B_2$, $A_2 \cap B_2$ and $A_2 \cap B_1$.*

Crossing Matching type. *$a = b = 2$, and w.l.o.g., $A_1 \cap B_1 \neq \emptyset$, $A_2 \cap B_2 \neq \emptyset$, but $A_1 \cap B_2 = \emptyset$. We have $|W_2| = |U_1| > 0$, $|W_1| = |U_2| > 0$, and W is a crossing matching cut of U in side A_1 w.r.t. U_2. Furthermore, if $A_2 \cap B_1 \neq \emptyset$, then $|U_1| \geq |U_2|$.*

Small type. *U is $(I, \kappa - 1)$-small, and the small sides of U are within W, or W is $(I, \kappa - 1)$-small, and the small sides of W are within U.*

▶ Remark 6. Minimum cuts with at least three sides (i.e., $a \geq 3$ or $b \geq 3$) are sometimes called *shredders*. It it known that there are $O(n)$ shredders [41, 47] and that, in the terminology of Theorem 5, two shredders have a *laminar* or *small* relationship.

Proof of Theorem 5. Suppose there is a single index i^* such that $W_{i^*} \neq \emptyset$ and $W_i = \emptyset$ for all $i \neq i^*$. It follows that $W \subseteq A_{i^*} \cup U$ is a laminar cut of U in side A_{i^*}. It remains to prove the other properties of the laminar type. By Lemma 3 there exists paths from any vertex in A_i, $i \neq i^*$, to $U \backslash W$ that are not blocked by W, so they all lie within one side of W; let us denote this side by B_{j^*}. Then $(U \backslash W) \cup (\cup_{i \neq i^*} A_i) \subseteq B_{j^*}$, and because $V = U \cup (\cup_{i=1}^{a} A_i)$, we obtain $B_{j^*} \backslash A_{i^*} = (U \backslash W) \cup (\cup_{i \neq i^*} A_i)$. Now that $U \subseteq W \cup B_{j^*}$ is laminar w.r.t. W, so based on the same reasoning we have $A_{i^*} \backslash B_{j^*} = (W \backslash U) \cup (\cup_{j \neq j^*} B_j)$.

We proceed under the assumption that such indices i^*, j^* do not exist, and without loss of generality assume that $W_1, W_2, U_1, U_2 \neq \emptyset$. We now wish to prove that *all* U_i, W_i are non-empty. Suppose $W_i \stackrel{\text{def}}{=} W \cap A_i = \emptyset$ were empty, then A_i would be contained within a side of W, say $A_i \subseteq B_j$. By Lemma 4 (intersection rule), whenever $A_i \cap B_j \neq \emptyset$, the set $W_i \cup T \cup U_j$ disconnects $A_i \cap B_j$ from the rest of the graph. It follows that

[10] The existence of *Small* cuts as a category – an *a priori* unnatural class – indicates that there may be other ways to capture all minimum vertex cuts through an entirely different classification system.

$$|W_i| + |T| + |U_j| = |T| + |U_j| \geq \kappa = |T| + \sum_{l=1}^{b} |U_l|\,,$$

which implies that U_j is the *only* non-empty U_*-set, contradicting $U_1, U_2 \neq \emptyset$. Therefore, $W_i \neq \emptyset$ for all i and similarly, $U_j \neq \emptyset$ for all j.

Define $\Omega = \{(i,j) \mid A_i \cap B_j \neq \emptyset\}$ to be the side-pairs whose intersections are non-empty. We consider the following possibilities, which are exhaustive.

1° There exist $(i,j),(i',j') \in \Omega$ such that $i \neq i'$, $j \neq j'$. Then by Lemma 4 (intersection rule)

$$\begin{aligned} &|W_i| + |T| + |U_j| \geq \kappa \\ \text{and}\quad &|W_{i'}| + |T| + |U_{j'}| \geq \kappa. \end{aligned}$$

On the other hand,

$$\begin{aligned} &|U_j| + |U_{j'}| + |T| \leq |U| = \kappa \\ \text{and}\quad &|W_i| + |W_{i'}| + |T| \leq |W| = \kappa. \end{aligned}$$

Thus all these inequalities must be equalities, and, adding the fact that all $W_i, U_j \neq \emptyset$, we conclude that $a = b = 2$, $|W_i| = |U_{j'}|$, $|W_{i'}| = |U_j|$. W.l.o.g. we fix $i = j = 1$, $i' = j' = 2$. See Figure 6.

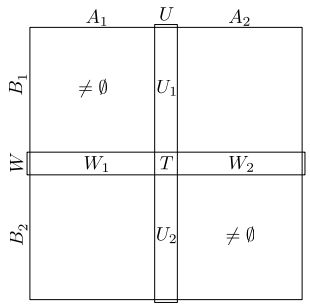


Figure 6 A depiction of cuts U, W in case 1°.

1.1° Suppose $A_1 \cap B_2 \neq \emptyset$ and $A_2 \cap B_1 \neq \emptyset$. Then $|W_i| + |U_j| \geq \kappa - |T|$ for every $\hat{\imath}, \hat{\jmath} \in \{1,2\}$, so we conclude that

$$|U_1| = |U_2| = |W_1| = |W_2| = \frac{\kappa - |T|}{2}.$$

Now that $W_i \cup T \cup U_j$ disconnects $A_i \cap B_j$ from the rest of the graph, $U_1 \cup T \cup U_2 = U$ disconnects $(A_1 \cap B_1) \cup W_1 \cup (A_1 \cap B_2) = A_1$ from $(A_2 \cap B_1) \cup W_2 \cup (A_2 \cap B_2) = A_2$, $W_1 \cup T \cup W_2 = W$ disconnects $(A_1 \cap B_1) \cup U_1 \cup (A_2 \cap B_1) = B_1$ from $(A_1 \cap B_2) \cup U_2 \cup (A_2 \cap B_2) = B_2$, we conclude that $(T; U_1, W_1, U_2, W_2)$ forms a 4-wheel.

1.2° Suppose $A_1 \cap B_2 = \emptyset$ (or symmetrically, that $A_2 \cap B_1 = \emptyset$). Then $A_1 = A_1 \cap (B_1 \cup W) = (A_1 \cap B_1) \cup W_1$. By Lemma 4, $W_1 \cup T \cup U_1$ separates $A_1 \cap B_1$ from the rest of the graph. Since $U_2 \subseteq V \setminus ((A_1 \cap B_1) \cup (U_1 \cup T \cup W_1))$, it follows that $W_1 \cup T \cup U_1$ disconnects U_2 from $A_1 \cap B_1 = A_1 \setminus (W_1 \cup T \cup U_1)$, i.e., it is a matching cut of U in side A_1 w.r.t. U_2. See Figure 7. Because $W_2 = W \cap A_2 \neq \emptyset$ and $(W \cap A_1) \cup (U \setminus U_2) = W_1 \cup T \cup U_1$, W is a crossing matching cut of U in side A_1 w.r.t. U_2.

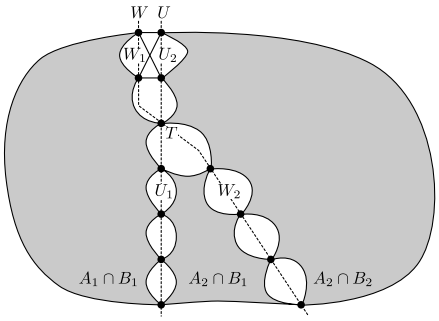


Figure 7 A depiction of the cuts U, W in case 1.2°.

If $A_2 \cap B_1 \neq \emptyset$, by Lemma 4 (intersection rule) $|U_1|+|W_2|+|T| \geq \kappa = |W_1|+|T|+|W_2|$, so $|U_1| \geq |U_2|$.

2° Suppose there exists a $j^\sharp$ such that $\forall i.\forall j \neq j^\sharp.(i,j) \notin \Omega$, i.e., $A_i \cap B_j = \emptyset$. This implies that $\cup_{j \neq j^\sharp} B_j \subseteq U$, and because $U_{j^\sharp} \neq \emptyset$, $\left|\cup_{j \neq j^\sharp} B_j\right|$ is *strictly* smaller than κ. Therefore W is a (I, $\kappa - 1$)-small cut, and all the small sides of W are within U.

3° There exists $i^\sharp$ such that $\forall i \neq i^\sharp.\forall j.(i,j) \notin \Omega$. Symmetric to case 2°; U is (I, $\kappa - 1$)-small, and all the small sides of U are within W.

4° $\Omega = \emptyset$. Then $\cup_{i=1}^{a} A_i \subseteq W$, so $|V| = |U \cup (\cup_{i=1}^{a} A_i)| \leq |U \cup W| \leq 2\kappa$. This is possible, but not one we consider as it contradicts our initial assumption that $n > 4\kappa$. ◀

▶ **Corollary 7** (cf. [41, 47, 53])**.** *If U is a κ-cut that is not (I, $\kappa - 1$)-small and has at least 3 sides, then all other κ-cuts have a laminar type relation with U, or are themselves (I, $\kappa - 1$)-small cuts.*

▶ **Corollary 8.** *Suppose U is a κ-cut that is not (I, $\kappa - 1$)-small, with exactly two sides A and B. Suppose W is a κ-cut with sides K, L (and possibly others), such that $W \cap A \neq \emptyset$, $W \cap B \neq \emptyset$, $A \subseteq K \cup W$, and $L \cap U \neq \emptyset$. Then W only has two sides, and W is a crossing matching cut of U in side A w.r.t. $L \cap U$.*

▶ **Corollary 9.** *Define* $\text{Cuts}_{C;D}$ *to be the set of all κ-cuts that disconnect disjoint, non-empty vertex sets C and D. If* $\text{Cuts}_{C;D} \neq \emptyset$*, it contains a unique minimal element* $\text{MinCut}_{C;D}$*, such that for any cut $U \in \text{Cuts}_{C;D}$,* $\text{Region}_{\text{MinCut}_{C;D}}(C) \subseteq \text{Region}_U(C)$.

Theorem 5 classifies the pairwise relationship between two minimum κ-cuts. In Sections 3.1–3.4 we further explore the properties of wheel cuts, (crossing) matching cuts, laminar cuts, and small cuts.

3.1 Wheels and Wheel Cuts

Recall that a w-wheel $(T; C_1, \ldots, C_w)$ satisfied, by definition, the property that $C_i \cup T \cup C_{i+2}$ formed a κ-cut, but did not say anything explicitly about $C(i,j) = C_i \cup T \cup C_j$. Lemma 10 proves that these are also cuts, and bounds their number of sides.

▶ **Lemma 10.** *Suppose $(T; C_1, C_2, \ldots, C_w)$ forms a w-wheel with sectors $S_1, S_2, \ldots, S_w$. (Subscripts are modulo w.) For any $i \neq j$, $C(i,j)$ is a κ-cut that disconnects $D(i,j)$ from the rest of the graph. Moreover, when $j - i \notin \{1, w-1\}$, $C(i,j)$ has exactly two sides, which are $D(i,j)$ and $D(j,i)$. Furthermore, $|C_i| = \frac{\kappa - |T|}{2}$.*

▶ **Theorem 11.** *Suppose $(T; C_1, C_2, \ldots, C_w)$ forms a w-wheel with sectors $S_1, S_2, \ldots, S_w$. (Subscripts are given by modulo w.) Let X be any minimum κ-cut. Then one of the following is true:*

1° *$X = C(i,j)$ for some $i \neq j$.*

2° *$X \subseteq C(i,i+1) \cup S_i$ for some i, i.e., X is a laminar cut of $C(i,i+1)$.*

3° *X has crossing matching type relation with some $C(i,i+1)$ or some $C(i,i+2)$.*

4° *X is a $(I, \kappa-1)$-small cut.*

5° *$(T; C_1, C_2, \ldots, C_w)$ is a small wheel.*

6° *There exists $i < j$, such that $X \subseteq S_i \cup T \cup S_j$, and $(T; C_1, \ldots, C_i, X \cap S_i, C_{i+1}, \ldots, C_j, X \cap S_j, C_{j+1}, \ldots, C_w)$ forms a $(w+2)$-wheel; or there exists $i \neq j$, $X \subseteq S_i \cup T \cup C_j$, and $(T; C_1, \ldots, C_i, X \cap S_i, C_{i+1}, \ldots, C_w)$ forms a $(w+1)$-wheel. In other words, the wheel $(T; C_1, C_2, \ldots, C_w)$ is a subwheel of some other wheel.*

3.2 Matching Cuts and Crossing Matching Cuts

Define $N(P)$ to be the neighborhood of $P \subset V$ and $N_A(P) \stackrel{\text{def}}{=} N(P) \cap A$.

▶ **Theorem 12.** *Let U be an arbitrary κ-cut and A a side of U.*

1° *If there exists a matching κ-cut of U in side A w.r.t. P, then it is $W = (U \backslash P) \cup N_A(P)$, and $|P| = |N_A(P)| < |A|$. In particular, $\mathrm{Match}_{U;A}(P) = N_A(P)$.*

2° *When there is such a matching cut W, G contains a matching between P and its neighborhood $N_A(P) = \mathrm{Match}_{U;A}(P)$ in side A.*

3° *Suppose $\mathrm{Match}_{U;A}(P)$ and $\mathrm{Match}_{U;A}(Q)$ exist. If $P \cap Q \neq \emptyset$, then $\mathrm{Match}_{U;A}(P \cap Q)$ exists, and*

$$\mathrm{Match}_{U;A}(P \cap Q) = \mathrm{Match}_{U;A}(P) \cap \mathrm{Match}_{U;A}(Q).$$

If $|A| > |P \cup Q|$, then $\mathrm{Match}_{U;A}(P \cup Q)$ exists, and

$$\mathrm{Match}_{U;A}(P \cup Q) = \mathrm{Match}_{U;A}(P) \cup \mathrm{Match}_{U;A}(Q).$$

Fix a κ-cut U and a side A of U. Define $\Theta = \{P \mid \mathrm{Match}_{U;A}(P) \text{ exists}\}$. According to Part 3° of Theorem 12, Θ is closed under union and intersection, and is therefore characterized by its minimal elements. Define $\Theta^* = \{\cap_{u \in P, P \in \Theta} P \mid u \in U\}$. It can be seen from the definition that $\cap_{u \in P, P \in \Theta} P$ corresponds to the minimum matching cut for vertex u.

In the most extreme case Θ may have $2^\kappa - 1$ elements (e.g., if the graph induced by $U \cup N_A(U)$ is a matching), which may be prohibitive to store explicitly. From definition we know that $|\Theta^*| \leq \kappa$, so it works as a good compression for Θ. Lemmas 13 and 14 also highlights some ways in which Θ^* is a sufficient substitute for Θ.

▶ **Lemma 13.** *Let U be a κ-cut and let Θ be defined w.r.t. the matching cuts of U in a side A. Suppose that $P \in \Theta^*$ and $P \subseteq Q \in \Theta$, and that W is a crossing matching cut of U in side A w.r.t. Q. Then $(W \setminus \mathrm{Match}_{U;A}(Q)) \cup (Q \backslash P) \cup \mathrm{Match}_{U;A}(P)$ is also a crossing matching cut of U in A w.r.t. P. Moreover, if $Q = P_1 \cup P_2 \cup \cdots \cup P_\ell$ where each $P_i \in \Theta^*$, then any pair disconnected by W is also disconnected by some $(W \setminus \mathrm{Match}_{U;A}(Q)) \cup (Q \backslash P_i) \cup \mathrm{Match}_{U;A}(P_i)$.*

▶ **Lemma 14.** *Let U be a κ-cut with two sides A and B, and let Θ be defined w.r.t. its matching cuts in side A. For $P \in \Theta^*$, define $U^*(P)$ to be the cut separating P from $A \setminus \mathrm{Match}_{U;A}(P)$ minimizing $\left|\mathrm{Side}_{U^*(P)}(P)\right|$.*

1° *$U^*(P)$ is either a crossing matching cut of U in side A w.r.t. P, or else there is no such crossing matching cut and $U^*(P) = (U \backslash P) \cup \mathrm{Match}_{U;A}(P)$ is a matching cut.*

2° *Suppose X is a crossing matching cut of U in side A w.r.t. P. If u, v are separated by X, then they are also separated by either $U^*(P)$ or $(X \setminus \mathrm{Match}_{U;A}(P)) \cup P$, which is a laminar cut of U.*

3.3 Laminar Cuts

In this section we analyze the structure of laminar cuts. Throughout this section, U refers to a cut that is *not* (I, $\kappa-1$)-small, *not* a wheel cut $C(i,j)$ in some wheel, and has a side A with $|A| > 2\kappa$.

Consider the set of all cuts W that are laminar w.r.t. U, contained in $U \cup A$ and not (I, $\kappa-1$)-small. It follows that W has a side, call it $S(W)$, that contains $U \backslash W$ and all other sides of U.[11] Define $R(W)$ to be the region containing all other sides of W beside $S(W)$. We call W a *maximal* laminar cut of U if there does not exist another laminar cut W' such that $R(W) \subseteq R(W')$.

▶ **Theorem 15.** *Let U be the reference cut.*

1. *If there exist matching cuts of U in side A, define Θ^* w.r.t. U, A, define $Q = \cup_{P\in\Theta^*} P$, and let $X = (U \backslash Q) \cup \mathrm{Match}_{U;A}(Q)$ be the matching cut in side A having the smallest intersection with U. Then every laminar cut W of U in side A is*
   - **(i)** *a laminar cut of X in region $A \backslash \mathrm{Match}_{U;A}(Q)$, or*
   - **(ii)** *a matching cut of U, or*
   - **(iii)** *a crossing matching cut of X.*
2. *If there are no matching cuts of U in side A, every laminar cut of U in side A is a maximal laminar cut, or a laminar cut of some maximal laminar cut W_i in a side of $R(W_i)$. Moreover, whenever W_i, W_j are distinct maximal laminar cuts, $R(W_i) \cap R(W_j) = \emptyset$.*

3.4 Small Cuts

Fix a vertex u and a threshold $t \leq \lceil \frac{n-\kappa}{2} \rceil$. Define $\mathrm{Sm}_t(u)$ to be a cut U minimizing $|\mathrm{Side}_U(u)|$ with $|\mathrm{Side}_U(u)| \leq t$. We show that $\mathrm{Sm}_t(u)$, if it exists, is unique. Also, note this is not immediately derived using intersection-union(submodularity) property of cuts, or lemma 4.

▶ **Theorem 16.** *If there exists a (III, t)-small cut that is small w.r.t. u, then there exists a* unique *such cut, denoted $\mathrm{Sm}_t(u)$, such that for any other cut U, $u \notin U$, $\mathrm{Side}_{\mathrm{Sm}_t(u)} \subseteq \mathrm{Side}_U(u)$.*

4 A Data Structure for $(\kappa+1)$-Connectivity Queries

In this section we design an efficient data structure that, given u, v, answers $(\kappa+1)$-connectivity queries, i.e., reports that $\kappa(u,v) = \kappa$ and produces a minimum κ-cut separating u, v, or reports that $\kappa(u,v) \geq \kappa+1$.

We work with the mixed-cut definition of $\kappa(u,v)$ (see Remark 1), which is the minimum size set of vertices and edges that need to be removed to disconnect u and v, or equivalently, the maximum size set of internally vertex-disjoint paths joining u and v.[12]

▶ **Theorem 17.** *Given a κ-connected graph G, we can construct in $\tilde{O}(m + \mathrm{poly}(\kappa)n)$ time a data structure occupying $O(\kappa n)$ space that answers the following queries. Given $u, v \in V(G)$, report whether $\kappa(u,v) = \kappa$ or $\geq \kappa+1$ in $O(1)$ time. If $\kappa(u,v) = \kappa$, report a κ-cut separating u, v in $O(\kappa)$ time.*

---

[11] $S(W)$ is exactly B_{j^*} of Theorem 5, if using its notation on U and W.

[12] If $\{u,v\} \notin E(G)$ and $\kappa(u,v) = \kappa$, then there exists $U \subset V$, $|U| = \kappa$, such that removing U disconnects u, v. If $\{u,v\} \in E(G)$ then there exists $U \subset V$, $|U| = \kappa - 1$, such that removing U and $\{u,v\}$ disconnects u, v. In this case the single-edge path $\{u,v\}$ would count for one of the κ internally vertex disjoint paths, the other $\kappa-1$ passing through distinct vertices of U.

In $O(m)$ time, the Nagamochi-Ibaraki [50] algorithm produces a subgraph G' that has arboricity $\kappa+1$[13] and hence at most $(\kappa+1)n$ edges, such that $\kappa_{G'}(u,v)=\kappa_G(u,v)$ whenever $\kappa_G(u,v)\le\kappa+1$, and $\kappa_{G'}(u,v)\ge\kappa+1$ whenever $\kappa_G(u,v)\ge\kappa+1$. Without loss of generality we may assume G is the *output* of the Nagamochi-Ibaraki algorithm.

Data Structure

Throughout this section we fix the threshold $t=\lceil\frac{n-\kappa}{2}\rceil$. Define $\mathrm{Sm}(u)=\mathrm{Sm}_t(u)$ to be the unique minimum κ-cut with $\mathrm{Side}_{\mathrm{Sm}(u)}(u)\le t$, if any such cut exists, and $\mathrm{Sm}(u)=\perp$ otherwise. The data structure stores, for each $u\in V(G)$, $\mathrm{Sm}(u)$, $|\mathrm{Side}_{\mathrm{Sm}(u)}(u)|$, a $O(\log n)$-bit identifier for $\mathrm{Side}_{\mathrm{Sm}(u)}(u)$, and for each vertex $v\in N(u)\cap\mathrm{Sm}(u)$, a bit $b_{u,v}$ indicating whether $\{\{u,v\}\}\cup\mathrm{Sm}(u)\backslash\{v\}$ is a mixed cut disconnecting u and v. Furthermore, when $|\mathrm{Side}_{\mathrm{Sm}(u)}(u)|\le\kappa-1$, we store $\mathrm{Side}_{\mathrm{Sm}(u)}(u)$ explicitly. When $\mathrm{Sm}(u)=\perp$ we will say $\mathrm{Side}_{\mathrm{Sm}(u)}(u)=G$ and hence $\left|\mathrm{Side}_{\mathrm{Sm}(u)}\right|=n$. The total space is $O(\kappa n)$.

Connectivity Queries

The query algorithm proceeds to the first applicable case. Note in the following, $\mathrm{Sm}(u)$ may be $\perp$, and for all vertices v, we define $v\notin\perp$.

Case I: $\mathrm{Sm}(u)=\mathrm{Sm}(v)$ and $\mathrm{Side}_{\mathrm{Sm}(u)}(u)=\mathrm{Side}_{\mathrm{Sm}(v)}(v)$. Then $\kappa(u,v)\ge\kappa+1$.

Case II: $u\notin\mathrm{Sm}(v)$ and $v\notin\mathrm{Sm}(u)$. Then $\kappa(u,v)=\kappa$. Without loss of generality suppose that $\left|\mathrm{Side}_{\mathrm{Sm}(u)}(u)\right|\le\left|\mathrm{Side}_{\mathrm{Sm}(v)}(v)\right|$. Then $\mathrm{Sm}(u)$ is a κ-cut separating u and v.

Case III: $v\in\mathrm{Sm}(u)\cap N(u)$, or the reverse. The bit $b_{u,v}$ indicates whether $\kappa(u,v)\ge\kappa+1$ or $\kappa(u,v)=\kappa$, in which case $\{\{u,v\}\}\cup\mathrm{Sm}(u)\backslash\{v\}$ is the κ-cut.

Case IV: $v\in\mathrm{Sm}(u),u\in\mathrm{Sm}(v)$. Then $\kappa(u,v)\ge\kappa+1$.

Case V: $v\in\mathrm{Sm}(u),u\notin\mathrm{Sm}(v)$, or the reverse. If $\left|\mathrm{Side}_{\mathrm{Sm}(v)}(v)\right|\le\kappa-1$, directly check whether $u\in\mathrm{Side}_{\mathrm{Sm}(v)}(v)$. If so then $\kappa(u,v)\ge\kappa+1$; if not then $\mathrm{Sm}(v)$ disconnects them. Thus $\left|\mathrm{Side}_{\mathrm{Sm}(v)}(v)\right|\ge\kappa$. If $\left|\mathrm{Side}_{\mathrm{Sm}(v)}(v)\right|\le\left|\mathrm{Side}_{\mathrm{Sm}(u)}(u)\right|$ then $\mathrm{Sm}(v)$ is a κ-cut separating u and v, and otherwise $\kappa(u,v)\ge\kappa+1$.

Lemmas 18, 19, and Theorem 20 establish the *correctness* of the query algorithm. Its construction algorithm is described and analyzed in Section 4.1.

▶ **Lemma 18.** *If $v\in\mathrm{Side}_{\mathrm{Sm}(u)}(u)$, then either $\mathrm{Sm}(v)=\mathrm{Sm}(u)$ or $\mathrm{Sm}(v)$ is a laminar cut of $\mathrm{Sm}(u)$ with $\mathrm{Side}_{\mathrm{Sm}(v)}(v)\subset\mathrm{Side}_{\mathrm{Sm}(u)}(u)$.*

▶ **Lemma 19.** *Suppose u and v are not $(\kappa+1)-$connected, i.e., $\kappa(u,v)=\kappa$. If $\{u,v\}\notin E(G)$, then they are disconnected by $\mathrm{Sm}(u)$ or $\mathrm{Sm}(v)$, and if $\{u,v\}\in E(G)$, then they are disconnected by $\{(u,v)\}\cup\mathrm{Sm}(u)\backslash\{v\}$ or $\{(u,v)\}\cup\mathrm{Sm}(v)\backslash\{u\}$.*

Proof. First suppose $\{u,v\}\notin E(G)$ and let X be any cut separating u and v. When $t=\lceil\frac{n-\kappa}{2}\rceil$ either $|\mathrm{Side}_X(u)|\le t$ or $|\mathrm{Side}_X(v)|\le t$. W.l.o.g. suppose it is the former, then $\mathrm{Sm}(u)$ exists and by Theorem 16, $\mathrm{Side}_{\mathrm{Sm}(u)}(u)\subseteq\mathrm{Side}_X(u)$, so $\mathrm{Sm}(u)$ also separates u and v.

If $\{u,v\}\in E(G)$, suppose $(\kappa-1)$ vertices $W=\{w_1,w_2,\ldots,w_{\kappa-1}\}$ and $\{u,v\}$ disconnect u and v. After removing W from the graph, $G\backslash W$ is still connected. By deleting the edge $\{u,v\}$, the graph breaks into exactly two connected components, say A and B with $u\in A$ and $v\in B$. Then $W\cup\{u\}$ forms a κ-cut with $\mathrm{Side}_{W\cup\{u\}}(v)=B$, and $W\cup\{v\}$

[13] Namely, G' is a union of k forests, as mentioned before.

also forms a κ-cut with $\text{Side}_{W\cup\{v\}}(u) = A$. Clearly we have $n = |W| + |A| + |B| = \kappa - 1 + |A| + |B|$. W.l.o.g. suppose $|A| \leq |B|$, then $|A| \leq \lfloor \frac{n-\kappa+1}{2} \rfloor = \lceil \frac{n-\kappa}{2} \rceil = t$. Thus $\text{Sm}(u)$ exists, $\text{Side}_{\text{Sm}(u)}(u) \subseteq \text{Side}_{W\cup\{v\}}(u)$, and $\text{Sm}(u)$ is either $W \cup \{v\}$ or a laminar cut of $W \cup \{v\}$ in side A. Since $\{u, v\} \in E(G)$, we have $v \in \text{Sm}(u)$. If we remove $\{u, v\}$ from G, then any path from u to v goes through a vertex in W, but any path from u to a vertex in W goes through a vertex in $\text{Sm}(u)\backslash\{v\}$. Therefore, $\{\{u, v\}\} \cup \text{Sm}(u)\backslash\{v\}$ is a mixed cut separating u, v as it blocks all u-v paths. ◀

▶ **Theorem 20.** *The query algorithm correctly answers $(\kappa + 1)$-connectivity queries.*

Proof. Suppose the algorithm terminates in Case I. It follows that $u \notin \text{Sm}(v), v \notin \text{Sm}(u)$, and neither $\text{Sm}(u)$ nor $\text{Sm}(v)$ disconnect u and v. Lemma 19 implies that $\kappa(u, v) \geq \kappa + 1$.

In Case II, if $\text{Sm}(u) \neq \perp$ but $\text{Sm}(v) = \perp$ then $\text{Sm}(u)$ is the cut separating u, v and since $\left|\text{Side}_{\text{Sm}(u)}(u)\right| < \left|\text{Side}_{\text{Sm}(v)}(v)\right| = n$, then the query is answered correctly. If both $\text{Sm}(u), \text{Sm}(v) \neq \perp$, then by Lemma 18, $v \notin \text{Side}_{\text{Sm}(u)}(u)$ and once again the query is answered correctly.

In Case III, by Lemma 19, if u and v are separated by a κ-cut, they are separated by $\{\{u, v\}\} \cup \text{Sm}(u)\backslash\{v\}$ (if $\text{Sm}(u) \neq \perp$) or $\{\{u, v\}\} \cup \text{Sm}(v)\backslash\{u\}$ (if $\text{Sm}(v) \neq \perp$), and this information is stored in the bit $b_{u,v}, b_{v,u}$.

If we get to Case IV then $\{u, v\} \notin E(G)$ and neither $\text{Sm}(u)$ nor $\text{Sm}(v)$ separate u, v, hence by Lemma 19, $\kappa(u, v) \geq \kappa + 1$ and the query is answered correctly.

Case V is the most subtle. Because $v \in \text{Sm}(u)$ and $\{u, v\} \notin E(G)$, Lemma 19 implies that *if* $\kappa(u, v) = \kappa$, then u, v must be separated by $\text{Sm}(v)$. If $\text{Sm}(v) = \perp$ then $\kappa(u, v) \geq \kappa + 1$ and the query is answered correctly. If $|\,\text{Side}_{\text{Sm}(v)}(v)| \leq \kappa - 1$ then the query explicitly answers the query correctly by direct lookup. Thus, we proceed under the assumption that $\text{Sm}(v) \neq \perp$ exists and is not small.

If $u \in \text{Side}_{\text{Sm}(v)}(v)$ then $\text{Sm}(v)$ does not disconnect u and v, and by Lemma 18, $\left|\text{Side}_{\text{Sm}(v)}(v)\right| > \left|\text{Side}_{\text{Sm}(u)}(u)\right|$, so the query is handled correctly in this case.

If $u \notin \text{Side}_{\text{Sm}(v)}(v)$ then $\text{Sm}(v)$ separates u and v, so we must argue that $\left|\text{Side}_{\text{Sm}(v)}(v)\right| \leq \left|\text{Side}_{\text{Sm}(u)}(u)\right|$ for the query algorithm to work correctly. It cannot be that $\text{Sm}(v)$ and $\text{Sm}(u)$ have a laminar relation, so by Theorem 5 they must have a crossing matching, wheel, or small type relation. If they have the small-type relation then the small sides of $\text{Sm}(u)$ are contained in $\text{Sm}(v)$ (contradicting $u \notin \text{Sm}(v)$) or the small sides of $\text{Sm}(v)$ are contained in $\text{Sm}(u)$, but we have already ruled out this case. Thus, the remaining cases to consider are wheel and crossing matching type.

Suppose $\text{Sm}(u), \text{Sm}(v)$ form a 4-wheel $(T; C_1, C_2, C_3, C_4)$. Then $u \notin \text{Sm}(v)$ appears in a sector of the wheel, say S_1. Then $C(1, 2)$ is a cut violating the minimality of $\text{Sm}(u) = C(1, 3)$.

Suppose $\text{Sm}(u), \text{Sm}(v)$ have a crossing matching type relation. Let $A_1 = \text{Side}_{\text{Sm}(u)}(u)$ and A_2 be the other side of $\text{Sm}(u)$, and $B_1 = \text{Side}_{\text{Sm}(v)}(v)$ and B_2 be the other side of $\text{Sm}(v)$. Then $u \in A_1 \cap B_2$, and it must be that the diagonal quadrant $A_2 \cap B_1 = \emptyset$. Suppose otherwise, i.e., $A_2 \cap B_1 \neq \emptyset$, and let $X = (\text{Sm}(u) \cap B_2) \cup (\text{Sm}(v) \cap A_1) \cup (\text{Sm}(u) \cap \text{Sm}(v))$. Then by Corollary 9 X is a κ-cut with $\text{Side}_X(u) = A_1 \cap B_2$, contradicting the minimality of $\text{Sm}(u)$. Thus, $\text{Sm}(v)$ is a crossing matching cut of $\text{Sm}(u)$ in side A_2 w.r.t. some $Q \subseteq \text{Sm}(u) \cap B_1$ with $v \in Q$. By Theorem 5 and $u \in A_1 \cap B_2 \neq \emptyset$, we have

$$|A_1 \cap \text{Sm}(v)| = |B_2 \cap \text{Sm}(u)| \geq |A_2 \cap \text{Sm}(v)| = |B_1 \cap \text{Sm}(u)| = |Q|.$$

Thus,

$$|\,\text{Side}_{\text{Sm}(u)}(u)| = |A_1| > |(A_1 \cap B_1) \cup Q| = |\,\text{Side}_{\text{Sm}(v)}(v)|,$$

establishing the correctness in the crossing matching case. (The strictness of the inequality is because $A_1 \cap B_2 \neq \emptyset$.) ◀

Refer to the full version for a $\tilde{O}(m + \text{poly}(\kappa)n)$-time algorithm to construct this data structure, in particular, to find all minimal cuts $\{\text{Sm}(u)\}_{u \in V}$.

4.1 Construction of the Data Structure

We assume Nagamochi-Ibaraki sparsification [50] has already been applied, so G has arboricity $\kappa + 1$ and $O(\kappa n)$ edges. We use the recent Forster et al. [23] algorithm for computing the connectivity $\kappa = \kappa(G)$ in $\tilde{O}(\text{poly}(\kappa)n)$ time and searching for κ-cuts.

▶ **Corollary 21** (Consequence of Forster, Nanongkai, Yang, Saranurak, and Yingchareonthawornchai [23]). *Given $x \in V(G)$ and an integer $s \leq \lceil \frac{n-\kappa}{2} \rceil$, we can, with high probability $1 - 1/\text{poly}(n)$, compute $\text{Sm}_s(x)$ in $\tilde{O}(|\text{Side}_{\text{Sm}_s(x)}(x)| \cdot \kappa^3)$ time, or determine that $\text{Sm}_s(x)$ does not exist in $\tilde{O}(s\kappa^3)$ time.*

We call the procedure of Corollary 21 FindSmall(x, s).

▶ **Definition 22.** *Let U be a cut, A a side of U. We use the notation $\overline{A} = G \backslash (U \cup A)$ to be the region of all other sides of U. Define $G(U, \overline{A})$ to be the graph induced by $U \cup \overline{A}$, supplemented with a κ-clique on U. If W is a cut in $G(U, \overline{A})$, define $\text{Side}_W^{G(U,\overline{A})}(x)$ to be $\text{Side}_W(x)$ in the graph $G(U, \overline{A})$.*

Lemma 23 is useful for constructing the algorithm in lemma 24.

▶ **Lemma 23.** *Let U be a κ-cut, A be a side of U, and W be a set of κ vertices in $G(U, \overline{A})$. Then W is a κ-cut in $G(U, \overline{A})$ if and only if W is a laminar cut of U in one of the sides of $\overline{A}$. Moreover, when W is such a cut, for any vertex $u \in U \backslash W$,*

$$\text{Side}_W(u) = \text{Side}_W^{G(U,\overline{A})}(u) \cup A.$$

Lemma 24 shows how, beginning with a cut X where $\text{Side}_X(u)$ is small, can find another cut Y (if one exists) where $\text{Side}_Y(u)$ is about M, in $\tilde{O}(M\kappa^4)$ time. The difficulty is that there could be an unbounded number of cuts "between" X and Y that would prevent the FindSmall algorithm from finding Y directly.

▶ **Lemma 24.** *For any integer $M \leq t/2$, vertex u, and cut X with $A = \text{Side}_X(u), |A| \leq 2M$, the algorithm* Expand$(u, A, M)$ *runs in time $\tilde{O}(M\kappa^4)$ and, w.h.p., returns a cut Y satisfying the following properties.*

- $\text{Side}_X(u) \subseteq \text{Side}_Y(u)$.
- $|\text{Side}_Y(u)| \leq 2M$.
- *If there exists a cut Z that is (III, M)-small w.r.t. u, then $|\text{Side}_Y(u)| \geq |\text{Side}_Z(u)|$.*

The content of Expand(u, A, M) *is given below.*

Initially $Y \leftarrow X$. While $|\text{Side}_Y(u)| < M$,

a. *For each vertex $v \in Y$, in parallel,*
   i. *In the graph $G(Y, \overline{\text{Side}_Y(u)})$, run* FindSmall$(v, M)$.

b. *The moment any call to* FindSmall *halts in step (i) with a cut W, stop all such calls and set $Y \leftarrow W$. If all $|Y|$ calls to* FindSmall *run to completion without finding a cut, halt and return Y.*

We use Corollary 25 to find $\text{Sm}_t(u)$ for potentially many vertices u in bulk.

▶ **Corollary 25** (Consequence of Picard and Queyrenne [56]). *Fix two disjoint, non-empty vertex sets C and D. In $O(\kappa^2(n - |C| - |D|))$ time, we can output a cut $S(v)$ for every $v \in V \backslash (C \cup D)$, such that if $\text{Sm}(v)$ exists and $C \subseteq \text{Side}_{\text{Sm}(v)}(v)$, then $S(v) = \text{Sm}(v)$.*

We are now ready to present the entire construction algorithm.

Preamble. The algorithm maintains some κ-cut $T(u)$ for each u, which is initially $\perp$, and stores $|\operatorname{Side}_{T(u)}(u)|$. If the algorithm makes no errors, $T(u) = \operatorname{Sm}_t(u) = \operatorname{Sm}(u)$ at the end of the computation. The procedure $\mathsf{Update}(u, U)$ updates $T(u) \leftarrow U$ if U is a better cut, i.e., $|\operatorname{Side}_U(u)| \leq \min\{|\operatorname{Side}_{T(u)}(u)| - 1, t\}$, and does nothing otherwise. $\mathsf{Update}(A, U)$ is short for $\mathsf{Update}(u, U)$ for all $u \in A$.

Step1: very small cuts. For each $u \in V$, let $U \leftarrow \mathsf{FindSmall}(u, 100\kappa)$ and then $\mathsf{Update}(u, U)$. This takes $\tilde{O}(n\kappa^4)$ time.

Step2: unbalanced cuts. For each index i such that $100\kappa < 2^i \leq t$, let $\alpha = 2^i$, and pick a uniform sample $V_i \subset V$ of size $(n \log n)/\alpha$.[14] For each $u \in V_i$, compute $\operatorname{Sm}_\alpha(u) \leftarrow \mathsf{FindSmall}(u, \alpha)$. If $\operatorname{Sm}_\alpha(u) = \operatorname{Sm}(u) \neq \perp$, we first do an $\mathsf{Update}(\operatorname{Side}_{\operatorname{Sm}(u)}(u), \operatorname{Sm}(u))$, then compute $Y \leftarrow \mathsf{Expand}(u, \operatorname{Sm}(u), \alpha)$. For each $v \in Y$, compute $W_v \leftarrow \mathsf{FindSmall}(w, \alpha)$ and then $\mathsf{Update}(v, W_v)$. We then run the algorithm of Corollary 25 with $C = \operatorname{Side}_{\operatorname{Sm}(u)}(u)$ and $D = V \backslash (Y \cup \operatorname{Side}_Y(u))$, which returns a set of cuts $\{S(v)\}_{v \in V \backslash (C \cup D)}$. For each such $v \in \operatorname{Side}_Y(u) \backslash \operatorname{Side}_{\operatorname{Sm}(u)}(u)$, do an $\mathsf{Update}(v, S(v))$. For each index i, the running time is $\tilde{O}(|V_i| \cdot \alpha\kappa^4) = \tilde{O}(n\kappa^4)$, which is $\tilde{O}(n\kappa^4)$ overall.

Step3: balanced cuts. Sample $O(\log n)$ pairs $(x, y) \in V^2$. For each such pair, compute $U \leftarrow \mathsf{FindSmall}(x, t)$. If $U \neq \perp$, apply the algorithm of Corollary 25 to $C = \operatorname{Side}_{\operatorname{Sm}(x)}(x)$ and $D = \{y\}$, which returns a set $\{S(v)\}$. Then do an $\mathsf{Update}(v, S(v))$ for every $v \in V \backslash (C \cup D)$. By Corollary 21 this takes $\tilde{O}(\kappa^3 n)$ time.

Step4: adjacent vertices At this point it should be the case that $T(u) = \operatorname{Sm}(u)$ for all u. For each $v \in T(u) \cap N(u)$ compute and set the bit $b_{u,v}$. (This information can be extracted from the calls to FindSmall and the algorithm of Corollary 25 in the same time bounds.)

▶ **Lemma 26.** *Suppose* $\operatorname{Sm}(u)$ *and* $\operatorname{Sm}(v)$ *exists,* $u \in \operatorname{Side}_{\operatorname{Sm}(v)}(v)$*, and suppose there is a cut* W *such that* $\kappa \leq \left|\operatorname{Side}_{\operatorname{Sm}(v)}(v)\right| < |\operatorname{Side}_W(u)| \leq t$*. Then* $v \in W \cup \operatorname{Side}_W(u)$*.*

Lemma 26 is critical to proving the correctness of the algorithm's search strategy.

▶ **Theorem 27.** *The construction algorithm correctly computes* $\{\operatorname{Sm}(v)\}_{v \in V}$ *and runs in time* $\tilde{O}(n\kappa^4)$*.*

Proof. If $|\operatorname{Side}_{\operatorname{Sm}(v)}(v)| \leq 100\kappa$, then $T(v) = \operatorname{Sm}(v)$ after Step 1, with high probability.

Suppose that $|\operatorname{Side}_{\operatorname{Sm}(v)}(v)| \in [2^j, 2^{j+1}]$ and $2^{j+1} \leq t$. Then with high probability, at least one vertex $x \in V_j$ is sampled in Step 2 such that $x \in \operatorname{Side}_{\operatorname{Sm}(v)}(v)$. Step 2 (Expand) computes a cut Y such that $|\operatorname{Side}_Y(x)| \geq |\operatorname{Side}_{\operatorname{Sm}(v)}(v)|$, so by Lemma 26, either $v \in \operatorname{Side}_Y(x)$ or $v \in Y$. In the former case $\operatorname{Sm}(v)$ is computed using the Corollary 25 algorithm. In the latter case $\operatorname{Sm}(v)$ is computed directly using FindSmall.

Finally, if $\operatorname{Sm}(v)$ is balanced, say $|\operatorname{Side}_{\operatorname{Sm}(v)}(v)| \geq t/4$, then w.h.p. we would pick a pair (x, y) in Step 3 such that $x \in \operatorname{Side}_{\operatorname{Sm}(v)}(v)$ and $y \in V \backslash (\operatorname{Sm}(v) \cup \operatorname{Side}_{\operatorname{Sm}(v)}(v))$. If this holds the algorithm of Corollary 25 correctly computes $\operatorname{Sm}(v)$. ◀

[14] The Forster et al. [23] algorithm samples vertices proportional to their degree. Note that after the Nagamochi-Ibaraki [50] sparsification, the minimum degree is at least κ and the density of every induced subgraph is at most $\kappa + 1$, so it is equally effective to do vertex sampling.

5 Conclusion

This paper was directly inspired by the extended abstract of Cohen, Di Battista, Kanevsky, and Tamassia [11]. Our goal was to substantiate the main claims of this paper, and to simplify and improve the data structure that answers $(\kappa+1)$-connectivity queries.

We believe that our structural theorems can, ultimately, be used to develop even more versatile vertex-cut data structures. For one example, is it possible to succinctly represent *all* minimum vertex cuts so that the following queries can be answered efficiently?
Is-it-a-cut?$(u_1,\ldots,u_\kappa)$**:** Return *true* iff $\{u_1,\ldots,u_\kappa\}$ forms a κ-cut.
List-cuts(u)**:** Return all the κ-cuts containing u.
Note that if all minimum cuts have a wheel or laminar relationship, then they can be represented as a tree, as in [4, 11]. Whether there is a clean representation that *also* captures all *small* and *crossing matching* cuts is an open problem.

We assumed throughout the paper that κ was not too large, specifically $\kappa < n/4$. When $n < 2\kappa$, *all* cuts are (I,κ)-small by our classification, and the classification theorem (Theorem 5) says very little about the structure of such cuts. Understanding the structure of minimum vertex cuts when κ is large, relative to n, is an interesting open problem.

References

1 Amir Abboud, Robert Krauthgamer, and Ohad Trabelsi. Cut-equivalent trees are optimal for min-cut queries. In *Proceedings of the 61st IEEE Annual Symposium on Foundations of Computer Science (FOCS)*, pages 105–118, 2020. doi:10.1109/FOCS46700.2020.00019.

2 S. Baswana, K. Choudhary, and L. Roditty. Fault tolerant subgraph for single source reachability: generic and optimal. In *Proceedings of the 48th Annual ACM Symposium on Theory of Computing (STOC)*, pages 509–518, 2016. doi:10.1145/2897518.2897648.

3 Joshua D. Batson, Daniel A. Spielman, and Nikhil Srivastava. Twice-Ramanujan sparsifiers. *SIAM J. Comput.*, 41(6):1704–1721, 2012. doi:10.1137/090772873.

4 G. Di Battista and R. Tamassia. On-line maintenance of triconnected components with SPQR-trees. *Algorithmica*, 15:302–318, 1996.

5 Giuseppe Di Battista and Roberto Tamassia. Incremental planarity testing (extended abstract). In *Proceedings of the 30th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 436–441, 1989. doi:10.1109/SFCS.1989.63515.

6 A. A. Benczúr. Counterexamples for directed and node capacitated cut-trees. *SIAM J. Comput.*, 24(3):505–510, 1995.

7 A. A. Benczúr and M. X. Goemans. Deformable polygon representation and near-mincuts. In M. Grötschel and G. O. H. Katona, editors, *Building Bridges: Between Mathematics and Computer Science*, volume 19 of *Bolyai Society Mathematical Studies*, pages 103–135. Springer, 2008.

8 András A. Benczúr and David R. Karger. Randomized approximation schemes for cuts and flows in capacitated graphs. *SIAM J. Comput.*, 44(2):290–319, 2015. doi:10.1137/070705970.

9 Chung-Kuan Cheng and T. C. Hu. Ancestor tree for arbitrary multi-terminal cut functions. *Ann. Oper. Res.*, 33(3):199–213, 1991. doi:10.1007/BF02115755.

10 K. Choudhary. An optimal dual fault tolerant reachability oracle. In *Proceedings 43rd Int'l Colloq. on Automata, Languages, and Programming (ICALP)*, 2016.

11 Robert F Cohen, Giuseppe Di Battista, Arkady Kanevsky, and Roberto Tamassia. Reinventing the wheel: an optimal data structure for connectivity queries (extended abstract). In *Proceedings of the 25th Annual ACM Symposium on Theory of Computing (STOC)*, pages 194–200, 1993.

12 William H. Cunningham and Jack Edmonds. A combinatorial decomposition theory. *Canadian J. Math.*, 32(3):734–765, 1980. doi:10.4153/CJM-1980-057-7.

13 E. A. Dinic, A. V. Karzanov, and M. V. Lomonosov. On the structure of the system of minimum edge cuts in a graph. *Studies in Discrete Optimization*, pages 290–306, 1976. (in Russian).

14 Y. Dinitz and Z. Nutov. A 2-level cactus model for the system of minimum and minimum+1 edge-cuts in a graph and its incremental maintenance. In *Proceedings 27th ACM Symposium on Theory of Computing (STOC)*, pages 509–518, 1995.

15 Y. Dinitz and Z. Nutov. A 2-level cactus tree model for the system of minimum and minimum+1 edge cuts of a graph and its incremental maintenance. Part I: the odd case. Unpublished manuscript, 1999.

16 Y. Dinitz and Z. Nutov. A 2-level cactus tree model for the system of minimum and minimum+1 edge cuts of a graph and its incremental maintenance. Part II: the even case. Unpublished manuscript, 1999.

17 Yefim Dinitz and Alek Vainshtein. The connectivity carcass of a vertex subset in a graph and its incremental maintenance. In *Proceedings of the 26th Annual ACM Symposium on Theory of Computing (STOC)*, pages 716–725, 1994. doi:10.1145/195058.195442.

18 Yefim Dinitz and Alek Vainshtein. Locally orientable graphs, cell structures, and a new algorithm for the incremental maintenance of connectivity carcasses. In *Proceedings of the 6th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 302–311, 1995. URL: http://dl.acm.org/citation.cfm?id=313651.313711.

19 Yefim Dinitz and Alek Vainshtein. The general structure of edge-connectivity of a vertex subset in a graph and its incremental maintenance. odd case. *SIAM J. Comput.*, 30(3):753–808, 2000. doi:10.1137/S0097539797330045.

20 R. Duan and S. Pettie. Connectivity oracles for failure prone graphs. In *Proceedings 42nd ACM Symposium on Theory of Computing*, pages 465–474, 2010.

21 R. Duan and S. Pettie. Connectivity oracles in graphs subject to vertex failures. *SIAM J. Comput.*, 49(6):1363–1396, 2020.

22 Donatella Firmani, Loukas Georgiadis, Giuseppe F. Italiano, Luigi Laura, and Federico Santaroni. Strong articulation points and strong bridges in large scale graphs. *Algorithmica*, 74(3):1123–1147, 2016. doi:10.1007/s00453-015-9991-z.

23 Sebastian Forster, Danupon Nanongkai, Liu Yang, Thatchaphol Saranurak, and Sorrachai Yingchareonthawornchai. Computing and testing small connectivity in near-linear time and queries via fast local cut algorithms. In *Proceedings of the 31st ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 2046–2065. SIAM, 2020. doi:10.1137/1.9781611975994.126.

24 András Frank and Tibor Jordán. Minimal edge-coverings of pairs of sets. *J. Comb. Theory, Ser. B*, 65(1):73–110, 1995. doi:10.1006/jctb.1995.1044.

25 Harold N Gabow. A representation for crossing set families with applications to submodular flow problems. In *Proceedings of the fourth annual ACM-SIAM Symposium on Discrete algorithms*, pages 202–211, 1993.

26 Harold N. Gabow. Using expander graphs to find vertex connectivity. *J. ACM*, 53(5):800–844, 2006. doi:10.1145/1183907.1183912.

27 Yu Gao, Jason Li, Danupon Nanongkai, Richard Peng, Thatchaphol Saranurak, and Sorrachai Yingchareonthawornchai. Deterministic graph cuts in subquadratic time: Sparse, balanced, and k-vertex. *CoRR*, abs/1910.07950, 2019. arXiv:1910.07950.

28 Loukas Georgiadis, Giuseppe F. Italiano, Luigi Laura, and Nikos Parotsidis. 2-edge connectivity in directed graphs. *ACM Trans. Algorithms*, 13(1):9:1–9:24, 2016. doi:10.1145/2968448.

29 Loukas Georgiadis, Giuseppe F. Italiano, Luigi Laura, and Nikos Parotsidis. 2-vertex connectivity in directed graphs. *Inf. Comput.*, 261:248–264, 2018. doi:10.1016/j.ic.2018.02.007.

30 Loukas Georgiadis, Giuseppe F. Italiano, and Nikos Parotsidis. Strong connectivity in directed graphs under failures, with applications. *SIAM J. Comput.*, 49(5):865–926, 2020. doi:10.1137/19M1258530.

31 R. E. Gomory and T. C. Hu. Multi-terminal network flows. *Journal of the Society for Industrial and Applied Mathematics*, 9, 1961.

32 Frieda Granot and Refael Hassin. Multi-terminal maximum flows in node-capacitated networks. *Discrete applied mathematics*, 13(2-3):157–163, 1986.

33 D. Gusfield and D. Naor. Efficient algorithms for generalized cut trees. In *Proceedings First ACM-SIAM Symposium on Discrete Algorithms*, pages 422–433, 1990.

34 Dan Gusfield and Dalit Naor. Extracting maximal information about sets of minimum cuts. *Algorithmica*, 10(1):64–89, 1993.

35 Refael Hassin and Asaf Levin. Flow trees for vertex-capacitated networks. *Discrete applied mathematics*, 155(4):572–578, 2007.

36 J. E. Hopcroft and R. E. Tarjan. Dividing a graph into triconnected components. *SIAM J. Comput.*, 2(3):135–158, 1973.

37 Tai-Hsin Hsu and Hsueh-I Lu. An optimal labeling for node connectivity. In *Proceedings of the 20th International Symposium on Algorithms and Computation (ISAAC)*, pages 303–310. Springer, 2009.

38 Rani Izsak and Zeev Nutov. A note on labeling schemes for graph connectivity. *Information processing letters*, 112(1-2):39–43, 2012.

39 Bill Jackson and Tibor Jordán. Independence free graphs and vertex connectivity augmentation. *Journal of Combinatorial Theory, Series B*, 94(1):31–77, 2005.

40 Tibor Jordán. On the optimal vertex-connectivity augmentation. *Journal of Combinatorial Theory, Series B*, 63(1):8–20, 1995.

41 Tibor Jordán. On the number of shredders. *Journal of Graph Theory*, 31(3):195–200, 1999.

42 A. Kanevsky, R. Tamassia, G. Di Battista, and J. Chen. On-line maintenance of the four-connected components of a graph. In *Proceedings 32nd IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 793–801, 1991.

43 B. M. Kapron, V. King, and B. Mountjoy. Dynamic graph connectivity in polylogarithmic worst case time. In *Proceedings of the 24th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1131–1142, 2013.

44 Michal Katz, Nir A Katz, Amos Korman, and David Peleg. Labeling schemes for flow and connectivity. *SIAM J. Comput.*, 34(1):23–40, 2004.

45 A. Korman. Labeling schemes for vertex connectivity. *ACM Trans. on Algorithms*, 6(2), 2010.

46 Jason Li, Danupon Nanongkai, Debmalya Panigrahi, Thatchaphol Saranurak, and Sorrachai Yingchareonthawornchai. Vertex connectivity in poly-logarithmic max-flows. In *Proceedings of the 53rd ACM Symposium on Theory of Computing (STOC)*, 2021.

47 Gilad Liberman and Zeev Nutov. On shredders and vertex connectivity augmentation. *Journal of Discrete Algorithms*, 5(1):91–101, 2007.

48 Saunders Mac Lane. A structural characterization of planar combinatorial graphs. *Duke Math. J.*, 3(3):460–472, 1937. doi:10.1215/S0012-7094-37-00336-3.

49 Karl Menger. Zur allgemeinen kurventheorie. *Fundamenta Mathematicae*, 10:96–115, 1927.

50 H. Nagamochi and T. Ibaraki. A linear-time algorithm for finding a sparse k-connected spanning subgraph of a k-connected graph. *Algorithmica*, 7(5&6):583–596, 1992.

51 Zeev Nutov. Approximating connectivity augmentation problems. *ACM Trans. Algorithms*, 6(1):5:1–5:19, 2009. doi:10.1145/1644015.1644020.

52 Zeev Nutov. Improved approximation algorithms for minimum cost node-connectivity augmentation problems. *Theory Comput. Syst.*, 62(3):510–532, 2018. doi:10.1007/s00224-017-9786-5.

53 Zeev Nutov and Masao Tsugaki. On (t, k)-shredders in k-connected graphs. *Ars Comb.*, 83, 2007.

54 M. Pătraşcu and M. Thorup. Planning for fast connectivity updates. In *Proceedings 48th IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 263–271, 2007.

55 Seth Pettie and Longhui Yin. The structure of minimum vertex cuts. *CoRR*, abs/2102.06805, 2021. arXiv:2102.06805.

56 J.-C. Picard and M. Queyranne. On the structure of all minimum cuts in a network and applications. In *Combinatorial Optimization II*, volume 13 of *Mathematical Programming Studies*, pages 8–16. Springer, 1980.

57 C.-P. Schnorr. Bottlenecks and edge connectivity in unsymmetrical networks. *SIAM J. Comput.*, 8(2):265–274, 1979.

58 R. E. Tarjan. Depth-first search and linear graph algorithms. *SIAM J. Comput.*, 1(2):146–160, 1972.

59 W. T. Tutte. A theory of 3-connected graphs. *Nederl. Akad. Wetensch. Proc. Ser. A 64 = Indag. Math.*, 23:441–455, 1961.

60 W. T. Tutte. *Connectivity in Graphs*. University of Toronto Press, 1966.

61 H. Whitney. Congruent graphs and the connectivity of graphs. *American J. Mathematics*, 54(1):150–168, 1932. doi:10.2307/2371086.

62 Hassler Whitney. Non-separable and planar graphs. *Trans. Amer. Math. Soc.*, 34(2):339–362, 1932. doi:10.2307/1989545.

Knapsack and Subset Sum with Small Items

Adam Polak
EPFL, Lausanne, Switzerland

Lars Rohwedder
EPFL, Lausanne, Switzerland

Karol Węgrzycki
Saarland University, Saarland Informatics Campus, Saarbrücken, Germany
Max Planck Institute for Informatics, Saarland Informatics Campus, Saarbrücken, Germany

Abstract

Knapsack and Subset Sum are fundamental NP-hard problems in combinatorial optimization. Recently there has been a growing interest in understanding the best possible pseudopolynomial running times for these problems with respect to various parameters.

In this paper we focus on the maximum item size s and the maximum item value v. We give algorithms that run in time $\mathcal{O}(n + s^3)$ and $\mathcal{O}(n + v^3)$ for the Knapsack problem, and in time $\widetilde{\mathcal{O}}(n + s^{5/3})$ for the Subset Sum problem.

Our algorithms work for the more general problem variants with multiplicities, where each input item comes with a (binary encoded) multiplicity, which succinctly describes how many times the item appears in the instance. In these variants n denotes the (possibly much smaller) number of *distinct* items.

Our results follow from combining and optimizing several diverse lines of research, notably proximity arguments for integer programming due to Eisenbrand and Weismantel (TALG 2019), fast structured $(\min, +)$-convolution by Kellerer and Pferschy (J. Comb. Optim. 2004), and additive combinatorics methods originating from Galil and Margalit (SICOMP 1991).

2012 ACM Subject Classification Theory of computation → Dynamic programming

Keywords and phrases Knapsack, Subset Sum, Proximity, Additive Combinatorics, Multiset

Digital Object Identifier 10.4230/LIPIcs.ICALP.2021.106

Category Track A: Algorithms, Complexity and Games

Funding *Adam Polak*: Supported by the Swiss National Science Foundation within the project *Lattice Algorithms and Integer Programming* (185030). Part of this work was done at Jagiellonian University, supported by Polish National Science Center grant 2017/27/N/ST6/01334.
Lars Rohwedder: Swiss National Science Foundation project 200021-184656.
Karol Węgrzycki: Project TIPEA that has received funding from the European Research Council (ERC) under the European Unions Horizon 2020 research and innovation programme (grant agreement No. 850979).

1 Introduction

In the Knapsack problem we are given a (multi-)set consisting of N items, where the i-th item has size s_i and value v_i, and a knapsack capacity t. The task is to find a subset of items with the maximum total value such that its total size does not exceed the capacity t. In the related Subset Sum problem we are given a (multi-)set of N positive integers and a target value t, and the task is to find a subset of integers with the total sum exactly equal to t. The Subset Sum problem can thus be seen as a decision variant of the Knapsack problem with the additional restriction that $s_i = v_i$ for every item i.

Knapsack and Subset Sum are fundamental problems in computer science and discrete optimization. They are studied extensively both from practical and theoretical points of view (see, e.g. [23] for a comprehensive monograph). The two problems are (weakly) NP-hard, and Bellman's seminal work on dynamic programming [7] gives pseudopolynomial $\mathcal{O}(Nt)$ time

algorithms for both of them. Recently there has been a growing interest in understanding the best possible pseudopolynomial running times for these problems with respect to various parameters, see, e.g., [9, 6, 16, 5, 11].

In this paper we consider binary multiplicity encoding of the Knapsack and Subset Sum instances. Each item i given in the input has a (binary encoded) positive integer multiplicity u_i, which denotes that up to u_i copies of this item can be used in a solution. In these variants n denotes the (possibly much smaller) number of distinct items and $N = \sum_{i\in[n]} u_i$.[1] Binary multiplicity encoding can be challenging because one requires the algorithm to run in polynomial time in the input size, which can be exponentially smaller compared to the naive encoding. A notable example of this setting is the breakthrough result of Goemans and Rothvoß [18] showing that Bin Packing with few different item sizes (and binary multiplicity encoding) can be solved in polynomial time.

Formally, Knapsack with multiplicities can be defined as an integer linear program: maximize $\sum_{i\in[n]} v_i x_i$ subject to $0 \leqslant x_i \leqslant u_i$, $x_i \in \mathbb{Z}$, and $\sum_{i\in[n]} s_i x_i \leqslant t$. Similarly, Subset Sum with multiplicities can be defined as a feasibility integer linear program with constraints $0 \leqslant x_i \leqslant u_i$, $x_i \in \mathbb{Z}$, and $\sum_{i\in[n]} s_i x_i = t$. Throughout the paper we use u to denote the maximum item multiplicity $\max_{i\in[n]} u_i$, and w.l.o.g. we assume that $u \leqslant t$.

1.1 Our results

We focus on pseudo-polynomial time algorithms with respect to the maximum item size $s = \max_{i\in[n]} s_i$ (or maximum item value $v = \max_{i\in[n]} v_i$, which is essentially equivalent). We note that s is a stronger parameter compared to t in the sense that s can be much smaller than t, but not vice versa. Yet, s is less well understood than t. In the regime where n is large compared to s, an $\mathcal{O}(n + \mathrm{poly}(s))$ time algorithm would be desirable. We show that the Knapsack problem can indeed be solved in such a time. Prior results (even for 0-1 Knapsack, that is, without multiplicities) only came with the form $\mathcal{O}(\mathrm{poly}(n) \cdot \mathrm{poly}(s))$ or $\mathcal{O}(\mathrm{poly}(s) \cdot \mathrm{poly}(t))$.

This raises the natural question what the best exponent in the polynomial is. In this paper we address the question from the upper bound side. We give algorithms for Knapsack running in $\mathcal{O}(n + s^3)$ and $\mathcal{O}(n + v^3)$ time, and for Subset Sum running in $\widetilde{\mathcal{O}}(n + s^{5/3})$ time[2]. Our algorithms are in the word RAM model, and we assume that each integer in the input fits in one word. In particular, arithmetic operations on integers of size polynomial in the sum of the input's integers require constant time.

Our first result is an algorithm for Knapsack. We use proximity techniques due to Eisenbrand and Weismantel [16] which allow us to prove that there is an efficiently computable solution that differs only very little from an optimal solution. Then we apply a fast algorithm for structured $(\min, +)$-convolution [22] to search for this optimal solution within the limited space. This results in a running time which is cubic in the maximum item size.

▶ **Theorem 1.1.** *Knapsack (with multiplicities) can be solved in deterministic $\mathcal{O}(n + s^3)$ time.*

The definition of (the decision variant of) the Knapsack problem is symmetric with respect to sizes and values. We give a simple transformation that allows us to apply our algorithm also to the case where the maximal item value (and not the maximal item size) is small.

---

[1] We use $[n]$ to denote $\{1, 2, \ldots, n\}$.

[2] Throughout the paper we use a $\widetilde{\mathcal{O}}(\cdot)$ notation to hide polylogarithmic factors.

▶ **Theorem 1.2.** *Knapsack (with multiplicities) can be solved in deterministic* $\mathcal{O}(n + v^3)$ *time.*

Theorem 1.1 already implies that Subset Sum can also be solved in $\mathcal{O}(n+s^3)$. Our algorithm uses as a subprocedure an $\mathcal{O}(n+st)$ time Knapsack algorithm (see Lemma 2.2). If we simply replaced it with a $\widetilde{\mathcal{O}}(N+t)$ time algorithm for Subset Sum [9], we would get a $\widetilde{\mathcal{O}}(n+s^2)$ time algorithm for Subset Sum. We improve on this by introducing a refined proximity argument that lets us further reduce an instance, where the maximum item multiplicity u can be of the order of s, to two instances with $u \ll s$ each. By combining this with additive combinatorics methods, originally developed by Galil and Margalit [17] and recently generalized by Bringmann and Wellnitz [11], we then obtain a subquadratic algorithm.

▶ **Theorem 1.3.** *Subset Sum (with multiplicities) can be solved in randomized* $\widetilde{\mathcal{O}}(n + s^{5/3})$ *time, with a one-sided error algorithm that returns a correct answer with high probability.*

All our algorithms can also retrieve a solution, without increasing the asymptotic running times. This is notable especially for our Subset Sum algorithm, in which we use as a black box the Bringmann-Wellnitz algorithm that gives only yes/no answers. We can deal with this presumable obstacle because we can afford to spend more time on retrieving a solution than the Bringmann-Wellnitz algorithm could.

A limitation of our algorithms is that they can provide an answer only for a single target value t at a time. Conversely, many (but not all) known Knapsack and Subset Sum algorithms can give answers for all target values between 0 and t at the same time. This limitation is however unavoidable: We aim at running times independent of the target value t, thus we cannot afford output size linear in t, because t cannot be bounded in terms of n and s only.

In the next section we discuss how our results fit a broader landscape of existing Knapsack and Subset Sum algorithms.

1.2 Related work

Pseudopolynomial time algorithms for Knapsack

Bellman [7] was the first to show that the Knapsack problem admits a pseudopolynomial time algorithm. He presented an $\mathcal{O}(Nt)$ time algorithm based on dynamic programming. Pisinger [28] gave an $\mathcal{O}(Nsv)$ time algorithm, which is an improvement for instances with both small sizes and small values. He proved that only *balanced feasible solutions* to Knapsack need to be considered in order to find an optimal solution. Then he used this observation to decrease the number of states of the dynamic program. His arguments may be thought of as an early example of proximity-based arguments.

Kellerer and Pferschy [22] studied approximation algorithms for Knapsack. As a subroutine they developed a $\widetilde{\mathcal{O}}(N+vp)$ time (exact) algorithm, where p denotes the optimal total value. Their algorithm can be easily modified to work in $\widetilde{\mathcal{O}}(N+st)$ time. Their approach, based on fast $(\min,+)$-convolution for structured (convex) instances was rediscovered and improved by Axiotis and Tzamos [5]. Bateni et al. [6] achieved the same $\widetilde{\mathcal{O}}(N+st)$ running time with a different method, which can be seen as a far-reaching refinement of Pisinger's idea [28]. They also developed the *prediction* technique, which let them achieve $\widetilde{\mathcal{O}}(N+vt)$ running time.

Eisenbrand and Weismantel [16] studied more general integer linear programs, and presented a $\widetilde{\mathcal{O}}(ns^2)$ time algorithm for Knapsack with multiplicities, based on proximity-based arguments. To the best of our knowledge they are the first to consider Knapsack with multiplicities. Subsequently Axiotis and Tzamos [5] improved logarithmic factors (in the

Table 1 Pseudopolynomial time algorithms for **Knapsack**. N is the total number of items, n is the number of distinct items, t is the knapsack capacity, s is the maximum size and v the maximum value of an item. Symbol (–) means that no non-trivial optimization is given for the respective regime; running times can still be derived from the trivial inequalities $n \leqslant N$ and $t \leqslant Ns$. We use symbol (‡) when Remark 1.4 applies and (†) when Remark 1.5 applies.

0-1 Knapsack	**with multiplicities**	**Reference**
$\mathcal{O}(Nt)$	$\widetilde{\mathcal{O}}(nt)^{\ddagger}$	Bellman [7]
$\mathcal{O}(Nsv)$	–	Pisinger [28]
$\widetilde{\mathcal{O}}(N+st)$	$\widetilde{\mathcal{O}}(n+st)^{\dagger}$	Kellerer and Pferschy [22], also [6, 5]
$\widetilde{\mathcal{O}}(N+vt)$	$\widetilde{\mathcal{O}}(n+vt)^{\dagger}$	Bateni et al. [6]
–	$\widetilde{\mathcal{O}}(ns^2 \min\{n,s\})$	Bateni et al. [6]
$\mathcal{O}(N \min\{s^2,v^2\})$	–	Axiotis and Tzamos [5]
–	$\widetilde{\mathcal{O}}(ns^2)$	Eisenbrand and Weismantel [16]
–	$\mathcal{O}(n+\min\{s^3,v^3\})$	**This paper**

non-multiplicity setting) and gave an $\mathcal{O}(Ns^2)$ time algorithm, which they also generalized to $\mathcal{O}(Nv^2)$ time. Bateni et al. [6] also explicitly consider the Knapsack problem with multiplicities and independently designed a $\widetilde{\mathcal{O}}(ns^2 \min\{n,s\})$ time algorithm.

Axiotis and Tzamos suggested [5, Footnote 2] that the fast convex convolution can be combined with proximity-based arguments of Eisenbrand and Weismantel [16] to obtain an algorithm for small items with running time independent of t. However, a direct application of Eisenbrand and Weismantel [16] proximity argument (see [16, Section 4.1]) reduces an instance to $t \leqslant \mathcal{O}(ns^2)$, which, in combination with $\mathcal{O}(N+st)$ algorithm, yields $\mathcal{O}(N+ns^3)$ runtime. Our algorithm improves it to $\mathcal{O}(n+s^3)$ by a more careful proximity argument and a convex convolution that explicitly handles negative items. Moreover, we show how to extend this reasoning to the multiplicity setting.

▶ Remark 1.4. Lawler [26] showed that the variant with multiplicities can be reduced to the 0-1 variant. His reduction transforms a multiset composed of at most u copies of each of n distinct numbers bounded by s into an instance of $\mathcal{O}(n \log u)$ numbers bounded by $\mathcal{O}(us)$. This easily enables us to adapt algorithms with no time dependence on s (e.g., the $\mathcal{O}(Nt)$ time algorithm of Bellman) into the setting with multiplicities (with logarithmic overhead).

▶ Remark 1.5. There is also a folklore reduction that enables us to bound $N \leqslant \widetilde{\mathcal{O}}(t)$ for the variant with multiplicities. For each $x \in [s]$ keep $\lfloor t/x \rfloor$ most profitable items of size $s_i = x$. This leaves us with at most $N \leqslant \mathcal{O}(t \log(t))$ items and does not increase the item sizes.

See Table 1 for a summary of the known results for Knapsack.

Pseudopolynomial time algorithms for Subset Sum

Subset Sum is a special case of Knapsack and we expect significantly faster algorithms for it. Pisinger's algorithm [28] runs in $\mathcal{O}(Ns)$ time for Subset Sum. The first improvement in all parameter regimes over the $\mathcal{O}(Nt)$ time algorithm of Bellman was given by Koiliaris and Xu [24]. They presented $\widetilde{\mathcal{O}}(\sqrt{Nt}+N)$, $\widetilde{\mathcal{O}}(N+t^{5/4})$ and $\widetilde{\mathcal{O}}(\Sigma)$ time deterministic algorithms for Subset Sum, where Σ is the total sum of items. A by now standard method, used by all these algorithms, is to encode an instance of Subset Sum as a convolution problem that can be solved using Fast Fourier Transform. Subsequently, Bringmann [9] presented a $\widetilde{\mathcal{O}}(N+t)$ randomized time algorithm for Subset Sum based on the color-coding technique. Jin and Wu [21] later gave an alternative $\widetilde{\mathcal{O}}(N+t)$ randomized time algorithm based on Newton's iterative method. Their proof is notable for being very compact.

Table 2 Pseudopolynomial time algorithms for **Subset Sum**. N is the total number of items, n is the number of distinct items, t is the target value, Σ is the sum of all items, s is the maximum item, and u is the maximum multiplicity of an item. Symbol (–) means that no non-trivial optimization is given for the respective regime; running times can still be derived from the trivial inequalities $n \leqslant N \leqslant nu$ and $t \leqslant Ns$. In ($\star$) the instance cannot have two items with the same size, the algorithm works only for $u = 1$. We use symbol (‡) when Remark 1.4 applies.

0-1 Subset Sum		**with Multiplicities**	**Reference**
$\mathcal{O}(Nt)$		$\widetilde{\mathcal{O}}(nt)^{\ddagger}$	Bellman [7]
$\mathcal{O}(Ns)$		–	Pisinger [28]
$\widetilde{\mathcal{O}}(N+\sqrt{Nt})$		$\widetilde{\mathcal{O}}(n+\sqrt{nt})^{\ddagger}$	Koiliaris and Xu [24]
$\widetilde{\mathcal{O}}(N+t^{5/4})$		$\widetilde{\mathcal{O}}(n+t^{5/4})^{\ddagger}$	Koiliaris and Xu [24]
$\widetilde{\mathcal{O}}(\Sigma)$		–	Koiliaris and Xu [24]
$\widetilde{\mathcal{O}}(N+t)$		$\widetilde{\mathcal{O}}(n+t)^{\ddagger}$	Bringmann [9]
$\widetilde{\mathcal{O}}(N+s^{3/2})$	$\star$	(not applicable)	Galil and Margalit [17]
–		$\widetilde{\mathcal{O}}(N+u^{1/2}s^{3/2})$	Bringmann and Wellnitz [11]
–		$\widetilde{\mathcal{O}}(n+s^{5/3})$	**This Paper**

From a different perspective, Galil and Margalit [17] used additive combinatorics methods to prove that Subset Sum can be solved in near linear time when $t \gg \Sigma s/N^2$ and all items are distinct. Very recently Bringmann and Wellnitz [11] generalized that result to multisets. Their algorithm combined with the $\widetilde{\mathcal{O}}(N+t)$ time algorithm [9] yields a $\widetilde{\mathcal{O}}(N+u^{1/2}s^{3/2})$ time algorithm for Subset Sum with multiplicities (cf., Lemma 2.4). For $u = 1$ this gives the currently fastest $\widetilde{\mathcal{O}}(N+s^{3/2})$ time algorithm (in terms of small s). With our $\widetilde{\mathcal{O}}(n+s^{5/3})$ time algorithm we improve upon their result for $u \gg s^{1/3}$. We note that even with the naive (not binary) multiplicity encoding our improvement is nontrivial, since the mentioned case with $u = 1$ requires that each item has a different size. For example, even an $\mathcal{O}(N+s^{1.99})$ time algorithm does not follow immediately from [9] when multiple items can have the same size. We discuss their additive combinatorics methods in more detail in Section 5.2.

See Table 2 for a summary of the known results for Subset Sum.

Lower bounds

Bringmann's Subset Sum algorithm [9], which runs in time $\widetilde{\mathcal{O}}(N+t)$, was shown to be near-optimal by using the modern toolset of fine-grained complexity. More precisely, any $t^{1-\varepsilon}2^{o(N)}$ algorithm for Subset Sum, for any $\varepsilon > 0$, would violate both the Strong Exponential Time Hypothesis [1] and the Set Cover Conjecture [14]. This essentially settles the complexity of the problem in the parameters N and t. These lower bounds use reductions that produce instances with $t = \widetilde{\Theta}(s)$, and therefore they do not exclude a possibility of a $\widetilde{\mathcal{O}}(N+s)$ time algorithm for Subset Sum. The question if such an algorithm exists is still a major open problem [4].

Bringmann and Wellnitz [11] excluded a possibility of a near-linear algorithm for Subset Sum in a *dense* regime. More precisely, they showed that, unless the Strong Exponential Time Hypothesis and the Strong k-Sum Hypothesis both fail, Subset Sum requires $(s\Sigma/(Nt))^{1-o(1)}$ time (where Σ is the total sum of items).

For the Knapsack problem Bellman's algorithm [7] remains optimal for the most natural parametrization by N and t. This was explained by Cygan at el. [15] and Künnemann et al. [25], who proved an $(N+t)^{2-o(1)}$ lower bound assuming the $(\min,+)$-Convolution

Conjecture. Their hardness constructions create instances of 0-1 Knapsack where s and t are $\widetilde{\Theta}(N)$. This is also the best lower bound known for Knapsack with multiplicities. In particular, an $\mathcal{O}(N + s^{2-\varepsilon})$ time algorithm is unlikely, and our $\mathcal{O}(n + s^3)$ upper bound leaves the gap for the best exponent between 2 and 3.

Other variants of Knapsack and Subset Sum

We now briefly overview other variants of Knapsack and Subset Sum, which are not directly related to our results. The Unbounded Knapsack problem is the special case with $u_i = \infty$, for all $i \in [n]$, and one can assume w.l.o.g. $N = n \leqslant s$. For that variant Tamir [31] presented an $\mathcal{O}(n^2 s^2)$ time algorithm. Eisenbrand and Weismantel [16] improved this result and gave an $\mathcal{O}(ns^2)$ time algorithm using proximity arguments. Bateni et al. [6] presented a $\widetilde{\mathcal{O}}(ns + s^2 \min\{n, s\})$ time algorithm. Then, an $\mathcal{O}(n + \min\{s^2, v^2\})$ algorithm for Unbounded Knapsack was given independently by Axiotis and Tzamos [5] and Jansen and Rohwedder [19]. Finally, Chan and He [13] gave a $\widetilde{\mathcal{O}}(ns)$ time algorithm. Unbounded Knapsack seems to be an easier problem than 0-1 Knapsack because algorithms do not need to keep track of which items are already used in partial solutions. Most of the Unbounded Knapsack techniques do not apply to 0-1 Knapsack.

In the polynomial space setting, Lokshtanov and Nederlof [27] presented a $\widetilde{\mathcal{O}}(N^4 sv)$ time algorithm for Knapsack and a $\widetilde{\mathcal{O}}(N^3 t)$ time algorithm for Subset Sum. The latter was subsequently improved by Bringmann [9], who gave a $\widetilde{\mathcal{O}}(Nt^{1+\varepsilon})$ time and $\widetilde{\mathcal{O}}(N \log t)$ space algorithm. Recently, Jin, Vyas and Williams [20] presented a $\widetilde{\mathcal{O}}(Nt)$ time and $\widetilde{\mathcal{O}}(\log(Nt))$ space algorithm (assuming a read-only access to $\widetilde{\mathcal{O}}(\log N \log\log N + \log t)$ random bits).

In the Modular Subset Sum problem, all subset sums are taken over a finite cyclic group $\mathbb{Z}_m$, for some given integer m. Koiliaris and Xu [24] gave a $\widetilde{\mathcal{O}}(m^{5/4})$ time algorithm for this problem, which was later improved by [4] to $\mathcal{O}(m \log^7 m)$. Axiotis et al. [3] independently with Cardinal and Iacono [12] simplified their algorithm and gave an $\mathcal{O}(m \log m)$ time randomized and $\mathcal{O}(m \operatorname{polylog}(m))$ deterministic time algorithms. Recently, Potępa [29] gave the currently fastest $\mathcal{O}(m \log(m)\alpha(m))$ deterministic algorithm for Modular Subset Sum (where $\alpha(m)$ is the inverse Ackerman function).

Bringmann and Nakos [10] designed a near-linear time algorithm for output sensitive Subset Sum by using additive combinatorics methods. Finally, Jansen and Rohwedder [19] considered the Unbounded Subset Sum problem and presented a $\widetilde{\mathcal{O}}(s)$ time algorithm.

2 Techniques

In this section we recall several known techniques from different fields, which we later combine as black boxes in order to get efficient algorithms in the setting with binary encoded multiplicities. We do not expect the reader to be familiar with all of them, and we include their brief descriptions for completeness. Nevertheless, it should be possible to skip reading this section and still get a high-level understanding of our results.

2.1 Proximity arguments

Now we introduce proximity arguments, which will allow us to avoid a dependency on the multiplicities u_i in the running time. Very similar arguments were used by Eisenbrand and Weismantel [16] for more general integer linear programs. We reprove them for our simpler case to make the paper self-contained.

We will show that we can efficiently compute a solution to Knapsack which differs from an optimal solution only in a few items. To this end, we define a ***maximal prefix solution*** to be a solution obtained as follows. We order the items by their efficiency, i.e. by the ratios v_i/s_i, breaking ties arbitrarily. Then, beginning with the most efficient item, we select items in the decreasing order of efficiency until the point when adding the next item would exceed the knapsack's capacity. At this point we stop and return the solution.

Note that a maximal prefix solution can be found in time $\mathcal{O}(n)$: We first select the median of v_i/s_i in time $\mathcal{O}(n)$ [8]. Then, we check if the sum of all the items that are more efficient than the median exceeds t. If so, we know that none of the other items are used in the maximal prefix solution. Otherwise, all the more efficient items are used. In both cases we can recurse on the remaining $n/2$ items. The running time is then of the form of a geometric sequence that converges to $\mathcal{O}(n)$.

▶ **Lemma 2.1** (cf., [16]). *Let p be a maximal prefix solution to Knapsack. There is an optimal solution z that satisfies $\|z-p\|_1 \leqslant 2s$, where $\|z-p\|_1$ denotes $\sum_{i\in[n]} |z_i - p_i|$.*

Proof. Let z be an optimal solution which minimizes $\|z-p\|_1$. If all the items fit into the knapsack, p and z must be equal. Otherwise, we can assume that the total sizes of both solutions, i.e. $\sum_{i\in[n]} s_i z_i$ and $\sum_{i\in[n]} s_i p_i$, are both between $t-s+1$ and t. In particular, we have that

$$-s < \sum_{i\in[n]} s_i(p_i - z_i) < s. \tag{1}$$

For the sake of the proof consider the following process. We start with the vector $z-p$, and we move its components towards zeros, carefully maintaining the bounds of (1). That is, in each step of the process, if the current sum of item sizes is positive, we reduce a positive component by 1; if the sum is negative, we increase a negative component by 1. The crucial idea is that during this process in no two steps we can have the same sum of item sizes. Otherwise, one could apply to z the additions and removals performed between the two steps, and therefore obtain another solution that is closer to p and is still optimal. Indeed, the optimiality follows from the fact that this operation does not increase the total size of the solution, and it also cannot decrease the value of the solution, because every item selected by p but not by z has efficiency no lower than every item selected by z but not by p. Hence, the number of steps, i.e., $\|z-p\|_1$, is bounded by $2s$. ◀

This lemma can be used to avoid the dependency on multiplicities $u_1, u_2, \ldots, u_n$ as follows. We compute a maximal prefix solution p. Then we know that there is an optimal solution z with

$$z_i \in \{0, \ldots, u_i\} \cap \{p_i - 2s, \ldots, p_i + 2s\} \quad \forall i \in [n].$$

Hence, we can obtain an equivalent instance by fixing the choice of some items. Formally, we remove $\max\{0, p_i - 2s\}$ many copies of item i and subtract their total size from t. If some item still has more than $4s$ copies, we can safely remove the excess. This shows that one can reduce a general knapsack instance to an instance with $u_i \leqslant 4s$ for all $i \in [n]$. In particular, a naive application of Bellman's algorithm would run in time $\mathcal{O}(t \cdot \sum_{i=1}^{n} u_i) \leqslant \mathcal{O}(n^2 s^3)$. Later in this paper we will apply the same proximity statement in more involved arguments.

2.2 Fast structured $(\min, +)$-convolution

Another technique that we use in this paper is a fast algorithm for structured instances of the $(\min, +)$-convolution problem. This technique was already applied in several algorithms for Knapsack [22, 6, 5]. We use it to find solutions for all knapsack capacities in $\{0, \dots, t\}$ in time $\mathcal{O}(n + st + t\log^2(t))$. We consider a slightly more general variant, where the values of items may also be negative and the knapsack constraint has to be satisfied with equality. To avoid confusion, we let $\bar{v}_i \in \mathbb{Z}$, $i \in [n]$, denote these possibly negative values of items.

▶ **Lemma 2.2.** *Let $\bar{v}_1, \bar{v}_2, \dots, \bar{v}_n \in \mathbb{Z}$. In time $\mathcal{O}(n + st + t\log^2(t))$ one can solve*

$$\max_{x \in \mathbb{Z}^n} \sum_{i \in [n]} \bar{v}_i x_i \qquad \text{subject to } \sum_{i \in [n]} s_i x_i = t' \qquad \text{and} \quad \forall_{i \in [n]}\ 0 \leqslant x_i \leqslant u_i, \tag{2}$$

for all $t' \in \{0, \dots, t\}$.

Proof. For $h \in [s+1]$ let $w^{(<h)} = \langle w_0^{(<h)}, w_1^{(<h)}, \dots, w_t^{(<h)} \rangle$ where $w_{t'}^{(<h)}$ denotes the value of an optimal solution to (2) for the knapsack capacity t' when restricting the instance to items i with $s_i < h$. If there is no solution satisfying the equality constraint with t', we let $w_{t'}^{(<h)} = -\infty$. Our goal is to compute the vectors $w^{(<1)}, w^{(<2)}, \dots, w^{(<h+1)}$ iteratively. We define the vector $w^{(h)} = \langle w_0^{(h)}, w_1^{(h)}, \dots, w_t^{(h)} \rangle$ which describes the optimal solutions solely of items j with $s_j = h$. For each i, the component $w_{t'}^{(h)}$ for $t' = ih$ is equal to the total value of the i most valuable items of size h, or to $-\infty$ if there are less than i items of size h. All components for indices not divisible by h are $-\infty$.

Hence, to compute $w^{(h)}$ it suffices to find the $\lceil t/h \rceil$ most valuable items of size h in the decreasing order of values. In time $\mathcal{O}(n+s)$ we partition the items by their size s_i. Extracting the t/h most valuable items for all $h \in [s]$ requires in total a time of $\mathcal{O}(n + \sum_{h \in [s]} t/h) \leqslant \mathcal{O}(n + t\log(t))$. Finally, sorting all sets takes in total $\mathcal{O}(\sum_{h \in [s]} t/h \cdot \log(t/h)) \leqslant \mathcal{O}(t\log^2(t))$ time.

Given $w^{(<h)}$ for some h we want to compute the vector $w^{(<h+1)}$ in time $\mathcal{O}(t)$. Then the lemma follows by iteratively applying this step. To this end we notice that $w^{(<h+1)}$ is precisely the $(\max, +)$-convolution of $w^{(h)}$ and $w^{(<h)}$, that is,

$$w_i^{(<h+1)} = \max\left\{ w_j^{(h)} + w_{i-j}^{(<h)} \mid j \in \{0, \dots, i\} \right\}.$$

While in general computing a $(\max, +)$-convolution is conjectured to require quadratic time [15, 25], in this case it can be done efficiently by exploiting the simple structure of $w^{(h)}$. For each remainder $r \in \{0, \dots, h-1\}$ we separately compute the entries of indices that are equal to r modulo h. We define the matrix $M \in \mathbb{Z}^{\lceil t/h \rceil \times \lceil t/h \rceil}$ with

$$M[i, j] = w_{jh+r}^{(<h)} + w_{(i-j)h}^{(h)},$$

where $w_{(i-j)h}^{(h)} = -\infty$ if $j > i$. We do not explicitly construct the matrix, but we can compute any entry of M in the constant time. To produce the vector $w^{(<h+1)}$ it suffices to find the maximum of each row of M. This can be done efficiently, since M is *inverse-Monge*, that is,

$$\begin{aligned} M[i, j] + M[i+1, j+1] &= w_{jh+r}^{(<h)} + w_{(i-j)h}^{(h)} + w_{jh+h+r}^{(<h)} + w_{(i-j)h}^{(h)} \\ &\geqslant w_{jh+r}^{(<h)} + w_{(i-j)h+h}^{(h)} + w_{jh+h+r}^{(<h)} + w_{(i-j)h-h}^{(h)} = M[i+1, j] + M[i, j+1]. \end{aligned}$$

Therefore, we can compute the row maxima in time $\mathcal{O}(t/h)$ with SMAWK algorithm [2]. This implies a total running time of $\mathcal{O}(t)$ for all remainders r and proves the lemma. ◀

2.3 Additive combinatorics

In this section, we introduce a near-linear time algorithm for dense instances of Subset Sum, more precisely, instances with $N^2 \gg us$.

The techniques behind the algorithm were introduced by Galil and Margalit [17], and recently generalized to the multiset setting by Bringmann and Wellnitz [11]. We focus on the modern description of [11], and show in Section 5.3 that in our application we can additionally report a solution.

▶ **Theorem 2.3** (Bringmann and Wellnitz [11]). *There exists $\lambda = \widetilde{\Theta}(us\Sigma/N^2)$ such that in time $\widetilde{\mathcal{O}}(N)$ we can construct a data structure that for any t satisfying $\lambda \leqslant t \leqslant \Sigma/2$ decides in time $\mathcal{O}(1)$ whether t is a subset sum.*

This is non-trivial when $\lambda \leqslant \Sigma/2$, that is when $N^2 \gg us$. We note that in the setting of Subset Sum with multiplicities Theorem 2.3 gives an $\widetilde{\mathcal{O}}(N + u^{1/2}s^{3/2})$ time algorithm. We denote by $\mathcal{S}(I)$ all subset sums of a multiset I, and we write $\Sigma(I) = \sum_{a \in I} a$.

▶ **Lemma 2.4.** *Given a multiset I of size N, in $\widetilde{\mathcal{O}}(N + s^{3/2}u^{1/2})$ time we can construct a data structure that, for any $t \in \mathbb{N}$,*
(a) *determines whether $t \in \mathcal{S}(I)$ in time $\mathcal{O}(1)$, and*
(b) *if $t \in \mathcal{S}(I)$, it finds $X \subseteq I$ with $\Sigma(X) = t$ in time $\widetilde{\mathcal{O}}(N + s^{3/2}u^{1/2})$.*

In order to prove that we can retrieve a solution with the given running time we will need to get into technical details behind the proof of Theorem 2.3. We do it in Section 5.3. Now, we sketch how to use Theorem 2.3 as a blackbox to give an $\widetilde{\mathcal{O}}(N + u^{1/2}s^{3/2})$ time algorithm that can only detect if there is a solution.

Proof of Lemma 2.4 (a). Let λ be defined as in Theorem 2.3. If the total sum of items Σ is bounded by $\widetilde{\mathcal{O}}(s^{3/2}u^{1/2})$, then we can use Bringmann's $\widetilde{\mathcal{O}}(N+t)$ time Subset Sum algorithm [9] to compute $\mathcal{S}(I)$. Therefore from now on we can assume that $s^{3/2}u^{1/2} \leqslant \widetilde{\mathcal{O}}(\Sigma) \leqslant \widetilde{\mathcal{O}}(Ns)$. In particular, this means that $\sqrt{us} \leqslant \widetilde{\mathcal{O}}(N)$. Hence,

$$\lambda \leqslant \widetilde{\mathcal{O}}\left(\frac{us\Sigma}{N^2}\right) \leqslant \widetilde{\mathcal{O}}\left(\frac{us^2}{N}\right) \leqslant \widetilde{\mathcal{O}}(u^{1/2}s^{3/2}).$$

This means that we can afford $\widetilde{\mathcal{O}}(\lambda)$ time. In time $\widetilde{\mathcal{O}}(N + \lambda)$ we find all subset sums in $\mathcal{S}(I) \cap [0, \lambda]$ using Bringmann's algorithm [9]. For $t \in [\lambda, \Sigma/2]$ we use Theorem 2.3 to decide in $\widetilde{\mathcal{O}}(N)$ time if $t \in \mathcal{S}(I)$. For $t > \Sigma/2$ we ask about $\Sigma - t$ instead. ◀

3 Knapsack with small item sizes

In this section we obtain an $\mathcal{O}(n + s^3)$ time algorithm for Knapsack by combining the proximity and convolution techniques.

Proof of Theorem 1.1. Let p be a maximal prefix solution. By Lemma 2.1 there is an optimal solution z with $\|z - p\|_1 \leqslant 2s$. We will construct an optimal solution x that is composed of three parts, that is, $x = p - x^- + x^+$. Our intuition is that x^+ is supposed to mimic the items that are included in z but not in p. We denote these items by $(z - p)_+$, where $(\cdot)_+$ takes for each component the maximum of it and 0. Likewise, x^- intuitively stands for $(p - z)_+$, the items in p but not in z.

To find x^+ and x^- we will invoke twice the $\mathcal{O}(n + st + t\log^2(t))$ time algorithm of Lemma 2.2. Let $\Delta = t - \sum_{i \in [n]} s_i p_i$, that is, the remaining knapsack capacity in the prefix solution. We can assume w.l.o.g. that $\Delta < s$, since otherwise p already includes all items and must be optimal. We use Lemma 2.2 to compute optimal solutions to the following integer programs for every $k \in \{0, \ldots, 2s^2 + \Delta\}$.

$$\max_{x\in\mathbb{Z}^n} \sum_{i\in[n]} v_i x_i \qquad \text{subject to} \qquad \sum_{i\in[n]} s_j x_i \leqslant k \quad \text{and} \quad \forall_{i\in[n]}\ 0 \leqslant x_i \leqslant u_i - p_i \tag{3}$$

$$\max_{x\in\mathbb{Z}^n} \sum_{i\in[n]} -v_i x_i \qquad \text{subject to} \qquad \sum_{i\in[n]} s_i x_i = k \quad \text{and} \quad \forall_{i\in[n]}\ 0 \leqslant x_i \leqslant p_i \tag{4}$$

We denote the resulting solutions by $x^+(k)$ and $x^-(k)$. Note, that formally the algorithm in Lemma 2.2 outputs solutions to the variant of (3) with equality, we can transform it to the above form with a single pass over the solutions.

For any k the solution $x(k) = p - x^-(k) + x^+(k+\Delta)$ is feasible. We compute values of all such solutions and select the best of them. To show that this is indeed an optimal solution, it suffices to show that for one such k the solution is optimal. Let $k = \sum_{i\in[n]} \max\{0, s_i(p_i - z_i)\} \leqslant 2s^2$, then $(x-z)_+$ is feasible for (3) with $k+\Delta$ and $(p-z)_+$ for (4) with k. Thus,

$$\begin{aligned}\sum_{i\in[n]} v_i(x(k))_i &= \sum_{i\in[n]} v_i p_i \quad - \sum_{i\in[n]} v_i (x^-(k))_i \quad + \sum_{i\in[n]} v_i (x^+(k+\Delta))_i \\ &\geqslant \sum_{i\in[n]} v_i p_i \quad - \sum_{i\in[n]} v_i \max\{0, p_i - z_i\} \quad + \sum_{i\in[n]} v_i \max\{0, z_i - p_i\} = \sum_{i\in[n]} v_i z_i.\end{aligned}$$

It remains to bound the running time. The maximal prefix solution can be found in time $\mathcal{O}(n)$. Each of the two calls to the algorithm of Lemma 2.2 takes time $\mathcal{O}(n + s^3 + s^2 \log^2(s)) = \mathcal{O}(n + s^3)$ and selecting the best solution among the $2s^2$ candidates takes time $\mathcal{O}(s^2)$. ◀

4 Knapsack with small item values

In this section we show that it is also possible to solve Knapsack in time $\mathcal{O}(n + v^3)$, proving Theorem 1.2. This can be derived directly from the $\mathcal{O}(n+s^3)$ time algorithm from the previous section. Essentially, we swap the item values and sizes by considering the complementary problem of finding the items that are not taken in the solution. Then our goal is to solve

$$\min_x \sum_{i\in[n]} v_i x_i \qquad \text{subject to} \sum_{i\in[n]} s_i x_i \geqslant \sum_{i\in[n]} u_i s_i - t \qquad \text{and} \qquad \forall_{i\in[n]} 0 \leqslant x_i \leqslant u_i. \tag{5}$$

Suppose we are satisfied with any solution that has value at least some given $v^\star$. Then this can be solved by

$$\max_x \sum_{i\in[n]} s_i x_i \qquad \text{subject to} \sum_{i\in[n]} v_i x_i \leqslant \sum_{i\in[n]} u_i v_i - v^\star \qquad \text{and} \qquad \forall_{i\in[n]} 0 \leqslant x_i \leqslant u_i.$$

Notice that this is now a Knapsack problem with item sizes bounded by v. Hence, our previous algorithm can solve it in time $\mathcal{O}(n + v^3)$. It remains to find the optimum of (5) and use it for the value of $v^\star$. Notice that the maximal prefix solution p gives a good estimate of this $v^\star$, because its value is between $v^\star - v + 1$ and $v^\star$. Thus, one could in a straight-forward way implement a binary search for $v^\star$ and this would increase the running time only by a factor of $\log(v)$, but we can avoid this and get an $\mathcal{O}(n + v^3)$ time algorithm.

It is enough to devise an algorithm that in time $\mathcal{O}(n + v^3)$ computes a solution for each of the v potential values values of $v^\star$ at once. Then we can return the largest $v^\star$ for which the solution requires a knapsack of size at most t. Fortunately, our original Theorem 1.1 can compute solution to every $t' \in \{t - v, t - v + 1, \ldots, t\}$ and the $\mathcal{O}(n + v^3)$ time algorithm for Knapsack follows.

We include a small modification of Knapsack algorithm from Section 3 for completeness.

▷ Claim 4.1. In $\mathcal{O}(n + s^3)$ time we can compute an optimal solution to Knapsack for every $t' \in \{t - s, t - s + 1, \dots, t\}$.

Proof. Recall that the intermediate solutions $x^+(k)$ and $x^-(k)$ depend only on the maximal prefix solution p and the only property of p that is needed is that it differs from the optimal solution by $\mathcal{O}(s)$ items. Notice that the maximal prefix solutions with respect to each of the values t' above differ only by at most s items. Hence, p, the prefix solution for t, differs from each of the optimal solutions only by $\mathcal{O}(s)$. Hence, we only need to compute $x^+(k)$ and $x^-(k)$ once. Given these solutions the remaining computation takes only $\mathcal{O}(s^2)$ for each t'; thus, $\mathcal{O}(s^3)$ in total. ◁

5 Subset Sum

In this section we give a $\widetilde{\mathcal{O}}(n+s^{5/3})$ time algorithm for Subset Sum with multiplicties proving Theorem 1.3. Our algorithm is a combination of additive combinatorics and proximity arguments. Throughout this section, we denote by $\mathcal{S}(A)$ the set of all subset sums of a (multi-)set of integers A. In other words, $t \in \mathcal{S}(A)$ if there exists some $B \subseteq A$ with $\Sigma(B) = t$.

Algorithm 1 $\widetilde{\mathcal{O}}(n + s^{5/3})$ time algorithm for Subset Sum with multiplicties.

Algorithm : `SubsetSum`(I, t).
Output : Multiset $X \subseteq I$ with $\Sigma(X) = t$ or `NO` if such a multiset does not exists.

Preprocess I using Lemma 2.1 so that $u_i \leqslant O(s)$ for all $i \in [n]$
Set $k := \lfloor s^{1/3} \rfloor$
Construct $I^{\uparrow} := \{(\max\{0, \lfloor u_i/k \rfloor - 8\}, s_i) \ : \ (u_i, s_i) \in I\}$
Construct $I^{\downarrow} := \{(u_i - k \cdot \max\{0, \lfloor u_i/k \rfloor - 8\}, s_i) \ : \ (u_i, s_i) \in I\}$
Construct oracle to $\mathcal{S}(I^{\downarrow})$ `// with Lemma 2.4`
Construct set of candidates $\mathcal{C}(I^{\uparrow}) \subseteq \mathcal{S}(I^{\uparrow})$ `// with Lemma 5.1`
foreach $t' \in \mathcal{C}(I^{\uparrow})$ **do**
  if $t - k \cdot t' \in \mathcal{S}(I^{\downarrow})$ **then**
    Recover $A \subseteq I^{\uparrow}$ with $\Sigma(A) = t'$
    Recover $B \subseteq I^{\downarrow}$ with $\Sigma(B) = t - k \cdot t'$
    return $(k \cdot A) \cup B$
return `NO`

We now give a high level overview of the algorithm (see Algorithm 1). In the following we assume w.l.o.g. that $u \leqslant \mathcal{O}(s)$ by using the preprocessing described after Lemma 2.1. First, we split the instance I into two parts $I^{\uparrow}$ and $I^{\downarrow}$: Let $k = \lfloor s^{1/3} \rfloor$. For every item s_i with multiplicity u_i we add $u_i^{\uparrow} = \max\{0, \lfloor u_i/k \rfloor - 8\} = \mathcal{O}(s^{2/3})$ many items of size s_i into multiset $I^{\uparrow}$. The rest of the items of size s_i, i.e., $u_i^{\downarrow} = u_i - k \cdot u_i^{\uparrow} = \mathcal{O}(s^{1/3})$ many, are added to multiset $I^{\downarrow}$. Intuitively, $I^{\uparrow}$ stands for taking bundles of k items. The set $I^{\downarrow}$ consists of the remaining items. In particular, it holds that:

$$\mathcal{S}(I) = \left\{kt^{\uparrow} + t^{\downarrow} \ : \ t^{\uparrow} \in \mathcal{S}(I^{\uparrow}) \text{ and } t^{\downarrow} \in \mathcal{S}(I^{\downarrow})\right\}.$$

Our goal is to decide whether there exists an integer $t' \in \mathcal{S}(I^{\uparrow})$ with the property that $t - kt' \in \mathcal{S}(I^{\downarrow})$. The strategy of the algorithm is as follows: we will use proximity arguments to bound the number of candidates for such a $t' \in \mathcal{S}(I^{\uparrow})$ and efficiently enumerate them in time $\widetilde{\mathcal{O}}(s^{5/3})$.

▶ **Lemma 5.1.** *In time $\widetilde{\mathcal{O}}(s^{5/3})$, we can construct $\mathcal{C}(I^\uparrow) \subseteq \mathcal{S}(I^\uparrow)$ of size $|\mathcal{C}(I^\uparrow)| \leqslant \widetilde{\mathcal{O}}(s^{5/3})$ with the property that if $t \in \mathcal{S}(I)$, then there exists $t' \in \mathcal{C}(I^\uparrow)$ and $t - kt' \in \mathcal{S}(I^\downarrow)$. Moreover, for any $t' \in \mathcal{C}(I^\uparrow)$ we can find $X \subseteq I^\uparrow$ with $\Sigma(X) = t'$ in time $\widetilde{\mathcal{O}}(s^{5/3})$.*

We will prove this lemma in Section 5.1. To check the condition $t - kt' \in \mathcal{S}(I^\downarrow)$ we observe that the set $I^\downarrow$ has bounded multiplicity and large density. To accomplish that we use Lemma 2.4. It uses a recent result of Bringmann and Wellnitz [11] and enables us to decide in constant time if $t - kt' \in \mathcal{S}(I^\downarrow)$ for any t' after a preprocessing that requires time $\widetilde{\mathcal{O}}(|I^\downarrow| + s^{3/2}(u^\downarrow)^{1/2}) \leqslant \widetilde{\mathcal{O}}(s^{5/3})$ because $|I^\downarrow| \leqslant \mathcal{O}(s \cdot u^\downarrow)$ and $u^\downarrow \leqslant \mathcal{O}(s^{1/3})$. We extend their methods to be able to construct the solution within $\widetilde{\mathcal{O}}(s^{5/3})$ time (see Section 5.3 for the proof of Lemma 2.4).

Now, we analyse the correctness of Algorithm 1. The running time is bounded by $\widetilde{\mathcal{O}}(n + s^{5/3})$ because the number of candidates is $|\mathcal{C}(I^\uparrow)| \leqslant \widetilde{\mathcal{O}}(s^{5/3})$ by Lemma 5.1. The algorithm is correct because set $\mathcal{C}(I^\uparrow)$ has the property that if an answer to the Subset Sum is positive, then there exists $t' \in \mathcal{C}(I^\uparrow)$ with $t - k \cdot t' \in \mathcal{S}(I^\downarrow)$

Moreover, when the answer is positive we have $t' \in \mathcal{C}(I^\uparrow)$ with $t - k \cdot t' \in \mathcal{S}(I^\downarrow)$. We use Lemma 5.1 to recover $A \subseteq I^\uparrow$ with $\Sigma(A) = t'$. Then we use Lemma 2.4 to find $B \subseteq I^\downarrow$ with $\Sigma(B) = t - k \cdot t'$. We construct a final solution $(k \cdot A) \cup B$ by unbundling items in A (duplicating them k times) and joining them with set B. This concludes the proof of Theorem 1.3.

5.1 Finding a small set of candidates

In this section we derive a small set of candidates $\mathcal{C}(I^\uparrow)$ and prove Lemma 5.1. This is based on the proximity result (Lemma 2.1). Recall that for p, a maximal prefix solution, we know that there exists some feasible solution that differs only by $\mathcal{O}(s)$ items (if the instance is feasible). Suppose we split p between $I^\uparrow$ and $I^\downarrow$ into $p^\uparrow$ and $p^\downarrow$. As each of the items in $I^\uparrow$ stands for k items in I, one might expect that the part of the optimal solution that comes from $I^\uparrow$ differs from $p^\uparrow$ by only $\mathcal{O}(s/k)$ items. This is not necessarily true if $p^\uparrow$ and $p^\downarrow$ are chosen unfavorably. Fortunately, we can show that with a careful choice of $p^\uparrow$ and $p^\downarrow$ it can be guaranteed.

▶ **Definition 5.2** (Robust split)**.** *Let $p = (p_1, \ldots, p_n) \in \mathbb{N}^n$ be a maximal prefix solution. Let $p^\uparrow, p^\downarrow \in \mathbb{N}^n$ be defined by*

$$p_i^\uparrow = \begin{cases} 0 & \text{if } p_i \leqslant 4k \text{ or } u_i \leqslant 8k, \\ \lfloor u_i/k \rfloor - 8 & \text{if } p_i \geqslant u_i - 4k \text{ and } u_i > 8k, \\ \lfloor p_i/k \rfloor - 2 & \text{if } p_i \in (4k, u_i - 4k) \text{ and } u_i > 8k, \end{cases}$$

and $p_i^\downarrow = p_i - kp_i^\uparrow$, for every $i \in [n]$. We call $p^\uparrow, p^\downarrow$ the robust split *of p.*

An important property of the choice of $p^\uparrow$ and $p^\downarrow$ is that there is some slack for $p_i^\downarrow$. Namely, if we were to change p_i slightly (say, by less than k) then we only need to change $p_i^\downarrow$ (and do not need to change $p_i^\uparrow$) to maintain $p_i = kp_i^\uparrow + p_i^\downarrow$.

▶ **Lemma 5.3.** *Let p be a maximal prefix solution and let $p^\uparrow, p^\downarrow$ be its robust split. If $t \in \mathcal{S}(I)$, then there are solutions $x^\uparrow, x^\downarrow$ of $I^\uparrow$ and $I^\downarrow$ such that:*

$$\sum_{i \in [n]} (kx_i^\uparrow + x_i^\downarrow)s_i = t \quad \text{and} \quad \|x^\uparrow - p^\uparrow\|_1 \leqslant \mathcal{O}(s/k).$$

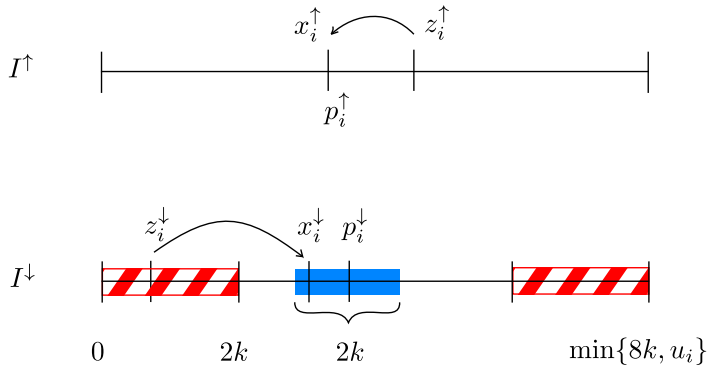


Figure 1 Schematic idea behind the proof of Lemma 5.3 in the case $u_i > 8k$. We select $p_i^\downarrow$ such that it is never in the red hatched regions. The property of $p^\uparrow, p^\downarrow$ is that any solution that differs by at most k elements can take the same elements from $I^\uparrow$ as $p_i^\uparrow$. In that case the situation where the optimal solution is of the form z (in the figure) can always be avoided to instead get the situation with x.

Proof. The proof consists of straightforward, but tedious, calculations; see Figure 1 for an intuition.

By Lemma 2.1 we know that there is a feasible solution x with $\|x - p\|_1 \leqslant \mathcal{O}(s)$. Let us use this solution and construct $x^\uparrow$ and $x^\downarrow$. We consider each index $i \in [n]$ individually and show that there is a split of x_i into $x_i^\uparrow$, $x_i^\downarrow$ which are feasible for $I^\uparrow$, $I^\downarrow$, that is, they satisfy the bounds $[0, u_i^\uparrow]$ and $[0, u_i^\downarrow]$, and we have that $|x_i^\uparrow - p_i^\uparrow| \leqslant 19|x_i - p_i|/k$. This implies the lemma. To this end, consider two cases based on $|x_i - p_i|$.

Case 1: $|x_i - p_i| < k$. In this case we set $x_i^\uparrow = p_i^\uparrow$ and $x_i^\downarrow = x_i - kx_i^\uparrow$. This choice does not contribute anything to the norm $\|x^\uparrow - p^\uparrow\|_1$ because $x_i^\uparrow = p_i^\uparrow$. We need to show that $x_i^\uparrow$ and $x_i^\downarrow$ are feasible solutions to the subset sum instances $I^\uparrow$ and $I^\downarrow$.

Clearly, $x_i^\uparrow$ is between 0 and $u_i^\uparrow = \max\{\lfloor u_i/k \rfloor - 8, 0\}$ in the first two cases of Definition 5.2. In the last case, we have that $4k < p_i < u_i - 4k$ and therefore $0 \leqslant \lfloor p_i/k \rfloor - 2 = p_i^\uparrow$. Furthermore, $p_i^\uparrow = \lfloor p_i/k \rfloor - 2 < p_i/k - 3 < u_i/k - 7 < \lfloor u_i/k \rfloor - 8 = u_i^\uparrow$.

Therefore, it remains to show that $x_i^\downarrow$ is feasible for $I^\downarrow$. More precisely, we will prove that:

$$0 \leqslant x_i - kp_i^\uparrow \leqslant u_i - k \cdot \max\{\lfloor u_i/k \rfloor - 8, 0\}.$$

To achieve that, we will crucially rely on our choice for $p^\uparrow$ in Definition 5.2.

Case 1a: $u_i \leqslant 8k$ **or** $p_i \leqslant 8k$. When $u_i \leqslant 8k$ or $p_i \leqslant 8k$ then the claim follows because $p_i^\uparrow = x_i^\uparrow = 0$.

Case 1b: $p_i \geqslant u_i - 4k$ **and** $u_i > 8k$. In this case we have $p_i^\uparrow := \lfloor u_i/k \rfloor - 8$. Inequality $x_i - kp_i^\uparrow \geqslant 0$ follows from $x_i > p_i - k \geqslant u_i - 5k \geqslant k\lfloor u_i/k \rfloor - 8k = kp_i^\uparrow$. Next, we use the fact $x_i \leqslant u_i$ to conclude $x_i - k\lfloor u_i/k \rfloor + 8k \leqslant u_i - k\lfloor u_i/k \rfloor + 8k$.

Case 1c: $4k < p_i < u_i - 4k$ **and** $u_i > 8k$. In this case we have $p_i^\uparrow := \lfloor p_i/k \rfloor - 2$. The inequality $x_i > p_i - k \geqslant k\lfloor p_i/k \rfloor - k = kp_i^\uparrow + k$ shows that $x_i - kp_i^\uparrow \geqslant 0$. Next, we use inequality $p_i - k\lfloor p_i/k \rfloor < k$ to show that

$$x_i - k\lfloor p_i/k \rfloor \leqslant p_i - k\lfloor p_i/k \rfloor + k \leqslant 2k < \underbrace{u_i - k\lfloor u_i/k \rfloor}_{\geqslant 0} + 8k.$$

Case 2: $|x_i - p_i| \geqslant k$. We set $x_i^{\uparrow} := \min\{\lfloor x_i/k \rfloor, u_i^{\uparrow}\}$ and $x_i^{\downarrow} := x_i - kx_i^{\uparrow}$. Clearly, $0 \leqslant x_i^{\uparrow} \leqslant u_i^{\uparrow}$. Furthermore, $x_i^{\downarrow} \geqslant 0$ and if $x_i^{\uparrow} = u_i^{\uparrow}$ then $x_i^{\downarrow} \leqslant u_i - ku_i^{\uparrow} = u_i^{\downarrow}$; otherwise, $x_i^{\downarrow} = x_i - k\lfloor x_i/k \rfloor < k < u_i^{\downarrow}$.

It remains to bound the difference between $x_i^{\uparrow}$ and $p_i^{\uparrow}$. Note that because of our choice of $p^{\uparrow}$ we have $|p_i^{\uparrow} - \lfloor p_i/k \rfloor| \leqslant 8$. Also, $|u_i^{\uparrow} - \lfloor u_i/k \rfloor| \leqslant 8$. Therefore,

$$|x_i^{\uparrow} - p_i^{\uparrow}| \leqslant \left|\min\{\lfloor x_i/k \rfloor, u_i^{\uparrow}\} - \lfloor p_i/k \rfloor\right| + 8 \leqslant \Big|\underbrace{\min\{\lfloor x_i/k \rfloor, \lfloor u_i/k \rfloor\}}_{=\lfloor x_i/k \rfloor} - \lfloor p_i/k \rfloor\Big| + 16$$
$$\leqslant |x_i - p_i|/k + 18.$$

Recall that we assumed $|x_i - p_i| \geqslant k$. Thus,

$$|x_i^{\uparrow} - p_i^{\uparrow}| \leqslant 19|x_i - p_i|/k. \qquad \blacktriangleleft$$

Now we are ready to proceed with the algorithmic part and prove Lemma 5.1.

Proof of Lemma 5.1. Let t_p be the value of $p^{\uparrow}$ in $I^{\uparrow}$, that is, $t_p := \sum_{i\in[n]} p_i^{\uparrow} s_i$. Let A^- be the multiset of numbers selected in $p^{\uparrow}$. Moreover, let A^+ denote all other elements in $I^{\uparrow}$. We now compute

$$S^+ := \mathcal{S}(A^+) \cap [c \cdot s^{5/3}] \quad \text{and} \quad S^- := \mathcal{S}(A^-) \cap [c \cdot s^{5/3}].$$

Here c is a constant that we will specify later. These sets can be computed in time $\widetilde{\mathcal{O}}(s^{5/3})$ with Bringmann's algorithm [9]. Next, using FFT we compute the sumset

$$\mathcal{C}(I^{\uparrow}) := \{t_p + a - b \mid a \in S^+, b \in S^-\}.$$

Any element in $\mathcal{C}(I^{\uparrow})$ is an integer of the form $t_p + a - b$, integer a is the sum of elements not in $p^{\uparrow}$, and integer $t_p - b$ is the contribution of elements in $p^{\uparrow}$. This operation takes time $\widetilde{\mathcal{O}}(s^{5/3})$ because the range of values of S^+ and S^- is bounded by $\mathcal{O}(s^{5/3})$. We return set $\mathcal{C}(I^{\uparrow})$ as the set of possible values of the candidates. To recover a solution we will use the fact that Bringmann's algorithm can recover solutions and the property that we can find a witness to FFT computation in linear time. Since S^+ and S^- are subsets of $[c \cdot s^{5/3}]$ we know that $\mathcal{C}(I^{\uparrow})$ is a subset of $\{t_p - c \cdot s^{5/3}, \ldots, t_p + c \cdot s^{5/3}\}$. In particular, $|\mathcal{C}(I^{\uparrow})| \leqslant \widetilde{\mathcal{O}}(s^{5/3})$.

It remains to to show that $\mathcal{C}(I^{\uparrow})$ contains some t' such that $t - kt' \in \mathcal{S}(I^{\downarrow})$ if $t \in \mathcal{S}(I)$. By Lemma 5.3 it suffices to show that $\mathcal{C}(I^{\uparrow})$ contains all values $\sum_{i\in[n]} x_i^{\uparrow} s_i$ for $x^{\uparrow}$ with $\|x^{\uparrow} - p^{\uparrow}\|_1 \leqslant c \cdot s/k$, where c is the constant in the lemma. This holds because S^+ contains $\sum_{i\in[n]:x_i^{\uparrow}\geqslant p_i^{\uparrow}} (x_i^{\uparrow} - p_i^{\uparrow}) s_i$ and S^- contains $\sum_{i\in[n]:x_i^{\uparrow}\leqslant p_i^{\uparrow}} (p_i^{\uparrow} - x_i^{\uparrow}) s_i$. $\blacktriangleleft$

5.2 Introduction to additive combinatorics methods

In this section we review the structural ideas behind the proof of Theorem 2.3. Next, in Section 5.3 we show how to use them to recover a solution to Subset Sum with multiplicities.

The additive combinatorics structure that we explore is present in the regime when $N^2 \gg us$. We formalize this assumption as follows:

▶ **Definition 5.4** (Density). *We say that a multiset X is δ-dense if it satisfies $|X|^2 \geqslant \delta us$.*

Note that if all numbers in X are divisible by the same integer d, then the solutions to Subset Sum are divisible by d. Intuitively, this situation is undesirable, because our goal is to exploit the density of the instance. With the next definition we quantify how close we are to the case where almost all numbers in X are divisible by the same number.

▶ **Definition 5.5** (Almost Divisor). *We write $X(d) := X \cap d\mathbb{Z}$ to denote the multiset of all numbers in X that are divisible by d and $\overline{X(d)} := X \setminus X(d)$ to denote the multiset of all numbers in X not divisible by d. We say that an integer $d > 1$ is an α-almost divisor of X if $|\overline{X(d)}| \leqslant \alpha u \Sigma(X)/|X|^2$.*

Bringmann and Wellnitz [11] show that this situation is not the hardest case.

▶ **Lemma 5.6** (Algorithmic Part of [11]). *Given an $\widetilde{\Theta}(1)$-dense multiset X of size N in time $\widetilde{\mathcal{O}}(N)$ we can compute an integer $d \geqslant 1$, such that $X' := X(d)/d$ is $\widetilde{\Theta}(1)$-dense and has no $\widetilde{\Theta}(1)$-almost divisors.*

They achieve that with novel prime factorization techniques. This lemma allows them essentially to reduce to the case that there are no $\widetilde{\Theta}(1)$-almost divisors. We will give more details on this in the end of Section 2.3. In our proofs we use the Lemma 5.6 as a blackbox. Next, we focus on the structural part of their arguments.

Structural part

The structural part of [11] states the surprising property. If we are given a sufficiently dense instance with no almost divisors then every set with target within the given region is attainable.

▶ **Theorem 5.7** (Structural Part of [11]). *If X is $\widetilde{\Theta}(1)$-dense and has no $\widetilde{\Theta}(1)$-almost divisor then $[\lambda_X, \dots, \Sigma(X) - \lambda_X] \subseteq \mathcal{S}(X)$ for some $\lambda_X = \widetilde{\Theta}(us\Sigma(X)/|X|^2))$.*

Therefore, Bringmann and Wellnitz [11] after the reduction to the almost-divisor-free setting can simply output `YES` on every target in the selected region. Our goal is to recover the solution to subset sum and therefore, we need to get into the details of this proof and show that we can efficiently construct it. The crucial insight into this theorem is the following decomposition of a dense multiset.

▶ **Lemma 5.8** (Decomposition, see [11, Theorem 4.35]). *Let X be a $\widetilde{\Theta}(1)$-dense multiset of size n that has no $\widetilde{\Theta}(1)$-almost divisor. Then there exists a partition $X = R \uplus A \uplus G$ and an integer $\kappa = \widetilde{\mathcal{O}}(u\Sigma(X)/|X|^2)$ such that:*

- *set $\mathcal{S}(R)$ is κ-complete, i.e., $\mathcal{S}(R) \bmod \kappa = \mathbb{Z}_\kappa$,*
- *set $\mathcal{S}(A)$ contains an arithmetic progression $\mathcal{P}$ of length $2s$ and step size κ satisfying $\max\{\mathcal{P}\} \leqslant \widetilde{\mathcal{O}}(us\Sigma(X)/|X|^2)$,*
- *the multiset G has sum $\Sigma(G) \geqslant \Sigma(X)/2$.*

Now, we sketch the proof of Theorem 5.7 with the decomposition from Lemma 5.8. This is based on the proof in [11].

Sketch of the proof of Theorem 5.7 assuming Lemma 5.8. We show that any target $t \in [\lambda_X, \dots, \Sigma(X) - \lambda_X]$ is a subset sum of X. For that we will assume without loss of generality that $t \leqslant \Sigma(X)/2$ (note that t is a subset sum if and only if $\Sigma(X) - t$ is). By Lemma 5.8 we get a partition of X into $R \uplus A \uplus G$. We know that $\mathcal{S}(A)$ contains an arithmetic progression $\mathcal{P} \subseteq \mathcal{S}(A)$, with $\mathcal{P} = \{a + \kappa, a + 2\kappa, \dots, a + 2s\kappa\}$. We construct a subset of X that sums to t as follows. First, we greedily pick $G' \subseteq G$ by iteratively adding elements until:

$$t - \Sigma(G') \in [a + \kappa(s+1), a + \kappa(s+1) + s].$$

This is possible because the largest element is bounded by s, t is at most $\Sigma(X)/2 \leqslant \Sigma(G)$, and λ_X is selected such that:

$$t \geqslant \lambda_X \geqslant a + \kappa(s+1).$$

The next step is to select a subset $R' \subseteq R$ that sums up to a number congruent to $(t - \Sigma(G') - a)$ modulo κ. Recall, that set R is κ-complete, hence such a set must exist. Moreover, w.l.o.g. $R' < \kappa$, and $\Sigma(R') < \kappa s$. Therefore, we need extra elements of total sum

$$t - \Sigma(G' \cup R') \in [a + \kappa, a + 2\kappa s], \quad \text{and} \quad t - \Sigma(G' \cup R') \equiv a \mod \kappa.$$

Finally, we note that this is exactly the range of elements of the arithmetic progression $\mathcal{P}$. It means that we can pick a subset $A' \subseteq A$, that gives the appropriate element of the arithmetic progression $\mathcal{P}$ and $t = \Sigma(G' \cup R' \cup A')$. ◀

5.3 Recovering a solution

In this section, we show how to recover a solution to Subset Sum. We need to overcome several technical difficulties. First, we need to reanalyze Lemma 5.8 and show that the partition $X = A \uplus R \uplus G$ can be constructed efficiently. This step follows directly from [11]. However, we do not know of an efficient way to construct κ and a. We show that we do not really need it. Intuitively, for our application we can afford to spend a time $\widetilde{\mathcal{O}}(N + \lambda_I)$. This observation enables us to use the $\widetilde{\mathcal{O}}(N + t)$ time algorithm of Bringmann [9] to reconstruct the solution. We commence with the observation that the decomposition into $R \uplus A \uplus G$ can be constructed within the desired time.

▷ Claim 5.9 (Recovering decomposition). Let X be a $\widetilde{\Theta}(1)$-dense multiset that has no $\widetilde{\Theta}(1)$-almost divisor. Let $K = 42480 \cdot u \cdot \Sigma(X) \log(2u)/|X|^2$. Then in $\widetilde{\mathcal{O}}(N + u\Sigma(X)/|X|^2)$ time we can explicitly find a partition $X = R \uplus A \uplus G$ such that:

- set $\mathcal{S}(R)$ is d-complete for any $d \leqslant K$,
- there exists an integer $\kappa \leqslant K$ such that the set $\mathcal{S}(A)$ contains an arithmetic progression $\mathcal{P}$ of length $2s$ and step size κ satisfying $\max\{\mathcal{P}\} \leqslant \widetilde{\mathcal{O}}(us\Sigma(X)/|X|^2)$,
- the multiset G has sum $\Sigma(G) \geqslant \Sigma(X)/2$.

Proof of Claim 5.9 based on arguments from [11]. We follow the proof of Theorem 4.35 in [11]. We focus on the construction of partition X and present only how the construction follows from [11].

To construct the set R, Bringmann and Wellnitz use [11, Theorem 4.20]. We start by picking an arbitrary subset $R' \subseteq X$ of size $\tau = \widetilde{\Theta}(u\Sigma(X)/|X|^2)$. Next we generate the set S of all prime numbers p with $p \leqslant \tau$. We can do this in $\widetilde{\mathcal{O}}(\tau)$ time by the sieve of Eratosthenes algorithm [30]. Then, we compute the prime factorization of every number in R' by [11, Theorem 3.8] in $\widetilde{\mathcal{O}}(\tau)$ time. This enables us to construct the set P of primes p with $p \leqslant \tau$ such that every $p \in P$ does not divide at least $\tau/2$ numbers in R'. Bringmann and Wellnitz [11] show that $|P| \leqslant 2 \log s$. Next, for every $p \in P$ we select an arbitrary subset $R_p \subseteq \overline{X(p)}$ of size $|R_p| = \tau$. This can be done in $\widetilde{\mathcal{O}}(N)$ time because $|P| = \widetilde{\mathcal{O}}(1)$. Finally Bringmann and Wellnitz [11] construct $R = R' \cup \left(\bigcup_{p \in P} R_p\right)$. See Theorem 4.20 in [11] for a proof that the constructed R is d-complete for any $d \leqslant K$.

To construct set A we do exactly the same as Bringmann and Wellnitz [11] and we pick at most $\lfloor n/4 \rfloor$ smallest elements from $X \setminus R$. At the end we set $G = X \setminus (R \cup A)$. See [11] for a proof that A and G have a desired properties. ◁

Now, we are ready to prove our result about recovering the solution

▶ **Lemma 5.10** (Restatement of Lemma 2.4). *Given a multiset I of N elements, in $\widetilde{\mathcal{O}}(N + s^{3/2}u^{1/2})$ time we can construct a data structure that for any $t \in \mathbb{N}$ can decide if $t \in \mathcal{S}(I)$ in time $\mathcal{O}(1)$. Moreover if $t \in \mathcal{S}(I)$ in $\widetilde{\mathcal{O}}(N + s^{3/2}u^{1/2})$ time we can find $X \subseteq I$ with $\Sigma(X) = t$.*

In the rest of this section we prove Lemma 2.4. We assume that all considered t are at most $\Sigma(I)/2$ because for larger t we can ask about $\Sigma(I) - t$ instead. Let λ_I and be defined as in Theorem 2.3. When the total sum of the elements $\Sigma(I)$ is bounded by $\widetilde{\mathcal{O}}(s^{3/2}u^{1/2})$, Bringmann's algorithm [9] computes $\mathcal{S}(I)$ and retrieves the solution in the declared time. Therefore, we can assume that $s^{3/2}u^{1/2} \leqslant \widetilde{\mathcal{O}}(|I|s)$. In particular, this means that $\sqrt{us} \leqslant \widetilde{\mathcal{O}}(|I|)$ and the set I is $\widetilde{\Theta}(1)$-dense. Hence,

$$\lambda_I \leqslant \widetilde{\mathcal{O}}\left(\frac{us\Sigma(I)}{|I|^2}\right) \leqslant \widetilde{\mathcal{O}}\left(\frac{us^2}{|I|}\right) \leqslant \widetilde{\mathcal{O}}(u^{1/2}s^{3/2}).$$

This means, that we can afford $\widetilde{\mathcal{O}}(\lambda_I)$ time. In time $\widetilde{\mathcal{O}}(|I| + \lambda_I)$ we can find all subset sums in $\mathcal{S}(I)$ that are smaller than $\widetilde{\Theta}(\lambda_I)$ using Bringmann's algorithm. Additionally, when $t \leqslant \widetilde{\Theta}(\lambda_I)$ Bringmann's algorithm can find $X \subseteq I$ in the output sensitive time.

We are left with answering queries about targets greater than λ_I. To achieve that within $\widetilde{\mathcal{O}}(\lambda_I)$-time preprocessing we use the Additive Combinatorics result from [11].

Observe, that if we were interested in a data structure that works in $\widetilde{\mathcal{O}}(N + \lambda_I) \leqslant \widetilde{\mathcal{O}}(N + s^{3/2}u^{1/2})$ and *decides* if $t \in \mathcal{S}(I)$ we can directly use [11] as a blackbox. Therefore, from now, we show that we can also reconstruct the solution in $\widetilde{\mathcal{O}}(N + \lambda_I)$ time.

▷ Claim 5.11. We can find a set $X \subseteq I$ with $\Sigma(X) = t$ in $\widetilde{\mathcal{O}}(N + \lambda_I)$ time for any $t \in [\lambda_I, \Sigma(I)/2]$ if such an X exists.

Proof. First, we use Lemma 5.6 to find an integer $d \geqslant 1$, such that set $I' := I(d)/d$ has no $\widetilde{\Theta}(1)$-almost divisor and is $\widetilde{\Theta}(1)$-dense. Observe, that the integer d can be found in $\widetilde{\mathcal{O}}(N)$ time and set I' can be constructed in $\widetilde{\mathcal{O}}(N)$ time.

Bringmann and Wellnitz [11, Theorem 3.5] prove that (recall that I is $\widetilde{\Theta}(1)$-dense and $t \geqslant \lambda_I$):

$$t \in \mathcal{S}(I) \text{ if and only if } t \bmod d \in (\mathcal{S}(I) \bmod d).$$

Therefore, the first step of our algorithm is to use Axiotis et al. [4] to decide if $t \bmod d \in (\mathcal{S}(I) \bmod d)$ and recover set $D \subseteq I \setminus I(d)$ if such a set exists (Axiotis et al. [4] enables to recover solution and by density assumption it works in $\widetilde{\mathcal{O}}(N)$ time).

If such a set exists by reasoning in [11] we know that a solution must exist (otherwise we output `NO`). We are left with recovering set $K \subseteq I'$ with $\Sigma(K) := (t - \Sigma(D))/d$.

Next, we use Claim 5.9 to find a partition $I' = R \uplus A \uplus G$. We can achieve that in $\widetilde{\mathcal{O}}(u\Sigma(I')/|I'|^2) \ll \widetilde{\mathcal{O}}(N + \lambda_I)$ time. Ideally, we would like to repeat the reasoning presented in the proof of Theorem 5.7. Unfortunately, we do not know how to explicitly construct a and κ within the given time. Nevertheless, Theorem 5.7 guarantees that $t \in \mathcal{S}(I)$ and that such integers a and κ exist.

Based on the properties of sets in Theorem 5.7 we have

$$t \in \mathcal{S}(G \uplus R \uplus A) = \mathcal{S}(G) \oplus (\mathcal{S}(R \cup A) \cap [0, \widetilde{\mathcal{O}}(\lambda_I)]).$$

With a Bringmann's algorithm we can compute the set $T := \mathcal{S}(R \cup A) \cap [0, \widetilde{\mathcal{O}}(\lambda_I)]$ in $\widetilde{\mathcal{O}}(N + \lambda_I)$ time. Now, recall that in the proof of Theorem 5.7 we have chosen set $G' \subseteq G$ greedily to satisfy $\Sigma(G') \geqslant t - a - \kappa(s + 1)$ for some a, κ and s. Therefore it is enough to iterate over every greedily chosen $G' \subseteq G$ and check whether $t - \Sigma(G') \in T$. Notice that there are only N options for G' that can be generated in $\mathcal{O}(N + \lambda_I)$ time. For each of them we can check whether $t - \Sigma(G') \in T$ in $\mathcal{O}(1)$ time because we have access to T. If we find such a set, we just report $K := G' \cup T'$, where $T' \subseteq R \cup A$ with $\Sigma(T') + \Sigma(G') = t$. ◁

This concludes the proof of Lemma 2.4.

References

1 Amir Abboud, Karl Bringmann, Danny Hermelin, and Dvir Shabtay. SETH-based lower bounds for subset sum and bicriteria path. In *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2019*, pages 41–57. SIAM, 2019. doi:10.1137/1.9781611975482.3.

2 Alok Aggarwal, Maria M. Klawe, Shlomo Moran, Peter W. Shor, and Robert E. Wilber. Geometric applications of a matrix-searching algorithm. *Algorithmica*, 2:195–208, 1987. doi:10.1007/BF01840359.

3 Kyriakos Axiotis, Arturs Backurs, Karl Bringmann, Ce Jin, Vasileios Nakos, Christos Tzamos, and Hongxun Wu. Fast and simple modular subset sum. In *4th Symposium on Simplicity in Algorithms, SOSA 2021*, pages 57–67. SIAM, 2021. doi:10.1137/1.9781611976496.6.

4 Kyriakos Axiotis, Arturs Backurs, Ce Jin, Christos Tzamos, and Hongxun Wu. Fast modular subset sum using linear sketching. In *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2019*, pages 58–69, 2019. doi:10.1137/1.9781611975482.4.

5 Kyriakos Axiotis and Christos Tzamos. Capacitated dynamic programming: Faster knapsack and graph algorithms. In *46th International Colloquium on Automata, Languages, and Programming, ICALP 2019*, volume 132 of *LIPIcs*, pages 19:1–19:13, 2019. doi:10.4230/LIPIcs.ICALP.2019.19.

6 MohammadHossein Bateni, MohammadTaghi Hajiaghayi, Saeed Seddighin, and Cliff Stein. Fast algorithms for knapsack via convolution and prediction. In *Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2018*, pages 1269–1282, 2018. doi:10.1145/3188745.3188876.

7 Richard Bellman. *Dynamic Programming*. Princeton University Press, Princeton, NJ, USA, 1957.

8 Manuel Blum, Robert W. Floyd, Vaughan R. Pratt, Ronald L. Rivest, and Robert Endre Tarjan. Time bounds for selection. *J. Comput. Syst. Sci.*, 7(4):448–461, 1973. doi:10.1016/S0022-0000(73)80033-9.

9 Karl Bringmann. A near-linear pseudopolynomial time algorithm for subset sum. In *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2017*, pages 1073–1084. SIAM, 2017. doi:10.1137/1.9781611974782.69.

10 Karl Bringmann and Vasileios Nakos. Top-k-convolution and the quest for near-linear output-sensitive subset sum. In *Proccedings of the 52nd Annual ACM SIGACT Symposium on Theory of Computing, STOC 2020*, pages 982–995. ACM, 2020. doi:10.1145/3357713.3384308.

11 Karl Bringmann and Philip Wellnitz. On near-linear-time algorithms for dense subset sum. In *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms, SODA 2021*, pages 1777–1796. SIAM, 2021. doi:10.1137/1.9781611976465.107.

12 Jean Cardinal and John Iacono. Modular subset sum, dynamic strings, and zero-sum sets. In *4th Symposium on Simplicity in Algorithms, SOSA 2021*, pages 45–56. SIAM, 2021. doi:10.1137/1.9781611976496.5.

13 Timothy M. Chan and Qizheng He. More on change-making and related problems. In *28th Annual European Symposium on Algorithms, ESA 2020*, pages 29:1–29:14, 2020. doi:10.4230/LIPIcs.ESA.2020.29.

14 Marek Cygan, Holger Dell, Daniel Lokshtanov, Dániel Marx, Jesper Nederlof, Yoshio Okamoto, Ramamohan Paturi, Saket Saurabh, and Magnus Wahlström. On problems as hard as CNF-SAT. *ACM Trans. Algorithms*, 12(3):41:1–41:24, 2016. doi:10.1145/2925416.

15 Marek Cygan, Marcin Mucha, Karol Węgrzycki, and Michał Włodarczyk. On problems equivalent to $(\min, +)$-convolution. *ACM Trans. Algorithms*, 15(1):14:1–14:25, 2019. doi:10.1145/3293465.

16 Friedrich Eisenbrand and Robert Weismantel. Proximity results and faster algorithms for integer programming using the Steinitz lemma. *ACM Trans. Algorithms*, 16(1):5:1–5:14, 2020. doi:10.1145/3340322.

17 Zvi Galil and Oded Margalit. An almost linear-time algorithm for the dense subset-sum problem. *SIAM J. Comput.*, 20(6):1157–1189, 1991. doi:10.1137/0220072.

18 Michel X. Goemans and Thomas Rothvoss. Polynomiality for bin packing with a constant number of item types. *J. ACM*, 67(6):38:1–38:21, 2020. doi:10.1145/3421750.

19 Klaus Jansen and Lars Rohwedder. On integer programming and convolution. In *10th Innovations in Theoretical Computer Science Conference, ITCS 2019*, pages 43:1–43:17, 2019. doi:10.4230/LIPIcs.ITCS.2019.43.

20 Ce Jin, Nikhil Vyas, and Ryan Williams. Fast low-space algorithms for subset sum. In *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms, SODA 2021*, pages 1757–1776. SIAM, 2021. doi:10.1137/1.9781611976465.106.

21 Ce Jin and Hongxun Wu. A simple near-linear pseudopolynomial time randomized algorithm for subset sum. In *2nd Symposium on Simplicity in Algorithms, SOSA@SODA 2019*, pages 17:1–17:6, 2019. doi:10.4230/OASIcs.SOSA.2019.17.

22 Hans Kellerer and Ulrich Pferschy. Improved dynamic programming in connection with an FPTAS for the knapsack problem. *J. Comb. Optim.*, 8(1):5–11, 2004. doi:10.1023/B:JOCO.0000021934.29833.6b.

23 Hans Kellerer, Ulrich Pferschy, and David Pisinger. *Knapsack problems*. Springer, 2004. doi:10.1007/978-3-540-24777-7.

24 Konstantinos Koiliaris and Chao Xu. Faster pseudopolynomial time algorithms for subset sum. *ACM Trans. Algorithms*, 15(3):40:1–40:20, 2019. doi:10.1145/3329863.

25 Marvin Künnemann, Ramamohan Paturi, and Stefan Schneider. On the fine-grained complexity of one-dimensional dynamic programming. In *44th International Colloquium on Automata, Languages, and Programming, ICALP 2017*, pages 21:1–21:15, 2017. doi:10.4230/LIPIcs.ICALP.2017.21.

26 Eugene L. Lawler. Fast approximation algorithms for knapsack problems. *Math. Oper. Res.*, 4(4):339–356, 1979. doi:10.1287/moor.4.4.339.

27 Daniel Lokshtanov and Jesper Nederlof. Saving space by algebraization. In *Proceedings of the 42nd ACM Symposium on Theory of Computing, STOC 2010*, pages 321–330. ACM, 2010. doi:10.1145/1806689.1806735.

28 David Pisinger. Linear time algorithms for knapsack problems with bounded weights. *J. Algorithms*, 33(1):1–14, 1999. doi:10.1006/jagm.1999.1034.

29 Krzysztof Potępa. Faster deterministic modular subset sum, 2020. arXiv:2012.06062.

30 Jonathan Sorenson. An introduction to prime number sieves. Technical report, University of Wisconsin-Madison Department of Computer Sciences, 1990.

31 Arie Tamir. New pseudopolynomial complexity bounds for the bounded and other integer knapsack related problems. *Oper. Res. Lett.*, 37(5):303–306, 2009. doi:10.1016/j.orl.2009.05.003.

Multiple Random Walks on Graphs: Mixing Few to Cover Many

Nicolás Rivera
Department of Computer Science & Technology, University of Cambridge, UK
Instituto de Ingeniería Matemática, University of Valparaíso, Chile

Thomas Sauerwald
Department of Computer Science & Technology, University of Cambridge, UK

John Sylvester
Department of Computer Science & Technology, University of Cambridge, UK
School of Computing Science, University of Glasgow, UK

Abstract

Random walks on graphs are an essential primitive for many randomised algorithms and stochastic processes. It is natural to ask how much can be gained by running k multiple random walks independently and in parallel. Although the cover time of multiple walks has been investigated for many natural networks, the problem of finding a general characterisation of multiple cover times for *worst-case* start vertices (posed by Alon, Avin, Koucký, Kozma, Lotker, and Tuttle in 2008) remains an open problem.

First, we improve and tighten various bounds on the *stationary* cover time when k random walks start from vertices sampled from the stationary distribution. For example, we prove an unconditional lower bound of $\Omega((n/k)\log n)$ on the stationary cover time, holding for any n-vertex graph G and any $1 \leq k = o(n \log n)$. Secondly, we establish the *stationary* cover times of multiple walks on several fundamental networks up to constant factors. Thirdly, we present a framework characterising *worst-case* cover times in terms of *stationary* cover times and a novel, relaxed notion of mixing time for multiple walks called the *partial mixing time*. Roughly speaking, the partial mixing time only requires a specific portion of all random walks to be mixed. Using these new concepts, we can establish (or recover) the *worst-case* cover times for many networks including expanders, preferential attachment graphs, grids, binary trees and hypercubes.

2012 ACM Subject Classification Theory of computation → Random walks and Markov chains; Mathematics of computing → Stochastic processes

Keywords and phrases Multiple Random walks, Markov Chains, Random Walks, Cover Time

Digital Object Identifier 10.4230/LIPIcs.ICALP.2021.107

Category Track A: Algorithms, Complexity and Games

Related Version *Full Version*: `https://arxiv.org/abs/2011.07893` [39]

Funding N.R., T.S. & J.S. were supported by ERC Starting Grant no. 679660 (DYNAMIC MARCH). N.R. was supported by FONDECYT grant number 3210805. J.S. was partially supported by ESPRC grant number EP/T004878/1.

Acknowledgements We thank Jonathan Hermon for some interesting and useful discussions, and Przemysław Gordinowicz for his feedback on an earlier version of this paper.

1 Introduction

A random walk on a graph is a stochastic process that at each time step chooses a neighbour of the current vertex as its next state. The fact that a random walk visits every vertex of a connected, undirected graph in polynomial time was first used to solve the undirected

$s-t$ connectivity problem in logarithmic space [4]. Since then random walks have become a fundamental primitive in the design of randomised algorithms which feature in approximation algorithms and sampling [32, 40], load balancing [23, 42], searching [19, 33], resource location [24], property testing [12, 26, 27], graph parameter estimation [7, 11] and biological applications [8, 20].

The fact that random walks are local and memoryless (Markov property) ensures they require very little space and relatively unaffected by changes in the environment, e.g., dynamically evolving graphs or graphs with edge failures. These properties make random walks a natural candidate for parallelisation, where running parallel walks has the potential of lower time overheads. One early instance of this idea are space-time trade-offs for the undirected $s-t$ connectivity problem [9, 18]. Other applications involving multiple random walks are sublinear algorithms [13], local clustering [6, 43] or epidemic processes on networks [29, 38].

Given the potential applications of multiple random walks in algorithms, it is important to understand fundamental properties of multiple random walks. The *speed up*, first introduced in [5], is the ratio of the worst-case cover time by a single random walk to the cover time of k parallel walks. Following [5] and subsequent works [15, 16, 22, 25, 41] our understanding of when and why a speed up is present has improved. In particular, various results in [5, 15, 16] establish that as long as the lengths of the walks are not smaller than the mixing, the speed-up is linear in k. However, there are still many challenging open problems, for example, understanding the effect of different start vertices or characterising the magnitude of speed-up in terms of graph properties, a problem already stated in [5]: "*...which leads us to wonder whether there is some other property of a graph that characterises the speed-up achieved by multiple random walks more crisply than hitting and mixing times.*" Addressing the previous questions, we introduce new quantities and couplings for multiple random walks, that allow us to improve the state-of-the-art by refining, strengthening or extending results from previous works.

While there is an extensive body of research on the foundations of (single) random walks (and Markov chains), it seems surprisingly hard to transfer these results and develop a systematic theory of *multiple* random walks. One of the reasons is that processes involving multiple random walks often lead to questions about *short* random walks, e.g., shorter than the mixing time. Such short walks may arise in applications including generating random walk samples in massively parallel systems [28, 40], or in applications where random walk steps are expensive or subject to delays (e.g., when crawling social networks like Twitter [11]). The challenge of analysing *short* random walks (shorter than mixing or hitting time) has been mentioned not only in the area of multiple cover times (e.g., [15, Sec. 6]), but also in the contexts of concentration inequalities for random walks [31, p. 863] and property testing [13].

1.1 Our Contribution

Our first set of results provide several tight bounds on $t^{(k)}_{\mathsf{cov}}(\pi)$ in general (connected) graphs, where $t^{(k)}_{\mathsf{cov}}(\pi)$ is the expected time for each vertex to be visited by at least one of k independent walks each started from a vertex independently sampled from the stationary distribution π.

The main findings of Section 3 include:

- Proving general bounds of $\mathcal{O}\big(\big(\frac{|E|}{kd_{\mathsf{min}}}\big)^2 \log^2 n\big)$, $\mathcal{O}\big(\frac{\max_{v\in V} \mathbb{E}_\pi[\,\tau_v\,]}{k} \log n\big)$ and $\mathcal{O}\big(\frac{|E| \log n}{kd_{\mathsf{min}}\sqrt{1-\lambda_2}}\big)$ on $t^{(k)}_{\mathsf{cov}}(\pi)$, where d_{min} is the minimum degree, $\mathbb{E}_\pi[\,\tau_v\,]$ is the single-walk hitting time of $v \in V$ from a stationary start vertex and λ_2 is the second largest eigenvalue of the transition matrix of the walk. All three bounds are tight for certain graphs. The first

bound improves over [9], the second result is a Matthew's type bound for multiple random walks, and the third yields tight bounds for non-regular expanders such as preferential attachment graphs.

- We prove that for any graph G and $1 \leq k = o(n \log n)$, $t_{\mathsf{cov}}^{(k)}(\pi) = \Omega((n/k) \log n)$. Weaker versions of this bound were obtained in [16], holding only for certain values of k or under additional assumptions on the mixing time. Our result matches the fundamental $\Omega(n \log n)$ lower bound for single random walks ($k = 1$) [17], and generalises it in the sense that the total amount of work by all k stationary walks together for covering is always $\Omega(n \log n)$. We establish the $\Omega((n/k) \log n)$ bound by reducing the multiple walk process to a single, reversible Markov chain, and applying a general lower bound on stationary cover times [3].
- A technical tool that provides a bound on the lower tail of the cover time by k walks from stationary for graphs with a large and (relatively) symmetric set of hard to hit vertices (Lemma 9). When applied to 2d tori and binary trees this yields a tight lower bound.

In Section 4 we introduce a novel quantity for multiple walks we call *partial mixing*. Intuitively, instead of mixing all (or at least half) of the k walks, we only need to mix a specified number $\tilde{k}$ of them. We put this idea on a more formal footing and prove min-max theorems which relate worst case cover times $t_{\mathsf{cov}}^{(k)}$ to partial mixing times $t_{\mathsf{mix}}^{(\tilde{k},k)}$ and stationary cover times:

- For any graph G and any $1 \leq k \leq n$, we prove that:

$$t_{\mathsf{cov}}^{(k)} \leq 16 \cdot \min_{1 \leq \tilde{k} < k} \max \left(t_{\mathsf{mix}}^{(\tilde{k},k)}, t_{\mathsf{cov}}^{(\tilde{k})}(\pi) \right).$$

For now, we omit details such as the definition of the partial mixing time $t_{\mathsf{mix}}^{(\tilde{k},k)}$ as well as some max-min characterisations that serve as lower bounds (these can be found in Section 4). Intuitively these characterisations suggest that for any number of walks k, there is an "optimal" choice of $\tilde{k}$ so that one first waits until $\tilde{k}$ out of the k walks are mixed, and then considers only these $\tilde{k}$ stationary walks when covering the remainder of the graph.

This argument involving mixing only some walks extends and generalises previous results that involve mixing all (or at least a constant portion) of the k walks [5, 15, 16]. Previous approaches only imply a linear speed-up as long as the lengths of the walks are not shorter than the mixing time of a *single* random walk. In contrast, our characterisation may still yield tight bounds on the cover time for random walks that are much shorter than the mixing time.

To demonstrate how our insights can be used, we derive worst case cover times for several well-known graph classes. Due to space limitations we could not include this in the main body of this work, however we have summarised our results in Table 1. The corresponding results with full proofs can be found in corresponding section of the full paper [39]. As a first step to calculating $t_{\mathsf{cov}}^{(k)}$, we determine the stationary cover time; this is based on our bounds from Section 3. Secondly, we derive lower and upper bounds on the partial mixing times. Finally, with the stationary cover times and partial mixing times at hand, we can apply the characterisations from Section 4 to infer lower and upper bounds on the worst case stationary times. For some of those graphs the worst case cover times were already known before, while for, e.g., binary trees and preferential attachment graphs, our bounds are new.

- For the graph families of binary trees, cycles, d-dim. tori ($d = 2$ and $d \geq 3$), hypercube, clique, and (possibly non-regular) expanders we determine the cover time up to constants, for both worst-case and stationary start vertices (see Table 1 for the quantitative results).

We believe that this new methodology constitutes some progress towards the open question of Alon et al. [5] about a characterisation of worst-case cover times.

1.2 Novelty of Our Techniques

While a lot of the proof techniques in previous work [5, 15, 16, 41] are based on direct arguments such as mixing time (or relaxation time), our work introduces a number of new methods which, to the best of our knowledge, have not been used in the analysis of cover time of multiple walks before. In particular, one important novel concept is the introduction of the *partial mixing time*. The idea is that instead of waiting for all (or a constant portion of) k walks to mix, we can just mix some $\tilde{k} \leq k$ walks to reap the benefits of coupling these $\tilde{k}$ walks to stationary walks. This then presents a delicate balancing act where one must find an optimal $\tilde{k}$ minimising the overall bound on the cover time, for example in expanders the optimal $\tilde{k}$ is linear in k whereas in binary trees it is approximately $\sqrt{k}$, and for the cycle it is roughly $\log k$. This turning point reveals something about the structure of the graph and our results relating partial mixing to hitting time of sets helps one find this. Another tool we frequently use is a reduction to random walks with geometric resets, similar to a PageRank process, which allows us to relate *multiple* walks from stationary to a *single* reversible Markov chain.

2 Notation & Preliminaries

Throughout $G = (V, E)$ will be a finite undirected, connected graph with $n := |V|$ vertices and $m := |E|$ edges. For any $k \geq 1$, let $X_t = \left(X_t^{(1)}, \ldots, X_t^{(k)}\right)$ be *multiple random walk process*, where each $X_t^{(i)}$ is an independent random walk on G. Let

$$\mathbb{E}_{u_1,\ldots,u_k}[\,\cdot\,] := \mathbb{E}[\,\cdot \mid X_0 = (u_1, \ldots, u_k)\,]$$

denote the conditional expectation where, for each $1 \leq i \leq k$, $X_0^{(i)} = u_i \in V$ is the start vertex of the i^{th} walk. Unless mentioned otherwise, walks will be *lazy*, i.e., at each step the walk stays at its current location with probability $1/2$, and otherwise moves to a neighbour chosen uniformly at random. We let the random variable $\tau_{\mathsf{cov}}^{(k)}(G) = \inf\{t : \bigcup_{i=0}^{t}\{X_i^{(1)}, \ldots, X_i^{(k)}\} = V\}$ be the first time every vertex of the graph has been visited by some walk $X_t^{(i)}$. For $u_1, \ldots, u_k \in V$ let

$$t_{\mathsf{cov}}^{(k)}((u_1, \ldots, u_k), G) = \mathbb{E}_{u_1,\ldots,u_k}\left[\tau_{\mathsf{cov}}^{(k)}(G)\right], \qquad t_{\mathsf{cov}}^{(k)}(G) = \max_{u_1,\ldots,u_k \in V} t_{\mathsf{cov}}^{(k)}((u_1, \ldots, u_k), G)$$

denote the cover time of k walks from $(u_1, \ldots, u_k)$ and the cover time of k walks from worst case start positions respectively. For simplicity, we drop G from the notation if the underlying graph is clear from the context. We shall use π to denote the stationary distribution of a single random walk on on a graph G, for $v \in V$ this is given by $\pi(v) = \frac{d(v)}{2m}$ which is the degree over twice the number of edges. We use π^k, which is a distribution on V^k given by the product measure of π with itself, to denote the stationary distribution of a multiple random walk. For a probability distribution μ on V let $\mathbb{E}_{\mu^k}[\,\cdot\,]$ denote expectation with respect to k walks where each start vertex is sampled independently from μ and

$$t_{\mathsf{cov}}^{(k)}(\mu, G) = \mathbb{E}_{\mu^k}\left[\tau_{\mathsf{cov}}^{(k)}(G)\right].$$

In particular $t_{\mathsf{cov}}^{(k)}(\pi, G)$ denotes the expected cover time from independent stationary start vertices. For a set $S \subseteq V$ we define

$$\tau_S^{(k)} = \inf\{t : \text{ there exists } 1 \leq i \leq k \text{ such that } X_t^{(i)} \in S\}$$

Table 1 All results above are $\Theta(\cdot)$, that is bounded above and below by a multiplicative constant, apart from the mixing time of expanders which is only bounded from above. PA above is the preferential attachment process where each vertex has m initial links, the results hold w.h.p., see [10, 34]. Cells shaded in Yellow are new results proved in this paper with the exception that for $k = \mathcal{O}(\log n)$ upper bounds on the stationary cover time for binary trees, expanders and preferential attachment graphs can be deduced from general bounds for the worst case cover time in [5]. Cells shaded Gray in the second to last column are known results we re-prove in this paper using our partial mixing time results, for the 2-dim grid we only re-prove upper bounds. References for the second to last column are given in the corresponding section in the full version, except for the Barbell, see [15, Page 2]. The Barbell consists of two cliques on $n/2$ vertices connected by single edge; we include this in the table as an interesting example where the speed up by stationary walks is exponential in k. All other results for single walks can be found in [2], for example.

Graph family	Cover t_{cov}	Hitting t_{hit}	Mixing t_{mix}	k-Cover Time, where $2 \le k \le n$	
				Worst case $t_{\mathsf{cov}}^{(k)}$	From π^k, $t_{\mathsf{cov}}^{(k)}(\pi)$
Binary tree	$n \log^2 n$	$n \log n$	n	$(n/k)\log^2 n$ **if** $k \le \log^2 n$. $(n/\sqrt{k})\log n$ **if** $k \ge \log^2 n$.	$\frac{n \log n}{k} \log\left(\frac{n \log n}{k}\right)$
Cycle	n^2	n^2	n^2	$\frac{n^2}{\log k}$	$\left(\frac{n}{k}\right)^2 \log^2 k$
2-Dim. Tori	$n \log^2 n$	$n \log n$	n	$(n/k)\log^2 n$ **if** $k \le \log^2 n$. $\frac{n}{\log(k/\log^2 n)}$ **if** $k \ge \log^2 n$.	$\frac{n \log n}{k} \log\left(\frac{n \log n}{k}\right)$
d-Dim. Tori $d \ge 3$	$n \log n$	n	$n^{2/d}$	$(n/k)\log n$ **if** $k \le n^{1-2/d}\log n$. $\frac{n^{2/d}}{\log(k/(n^{1-2/d}\log n))}$ **if** $k \ge n^{1-2/d}\log n$.	$\frac{n}{k}\log n$
Hypercube	$n \log n$	n	$\log n \log\log n$	$(n/k)\log n$ **if** $k \le n/\log\log n$. $\log n \log\log n$ **if** $k \ge n/\log\log n$.	$\frac{n}{k}\log n$
Expanders	$n \log n$	n	$\mathcal{O}(\log n)$	$\frac{n}{k}\log n$	$\frac{n}{k}\log n$
PA, $m \ge 2$	$n \log n$	n	$\mathcal{O}(\log n)$	$\frac{n}{k}\log n$	
Barbell	n^2	n^2	n^2	n^2/k	$\frac{2^{-k}n^2}{k} + \frac{n \log n}{k}$

as the first time the set S is visited by any of the k independent random walks, if $S = \{v\}$ is a singleton set we use τ_v, dropping brackets. Let

$$t_{\mathsf{hit}}^{(k)}(G) = \max_{u_1,\ldots,u_k \in V} \max_{v \in V} \mathbb{E}_{u_1,\ldots,u_k} \left[\tau_v^{(k)} \right]$$

be the worst case vertex to vertex hitting time. When talking about a single random walk we drop the (1) index, i.e. $t_{\mathsf{cov}}^{(1)}(G) = t_{\mathsf{cov}}(G)$; we also drop G from the notation when the graph is clear. If we wish the graph G to be clear we shall also use the notation $\mathbb{P}_{u,G}[\cdot]$ and $\mathbb{E}_{u,G}[\,\cdot\,]$. For a single random walk X_t with stationary distribution π and $x \in V$, let $d(t)$ and $s_x(t)$ be the total variation and separation distances for X_t given by

$$d(t) = \max_{x \in V} ||P_{x,\cdot}^t - \pi||_{TV} \quad \text{and} \quad s_x(t) = \max_{y \in V} \left[1 - \frac{P_{x,y}^t}{\pi(y)} \right],$$

where $P_{x,\cdot}^t$ is the t-step probability distribution of a random walk starting from x and, for probability measures μ, ν, $||\mu - \nu||_{TV} = \frac{1}{2} \sum_{x \in V} |\mu(x) - \nu(x)|$ is the total variation distance. Let $s(t) = \max_{x \in V} s_x(t)$, then for $0 \le \varepsilon \le 1$ the mixing and separation times [30, (4.32)] are

$$t_{\mathsf{mix}}(\varepsilon) = \inf\{t : d(t) \le \varepsilon\} \qquad \text{and} \qquad t_{\mathsf{sep}}(\varepsilon) = \inf\{t : s(t) \le \varepsilon\}, \tag{1}$$

and $t_{\mathsf{mix}} := t_{\mathsf{mix}}(1/4)$ and $t_{\mathsf{sep}} = t_{\mathsf{sep}}(1/e)$. A strong stationary time (SST) σ, see [30, Ch. 6] or [1], is a randomised stopping time for a Markov chain Y_t on V with stationary distribution π if

$$\mathbb{P}_u[Y_\sigma = v \mid \sigma = k] = \pi(v) \qquad \text{for any } u, v \in V \text{ and } k \ge 0. \tag{2}$$

Let $t_{\mathsf{rel}} = \frac{1}{1-\lambda_2}$ be the relaxation time of G, where λ_2 is the second largest eigenvalue of the transition matrix of the (lazy) random walk on G.

For random variables Y, Z we say that Y dominates Z ($Y \succeq Z$) if $\mathbb{P}[Y \ge x] \ge \mathbb{P}[Z \ge x]$ for all x.

3 Multiple Stationary Cover Times

We shall state our general upper and lower bound results for multiple walks from stationary in Sections 3.1 & 3.2 respectively. All proofs can be found in the full version [39].

3.1 Upper Bounds

Broder, Karlin, Raghavan, and Upfal [9] showed that for any graph G and $k \ge 1$,

$$t_{\mathsf{cov}}^{(k)}(\pi) = \mathcal{O}\left(\left(\frac{m}{k} \right)^2 \log^3 n \right).$$

We prove a general bound which improves this bound by a multiplicative factor of $d_{\mathsf{min}}^2 \log n$ which may be $\Omega(n^2 \log n)$ for some graphs.

▶ **Theorem 1.** *For any graph G and any $k \ge 1$,*

$$t_{\mathsf{cov}}^{(k)}(\pi) = \mathcal{O}\left(\left(\frac{m}{k d_{\mathsf{min}}} \right)^2 \log^2 n \right).$$

This bound is tight for the cycle if $k = n^{\Theta(1)}$, see Table 1. Theorem 1 is proved by relating the probability a vertex v is not hit up to a certain time t to the expected number of returns to v by a walk of length t from v and applying a bound by Oliveira and Peres [35].

The next bound is analogous to Matthew's bound [2, Theorem 2.26] for the cover time of single random walks from worst case, however it is proved by a different method.

▶ **Theorem 2.** *For any graph G and any $k \geq 1$, we have*

$$t_{\mathsf{cov}}^{(k)}(\pi) = \mathcal{O}\left(\frac{\max_{v\in V} \mathbb{E}_\pi\left[\tau_v\right] \log n}{k}\right).$$

This bound is tight for many graphs, see Table 1. Since the acceptance of this paper, the stronger bound $t_{\mathsf{cov}}^{(k)}(\pi) = \mathcal{O}(t_{\mathsf{cov}}/k)$ has been proved by Hermon & Sousi [21]. This bound implies Theorem 2 by a simple application of the aforementioned Matthew's Bound for single random walks. A version of Theorem 2 for $t_{\mathsf{cov}}^{(k)}$ was established by Alon et al. [5] provided $k = \mathcal{O}(\log n)$, the restriction on k is necessary (for worst case) as witnessed by the cycle. Theorem 2 also gives the following explicit bound.

▶ **Corollary 3.** *For any graph G and any $k \geq 1$, we have*

$$t_{\mathsf{cov}}^{(k)}(\pi) = \mathcal{O}\left(\frac{m}{k d_{\mathsf{min}}} \sqrt{t_{\mathsf{rel}}} \log n\right).$$

Proof. Use $\max_{v\in V} \mathbb{E}_\pi\left[\tau_v\right] \leq 20m\sqrt{t_{\mathsf{rel}}+1}/d_{\mathsf{min}}$ from [35, Theorem 1] in Theorem 2. ◀

Notice that, for all values $k \geq 1$, this bound is tight for any expander with $d_{\min} = \Omega(m/n)$, such as preferential attachment graphs (see Table 1 and the full version for more details).

We also establish the following two bounds for classes of graphs with "not too large" return probabilities.

▶ **Lemma 4.** *Let G be any graph satisfying $\pi_{\mathsf{min}} = \Omega(1/n)$, $t_{\mathsf{rel}} = o(n)$ and $\sum_{i=0}^{t} P_{vv}^i = \mathcal{O}(1 + t\pi(v))$ for any $t \leq t_{\mathsf{rel}}$. Then for any $1 \leq k \leq n$,*

$$t_{\mathsf{cov}}^{(k)}(\pi) = \Theta\left(\frac{n}{k} \log n\right).$$

The bound above applied to a broad class of graphs with expander like properties but large relaxation time, this includes the hypercube and high dimensional grids. The following bound holds for graphs with sub-harmonic return times, this includes binary trees and 2d-grid/tori.

▶ **Lemma 5.** *Let G be any graph with $\sum_{i=0}^{t} P_{v,v}^i = \mathcal{O}(t/n + \log t)$ for any $t \leq n(\log n)^2$ for all $v \in V$ and $t_{\mathsf{mix}} = \mathcal{O}(n)$. Then for any $1 \leq k \leq (n \log n)/3$,*

$$t_{\mathsf{cov}}^{(k)}(\pi) = \mathcal{O}\left(\frac{n \log n}{k} \log\left(\frac{n \log n}{k}\right)\right).$$

3.2 Lower Bounds

Generally speaking, lower bounds for random walks are more challenging to derive than upper bounds. In particular, the problem of obtaining a lower bound for the cover time of a simple random walk on an undirected graph was open for many years [2]. This was finally resolved by Feige [17] who proved $t_{\mathsf{cov}} \geq (1 - o(1))n \log n$. We prove a generalisation of this bound, up to constants, that holds for k random walks which start from stationarity (thus also for worst case).

▶ **Theorem 6.** *There exists a constant $c > 0$ such that for any graph G and $1 \le k \le c \cdot n \log n$,*

$$t_{\mathsf{cov}}^{(k)}(\pi) \ge c \cdot \frac{n}{k} \cdot \log n.$$

We remark that in this section all results hold (and are proven) for *non-lazy* random walks, which by stochastic domination implies that the same result also holds for *lazy* random walks. Theorem 6 is tight, uniformly for all $1 \le k \le n$, for the hypercube, expanders and high-dimensional tori, see Table 1. We note that [16] proved this bound for any start vertices under the additional assumption that $k \ge n^{\epsilon}$, for some constant $\varepsilon > 0$. One can track the constants in the proof of Theorem 6 and show that $c > 2 \cdot 10^{-11}$, we have not optimised this but note that $c \le 1$ must hold in either condition of Theorem 6 due to the complete graph.

To prove this result we introduce the *geometric reset graph*, which allows us to couple the multiple random walk to a single walk to which we can apply a lower bound by Aldous [3]. The random reset graph is a small modification to a graph G which gives an edge-weighted graph $\widehat{G}(x)$ such that the simple random walk on $\widehat{G}(x)$ emulates a random walk on G with $\mathrm{Geo}(x)$ resets to stationarity, where $\mathrm{Geo}(x)$ is a geometric random variable with expectation $1/x$.

▶ **Definition 7** (The Geometric Reset Graph $\widehat{G}(x)$)**.** *For any graph G the undirected, edge-weighted graph $\widehat{G}(x)$, where $0 < x \le 1$, consists of all vertices $V(G)$ and one extra vertex z. All edges from G are included with edge-weight 1. Further, z is connected to each vertex $u \in G$ by an edge with edge-weight $x \cdot d(u)/(1-x)$, where $d(u)$ is the degree of vertex u in G.*

Given a graph with edge weights $\{w_e\}_{e \in E}$ the probability a *non-lazy* random walk moves from u to v is given by $w_{uv} / \sum_{w \in V} w_{uw}$. Thus the walk on $\widehat{G}(x)$ behaves as a random walk in G, apart from that in any step, it may move to the extra vertex z with probability $\frac{x \cdot d(u)/(1-x)}{x \cdot d(u)/(1-x) + d(u)} = x$. Once the walk is at z it moves back to a vertex $u \in V \backslash \{z\}$ with probability proportional to $d(u)$. Hence the stationary distribution $\widehat{\pi}$ of the random walk on $\widehat{G}(x)$ is proportional to π on G, and for the extra vertex z we have

$$\widehat{\pi}(z) = \frac{\sum_{v \in V} x d(u)/(1-x)}{\sum_{v \in V} d(u) + \sum_{v \in V} x d(u)/(1-x)} = \frac{x/(1-x)}{1 + x/(1-x)} = x.$$

Using the next lemma we can then obtain bounds on the multiple stationary cover time by simply bounding the cover time in the augmented graph $\widehat{G}(x)$ for some x.

▶ **Lemma 8.** *Let G be any graph G, $k \ge 1$ and $x = Ck/T$ where $C > 30$ and $T \ge 5Ck$. Then*

$$\mathbb{P}_{\pi^k, G}\left[\tau_{\mathsf{cov}}^{(k)} > \frac{T}{10Ck}\right] > \mathbb{P}_{\widehat{\pi}, \widehat{G}(x)}[\tau_{\mathsf{cov}} > T] - \exp\left(-\frac{Ck}{50}\right).$$

The coupling above will also be used later in the paper to prove a lower bound for the stationary cover time of the binary tree and 2-dimensional grid when k is small.

The next result we present utilises the second moment method to obtain a lower bound which works very well for $k = n^{\Theta(1)}$ walks on symmetric (e.g., transitive) graphs. In particular, we apply this to get tight lower bounds for cycles, 2-dim. tori and binary trees (see the corresponding section of the full version).

▶ **Lemma 9.** *Let G be any graph. Let $\alpha \in (0,1)$ be a fixed real constant and define $p_v(t) = \mathbb{P}_\pi[\tau_v \le t]$ for $t \ge 1$. Suppose there exists a subset $S \subseteq V$, and real numbers $p > 0$ and $0 \le \varepsilon < 1$ such that for all $v \in S$ we have $p(1-\varepsilon) \le p_v(t) \le p$, with $p \le \alpha(\log n)/k$, and that $\min_{v \in S} \pi(v) = \Omega(1/|S|)$. If in addition $p^2 k = o(1)$, then*

$$\mathbb{P}_{\pi^k}\left[\tau_{\mathsf{cov}}^{(k)} < t\right] = \mathcal{O}\left(\frac{(\log n)^2 n^{\alpha(1+\varepsilon)}}{k}\right).$$

4 Mixing Few Walks to Cover Many Vertices

In this section we present several bounds on $t_{\mathsf{cov}}^{(k)}$, the multiple cover time from worst case start vertices, based on $t_{\mathsf{cov}}^{(k)}(\pi)$, the multiple cover time from stationarity, and a new notion that we call *partial* mixing time. The intuition behind this is that on many graphs such as cycles or binary trees, only a certain number, say $\tilde{k}$ out of k walks will be able to reach vertices that are "far away" from the initial distribution. That means covering the whole graph hinges on how quickly these $\tilde{k}$ "mixed" walks cover the graph G, however, we also need to take into account the number of steps needed to "mix" those. Theorem 16 (see Subsection 4.2) makes this intuition more precise and suggests that the best strategy for covering a graph might be when $\tilde{k}$ is chosen so that the time to mix $\tilde{k}$ out of k walks and the stationary cover time of $\tilde{k}$ walks are approximately equal. The first subsection (Subsection 4.1) contains details of our new notions of mixing for multiple random walks, the second contains the bounds on worst-case cover times we derive from these and the third contains some bounds on the multiple mixing times.

4.1 Two Notions of Mixing for Multiple Random Walks

Recall the definition of strong stationary time (SST) given by (2) in Section 2. Then, for any graph G, and any $1 \leq \tilde{k} < k$, we define the *partial mixing time*:

$$\begin{aligned} t_{\mathsf{mix}}^{(\tilde{k},k)}(G) &= \inf\left\{ t \geq 1 : \textit{there exists an SST } \tau \text{ such that } \min_{v \in V} \mathbb{P}_v[\tau \leq t] \geq \tilde{k}/k \right\} \\ &= \inf\left\{ t \geq 1 : s(t) \leq 1 - \tilde{k}/k \right\}. \end{aligned} \tag{3}$$

Note that the two definitions above are equivalent by the following result.

▶ **Proposition 10** ([1, Proposition 3.2]). *If σ is an SST then $\mathbb{P}[\sigma > t] \geq s(t)$ for any $t \geq 0$. Furthermore there exists an SST for which equality holds.*

This notion of mixing, based on the idea of separation distance and strong stationary times for single walks, will be useful for establishing an upper bound on the worst case cover time. For lower bounds on the cover time we will now introduce another notion of mixing for multiple random walks based on a different property of mixing times of single walks.

For single random walks, there is a fundamental connection between mixing times and hitting times of sets. In particular if we let

$$t_{\mathsf{H}}(\alpha) = \max_{u \in V, S \subseteq V\,:\,\pi(S) \geq \alpha} \mathbb{E}_u[\tau_S], \qquad \text{and} \qquad t_{\mathsf{H}} := t_{\mathsf{H}}(1/4),$$

then the following theorem shows this *large-set hitting time* is equivalent to the mixing time.

▶ **Theorem 11** ([36] and independently [37]). *Let $\alpha < 1/2$. Then there exist positive constants $c(\alpha)$ and $C(\alpha)$ so that for every reversible chain*

$$c(\alpha) \cdot t_{\mathsf{H}}(\alpha) \leq t_{\mathsf{mix}}(\alpha) \leq C(\alpha) \cdot t_{\mathsf{H}}(\alpha).$$

In order to prove a lower bounds on the cover time we seek to replace the partial mixing time by an analogue of hitting times of large sets, adapted to multiple walks:

$$t_{\mathsf{large-hit}}^{(\tilde{k},k)}(G) := \min\left\{ t \geq 1 : \min_{u \in V, S \subseteq V\,:\,\pi(S) \geq 1/4} \mathbb{P}_u[\tau_S \leq t] \geq \frac{\tilde{k}}{k} \right\}. \tag{4}$$

Note that both of our mixing times, (4) and (3), are only defined for $\tilde{k} < k$. However, by the union bound, there exists a $C < \infty$ such that if we run k walks for $Ct_{\mathsf{mix}} \log k$ steps then all k walks will be close to uniform in total variation norm.

In the following four results, we present some simple relations between $t_{\mathsf{mix}}^{(\tilde{k},k)}$ and $t_{\mathsf{large-hit}}^{(\tilde{k},k)}$, and t_{mix}, where t_{mix} is the total variation mixing time for a single random walk given by (1).

First we show a simple upper bound in terms of the single walk mixing time.

▶ **Lemma 12.** *There exists a constant $C < \infty$ such that for any graph and $1 \le \tilde{k} < k$ we have*

(i) $t_{\mathsf{mix}}^{(\tilde{k},k)} \le 2 \cdot t_{\mathsf{mix}} \cdot \log\left(\frac{4k}{k-\tilde{k}}\right)$,

(ii) $t_{\mathsf{large-hit}}^{(\tilde{k},k)} \le C \cdot t_{\mathsf{mix}} \cdot \log\left(\frac{k}{k-\tilde{k}}\right)$.

The partial mixing time can be bounded from below quite simply by mixing time.

▶ **Lemma 13.** *For any graph and $1 \le \tilde{k} < k$ we have*

$$t_{\mathsf{mix}}^{(\tilde{k},k)} \ge t_{\mathsf{mix}}\left(1 - \frac{\tilde{k}}{k}\right).$$

We would prefer a bound in terms of $t_{\mathsf{mix}} := t_{\mathsf{mix}}(1/4)$ instead of $t_{\mathsf{mix}}(1 - \tilde{k}/k)$ as the former is easier to compute for most graphs. The following Lemma establishes such a lower bound for both notions of mixing time at the cost of a $\tilde{k}/k$ factor.

▶ **Lemma 14.** *There exists some constant $c > 0$ such that for any graph and $1 \le \tilde{k} < k$ we have*

(i) $t_{\mathsf{mix}}^{(\tilde{k},k)} \ge c \cdot \frac{\tilde{k}}{k} \cdot t_{\mathsf{mix}}$,

(ii) $t_{\mathsf{large-hit}}^{(\tilde{k},k)} \ge c \cdot \frac{\tilde{k}}{k} \cdot t_{\mathsf{mix}}$.

We leave as an open problem whether our two notions of mixing for multiple random walks are equivalent up to constants, but the next result gives partial progress in one direction.

▶ **Lemma 15.** *For any graph and $1 \le \tilde{k} < k/4$ we have*

$$t_{\mathsf{large-hit}}^{(\tilde{k},k)} \le t_{\mathsf{mix}}^{(4\tilde{k},k)} + 1 \le 2t_{\mathsf{mix}}^{(4\tilde{k},k)}.$$

4.2 Upper and Lower Bounds by Partial Mixing

Armed with our new notions of mixing time for multiple random walks from Section 4.1, we can now use them to prove upper and lower bounds on the worst case cover time in terms of stationary cover times and partial mixing times. We begin with the upper bound.

▶ **Theorem 16.** *For any graph G and any $1 \le k \le n$,*

$$t_{\mathsf{cov}}^{(k)} \le 16 \cdot \min_{1 \le \tilde{k} < k} \max\left(t_{\mathsf{mix}}^{(\tilde{k},k)}, t_{\mathsf{cov}}^{(\tilde{k})}(\pi)\right).$$

Proof. Fix any $1 \le \tilde{k} < k$. It suffices to prove that with k walks starting from arbitrary positions running for

$$t := t_{\mathsf{mix}}^{(\tilde{k},k)} + 2t_{\mathsf{cov}}^{(\tilde{k})}(\pi) \le 4 \cdot \max\left(t_{\mathsf{mix}}^{(\tilde{k},k)}, t_{\mathsf{cov}}^{(\tilde{k})}(\pi)\right)$$

steps, we cover G with probability at least $1/4$. Consider a single walk $X_1(t)$ on G. From (3), we have that at time $T = t_{\mathsf{mix}}^{(\tilde{k},k)}$ there exists a probability measure ν_v on V such that,

$$P_{v,w}^T = (1 - s_v(T))\pi(w) + s_v(T)\nu_v(w).$$

Therefore, we can generate $X_1(T)$ as follows: with probability $1 - s_v(T) \geq \tilde{k}/k$ we sample from π, otherwise we sample from ν_v. If we now consider k independent walks, the number of walks that are sampled at time T from π has a binomial distribution $\mathrm{Bin}(k, \tilde{k}/k)$ with k trials and probability $\tilde{k}/k$, whose expectation is $\tilde{k}$. Since the expectation $\tilde{k}$ is an integer it is equal to the median, thus with probability at least $1/2$, at least $\tilde{k}$ walks are sampled from the stationary distribution. Now, consider only the $\tilde{k}$ independent walks starting from π. After $2t_{\mathsf{cov}}(\pi, \tilde{k})$ steps, these walks will cover G with probability at least $1/2$, due to Markov's inequality.

We conclude that in t time steps, from any starting configuration of the k walks, the probability we cover the graph is at least $1/4$. Hence in expectation, after (at most) 4 periods of length t we cover the graph. ◀

This theorem improves on various results in [5] and [15] which bound the worst case cover time by mixing all k walks, and it also generalises a previous result in [16, Lemma 3.1], where most walks were mixed, i.e., $\tilde{k} = k/2$.

We also prove a lower bound for cover times, however this involves the related definition of partial mixing time based on the hitting times of large sets.

▶ **Theorem 17.** *For any graph G with $\pi_{\max} = \max_u \pi(u)$ and any $1 \leq k \leq n$,*

$$t_{\mathsf{cov}}^{(k)} \geq \frac{1}{16} \cdot \max_{1 \leq \tilde{k} < k} \min\left(t_{\mathsf{large-hit}}^{(\tilde{k},k)}, \frac{1}{\tilde{k}\pi_{\max}} \right).$$

Further, for any regular graph G any $\delta > 0$ fixed, there is a constant $C = C(\delta) > 0$ such that

$$t_{\mathsf{cov}}^{(k)} \geq C \cdot \max_{n^\delta \leq \tilde{k} < k} \min\left(t_{\mathsf{large-hit}}^{(\tilde{k},k)}, \frac{n \log n}{\tilde{k}} \right).$$

As we will see later, both Theorem 16 and Theorem 17 yield asymptotically tight (or tight up to logarithmic factors) lower and upper bounds for many concrete networks. To explain why this is often the case, note that both bounds include one non-increasing function in $\tilde{k}$ and one non-decreasing in function in $\tilde{k}$. That means both bounds are optimised when the two functions are as close as possible. Then balancing the two functions in the upper bound asks for $\tilde{k}$ such that $t_{\mathsf{mix}}^{(\tilde{k},k)} \approx t_{\mathsf{cov}}^{(\tilde{k})}(\pi)$. Similarly, balancing the two functions in the first lower bound demands $t_{\mathsf{large-hit}}^{(\tilde{k},k)} \approx n/\tilde{k}$ (assuming $\pi_{\max} = O(1/n)$). Hence for any graph G where $t_{\mathsf{mix}}^{(\tilde{k},k)} \approx t_{\mathsf{large-hit}}^{(\tilde{k},k)}$, and also $t_{\mathsf{cov}}^{(\tilde{k})}(\pi) \approx n/\tilde{k}$, the upper and lower bounds will be close. This turns out to be the case for many networks (see the corresponding section in the full version).

One exception where Theorem 17 is far from tight is the cycle, we shall also prove a min-max theorem but with a different notion of partial cover time which is tight for the cycle.

For a set $S \subseteq V$ we let $\tau_{\mathsf{cov}}^{(k)}(S)$ be the first time that every vertex in S has been visited by at least one of the k walks, thus $\tau_{\mathsf{cov}}^{(k)}(V) = \tau_{\mathsf{cov}}^{(k)}$. Then we define the set cover time

$$t_{\mathsf{large-cov}}^{(k)} = \min_{S:\pi(S)\geq 1/4} \max_{\mu} \mathbb{E}_{\mu^k}\left[\tau_{\mathsf{cov}}^{(k)}(S) \right],$$

where the first minimum is over all sets $S \subseteq V$ satisfying $\pi(S) \geq 1/4$ and the second is over all probability distributions μ on the set $\partial S = \{x \in S : \text{exists } y \in S^c,\ xy \in E\}$.

▶ **Theorem 18.** *For any graph G and any $1 \leq k \leq n$,*

$$t_{\mathsf{cov}}^{(k)} \geq \frac{1}{4} \cdot \max_{1 \leq \tilde{k} < k} \min \left(t_{\mathsf{large-hit}}^{(\tilde{k},k)}, t_{\mathsf{large-cov}}^{(k)} \right).$$

4.3 Geometric Lower Bounds on the Large-Hit and Large-Cover Times

We will now derive two useful lower bounds on $t_{\mathsf{large-hit}}^{(\tilde{k},k)}$, one based on the conductance of the graph, and a second one based on the distance to a large set the random walk needs to hit.

For two sets $A, B \subseteq V$ the *ergodic flow* $Q(A,B)$ is defined by $Q(A,B) = \sum_{a \in A, b \in B} \pi(a) P_{a,b}$, where $P_{a,b}$ denotes the transition matrix of a (lazy) single random walk. We define the *conductance* $\Phi(S)$ of a set $S \subseteq V$ with $\pi(S) \in (0, 1/2]$ to be

$$\Phi(S) = \frac{Q(S, S^c)}{\pi(S)} \quad \text{and let} \quad \Phi(G) = \min_{S \subseteq V, 0 < \pi(S) \leq 1/2} \Phi(S).$$

▶ **Lemma 19.** *For any graph G with conductance $\Phi(G)$, any $1 \leq \tilde{k} \leq k$,*

$$t_{\mathsf{large-hit}}^{(\tilde{k},k)} \geq \frac{\tilde{k}}{k} \cdot \frac{2}{\Phi(G)}.$$

We remark that a similar bound was used implicitly in [41, Proof of Theorem 1.1], where $t_{\mathsf{cov}}^{(k)} \geq \sqrt{\frac{n}{k \cdot \Phi(G)}}$ was shown.

The following lemmas will be useful to lower bound worst case cover times on cycles/tori.

▶ **Lemma 20.** *Let G be a d-dimensional torus with constant $d \geq 2$ (or cycle, $d = 1$), $u \in V$ and S be a set with $|S| \geq n/2$. Then for any $\tilde{k} \leq k/2$,*

$$t_{\mathsf{large-hit}}^{(\tilde{k},k)} = \Omega\left((\mathrm{dist}(u,S))^2 / \log(k/\tilde{k}) \right).$$

▶ **Lemma 21.** *Let $S \subseteq V$ be a subset of vertices with $\pi(S) \geq 1/4$, $t \geq 2$ be an integer and $k \geq 100$ such that for every $u \in S$,*

$$\sum_{s=0}^{t} P_{u,u}^s \geq 32 \cdot t \cdot \pi(u) \cdot k.$$

Then for any starting distribution μ of $k/8$ walks,

$$\mathbb{E}_{\mu^k} \left[\tau_{\mathsf{cov}}^{(k)}(S) \right] \geq t/5.$$

5 Conclusion & Open Problems

In this work, we derived several new bounds on multiple stationary and worst-case cover times. We also introduced a new quantity called *partial mixing time*, which extends the definition of mixing time from single random walks to multiple random walks. By means of a min-max characterisation, we proved that the partial mixing time connects the stationary and worst-case cover times, leading to tight lower and upper bounds for many graph classes.

In terms of worst-case bounds, Theorem 1 implies that for any regular graph G and any $k \geq 1$, $t_{\mathsf{cov}}^{(k)}(\pi) = \mathcal{O}\left(\left(\frac{n}{k}\right)^2 \log^2 n \right)$. This bound is tight for the cycle when k is polynomial in n but not for smaller k. We suspect that for any $k \geq 1$ the cycle is (asymptotically) the worst case for $t_{\mathsf{cov}}^{(k)}(\pi)$ amongst regular graphs, which suggests $t_{\mathsf{cov}}^{(k)}(\pi) = \mathcal{O}\left(\left(\frac{n}{k}\right)^2 \log^2 k \right)$.

Some of our results have been only proven for the independent stationary case, but it seems plausible they extend to the case where the k random walks start from the *same* vertex. For example, extending the bound $t_{\mathsf{cov}}^{(k)}(\pi) = \Omega((n/k)\log n)$ to this case would be very interesting.

In Theorem 2 we prove $t_{\mathsf{cov}}^{(k)}(\pi) = \mathcal{O}((\max_{v\in V} \mathbb{E}_\pi[\tau_v]\log n)/k)$, can we prove the stronger bound $t_{\mathsf{cov}}^{(k)}(\pi) = \mathcal{O}(1/k \cdot t_{\mathsf{cov}}(\pi, G))$ without assuming anything on the mixing time of G?

Although our min-max characterisations involving partial mixing time yields tight bounds for many natural graph classes, it would be interesting to establish a general approximation guarantee (or find graph classes that serve as counter-examples). For the former, we believe techniques such as Gaussian Processes and Majorising Measures used in the seminal work of Ding, Lee and Peres [14] could be very useful.

References

1. David Aldous and Persi Diaconis. Strong uniform times and finite random walks. *Adv. in Appl. Math.*, 8(1):69–97, 1987. doi:10.1016/0196-8858(87)90006-6.
2. David Aldous and James Allen Fill. Reversible Markov chains and random walks on graphs, 2002. Unfinished monograph, recompiled 2014. URL: https://www.stat.berkeley.edu/~aldous/RWG/book.html.
3. David J. Aldous. Lower bounds for covering times for reversible Markov chains and random walks on graphs. *J. Theoret. Probab.*, 2(1):91–100, 1989. doi:10.1007/BF01048272.
4. Romas Aleliunas, Richard M. Karp, Richard J. Lipton, László Lovász, and Charles Rackoff. Random walks, universal traversal sequences, and the complexity of maze problems. In *20th Annual Symposium on Foundations of Computer Science, San Juan, Puerto Rico, 29-31 October 1979*, pages 218–223. IEEE Computer Society, 1979. doi:10.1109/SFCS.1979.34.
5. Noga Alon, Chen Avin, Michal Koucký, Gady Kozma, Zvi Lotker, and Mark R. Tuttle. Many random walks are faster than one. *Combin. Probab. Comput.*, 20(4):481–502, 2011. doi:10.1017/S0963548311000125.
6. Reid Andersen, Fan R. K. Chung, and Kevin J. Lang. Local graph partitioning using pagerank vectors. In *47th Annual IEEE Symposium on Foundations of Computer Science (FOCS 2006), 21-24 October 2006, Berkeley, California, USA, Proceedings*, pages 475–486. IEEE Computer Society, 2006. doi:10.1109/FOCS.2006.44.
7. Anna Ben-Hamou, Roberto I. Oliveira, and Yuval Peres. Estimating graph parameters via random walks with restarts. In Artur Czumaj, editor, *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2018, New Orleans, LA, USA, January 7-10, 2018*, pages 1702–1714. SIAM, 2018. doi:10.1137/1.9781611975031.111.
8. Lucas Boczkowski, Brieuc Guinard, Amos Korman, Zvi Lotker, and Marc P. Renault. Random walks with multiple step lengths. In Michael A. Bender, Martin Farach-Colton, and Miguel A. Mosteiro, editors, *LATIN 2018: Theoretical Informatics - 13th Latin American Symposium, Buenos Aires, Argentina, April 16-19, 2018, Proceedings*, volume 10807 of *Lecture Notes in Computer Science*, pages 174–186. Springer, 2018. doi:10.1007/978-3-319-77404-6_14.
9. Andrei Z. Broder, Anna R. Karlin, Prabhakar Raghavan, and Eli Upfal. Trading space for time in undirected s-t connectivity. *SIAM J. Comput.*, 23(2):324–334, 1994. doi:10.1137/S0097539790190144.
10. Colin Cooper and Alan Frieze. The cover time of the preferential attachment graph. *J. Combin. Theory Ser. B*, 97(2):269–290, 2007. doi:10.1016/j.jctb.2006.05.007.
11. Colin Cooper, Tomasz Radzik, and Yiannis Siantos. Estimating network parameters using random walks. *Social Netw. Analys. Mining*, 4(1):168, 2014. doi:10.1007/s13278-014-0168-6.
12. Artur Czumaj, Morteza Monemizadeh, Krzysztof Onak, and Christian Sohler. Planar graphs: Random walks and bipartiteness testing. *Random Struct. Algorithms*, 55(1):104–124, 2019. doi:10.1002/rsa.20826.

13 Artur Czumaj and Christian Sohler. Testing expansion in bounded-degree graphs. *Comb. Probab. Comput.*, 19(5-6):693–709, 2010. doi:10.1017/S096354831000012X.

14 Jian Ding, James R. Lee, and Yuval Peres. Cover times, blanket times, and majorizing measures. *Annals of Mathematics*, 175(3):1409–1471, 2012. doi:10.4007/annals.2012.175.3.8.

15 Klim Efremenko and Omer Reingold. How well do random walks parallelize? In Irit Dinur, Klaus Jansen, Joseph Naor, and José D. P. Rolim, editors, *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques, 12th International Workshop, APPROX 2009, and 13th International Workshop, RANDOM 2009, Berkeley, CA, USA, August 21-23, 2009. Proceedings*, volume 5687 of *Lecture Notes in Computer Science*, pages 476–489. Springer, 2009. doi:10.1007/978-3-642-03685-9_36.

16 Robert Elsässer and Thomas Sauerwald. Tight bounds for the cover time of multiple random walks. *Theoret. Comput. Sci.*, 412(24):2623–2641, 2011. doi:10.1016/j.tcs.2010.08.010.

17 Uriel Feige. A tight lower bound on the cover time for random walks on graphs. *Random Structures Algorithms*, 6(4):433–438, 1995. doi:10.1002/rsa.3240060406.

18 Uriel Feige. A spectrum of time-space trade-offs for undirected s-t connectivity. *J. Comput. Syst. Sci.*, 54(2):305–316, 1997. doi:10.1006/jcss.1997.1471.

19 Christos Gkantsidis, Milena Mihail, and Amin Saberi. Hybrid search schemes for unstructured peer-to-peer networks. In *INFOCOM 2005. 24th Annual Joint Conference of the IEEE Computer and Communications Societies, 13-17 March 2005, Miami, FL, USA*, pages 1526–1537. IEEE, 2005. doi:10.1109/INFCOM.2005.1498436.

20 Brieuc Guinard and Amos Korman. Tight bounds for the cover times of random walks with heterogeneous step lengths. In Christophe Paul and Markus Bläser, editors, *37th International Symposium on Theoretical Aspects of Computer Science, STACS 2020, March 10-13, 2020, Montpellier, France*, volume 154 of *LIPIcs*, pages 28:1–28:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPIcs.STACS.2020.28.

21 Jonathan Hermon and Perla Sousi. Covering a graph with independent walks, 2021. arXiv:2104.00665.

22 Andrej Ivaskovic, Adrian Kosowski, Dominik Pajak, and Thomas Sauerwald. Multiple random walks on paths and grids. In Heribert Vollmer and Brigitte Vallée, editors, *34th Symposium on Theoretical Aspects of Computer Science, STACS 2017, March 8-11, 2017, Hannover, Germany*, volume 66 of *LIPIcs*, pages 44:1–44:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2017. doi:10.4230/LIPIcs.STACS.2017.44.

23 David R. Karger and Matthias Ruhl. Simple efficient load balancing algorithms for peer-to-peer systems. In Phillip B. Gibbons and Micah Adler, editors, *SPAA 2004: Proceedings of the Sixteenth Annual ACM Symposium on Parallelism in Algorithms and Architectures, June 27-30, 2004, Barcelona, Spain*, pages 36–43. ACM, 2004. doi:10.1145/1007912.1007919.

24 David Kempe, Jon M. Kleinberg, and Alan J. Demers. Spatial gossip and resource location protocols. In Jeffrey Scott Vitter, Paul G. Spirakis, and Mihalis Yannakakis, editors, *Proceedings on 33rd Annual ACM Symposium on Theory of Computing, July 6-8, 2001, Heraklion, Crete, Greece*, pages 163–172. ACM, 2001. doi:10.1145/380752.380796.

25 Ralf Klasing, Adrian Kosowski, Dominik Pajak, and Thomas Sauerwald. The multi-agent rotor-router on the ring: a deterministic alternative to parallel random walks. In Panagiota Fatourou and Gadi Taubenfeld, editors, *ACM Symposium on Principles of Distributed Computing, PODC '13, Montreal, QC, Canada, July 22-24, 2013*, pages 365–374. ACM, 2013. doi:10.1145/2484239.2484260.

26 Akash Kumar, C. Seshadhri, and Andrew Stolman. Finding forbidden minors in sublinear time: A n^1/2+o(1)-query one-sided tester for minor closed properties on bounded degree graphs. In Mikkel Thorup, editor, *59th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2018, Paris, France, October 7-9, 2018*, pages 509–520. IEEE Computer Society, 2018. doi:10.1109/FOCS.2018.00055.

27 Akash Kumar, C. Seshadhri, and Andrew Stolman. Random walks and forbidden minors II: a poly($d\,\epsilon^{-1}$)-query tester for minor-closed properties of bounded degree graphs. In Moses Charikar and Edith Cohen, editors, *Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing, STOC 2019, Phoenix, AZ, USA, June 23-26, 2019*, pages 559–567. ACM, 2019. doi:10.1145/3313276.3316330.

28 Jakub Lacki, Slobodan Mitrovic, Krzysztof Onak, and Piotr Sankowski. Walking randomly, massively, and efficiently. In Konstantin Makarychev, Yury Makarychev, Madhur Tulsiani, Gautam Kamath, and Julia Chuzhoy, editors, *Proccedings of the 52nd Annual ACM SIGACT Symposium on Theory of Computing, STOC 2020, Chicago, IL, USA, June 22-26, 2020*, pages 364–377. ACM, 2020. doi:10.1145/3357713.3384303.

29 Henry Lam, Zhenming Liu, Michael Mitzenmacher, Xiaorui Sun, and Yajun Wang. Information dissemination via random walks in d-dimensional space. In Yuval Rabani, editor, *Proceedings of the Twenty-Third Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2012, Kyoto, Japan, January 17-19, 2012*, pages 1612–1622. SIAM, 2012. doi:10.1137/1.9781611973099.128.

30 David A. Levin, Yuval Peres, and Elizabeth L. Wilmer. *Markov chains and mixing times*. American Mathematical Society, Providence, RI, 2009. With a chapter by James G. Propp and David B. Wilson.

31 Pascal Lezaud. Chernoff-type bound for finite Markov chains. *The Annals of Applied Probability*, 8(3):849–867, 1998. doi:10.1214/aoap/1028903453.

32 László Lovász. Random walks on graphs: a survey. In *Combinatorics, Paul Erdős is eighty, Vol. 2 (Keszthely, 1993)*, volume 2 of *Bolyai Soc. Math. Stud.*, pages 353–397. János Bolyai Math. Soc., Budapest, 1996.

33 Qin Lv, Pei Cao, Edith Cohen, Kai Li, and Scott Shenker. Search and replication in unstructured peer-to-peer networks. In Kemal Ebcioglu, Keshav Pingali, and Alex Nicolau, editors, *Proceedings of the 16th international conference on Supercomputing, ICS 2002, New York City, NY, USA, June 22-26, 2002*, pages 84–95. ACM, 2002. doi:10.1145/514191.514206.

34 Milena Mihail, Christos H. Papadimitriou, and Amin Saberi. On certain connectivity properties of the internet topology. *J. Comput. Syst. Sci.*, 72(2):239–251, 2006. doi:10.1016/j.jcss.2005.06.009.

35 Roberto I. Oliveira and Yuval Peres. Random walks on graphs: new bounds on hitting, meeting, coalescing and returning. In Marni Mishna and J. Ian Munro, editors, *Proceedings of the Sixteenth Workshop on Analytic Algorithmics and Combinatorics, ANALCO 2019, San Diego, CA, USA, January 6, 2019*, pages 119–126. SIAM, 2019. doi:10.1137/1.9781611975505.13.

36 Roberto Imbuzeiro Oliveira. Mixing and hitting times for finite Markov chains. *Electron. J. Probab.*, 17:no. 70, 12, 2012. doi:10.1214/EJP.v17-2274.

37 Yuval Peres and Perla Sousi. Mixing times are hitting times of large sets. *J. Theoret. Probab.*, 28(2):488–519, 2015. doi:10.1007/s10959-013-0497-9.

38 Alberto Pettarin, Andrea Pietracaprina, Geppino Pucci, and Eli Upfal. Tight bounds on information dissemination in sparse mobile networks. In Cyril Gavoille and Pierre Fraigniaud, editors, *Proceedings of the 30th Annual ACM Symposium on Principles of Distributed Computing, PODC 2011, San Jose, CA, USA, June 6-8, 2011*, pages 355–362. ACM, 2011. doi:10.1145/1993806.1993882.

39 Nicolás Rivera, Thomas Sauerwald, and John Sylvester. Multiple random walks on graphs: Mixing few to cover many, 2020. arXiv:2011.07893.

40 Atish Das Sarma, Danupon Nanongkai, Gopal Pandurangan, and Prasad Tetali. Distributed random walks. *J. ACM*, 60(1):2:1–2:31, 2013. doi:10.1145/2432622.2432624.

41 Thomas Sauerwald. Expansion and the cover time of parallel random walks. In Andréa W. Richa and Rachid Guerraoui, editors, *Proceedings of the 29th Annual ACM Symposium on Principles of Distributed Computing, PODC 2010, Zurich, Switzerland, July 25-28, 2010*, pages 315–324. ACM, 2010. doi:10.1145/1835698.1835776.

42 Thomas Sauerwald and He Sun. Tight bounds for randomized load balancing on arbitrary network topologies. In *53rd Annual IEEE Symposium on Foundations of Computer Science, FOCS 2012, New Brunswick, NJ, USA, October 20-23, 2012*, pages 341–350. IEEE Computer Society, 2012. doi:10.1109/FOCS.2012.86.

43 Daniel A. Spielman and Shang-Hua Teng. A local clustering algorithm for massive graphs and its application to nearly linear time graph partitioning. *SIAM J. Comput.*, 42(1):1–26, 2013. doi:10.1137/080744888.

Detecting and Counting Small Subgraphs, and Evaluating a Parameterized Tutte Polynomial: Lower Bounds via Toroidal Grids and Cayley Graph Expanders

Marc Roth
Merton College, University of Oxford, UK

Johannes Schmitt
Mathematical Institute, University of Bonn, Germany

Philip Wellnitz
Max Planck Institute for Informatics, Saarland Informatics Campus (SIC), Saarbrücken, Germany

Abstract

Given a graph property Φ, we consider the problem EdgeSub(Φ), where the input is a pair of a graph G and a positive integer k, and the task is to decide whether G contains a k-edge subgraph that satisfies Φ. Specifically, we study the parameterized complexity of EdgeSub(Φ) and of its counting problem #EdgeSub(Φ) with respect to both approximate and exact counting. We obtain a complete picture for minor-closed properties Φ: the decision problem EdgeSub(Φ) always admits an FPT ("fixed-parameter tractable") algorithm and the counting problem #EdgeSub(Φ) always admits an FPTRAS ("fixed-parameter tractable randomized approximation scheme"). For exact counting, we present an exhaustive and explicit criterion on the property Φ which, if satisfied, yields fixed-parameter tractability and otherwise #W[1]-hardness. Additionally, most of our hardness results come with an almost tight conditional lower bound under the so-called Exponential Time Hypothesis, ruling out algorithms for #EdgeSub(Φ) that run in time $f(k) \cdot |G|^{o(k/\log k)}$ for any computable function f.

As a main technical result, we gain a complete understanding of the coefficients of toroidal grids and selected Cayley graph expanders in the homomorphism basis of #EdgeSub(Φ). This allows us to establish hardness of exact counting using the Complexity Monotonicity framework due to Curticapean, Dell and Marx (STOC'17). This approach does not only apply to #EdgeSub(Φ) but also to the more general problem of computing weighted linear combinations of subgraph counts. As a special case of such a linear combination, we introduce a parameterized variant of the Tutte Polynomial T_G^k of a graph G, to which many known combinatorial interpretations of values of the (classical) Tutte Polynomial can be extended. As an example, $T_G^k(2,1)$ corresponds to the number of k-forests in the graph G. Our techniques allow us to completely understand the parameterized complexity of computing the evaluation of T_G^k at every pair of rational coordinates (x, y). In particular, our results give a new proof for the #W[1]-hardness of the problem of counting k-forests in a graph.

2012 ACM Subject Classification Theory of computation → Parameterized complexity and exact algorithms; Theory of computation → Problems, reductions and completeness

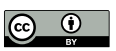
Keywords and phrases Counting complexity, parameterized complexity, Tutte polynomial

Digital Object Identifier 10.4230/LIPIcs.ICALP.2021.108

Category Track A: Algorithms, Complexity and Games

Related Version *Full Version*: https://arxiv.org/abs/2011.03433

Funding *Johannes Schmitt*: The second author was supported by the SNF early postdoc mobility grant 184245.



Acknowledgements We thank Alina Vdovina and Norbert Peyerimhoff for explaining their construction of 2-group Cayley graph expanders [54]. We thank Johannes Lengler and Holger Dell for helpful discussions on early drafts of this work. The second author was supported by the SNF early postdoc mobility grant 184245 and thanks the Max Planck Institute for Mathematics in Bonn for its hospitality.

1 Extended Abstract

Be it searching for cliques in social networks or understanding protein-protein interaction networks, many interesting real-life problems boil down to finding (or counting) small patterns in large graphs. Hence, to no surprise, finding (and counting) small patterns in large graphs are among the most well-studied computational problems in the fields of database theory [12, 40, 32, 13, 27], molecular biology and bioinformatics [38, 1, 56, 59], and network science [60, 52, 53]. In fact, already in the 1970s, the relevance of *finding* patterns became apparent in the context of finding cliques, finding Hamiltonian paths, or finding specific subgraphs in general [19, 18, 62, 12]. However, with the advent of motif counting for the frequency analysis of small structures in complex networks [52, 53], it became evident that detecting the existence of a pattern graph is not enough; we also need to *count* all of the occurrences of the pattern.

In this work, our patterns are (not necessarily induced) edge subgraphs that satisfy a certain graph property: for instance, given a graph, we want to count all occurrences of edge subgraphs that are are planar or connected.

From a classical point of view, often the problem of *finding* patterns is already NP-hard: prime examples include the aforementioned problems of finding (maximum) cliques or Hamiltonian paths. However, for the task of network motif counting, the patterns are (almost) always much smaller than the network itself (see [52, 53, 1]). This motivates a *parameterized* view: can we obtain fast algorithms to compute the number of occurrences of "small" patterns? If we cannot, can we at least obtain fast (randomized) algorithms to compute an *estimate* of this number? And if we cannot even do this, can we at least obtain fast algorithms to *detect* an occurrence? In this work, we completely answer all of the above questions for patterns that are specified by *minor-closed* graph properties (such as planarity) or selected other graph properties (such as connectivity).

As it turns out, the techniques we develop for answering the above questions are quite powerful: they easily generalize to a parameterized version of the Tutte polynomial. Specifically, our techniques allow us to completely understand at which rational points we can evaluate said parameterized Tutte polynomial in reasonable time, and at which rational points this is not feasible. This dichotomy turns out to be similar, but not equal, to the complexity landscape of the classical Tutte polynomial due to Jaeger et al. [42].

Parameterized Counting and Hardness

By now, *counting complexity theory* is a well established subfield of theoretical computer science. Already in the 1970s, Valiant started a formal study of counting problems when investigating the complexity of the permanent [63, 64]: counting the number of perfect matchings in a graph is #P-complete, and hence harder than any problem in the polynomial-time hierarchy PH by Toda's Theorem [61]. In contrast, *detecting* a perfect matching in a graph is much easier and can be done in polynomial time [33]. Hence, counting problems can be much harder than their decision problem counterparts.

As an attempt to overcome the hardness of counting problems in general, the focus shifted to a multivariate or *parameterized* view on these problems. Consider for example the following problem: given a query φ of size k and a database B of size n, we want to count the number of answers to φ in B. If we make the very reasonable assumption that k is much smaller than n, then we may consider an algorithm running in time $O(2^k \cdot n)$ as *tractable*. Note that in particular, such an algorithm may even outperform an algorithm running in time $O(n^2)$. Also consider [39] for a more detailed and formal discussion.
Formally, given a problem P and a *parameterization* κ that maps each instance I of P to a parameter $\kappa(I)$, we say that P is *fixed-parameter tractable* (FPT) with respect to κ, if there is an algorithm that solves each instance I of size n in time $f(\kappa(I)) \cdot n^{O(1)}$, for some computable function f. This notion was introduced by Downey and Fellows in the early 1990s [29, 30] and has itself spawned a rich body of literature (see [35, 31, 23]). In the context of the problems of detecting and counting small patterns in large networks, we parameterize by the size of the pattern: given a pattern of size k and a network of size n, we aim for algorithms that run in time $f(k) \cdot n^{O(1)}$, for some computable function f. However, for some patterns, even this goal is too ambitious: it is widely believed that even finding a clique of size k is not fixed-parameter tractable; in particular, an FPT algorithm for finding a clique of size k would also imply a breakthrough result for the Satisfiability Problem and thereby refute the widely believed Exponential Time Hypothesis [15, 16]. If a problem P is at least as hard as finding a clique (or counting all cliques) of size k, we say that P is $\mathsf{W}[1]$-hard (or $\#\mathsf{W}[1]$-hard, respectively).

For such a $(\#)\mathsf{W}[1]$-hard problem, the hope is to (significantly) improve upon the naive brute-force algorithm, which runs in time $n^{O(k)}$ for the problems considered in this work. However, in view of the aforementioned reduction from the Satisfiability Problem to the problem of finding cliques of size k [14, 15], we can see that for finding cliques this, too, would require a breakthrough for the Satisfiability Problem, which, again, is believed to be unlikely [41]. In our paper, via suitable reductions from the problem of finding cliques, we establish that exact algorithms significantly faster than the brute-force algorithms are unlikely for the problems we study.

Parameterized Detection and Counting of Edge Subgraphs

Vertex-induced subgraphs as patterns are notoriously hard to detect or to count. The long line of research on this problem [47, 17, 43, 44, 51, 45, 21, 57, 28, 58] showed that this holds even if the patterns are significantly smaller than the host graphs, as witnessed by $\mathsf{W}[1]$ and $\#\mathsf{W}[1]$-hardness results and almost tight conditional lower bounds. In case of exact counting, it is in fact an open question whether there are non-trivial instances of induced subgraph counting that admit efficient algorithms; recent work [58] supports the conjecture that no such instances exist.

In search for fast algorithms, in this work, we hence consider a related, but different version of network-motif counting: for a computable graph property Φ, in the problem $\#\textsc{EdgeSub}(\Phi)$ we are given a graph G and a positive integer k, and the task is to compute the number of (not necessarily induced) edge subgraphs[1] with k edges in G that satisfy Φ. Similarly, we write $\textsc{EdgeSub}(\Phi)$ for the corresponding decision problem. Then, in contrast to the case of counting vertex-induced subgraphs, for $(\#)\textsc{EdgeSub}(\Phi)$, we identify non-trivial

[1] Recall that an edge subgraph G' of a graph G may have fewer edges than the subgraph of G that is induced by the vertices of G'.

properties Φ for which (#)EdgeSub(Φ) is fixed-parameter tractable; we discuss this in more detail later. First, however, let us take a detour to elaborate more on what is known already for (#)EdgeSub(Φ).

If the property Φ is satisfied by at most a single graph for each value of the parameter k, the decision problem EdgeSub(Φ) becomes the subgraph isomorphism problem. Hence, naturally there is a vast body of known techniques and results for special properties Φ: for FPT algorithms, think of the Colour-Coding technique by Alon, Yuster and Zwick [3], the "Divide and Colour"-technique [16], narrow sieving [7], representative sets [36], or "extensor-coding" [9] to name but a few. For hardness results, apart from the aforementioned example of detecting a clique, Lin quite recently established that detecting a k-biclique is also W[1]-hard [48]. However, a complete understanding of the parameterized decision version of the subgraph isomorphism is one of the major open problems of parameterized complexity theory [31, Chapter 33.1], that is still to be solved.

In the setting of parameterized *counting*, the situation is much better understood: Flum and Grohe [34] proved #EdgeSub(Φ) to be #W[1]-hard when Φ is the property of being a cycle, or the property of being a path. Curticapean [20] established the same result for the property of being a matching. In [22], Curticapean and Marx established a complete classification in case Φ does not hold on two different graphs with the same number of edges, which is essentially the parameterized subgraph counting problem. In particular, they identified a bound on the matching number as the tractability criterion. In a later work, together with Dell [21], they presented what is now called the framework of Complexity Monotonicity, which can be considered to be one of the most powerful tools in the field of parameterized counting problems. Note that this does not classify the decision version, as #W[1]-hardness for a counting problem does not imply W[1]-hardness for the corresponding decision problem.

In contrast to the parameterized subgraph detection/counting problems, the problem (#)EdgeSub(Φ) allows to search for more general patterns. For example, while the (parameterized) complexity of counting all subgraphs of a graph G isomorphic to a *fixed* connected graph H with k edges is fully understood [22], the case of counting all connected k-edge subgraphs of a graph G remained open so far. As one of our main results, we completely understand the problem #EdgeSub(Φ) for the property Φ = connectivity. In what follows, we present our results, followed by an exposition of the most important techniques. Due to the space constraints, we have to defer the proofs to the full version.

Main Results

In a first part, we present our results on (#)EdgeSub(Φ); we continue with a definition and our results for a parameterized Tutte polynomial in a second part.

Our main results on (#)EdgeSub(Φ) can be categorized in roughly three categories: (1) exact algorithms and hardness results for the counting problem; (2) approximation algorithms for the counting problem; and (3) algorithms for the decision problem. For minor-closed properties Φ, we obtain exhaustive results for all three categories, for other (classes of) properties that we study, we obtain partial criteria. For an overview over our results on #EdgeSub(Φ), also consider Table 1; we go into more detail in the following.

Complete Classification for Minor-Closed Properties

Let us start with the case where the graph property Φ is closed under taking minors, that is, if Φ holds for a graph, then Φ still holds after removing vertices or edges, or after contracting edges. For minor-closed properties Φ, we obtain a complete picture of the complexity of

Table 1 An overview of the complexity of (#)EdgeSub(Φ) for different classes and examples of properties Φ, with respect to exact counting, approximate counting and decision. See further below for the definition of the matching and star criterion. All run-time lower bounds rely on the Exponential Time Hypothesis, and the absence of FPTRASes relies on the assumption that W[1] does not coincide with FPT under randomised parameterized reductions. We write "mixed" whenever the respective classes contain both tractable properties and hard properties. The known results about the clique problem are added for completeness; note that W[1]-hardness of decision immediately rules out an FPTRAS for approximate counting under the previous assumptions.

Property Φ	Exact Counting	Apx. Counting	Decision
Minor-closed[†] (e.g. Φ = planarity)	#W[1]-hard not in $f(k) \cdot \lvert G\rvert^{o(k/\log k)}$ (Main Theorem 1)	FPTRAS (Main Theorem 1)	FPT (Main Theorem 1)
Φ = connectivity	#W[1]-hard not in $f(k) \cdot \lvert G\rvert^{o(k/\log k)}$ (Main Theorem 2)	FPTRAS (follows from [26])	FPT (easy)
Φ = Hamiltonicity	#W[1]-hard not in $f(k) \cdot \lvert G\rvert^{o(k/\log k)}$ (Main Theorem 2)	unknown	unknown
Φ = Eulerianity	#W[1]-hard not in $f(k) \cdot \lvert G\rvert^{o(k/\log k)}$ (Main Theorem 2)	unknown	unknown
Φ = claw-freeness	#W[1]-hard not in $f(k) \cdot \lvert G\rvert^{o(k/\log k)}$ (Main Theorem 2)	unknown	unknown
Bounded matching number	FPT (Proposition 1.2)	FPTRAS (by exact counting)	FPT (by exact counting)
Bounded treewidth	mixed[‡]	FPTRAS (Main Theorem 3)	FPT (follows from [55])
Matching crit. **and** star crit.	mixed[*]	FPTRAS (Main Theorem 3)	FPT (Main Theorem 4)
Matching crit. **or** star crit.	mixed[‡]	mixed[§]	FPT (Main Theorem 4)
$\Phi = \Psi$ (see full version)	#W[1]-hard (full version)	no FPTRAS (full version)	FPT (full version)
Φ = Clique	#W[1]-hard ([34])	no FPTRAS (implicitly by [30])	W[1]-hard ([30])

[†] We assume that the minor-closed property Φ does not have bounded matching number, is not trivially true and that each forbidden minor has a vertex of degree at least 3.

[‡] Φ = true and Φ = false always yield fixed-parameter tractability of exact counting. $\Phi(H) = 1 \Leftrightarrow H$ is a matching yields #W[1]-hardness of exact counting [20]; note that the latter property is of bounded treewidth and satisfies the matching criterion.

[*] Φ = true always yields fixed-parameter tractability of exact counting. $\Phi(H) = 1 \Leftrightarrow (H$ is a matching or a star) yields #W[1]-hardness by Theorem 1.4; note that the latter property satisfies the matching criterion and the star criterion.

[§] Φ = true always yields an FPTRAS for approximate counting. $\Phi = \Psi$ (from the full version) does not allow for an FPTRAS while satisfying the matching criterion.

#EdgeSub(Φ) and EdgeSub(Φ). In what follows, we say that a property Φ has *bounded matching number* if there is a constant bound on the size of a largest matching in graphs satisfying Φ.

▶ **Main Theorem 1.** *Let Φ denote a minor-closed graph property.*

1. Exact Counting*: If Φ is either trivially true or of bounded matching number, then the (exact) counting version #EdgeSub(Φ) is fixed-parameter tractable. Otherwise, the problem #EdgeSub(Φ) is #W[1]-hard. If, additionally, each forbidden minor of Φ has a vertex of degree at least 3, and the Exponential Time Hypothesis holds, then #EdgeSub(Φ) cannot be solved in time $f(k) \cdot |G|^{o(k/\log k)}$, for any function f.*
2. Approximate Counting*: The problem #EdgeSub(Φ) always has a fixed-parameter tractable randomised approximation scheme (FPTRAS).*[2]
3. Decision: *The problem EdgeSub(Φ) is always fixed-parameter tractable.*

Consider for example the property Φ of being planar: planar graphs do not have bounded matching number. Additionally, by Kuratowski's Theorem, the forbidden minors of planar graphs are the 3-biclique $K_{3,3}$ and the 5-clique K_5. Since both $K_{3,3}$ and K_5 contain a vertex of degree at least 3, we conclude that computing the number of planar subgraphs with k edges in a graph G is #W[1]-hard and, assuming ETH, cannot be solved in time $f(k) \cdot |G|^{o(k/\log k)}$ for any function f. In sharp contrast, approximating the number of planar subgraphs with k edges in a graph, as well as deciding whether there is such a planar subgraph can be done efficiently. We obtain Main Theorem 1 as a combination of our (more general) results for each of the three settings that we study; we discuss these results next.

Results for Exact Counting

Let us return to the case of arbitrary graph properties Φ. Without any further assumptions on Φ, the naive algorithm for #EdgeSub(Φ) on the input (k, G) proceeds by enumerating the k-edge subsets of G and counting the number of cases where the corresponding subgraph satisfies Φ. This leads to a running time of the form $f(k) \cdot |V(G)|^{2k+O(1)}$. However, at least the linear constant in the exponent can be substantially improved using the currently fastest known algorithm for counting subgraphs with k edges due to Curticapean, Dell and Marx [21]. In the full version, we show that it easily extends to the case of #EdgeSub(Φ):

▶ **Proposition 1.1.** *Let Φ denote a computable graph property. Then #EdgeSub(Φ) can be solved in time $f(k) \cdot |V(G)|^{0.174k+o(k)}$, where f is some computable function.*

On the other hand, it was shown by Curticapean and Marx [22] that for the property Φ of being a matching, the problem #EdgeSub(Φ) cannot be solved in time $f(k) \cdot |V(G)|^{o(k/\log k)}$ for any function f, unless ETH fails. In other words, asymptotically and up to a factor of $1/\log k$, the exponent of $|V(G)|$ in the running time of #EdgeSub(Φ) cannot be improved without posing any restriction on Φ.

The goal is hence to identify properties Φ for which the algorithm in Proposition 1.1 *can* be (significantly) improved. In the best possible outcome, we hope to identify the properties for which the exponent of $|V(G)|$ does not depend on k; those cases are precisely the fixed-parameter tractable ones. An easy consequence of known results for subgraph counting (see for instance [22]) establishes the following tractability criterion; we include the proof only for the sake of completeness in the full version:

[2] The formal definition is given in the full version; intuitively an FPTRAS is the parameterized equivalent of a fully polynomial-time randomised approximation scheme (FPRAS).

▶ **Proposition 1.2.** *Let Φ denote a computable graph property satisfying that there is $M > 0$ such that for all k either the graphs with k edges satisfying Φ or the graphs with k edges satisfying $\neg\Phi$ have matching number bounded by M. Then #EdgeSub(Φ) is fixed-parameter tractable.*

Examples of properties satisfying the tractability criterion of Proposition 1.2 include, among others, the property of being a star, or the complement thereof. We conjecture that all remaining properties induce #W[1]-hardness and rule out any algorithm running in time $f(k) \cdot |G|^{o(k/\log k)}$ for any function f, unless ETH fails.[3] For the case of minor-closed graph properties, we have seen above that this conjecture holds.

Further, the techniques we develop to prove hardness of #EdgeSub(Φ) for minor-closed properties Φ in Main Theorem 1 can also be applied directly to show hardness for other specific properties Φ. Below, we record several natural examples of such properties which are covered by our methods.

▶ **Main Theorem 2.** *Consider the following graph properties.*

- $\Phi_C(H) = 1$ *if and only if H is connected.*
- $\Phi_H(H) = 1$ *if and only if H is Hamiltonian.*
- $\Phi_E(H) = 1$ *if and only if H is Eulerian.*
- $\Phi_{CF}(H) = 1$ *if and only if H is claw-free.*

For $\Phi \in \{\Phi_C, \Phi_H, \Phi_E, \Phi_{CF}\}$, the problem #EdgeSub($\Phi$) is #W[1]-hard. Further, unless ETH fails, the problem #EdgeSub(Φ) cannot be solved in time $f(k) \cdot |G|^{o(k/\log k)}$ for any function f.

Results for Approximate Counting and Decision

Our results on exact counting indicate that we have to relax the problem if we aim for tractability results for a larger variety of properties. One approach is to only ask for an *approximate* count of the number of k-edge subgraphs satisfying Φ. Tractability of approximation in the parameterized setting is given by the notion of a *fixed-parameter tractable randomized approximation scheme* (FPTRAS) as introduced by Arvind and Raman [5]. While we give the formal definition in full version, it suffices for now to think of an FPTRAS as a fixed-parameter tractable algorithm that can compute an arbitrarily good approximation of the answer with high probability. Readers familiar with the classical notions of approximate counting algorithms should think of an FPTRAS as an FPRAS in which we additionally allow a factor of $f(k)$ in the running time, for any computable function f.

For the statement of our results, we say that a property Φ satisfies the *matching criterion* if it is true for all but finitely many matchings, and we say that it satisfies the *star criterion* if it is true for all but finitely many stars. Furthermore, we say that Φ has bounded treewidth if there is a constant upper bound on the treewidth of graphs that satisfy Φ.

▶ **Main Theorem 3.** *Let Φ denote a computable graph property. If Φ satisfies the matching criterion* ***and*** *the star criterion, or if Φ has bounded treewidth, then #EdgeSub(Φ) admits an FPTRAS.*

For example, the property of being planar satisfies both, the star and the matching criterion. Moreover, we can show that every minor-closed graph property Φ has either bounded treewidth or satisfies matching and star criterion, and thus always admits an

[3] Note that it does not matter whether we choose $|G|$ or $|V(G)|$ for the size of the large graph since we care about the asymptotic behaviour of the exponent.

FPTRAS. Additionally, if not only exact but also approximate counting is intractable, we ask whether we can at least obtain an efficient algorithm for the decision version EdgeSub(Φ). Again, we obtain a tractability criterion; observe the subtle difference in the tractability criterion compared to Main Theorem 3.

▶ **Main Theorem 4.** *Let Φ denote a computable graph property. If Φ satisfies the matching criterion* ***or*** *the star criterion, or if Φ has bounded treewidth, then* EdgeSub(Φ) *is fixed-parameter tractable.*

As an easy corollary, we can conclude that for monotone, that is, subgraph-closed properties Φ, the problem EdgeSub(Φ) is always fixed-parameter tractable.[4]

For many previously studied problems, the complexity analysis of approximate counting and decision were related: often an algorithm solving one setting can be used to solve the other setting [51, 26]. However, in our results Main Theorems 3 and 4 we see an asymmetry between the two settings: it suffices for Φ to satisfy only one of the star and the matching criterion to induce tractability of the decision version, but we require satisfaction of both for approximate counting. One might expect that this reflects a shortcoming of our proof methods (and that in fact it suffices to check one of the criteria to have tractability of approximate counting). Interestingly, this is *not* the case:

▶ **Proposition 1.3.** *There is a computable graph property Ψ (see full version) that satisfies the matching criterion, but not the star criterion, such that* EdgeSub(Ψ) *is fixed-parameter tractable, but* #EdgeSub(Ψ) *does not admit an FPTRAS unless* W[1] *coincides with* FPT *(the class of all fixed-parameter tractable decision problems) under randomised parameterized reductions.*

Dichotomy for Evaluating a parameterized Tutte Polynomial

As a final part of the presentation of our main results, let us discuss our results on a parameterized Tutte polynomial.

The classical Tutte polynomial (as well as its specializations like the chromatic, flow or reliability polynomial) have received widespread attention, both from a combinatorial as well as a complexity theoretic perspective [42, 2, 65, 6, 37, 25, 10, 8]. The classical Tutte polynomial is of special interest from a complexity theoretic perspective, as the Tutte polynomial encodes a plethora of properties of a graph: prominent examples include the chromatic number, the number of acyclic orientations, and the number of spanning trees; we refer the reader to the work of Jaeger et al. [42] for a comprehensive overview. Formally, the Tutte polynomial is a bivariate graph polynomial defined as follows (see [42]):

$$T_G(x,y) := \sum_{A \subseteq E(G)} (x-1)^{k(A)-k(E(G))} \cdot (y-1)^{k(A)+\#A-\#V(G)},$$

where $k(S)$ is the number of connected components of the graph $(V(G), S)$. In the aforementioned work, Jaeger et al. [42] also classified the complexity of evaluating the Tutte Polynomial in every pair of (complex) coordinates, that is, for every pair (a, b), the complexity of computing the function $G \mapsto T_G(a, b)$ is fully understood.

[4] Every graph property has either bounded treewidth or unbounded matching number. In the latter case, if the property is additionally monotone, it must satisfy the matching criterion.

In this work, we consider the following parameterized version of the Tutte Polynomial by restricting to edge-subsets A in G of size k:

$$T_G^k(x,y) := \sum_{A \in \binom{E(G)}{k}} (x-1)^{k(A)-k(E(G))} \cdot (y-1)^{k(A)+k-\#V(G)} .$$

We observe that the parameterized Tutte polynomial can be seen as a weighted version of counting small k-edge subgraph patterns by assigning to each k-edge subset A of G the weight

$$(x-1)^{k(A)-k(E(G))} \cdot (y-1)^{k(A)+k-\#V(G)} .$$

Moreover, we point out that $T_G^k(x,y)$ is related to a generalization of the bases generating function for matroids [4]. By establishing a so-called deletion-contraction recurrence, we show that $T_G^k(x,y)$ has similar expressive power as its classical counterpart $T_G(x,y)$:

▶ **Main Theorem 5.** *For any graph G and positive integer k, the following graph invariants are encoded in $T_G^k(x,y)$:*

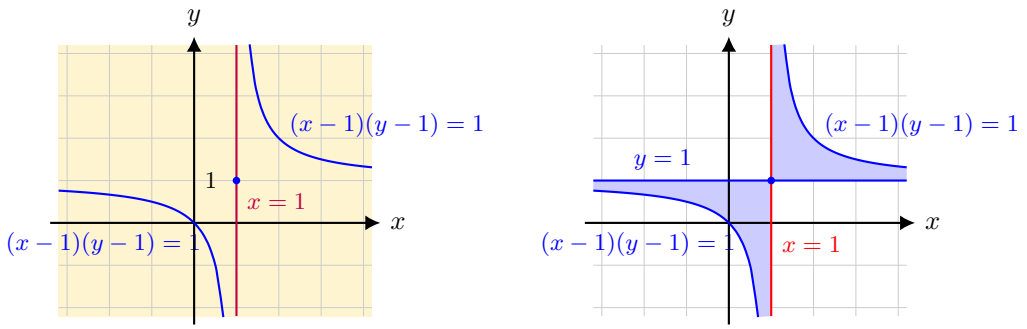
1. *$T_G^k(2,1)$ is the number of k-forests in G. In other words $T_G^k(2,1)$ corresponds to the problem #EdgeSub(Φ) for the property Φ of being a forest.*
2. *For each positive integer c, the values of $T_G^k(1-c,0)$ determine[5] the numbers of pairs (A,σ), where A is a k-edge subset of G, and σ is a proper c-colouring of $(V(G),A)$.*
3. *From $T_G^k(2,0)$ we can compute the numbers of pairs $(A,\vec{\eta})$, where A is a k-edge subset of G, and $\vec{\eta}$ is an acyclic orientation of $(V(G),A)$.*
4. *$T_G^k(2,0)$ also determines the number of k-edge subsets A of G, such that $(V(G),A)$ has even Betti Number (we give a formal definition of the Betti number in the full version).*
5. *$T_G^k(0,2)$ determines the number of k-edge subsets A of G, such that $(V(G),A)$ has an even number of components.*

Note that, while #EdgeSub(Φ) only allows us to count the number of subgraphs with k edges that satisfy Φ, the parameterized Tutte polynomial allows us to count more intricate objects, such as tuples of an edge-subset and a colouring (or acyclic orientation) on the induced graph. From a complexity theoretic point of view, we obtain a similar result as [42], albeit only for rational coordinates: for each *fixed* pair (x,y) of coordinates, we consider the problem receiving as input a graph G and a positive integer k and computing $T_G^k(x,y)$. Following the paradigm of this work, we choose k as a parameter, that is, we consider inputs in which k is significantly smaller than $|G|$.

▶ **Main Theorem 6.** *Let (x,y) denote a pair of rational numbers. The problem of computing $T_G^k(x,y)$ is solvable in polynomial-time if $x=y=1$ or $(x-1)(y-1)=1$, fixed-parameter tractable, but #P-hard, if $x=1$ and $y \neq 1$, and #W[1]-hard otherwise.*

The class #P is the counting version of NP [63, 64] and, in particular, the #P-hard cases in the above classification are not polynomial-time tractable unless the polynomial-time hierarchy collapses to P [61]. Consider Figure 1 for a depiction of the classification. Note that Main Theorem 6 yields #W[1]-hardness for each of the aforementioned problems from Main Theorem 5. Note further, that the tractable cases are similar, but not equal to the classical counterpart [42].

[5] They are equal up to trivial modifications; in particular, their complexities coincide.



(a) Points of the parameterized Tutte polynomial that can be computed in polynomial-time (blue) and that are fixed-parameter tractable, but #P-hard (red). Exact computation at any other point (yellow) is #W[1]-hard.

(b) Points of the parameterized Tutte polynomial that allow for an FPRAS (blue) and for an FPTRAS (red); all points on the boundary of the blue area are included. The complexity of approximation is open for all points outside of the coloured region.

Figure 1 Points of the parameterized Tutte polynomial that can be computed in FPT-time (a) exactly or (b) approximately. We emphasize that a full classification for exact counting is established, while the complexity of approximation remains open outside of the coloured area.

Moreover, our proof uses entirely different tools than [42] and illustrates the power and utility of the method presented in the subsequent discussion of our techniques.

Having fully classified the complexity of exact evaluation of the parameterized Tutte Polynomial, we also consider the complexity of approximate evaluation. We identify two regions bounded by the hyperbola $(x-1)(y-1)=1$ and the lines $x=1$ and $y=1$ as efficiently approximable; consider Figure 1b for a depiction.

▶ **Main Theorem 7.** *Let (x,y) denote a pair of rational numbers. If $0 \leq (x-1)(y-1) \leq 1$, then $T_G^k(x,y)$ has an FPTRAS. If additionally $x \neq 1$ or $y = 1$ then $T_G^k(x,y)$ even has a fully polynomial-time randomized approximation scheme (FPRAS).*

Techniques

Our Main Theorems 3, 4, and 7 are obtained easily: the proof of Main Theorem 3 is a standard application (see for instance [51]) of the Monte-Carlo approach, in combination with Ramsey's theorem, and Arvind and Raman's algorithm for approximately counting subgraphs of bounded treewidth [5]. The proof of Main Theorem 4 uses a standard parameterized Win-Win approach for graphs of bounded treewidth or bounded degree. Finally, the proof of Main Theorem 7 is an easy consequence of the work of Anari et al. [4] on approximate counting via log-concave polynomials.

Hence, in this technical discussion, we want to focus on the technique that enables us to prove the lower bounds for Main Theorems 1 and 2 and, perhaps surprisingly, also for Main Theorem 6.

As a main component, we use the Complexity Monotonicity framework of Curticapean, Dell and Marx [20]. Given a property Φ and a positive integer k, we write $\#\mathsf{EdgeSub}(\Phi, k \to \star)$ for the function that maps a graph G to the number of k-edge subgraphs of G that satisfy Φ. Using a well-known transformation via Möbius inversion [49, Chapter 5.2], we can show that there are rational numbers $a_1, \ldots, a_\ell$ and graphs $H_1, \ldots, H_\ell$ such that for each graph G we have

$$\#\mathsf{EdgeSub}(\Phi, k \to G) = \sum_{i=1}^{k} a_i \cdot \#\mathsf{Hom}(H_i \to G)\,, \tag{1}$$

where $\#\mathsf{Hom}(H_i \to G)$ is the number of graph homomorphisms from H_i to G. In other words, we can express $\#\mathsf{EdgeSub}(\Phi, k \to \star)$ as a finite linear combination of homomorphism counts. Here, we can then apply the Complexity Monotonicity framework [21], which asserts that computing a finite linear combination of homomorphism counts is *precisely as hard as* its hardest term (among the terms with a non-zero coefficient). However, the complexity of computing the number of homomorphisms from small pattern graphs to large host graphs is very well-understood [24, 50]. Roughly speaking, the higher the treewidth of the pattern graph, the harder the problem becomes; we make this formal in the full version.

Instead of our original problem #EdgeSub(Φ), we can thus consider the problem of computing linear combinations of graph homomorphism counts. In particular, to obtain hardness, it suffices to understand for which of the coefficients in equation (1) we have $a_i \neq 0$, depending on k and Φ.

Relying on the well-known fact that the Möbius function of the partition lattice alternates in sign, Curticapean, Dell, and Marx [21] observed that non-trivial cancellations cannot occur in equation (1) if, for each k, every k-edge graph that satisfies Φ must have the same number of vertices. Consequently, if the matching number is unbounded, those properties yield #W[1]-hardness. An example for such a property is the case of $\Phi(H) = 1$ if and only if H is a tree. In contrast, the intractability result for the case of $\Phi =$ acyclicity (that is, being a forest) turned out to be much harder to show [11], indicated by connections to parameterized counting problems in matroid theory.

In later work, the coefficients a_i were shown to have even more interesting structure: the coefficients a_i describe topological and algebraic invariants of the set of pattern graphs. For example, in [57] it was shown that the coefficient of the k-clique in case of counting vertex-induced subgraphs with property Φ is the reduced Euler characteristic of a simplicial complex associated with Φ and can thus, if non-zero, be used to establish evasiveness of certain graph properties [46].

In this work, we prove additional insights into said coefficients a_i.[6] For any graph H we give an explicit formula for its coefficient a_H in terms of a sum over the *fractures* on H, an additional combinatorial structure on a graph H resembling, to some extent, a gadget construction used for the classification of the subgraph counting problem [22] (see full version for details). Our most crucial insight is then that we can drastically simplify the expression of the coefficient a_H modulo a prime ℓ if H admits a vertex-transitive action of a group of order given by a power of ℓ. In this case, we obtain an action of the group on the set of fractures on H and in the formula for a_H all contributions from fractures not fixed by the group cancel out modulo ℓ.

In particular, we consider graphs H which are Cayley graphs of a finite group of prime power order and a symmetric set of generators. Since the Cayley graph of a group always has a natural vertex-transitive action of this group, such Cayley graphs always have the desired symmetry properties. We exploit this by showing that there is a constant number of fractures fixed by the group action. This in turn allows us to write (a_H modulo ℓ) as a finite sum of terms depending on the value of Φ on some explicit graphs.

[6] For technical reasons, the approach we describe below requires us to consider a coloured version of #EdgeSub(Φ), which is, however, shown to be interreducible with the uncoloured one.

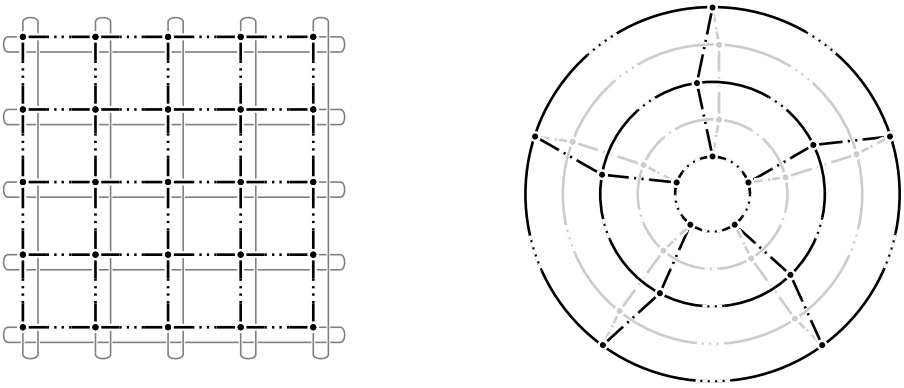


Figure 2 Two isomorphic representations of the toroidal grid $\circledcirc_\ell$: On the left hand side as a grid with connected endpoints, on the right hand side as a stylized torus.

Specifically, the first set of Cayley graphs we consider are the toroidal grids $\circledcirc_\ell$, which are depicted in Figure 2. Since the treewidth of $\circledcirc_\ell$ diverges with ℓ, we thus obtain a #W[1]-hardness result whenever the coefficient $a_{\circledcirc_\ell}$ does not vanish for infinitely many ℓ. Writing M_k for the matching of size k, P_2 for the path consisting of 2 edges, C_k for the cycle of length k, S_k for a sun (a cycle with dangling edges) of size k, and $\circledcirc_k$ for the toroidal grid of size k, our first main technical result reads as follows:

▶ **Theorem 1.4** (Simplified version)**.** *Let Φ denote a computable graph property and assume that infinitely many primes ℓ satisfy the equation*[7]

$$-6\Phi(M_{2\ell^2})+4\Phi(M_{\ell^2}+\ell C_\ell)+8\Phi(\ell^2 P_2)-\Phi(2\ell C_\ell)-2\Phi(\ell C_{2\ell})-4\Phi(\ell S_\ell)+\Phi(\circledcirc_\ell)\neq 0 \mod \ell\,. \quad (2)$$

Then #EdgeSub(Φ) *is* #W[1]*-hard.*

As a toy example for an application of Theorem 1.4, let us consider the property Φ of being connected. Observe that among the graphs in (2), only $\circledcirc_\ell$ is connected, and thus the sum is always 1 for $\ell \geq 2$. Thus, indeed the left-hand side of (2) is nonzero, proving that #EdgeSub(Φ) is #W[1]-hard.

Using Theorem 1.4, we can prove most of the #W[1]-hardness results of Main Theorem 1. However, using the toroidal grid $\circledcirc_\ell$ we cannot prove (almost) tight conditional lower bounds: the treewidth of $\circledcirc_\ell$ grows only with the *square-root* of the parameter k (that is the number of edges of the graph). To address this problem, we consider a second family of 4-regular Cayley graphs, constructed explicitly by Peyerimhoff and Vdovina [54], which have the additional property of being *expander graphs*. In particular, for these graphs, the treewidth grows *linearly* in the number of edges. This allows us to obtain almost tight conditional lower bounds. The variant of Theorem 1.4 for these Cayley graph expanders can be found in the full version of this paper.

The only drawback of the Cayley graphs from [54] is that the corresponding groups always have orders given by powers of 2 (in contrast to having arbitrary primes ℓ in Theorem 1.4). Hence, our criterion for hardness is the nonvanishing of some expression modulo 2. Ultimately, this is the reason why for the conditional lower bounds in Main Theorem 1 we need to exclude forbidden minors having a vertex of degree 2 or less.

[7] We write $+$ for (disjoint) graph union and ℓH for the graph consisting of ℓ disjoint copies of H. Further, we set $\Phi(H) = 1$ if H satisfies Φ and $\Phi(H) = 0$ otherwise.

Finally, to obtain Main Theorem 6, we express the parameterized Tutte polynomial at a rational point (x, y) as a linear combination of (fractures of) toroidal grids; the proof of Theorem 1.4 then essentially shows that this linear combination always contains a graph with unbounded treewidth, yielding #W[1]-hardness.

References

1. Noga Alon, Phuong Dao, Iman Hajirasouliha, Fereydoun Hormozdiari, and S. Cenk Sahinalp. Biomolecular network motif counting and discovery by color coding. *Bioinformatics*, 24(13):i241–i249, July 2008. doi:10.1093/bioinformatics/btn163.
2. Noga Alon, Alan M. Frieze, and Dominic Welsh. Polynomial Time Randomized Approximation Schemes for Tutte-Gröthendieck Invariants: The Dense Case. *Random Struct. Algorithms*, 6(4):459–478, 1995. doi:10.1002/rsa.3240060409.
3. Noga Alon, Raphael Yuster, and Uri Zwick. Color-Coding. *J. ACM*, 42(4):844–856, 1995. doi:10.1145/210332.210337.
4. Nima Anari, Kuikui Liu, Shayan Oveis Gharan, and Cynthia Vinzant. Log-concave polynomials II: high-dimensional walks and an FPRAS for counting bases of a matroid. In Moses Charikar and Edith Cohen, editors, *Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing, STOC 2019, Phoenix, AZ, USA, June 23-26, 2019*, pages 1–12. ACM, 2019. doi:10.1145/3313276.3316385.
5. Vikraman Arvind and Venkatesh Raman. Approximation Algorithms for Some Parameterized Counting Problems. In *Algorithms and Computation, 13th International Symposium, ISAAC 2002 Vancouver, BC, Canada, November 21-23, 2002, Proceedings*, pages 453–464, 2002. doi:10.1007/3-540-36136-7_40.
6. Andreas Björklund, Thore Husfeldt, Petteri Kaski, and Mikko Koivisto. Computing the Tutte Polynomial in Vertex-Exponential Time. In *49th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2008, October 25-28, 2008, Philadelphia, PA, USA*, pages 677–686. IEEE Computer Society, 2008. doi:10.1109/FOCS.2008.40.
7. Andreas Björklund, Thore Husfeldt, Petteri Kaski, and Mikko Koivisto. Narrow sieves for parameterized paths and packings. *J. Comput. Syst. Sci.*, 87:119–139, 2017. doi:10.1016/j.jcss.2017.03.003.
8. Andreas Björklund and Petteri Kaski. The Fine-Grained Complexity of Computing the Tutte Polynomial of a Linear Matroid. Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms, SODA 2021, Alexandria, VA, USA, January 10-13, 2021, to appear.
9. Cornelius Brand, Holger Dell, and Thore Husfeldt. Extensor-coding. In *Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2018, Los Angeles, CA, USA, June 25-29, 2018*, pages 151–164, 2018. doi:10.1145/3188745.3188902.
10. Cornelius Brand, Holger Dell, and Marc Roth. Fine-Grained Dichotomies for the Tutte Plane and Boolean #CSP. *Algorithmica*, 81(2):541–556, 2019. doi:10.1007/s00453-018-0472-z.
11. Cornelius Brand and Marc Roth. Parameterized Counting of Trees, Forests and Matroid Bases. In *Computer Science - Theory and Applications - 12th International Computer Science Symposium in Russia, CSR 2017, Kazan, Russia, June 8-12, 2017, Proceedings*, pages 85–98, 2017. doi:10.1007/978-3-319-58747-9_10.
12. Ashok K. Chandra and Philip M. Merlin. Optimal Implementation of Conjunctive Queries in Relational Data Bases. In *Proceedings of the 9th Annual ACM Symposium on Theory of Computing, May 4-6, 1977, Boulder, Colorado, USA*, pages 77–90, 1977. doi:10.1145/800105.803397.
13. Hubie Chen and Stefan Mengel. Counting Answers to Existential Positive Queries: A Complexity Classification. In *Proceedings of the 35th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems, PODS 2016, San Francisco, CA, USA, June 26 - July 01, 2016*, pages 315–326, 2016. doi:10.1145/2902251.2902279.

14 Jianer Chen, Benny Chor, Mike Fellows, Xiuzhen Huang, David W. Juedes, Iyad A. Kanj, and Ge Xia. Tight lower bounds for certain parameterized NP-hard problems. *Inf. Comput.*, 201(2):216–231, 2005. doi:10.1016/j.ic.2005.05.001.

15 Jianer Chen, Xiuzhen Huang, Iyad A. Kanj, and Ge Xia. Strong computational lower bounds via parameterized complexity. *J. Comput. Syst. Sci.*, 72(8):1346–1367, 2006. doi:10.1016/j.jcss.2006.04.007.

16 Jianer Chen, Joachim Kneis, Songjian Lu, Daniel Mölle, Stefan Richter, Peter Rossmanith, Sing-Hoi Sze, and Fenghui Zhang. Randomized Divide-and-Conquer: Improved Path, Matching, and Packing Algorithms. *SIAM Journal on Computing*, 38(6):2526–2547, 2009. doi:10.1137/080716475.

17 Yijia Chen, Marc Thurley, and Mark Weyer. Understanding the Complexity of Induced Subgraph Isomorphisms. In *Automata, Languages and Programming, 35th International Colloquium, ICALP 2008, Reykjavik, Iceland, July 7-11, 2008, Proceedings, Part I: Tack A: Algorithms, Automata, Complexity, and Games*, pages 587–596, 2008. doi:10.1007/978-3-540-70575-8_48.

18 Stephen A. Cook. The Complexity of Theorem-Proving Procedures. In *Proceedings of the 3rd Annual ACM Symposium on Theory of Computing, May 3-5, 1971, Shaker Heights, Ohio, USA*, pages 151–158, 1971. doi:10.1145/800157.805047.

19 Derek G. Corneil and C. C. Gotlieb. An Efficient Algorithm for Graph Isomorphism. *J. ACM*, 17(1):51–64, 1970. doi:10.1145/321556.321562.

20 Radu Curticapean. Counting matchings of size k is w[1]-hard. In Fedor V. Fomin, Rusins Freivalds, Marta Z. Kwiatkowska, and David Peleg, editors, *Automata, Languages, and Programming - 40th International Colloquium, ICALP 2013, Riga, Latvia, July 8-12, 2013, Proceedings, Part I*, volume 7965 of *Lecture Notes in Computer Science*, pages 352–363. Springer, 2013. doi:10.1007/978-3-642-39206-1_30.

21 Radu Curticapean, Holger Dell, and Dániel Marx. Homomorphisms are a good basis for counting small subgraphs. In *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2017, Montreal, QC, Canada, June 19-23, 2017*, pages 210–223, 2017. doi:10.1145/3055399.3055502.

22 Radu Curticapean and Dániel Marx. Complexity of Counting Subgraphs: Only the Boundedness of the Vertex-Cover Number Counts. In *55th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2014, Philadelphia, PA, USA, October 18-21, 2014*, pages 130–139, 2014. doi:10.1109/FOCS.2014.22.

23 Marek Cygan, Fedor V. Fomin, Lukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michal Pilipczuk, and Saket Saurabh. *Parameterized Algorithms*. Springer, 2015. doi:10.1007/978-3-319-21275-3.

24 Víctor Dalmau and Peter Jonsson. The complexity of counting homomorphisms seen from the other side. *Theor. Comput. Sci.*, 329(1-3):315–323, 2004. doi:10.1016/j.tcs.2004.08.008.

25 Holger Dell, Thore Husfeldt, Dániel Marx, Nina Taslaman, and Martin Wahlen. Exponential Time Complexity of the Permanent and the Tutte Polynomial. *ACM Trans. Algorithms*, 10(4):21:1–21:32, 2014. doi:10.1145/2635812.

26 Holger Dell, John Lapinskas, and Kitty Meeks. Approximately counting and sampling small witnesses using a colourful decision oracle. In Shuchi Chawla, editor, *Proceedings of the 2020 ACM-SIAM Symposium on Discrete Algorithms, SODA 2020, Salt Lake City, UT, USA, January 5-8, 2020*, pages 2201–2211. SIAM, 2020. doi:10.1137/1.9781611975994.135.

27 Holger Dell, Marc Roth, and Philip Wellnitz. Counting Answers to Existential Questions. In *46th International Colloquium on Automata, Languages, and Programming, ICALP 2019, July 9-12, 2019, Patras, Greece*, pages 113:1–113:15, 2019. doi:10.4230/LIPIcs.ICALP.2019.113.

28 Julian Dörfler, Marc Roth, Johannes Schmitt, and Philip Wellnitz. Counting Induced Subgraphs: An Algebraic Approach to #W[1]-hardness. In Peter Rossmanith, Pinar Heggernes, and Joost-Pieter Katoen, editors, *44th International Symposium on Mathematical Foundations of Computer Science, MFCS 2019, August 26-30, 2019, Aachen, Germany*, volume 138 of *LIPIcs*, pages 26:1–26:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019. doi:10.4230/LIPIcs.MFCS.2019.26.

29 Rodney G. Downey and Michael R. Fellows. Fixed-Parameter Tractability and Completeness I: Basic Results. *SIAM J. Comput.*, 24(4):873–921, 1995. doi:10.1137/S0097539792228228.

30 Rodney G. Downey and Michael R. Fellows. Fixed-Parameter Tractability and Completeness II: On Completeness for W[1]. *Theor. Comput. Sci.*, 141(1&2):109–131, 1995. doi:10.1016/0304-3975(94)00097-3.

31 Rodney G. Downey and Michael R. Fellows. *Fundamentals of Parameterized Complexity*. Texts in Computer Science. Springer, 2013. doi:10.1007/978-1-4471-5559-1.

32 Arnaud Durand and Stefan Mengel. Structural Tractability of Counting of Solutions to Conjunctive Queries. *Theory Comput. Syst.*, 57(4):1202–1249, 2015. doi:10.1007/s00224-014-9543-y.

33 Jack Edmonds. Paths, Trees, and Flowers. *Canadian Journal of Mathematics*, 17:449–467, 1965. doi:10.4153/CJM-1965-045-4.

34 Jörg Flum and Martin Grohe. The Parameterized Complexity of Counting Problems. *SIAM J. Comput.*, 33(4):892–922, 2004. doi:10.1137/S0097539703427203.

35 Jörg Flum and Martin Grohe. *Parameterized Complexity Theory*. Texts in Theoretical Computer Science. An EATCS Series. Springer, 2006. doi:10.1007/3-540-29953-X.

36 Fedor V. Fomin, Daniel Lokshtanov, and Saket Saurabh. Efficient Computation of Representative Sets with Applications in Parameterized and Exact Algorithms. In Chandra Chekuri, editor, *Proceedings of the Twenty-Fifth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2014, Portland, Oregon, USA, January 5-7, 2014*, pages 142–151. SIAM, 2014. doi:10.1137/1.9781611973402.10.

37 Leslie Ann Goldberg and Mark Jerrum. The Complexity of Computing the Sign of the Tutte Polynomial. *SIAM J. Comput.*, 43(6):1921–1952, 2014. doi:10.1137/12088330X.

38 Joshua A. Grochow and Manolis Kellis. Network Motif Discovery Using Subgraph Enumeration and Symmetry-Breaking. In Terry Speed and Haiyan Huang, editors, *Research in Computational Molecular Biology*, pages 92–106, Berlin, Heidelberg, 2007. Springer Berlin Heidelberg.

39 Martin Grohe. Parameterized Complexity for the Database Theorist. *SIGMOD Rec.*, 31(4):86–96, 2002. doi:10.1145/637411.637428.

40 Martin Grohe, Thomas Schwentick, and Luc Segoufin. When is the evaluation of conjunctive queries tractable? In *Proceedings on 33rd Annual ACM Symposium on Theory of Computing, July 6-8, 2001, Heraklion, Crete, Greece*, pages 657–666, 2001. doi:10.1145/380752.380867.

41 Russell Impagliazzo and Ramamohan Paturi. On the Complexity of k-SAT. *J. Comput. Syst. Sci.*, 62(2):367–375, 2001. doi:10.1006/jcss.2000.1727.

42 F. Jaeger, Dirk L. Vertigan, and Dominic J. A. Welsh. On the computational complexity of the Jones and Tutte polynomials. *Mathematical Proceedings of the Cambridge Philosophical Society*, 108(1):35–53, 1990. doi:10.1017/S0305004100068936.

43 Mark Jerrum and Kitty Meeks. The parameterised complexity of counting connected subgraphs and graph motifs. *J. Comput. Syst. Sci.*, 81(4):702–716, 2015. doi:10.1016/j.jcss.2014.11.015.

44 Mark Jerrum and Kitty Meeks. Some Hard Families of Parameterized Counting Problems. *TOCT*, 7(3):11:1–11:18, 2015. doi:10.1145/2786017.

45 Mark Jerrum and Kitty Meeks. The parameterised complexity of counting even and odd induced subgraphs. *Combinatorica*, 37(5):965–990, 2017. doi:10.1007/s00493-016-3338-5.

46 Jeff Kahn, Michael E. Saks, and Dean Sturtevant. A topological approach to evasiveness. *Combinatorica*, 4(4):297–306, 1984. doi:10.1007/BF02579140.

47 Subhash Khot and Venkatesh Raman. Parameterized complexity of finding subgraphs with hereditary properties. *Theor. Comput. Sci.*, 289(2):997–1008, 2002. doi:10.1016/S0304-3975(01)00414-5.

48 Bingkai Lin. The Parameterized Complexity of the k-Biclique Problem. *J. ACM*, 65(5):34:1–34:23, 2018. doi:10.1145/3212622.

49 László Lovász. *Large Networks and Graph Limits*, volume 60 of *Colloquium Publications*. American Mathematical Society, 2012. URL: http://www.ams.org/bookstore-getitem/item=COLL-60.

50 Dániel Marx. Can You Beat Treewidth? *Theory of Computing*, 6(1):85–112, 2010. doi:10.4086/toc.2010.v006a005.

51 Kitty Meeks. The challenges of unbounded treewidth in parameterised subgraph counting problems. *Discrete Applied Mathematics*, 198:170–194, 2016. doi:10.1016/j.dam.2015.06.019.

52 R. Milo, S. Shen-Orr, S. Itzkovitz, N. Kashtan, D. Chklovskii, and U. Alon. Network Motifs: Simple Building Blocks of Complex Networks. *Science*, 298(5594):824–827, 2002. doi:10.1126/science.298.5594.824.

53 Ron Milo, Shalev Itzkovitz, Nadav Kashtan, Reuven Levitt, Shai Shen-Orr, Inbal Ayzenshtat, Michal Sheffer, and Uri Alon. Superfamilies of evolved and designed networks. *Science*, 303(5663):1538–1542, 2004. doi:10.1126/science.1089167.

54 Norbert Peyerimhoff and Alina Vdovina. Cayley graph expanders and groups of finite width. *J. Pure Appl. Algebra*, 215(11):2780–2788, 2011. doi:10.1016/j.jpaa.2011.03.018.

55 Jürgen Plehn and Bernd Voigt. Finding minimally weighted subgraphs. In Rolf H. Möhring, editor, *Graph-Theoretic Concepts in Computer Science, 16rd International Workshop, WG '90, Berlin, Germany, June 20-22, 1990, Proceedings*, volume 484 of *Lecture Notes in Computer Science*, pages 18–29. Springer, 1990. doi:10.1007/3-540-53832-1_28.

56 Ofer Rahat, Uri Alon, Yaakov Levy, and Gideon Schreiber. Understanding hydrogen-bond patterns in proteins using network motifs. *Bioinformatics*, 25(22):2921–2928, September 2009. doi:10.1093/bioinformatics/btp541.

57 Marc Roth and Johannes Schmitt. Counting Induced Subgraphs: A Topological Approach to #W[1]-hardness. *Algorithmica*, 82(8):2267–2291, 2020. doi:10.1007/s00453-020-00676-9.

58 Marc Roth, Johannes Schmitt, and Philip Wellnitz. Counting Small Induced Subgraphs Satisfying Monotone Properties. 61th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2020, to appear.

59 Benjamin Schiller, Sven Jager, Kay Hamacher, and Thorsten Strufe. StreaM - A Stream-Based Algorithm for Counting Motifs in Dynamic Graphs. In Adrian-Horia Dediu, Francisco Hernández-Quiroz, Carlos Martín-Vide, and David A. Rosenblueth, editors, *Algorithms for Computational Biology*, pages 53–67, Cham, 2015. Springer International Publishing.

60 Falk Schreiber and Henning Schwöbbermeyer. Frequency Concepts and Pattern Detection for the Analysis of Motifs in Networks. In Corrado Priami, Emanuela Merelli, Pablo Gonzalez, and Andrea Omicini, editors, *Transactions on Computational Systems Biology III*, pages 89–104, Berlin, Heidelberg, 2005. Springer Berlin Heidelberg.

61 Seinosuke Toda. PP is as Hard as the Polynomial-Time Hierarchy. *SIAM J. Comput.*, 20(5):865–877, 1991. doi:10.1137/0220053.

62 Julian R. Ullmann. An algorithm for subgraph isomorphism. *J. ACM*, 23(1):31–42, 1976. doi:10.1145/321921.321925.

63 Leslie G. Valiant. The Complexity of Computing the Permanent. *Theor. Comput. Sci.*, 8:189–201, 1979. doi:10.1016/0304-3975(79)90044-6.

64 Leslie G. Valiant. The Complexity of Enumeration and Reliability Problems. *SIAM J. Comput.*, 8(3):410–421, 1979. doi:10.1137/0208032.

65 Dirk Vertigan. Bicycle Dimension and Special Points of the Tutte Polynomial. *J. Comb. Theory Ser. B*, 74(2):378–396, 1998. doi:10.1006/jctb.1998.1860.

The Greedy Algorithm Is *not* Optimal for On-Line Edge Coloring

Amin Saberi
Stanford University, CA, USA

David Wajc
Stanford University, CA, USA

Abstract

Nearly three decades ago, Bar-Noy, Motwani and Naor showed that no online edge-coloring algorithm can edge color a graph optimally. Indeed, their work, titled "the greedy algorithm is optimal for on-line edge coloring", shows that the competitive ratio of 2 of the naïve greedy algorithm is best possible online. However, their lower bound required bounded-degree graphs, of maximum degree $\Delta = O(\log n)$, which prompted them to conjecture that better bounds are possible for higher-degree graphs. While progress has been made towards resolving this conjecture for restricted inputs and arrivals or for random arrival orders, an answer for fully general *adversarial* arrivals remained elusive.

We resolve this thirty-year-old conjecture in the affirmative, presenting a $(1.9 + o(1))$-competitive online edge coloring algorithm for general graphs of degree $\Delta = \omega(\log n)$ under vertex arrivals. At the core of our results, and of possible independent interest, is a new online algorithm which rounds a fractional bipartite matching x online under vertex arrivals, guaranteeing that each edge e is matched with probability $(1/2 + c) \cdot x_e$, for a constant $c > 0.027$.

2012 ACM Subject Classification Theory of computation → Online algorithms

Keywords and phrases Online algorithms, edge coloring, greedy, online matching

Digital Object Identifier 10.4230/LIPIcs.ICALP.2021.109

Category Track A: Algorithms, Complexity and Games

Related Version *Full Version*: `https://arxiv.org/pdf/2105.06944.pdf`

Funding This research is partially supported by NSF award 1812919, ONR award N000141912550, and a gift from Cisco Research.

Acknowledgements We thank Janardhan Kulkarni for drawing our attention to [25].

1 Introduction

An edge coloring of a graph is a decomposition of its edge-set into few vertex-disjoint edge-sets (matchings), or *colors*. Edge coloring a graph of maximum degree Δ trivially requires at least Δ colors, and this is tight for bipartite graphs, by the century-old result of König [28]. For general graphs, Δ colors are not always sufficient (e.g., in odd-length cycles), yet $\Delta + 1$ colors are always sufficient, by Vizing's Theorem [35].

Algorithmically matching, or approximating, the optimal $\Delta(+1)$ colors needed to edge color a graph has been the focus of much concentrated effort, for numerous computational models. These include offline, online, distributed, parallel, and dynamic algorithms (see, e.g., [7–9, 11, 13, 25, 31, 34, 36] and references therein). These different models' specific challenges naturally impose limitations on the attainable approximations. For example, Holyer's Theorem [20] rules out efficient offline algorithms for computing an optimal edge coloring in general graphs, unless P=NP.

For online algorithms, the challenge is in making immediate and irrevocable decisions concerning edges' colors after only part of the input is revealed. For example, the input graph can either be revealed edge-by-edge (edge arrivals) or vertex-by-vertex (vertex arrivals), and an online algorithm must assign colors to edges after they are revealed, immediately and irrevocably. The measure of an online algorithm is its competitive ratio, which is the worst-case ratio of the number of colors used by the algorithm to those of the optimal offline algorithm, namely, Δ or $\Delta + 1$.

In both the edge-arrival and vertex-arrival settings, a simple greedy algorithm has competitive ratio 2. The natural question, then, is whether a better online algorithm exists. Some thirty years ago, Bar-Noy, Motwani and Naor [4] showed that this competitive ratio of 2 is best possible, and no online algorithm (randomized or deterministic) can do better, in either arrival model.

However, noting that their result only holds for bounded-degree n-node graphs, of maximum degree $\Delta = O(\log n)$, Bar-Noy et al. conjectured that better algorithms exist for graphs of sufficiently high maximum degree.

▶ **Conjecture 1.1** ([4]). *There exists a $(2 - \Omega(1))$-competitive online edge coloring algorithm under vertex arrivals in n-node graphs of maximum degree $\Delta = \omega(\log n)$.*

Bar-Noy et al. conjectured that the same holds under the more challenging edge-arrival model, and that moreover a $(1+o(1))$-competitive algorithm exists. These conjectures remain out of reach, though progress has been made on them over the years. For edge arrivals, a positive resolution of the stronger conjecture was achieved under the assumption of *random order* arrivals, where the input is generated adversarially, but its arrival order is randomly permuted by nature [1, 3, 5]. For adversarial vertex arrivals, Cohen et al. [9] showed that for *bipartite* graphs under one-sided vertex arrivals (vertices of one side are given, and the other side's vertices arrive), the conjectured $(1 + o(1))$-competitive ratio is achievable for $\Delta = \omega(\log n)$. Whether the competitive ratio of 2 of the greedy algorithm is optimal under *general* vertex arrivals, in *general* graphs, however, remained open.

We answer the above open question, resolving Conjecture 1.1 in the affirmative.

▶ **Theorem 1.2.** *There exists an online edge coloring algorithm which is $(1.897 + o(1))$-competitive w.h.p. on general n-node graphs with maximum degree $\Delta = \omega(\log n)$ under vertex arrivals.*

▶ Remark 1.3. For general Δ, the $o(1)$ term in the above theorem is of the form $\sqrt[\gamma]{\log n/\Delta}$, for some constant $\gamma > 0$. This implies a better than two approximation ratio for sufficiently large $\Delta = O(\log n)$. For simplicity of exposition, we do not elaborate on this point.

1.1 Techniques

To obtain our results, we combine and extend several previous algorithmic ideas.

Our starting point is the following natural recursive approach, due to Karloff and Shmoys [25], which reduces edge coloring a general graph G to edge coloring random bipartite subgraphs. Their idea was to assign each vertex to either side of a random subgraph uniformly, resulting in a bipartite subgraph H of G with maximum degree $\Delta/2 + o(\Delta)$ for $\Delta = \omega(\log n)$,

by standard tail bounds. Consequently, applying an α-approximate algorithm to the random bipartite graph and recursing on the remaining edges is easily shown to result in an edge coloring using $\alpha \cdot \Delta/2 + o(\Delta) + \alpha \cdot \Delta/4 + o(\Delta) \cdots = \alpha \cdot \Delta + o(\Delta)$ colors. Importantly for us, this approach, originally used in the context of NC algorithms by [25], is implementable online, by sampling the random bipartitions in advance. (See Section 5.)

At this point, one might be tempted to use the online algorithm of Cohen et al. [9] for these random bipartite subgraphs. Unfortunately, the reduction of Karloff and Shmoys [25] applied to online edge coloring with general vertex arrivals requires an online algorithm for bipartite graphs with *interleaved* arrivals, and not one-sided arrivals, as handled by [9]. To instantiate the Karloff-Shmoys approach, we therefore present a $(2-c)$-competitive edge coloring algorithm for interleaved vertex arrivals in bipartite graphs, which, when combined with the approach of [25], then extends to general graphs.

To obtain an edge-coloring algorithm for bipartite graphs under interleaved vertex arrival, we extend the approach of Cohen et al. [9], who showed that an $(\alpha + o(1))$-competitive edge coloring can be achieved by repeatedly applying a matching algorithm which matches each edge with probability $(1/\alpha)/\Delta$. For each vertex of degree $\Delta(1 - o(1))$, such a matching results in v being matched with probability $(1/\alpha) \cdot (1 - o(1))$. Repeating the above a super-logarithmic number of times (making use of $\Delta = \omega(\log n)$) therefore decreases the maximum degree of the graph at a rate of roughly one per α colors used. Cohen et al. used this approach with $\alpha = 1 + o(1)$, using an online matching algorithm from [10], on bipartite graphs under one-sided arrivals. We observe that this approach extends to arbitrary α and any arrival model, including interleaved vertex arrivals in bipartite graphs. (See Section 6.)

Motivated by the above discussion, we design an online matching algorithm for bipartite graphs under interleaved arrivals, which matches each edge with probability $(1/2 + c)/\Delta$, for some constant $c > 0$. More generally, and of possible independent interest, we design an online rounding algorithm for bipartite fractional matchings under interleaved vertex arrivals, with a multiplicative factor of $1/2 + c$. That is, we show how, given a bipartite graph G and a fractional matching x in G revealed vertex-by-vertex, one can output a randomized matching which matches each edge e in G with probability $(1/2 + c) \cdot x_e$. This extends a similar online rounding algorithm previously developed by the authors with Papadimitriou and Pollner [33] in the context of online stochastic optimization, but which only works under one-sided vertex arrivals, and is therefore insufficient for our needs. This new rounding algorithm is the technical meat of this paper, and is presented in Section 3.

Combining the above, we obtain Theorem 1.2, and the positive resolution of Conjecture 1.1.

1.2 Related Work

The first positive results for online edge coloring were under random order edge arrivals. In this setting, Aggarwal et al. [1] showed that a $(1 + o(1))$-competitive ratio is achievable in dense multigraphs with maximum degree $\Delta = \omega(n^2)$. Bahmani et al. [3] then showed that the greedy algorithm is sub-optimal for any graph of maximum degree $\Delta = \omega(\log n)$. Achieving the best of both these results, Bhattacharya et al. [5] recently obtained a $(1+o(1))$-competitive algorithm for graphs of maximum degree $\Delta = \omega(\log n)$. As stated above, the only prior algorithm which outperforms the greedy algorithm under *adversarial* arrivals is the algorithm of Cohen et al. [9] for bipartite graphs under one-sided vertex arrivals. In this work, we remove the assumption of bipartiteness and one-sided arrivals, and show how to outperform greedy in general graphs under arbitrary vertex arrivals.

Our work also ties into the long line of work on online matching, initiated by Karp, Vaizrani and Vazirani [26]. (See e.g., [2, 16, 18, 19, 21, 32] and references therein and [29] for a survey of earlier work.) Historically, most research on online matching considered bipartite

graphs with one-sided arrivals, due to applications in Internet advertising [17, 30]. A recent line of work considers such problems subject to interleaved vertex arrivals (motivated by more dynamic two-sided markets), as well as vertex arrivals in general graphs [2, 19, 21, 22, 37]. Our rounding algorithm for bipartite graphs with interleaved arrivals adds to the list of tools for tackling problems in this space.

Few of the works in the online (bipartite) matching literature rely on randomized rounding. At first blush, this seems surprising, given the integrality of the bipartite fractional matching polytope, and the multitude of competitive fractional algorithms for problems in this area [6, 17, 21, 22, 24, 37]. However, as pointed out in [12] and elaborated upon in [10], lossless rounding of a fractional matching x is impossible in online settings. In particular, outputting a matching $\mathcal{M}$ which matches each edge e in a bipartite graph with probability $\Pr[e \in \mathcal{M}] = x_e$ is impossible in online settings, though it is easy to do offline. A natural question, then, is what is the highest value of $\alpha < 1$ for which one can guarantee $\Pr[e \in \mathcal{M}] \geq \alpha \cdot x_e$ when rounding bipartite fractional matchings online. The batched OCRS of Ezra et al. [15] gives $\alpha = 1/2$, unfortunately too low for our purposes. In prior work [33], motivated by a variation of the online Bayesian selection problem, we improve this bound to $\alpha = 0.51$, though only for one-sided arrivals, which is insufficient for our needs here. In this work we generalize this result, achieving a slightly higher $\alpha = 0.527$, subject to *interleaved* vertex arrivals.

2 Preliminaries

The underlying (a priori unknown) input to our problem is an n-node graph $G = (V, E)$ of maximum degree Δ (with n and Δ both known). The vertices of G are revealed over time. For notational convenience, we associate the $n := |V|$ vertices with the numbers in $[n]$ by order of appearance, and denote by $u < v$ the fact that u arrives before v. When a vertex v arrives (at *time* v), all its edges (u, v) to its previously-arrived neighbors $u < v$ are revealed. After v arrives, and before arrival of vertex $v + 1$, an online edge coloring algorithm must decide, irrevocably, which color to assign to all edges (u, v) with $u < v$. The objective is to minimize the number of distinct colors used.

As outlined in the introduction, we will rely on the ability to edge color general graphs by recursively coloring random bipartite subgraphs, as first proposed by Karloff and Shmoys [25], in the context of NC algorithms. The extension and proof for online settings is essentially the same, and is provided, for completeness, in Section 5.

▶ **Lemma 2.1** (Implied by [25]). *Given an online edge coloring algorithm which is α-competitive w.h.p. on bipartite graphs of maximum degree $\Delta = \omega(\log n)$ under interleaved vertex arrivals, there exists an online edge coloring algorithm which is $(\alpha + o(1))$-competitive w.h.p. on* general *graphs of maximum degree $\Delta = \omega(\log n)$ under vertex arrivals.*

The following lemma, implied by the recent work of Cohen et al. [9], reduces α-competitive edge coloring to online matching algorithms which match each edge with probability $(1/\alpha)/\Delta$. The proof is is provided, for completeness, in Section 6.

▶ **Lemma 2.2** (Implied by [9]). *Let $\mathcal{A}$ be an online matching algorithm which on any (bipartite) graph of maximum degree $\Delta \leq \Delta'$ under vertex arrivals, matches each edge with probability at least $1/(\alpha\Delta')$. Then, there exists an online edge coloring algorithm $\mathcal{A}'$ which is $(\alpha + o(1))$-competitive w.h.p. for (bipartite) graphs of maximum degree $\Delta = \omega(\log n)$ under vertex arrivals.*

Motivated by Lemma 2.2, we show how to (approximately) round fractional matchings online. These are assignments of nonnegative $x_e \geq 0$ to edges $e \in E$, satisfying the fractional matching constraint, $\sum_{e \ni v} x_e \leq 1$ for all $v \in V$. This is a fractional relaxation of the

matching constraint, which stipulates that the degree of any vertex in a matching be at most one. Fittingly, we refer to $\sum_{w<v} x_{u,w}$ as the *fractional degree* of u before arrival of v (or at its arrival time, if $u = v$). We shall show how to round fractional matchings up to a multiplicative error of $\alpha < 2$. This rounding subroutine applied to the fractional matching assigning value $1/\Delta$ to each edge of the graph thus matches each edge with probability $1/(\alpha\Delta)$. Combined with lemmas 2.1 and 2.2, this yields our $(\alpha + o(1))\Delta$ coloring algorithm.

2.1 Negative Association

In our work we will need to bound positive correlations between variables. At the core of these proofs will be a use of *negatively associated* random variables. This section introduces this notion of negative dependence and its properties which we use.

▶ **Definition 2.3** ([23, 27]). *Random variables $X_1, \ldots, X_n$ are* negatively associated (NA) *if every two monotone nondecreasing functions f and g defined on disjoint subsets of the variables in $\vec{X}$ are negatively correlated. That is,*

$$\mathbb{E}[f \cdot g] \leq \mathbb{E}[f] \cdot \mathbb{E}[g]. \tag{1}$$

The following simple example of NA variables will prove useful for us.

▶ **Proposition 2.4** (0-1 Principle [14]). *Let $X_1, \ldots, X_n \in \{0, 1\}$ be binary random variables satisfying $\sum_i X_i \leq 1$ always. Then, the variables $X_1, \ldots, X_n$ are NA.*

Negative association is closed under several operations, allowing to construct more elaborate NA distributions from simpler NA distributions as above (see [14, 23, 27]).

▶ **Proposition 2.5** (Independent Union). *Let $X_1, \ldots, X_n$ be NA and $Y_1, \ldots, Y_m$ be NA, with $\{X_i\}_i$ independent of $\{Y_j\}_j$. Then, the variables $X_1, \ldots, X_n, Y_1, \ldots, Y_m$ are all NA.*

▶ **Proposition 2.6** (Function Composition). *Let $X_1, \ldots, X_n$ be NA variables, and let $f_1, \ldots, f_k$ be monotone nondecreasing functions defined on disjoint subsets of the variables in $\vec{X}$. Then the variables $f_1(\vec{X}), \ldots, f_k(\vec{X})$ are NA.*

An immediate corollary of negative association, obtained by considering the functions $f(\vec{X}) = X_i$ and $g(\vec{X}) = X_j$ for $i \neq j$, is pairwise negative correlation.

▶ **Proposition 2.7** (NA implies Negative Correlation). *Let $X_1, \ldots, X_n$ be NA variables. Then, for all $i \neq j$, we have that $\mathrm{Cov}(X_i, X_j) \leq 0$.*

2.2 Probability Basics

Here we include, for completeness, a number of basic probabilistic results used in this paper.

▶ **Proposition 2.8** (Chernoff Bound). *Let $X = \sum_i X_i$ be the sum of independent Bernoulli random variables $X_i \sim$ Bernoulli(p_i), with expectation $\mu := \mathbb{E}[X] = \sum_i p_i$. Then, for any $\epsilon \in (0, 1)$, and $\kappa \geq \mu$,*

$$\Pr[X \geq \kappa \cdot (1 + \epsilon)] \leq \exp\left(\frac{-\kappa \cdot \epsilon^2}{3}\right).$$

$$\Pr[X \leq \mu \cdot (1 - \epsilon)] \leq \exp\left(\frac{-\mu \cdot \epsilon^2}{2}\right).$$

▶ **Proposition 2.9** (Coupling). *Let $X_1, \ldots, X_m$ be binary random variables such that for all i and $\vec{x} \in \{0,1\}^{i-1}$,*

$$\Pr\left[X_i = 1 \,\middle|\, \bigwedge_{\ell \in [i-1]} (X_\ell = x_\ell)\right] \geq p_i.$$

If $\{Y_i \sim Bernoulli(p_i)\}_i$ are independent random variables, then for any $k \in \mathbb{R}$,

$$\Pr\left[\sum_i X_i \leq k\right] \leq \Pr\left[\sum_i Y_i \leq k\right].$$

▶ **Proposition 2.10.** *Let A and B be Bernoulli random variables. Then*

$$\mathrm{Cov}(A, B) = \mathrm{Cov}(1 - A, 1 - B).$$

3 Rounding Bipartite Fractional Matchings Online

In this section we present an online algorithm which (approximately) rounds a bipartite fractional matching under interleaved vertex arrivals. In what follows, we let $c \geq 0.027$ be the largest value below 0.03 satisfying

$$(1/2 - c)(1 - 4c)(1/2 - c - 6c/(1/2 - c)) - 2c \geq 0. \tag{2}$$

We note that this choice of $c \leq 0.03$ also satisfies the following.[1]

$$\min\{1/2 - c,\ 1 - 4c,\ 1 - 6c/(1/2 - c)^2\} \geq 0. \tag{3}$$

We show the following.

▶ **Theorem 3.1.** *There exists an online algorithm which, given an (unknown) bipartite graph G under interleaved vertex arrivals, together with a fractional matching x in G, outputs a random matching $\mathcal{M}$ matching each edge $e \in E$ with probability*

$$\Pr[e \in \mathcal{M}] = (1/2 + c) \cdot x_e \geq 0.527 \cdot x_e. \tag{4}$$

We now turn to describing the algorithm claimed by the above theorem.

3.1 Intuition and Algorithm

Before presenting our algorithm, we describe the approach used to obtain Theorem 3.1 under one-sided arrivals [33], and then discuss the new ideas needed to extend this result to interleaved arrivals.

Naturally, an edge (u, v) with $u < v$ (i.e., v arriving later than u) can only be matched if u is not already matched before the arrival of v. We denote by $F_{u,v}$ the event that u is free (i.e., is not matched in $\mathcal{M}$) prior to the arrival of v. The guarantee of Theorem 3.1 implies the following closed form for the probability of this event.

$$\Pr[F_{u,v}] = g(u, v) := 1 - \sum_{w<v} (1/2 + c) \cdot x_{u,w}. \tag{5}$$

[1] We encourage the reader to think of $c \to 0$, and note that inequalities (2) and (3) hold for sufficiently small constant $c > 0$. Our choice of $c \approx 0.027$ is simply the largest satisfying all these constraints.

To achieve marginal probabilities of $\Pr[(u,v) \in \mathcal{M}] = (1/2+c) \cdot x_{u,v}$, our first step is to have every arriving vertex v pick a random neighbor $u < v$ with probability $x_{u,v}$, and then, if u is free, we match (u,v) with probability $q_{u,v} := \min(1, (1/2+c)/g(u,v))$. For neighbors u of low fractional degree upon arrival of v, i.e., $\sum_{w<v} x_{u,w} \leq \frac{1/2-c}{1/2+c}$, this last probability is precisely $q_{u,v} = (1/2+c)/\Pr[F_{u,v}]$. Consequently, we match each such edge (u,v) with probability $\Pr[(u,v) \in \mathcal{M}] = x_{u,v} \cdot \Pr[F_{u,v}] \cdot (1/2+c)/\Pr[F_{u,v}] = (1/2+c) \cdot x_{u,v}$, as desired. For edges (u,v) for which u has *high* fractional degree, on the other hand, this only gives us $\Pr[(u,v) \in \mathcal{M}] \geq (1/2-c) \cdot x_{u,v}$, and this can be tight.

To increase the probability of an edge (u,v) to be matched to the desired $(1/2+c) \cdot x_{u,v}$, we repeat this process a second time, making a second pick, if v is not matched after its first pick. Here, we must argue that the variables $\{F_{u,v} \mid u < v\}$ do not have strong positive correlation. Indeed, if, as an extreme case, we had $F_{u,v} = F_{w,v}$ always for all $u, w < v$, and v had only high-degree neighbors (for which $q_{u,v} = 1$), then if v is not matched to its first pick, then all its neighbors must be matched, and v is therefore never matched as a second pick. This implies that a second pick does not increase $\Pr[(u,v) \in \mathcal{M}]$ in this case. As shown in [33], under one-sided arrivals, this problematic scenario does not occur, since the matched status of neighbors of v is rather weak. For interleaved arrivals, however, the underlying argument does not carry through, as we now explain.

3.1.1 Extension to Interleaved Arrivals

The key difference between one-sided and interleaved arrivals is that now we require small positive correlation between the matched statuses of every two nodes on the same side of the bipartition, rather than just nodes on the "offline side". For one-sided arrivals, the weak positive correlation between offline vertices was due to two factors. (1) low-degree offline vertices are matched only due to semi-adaptive matching choices, where precisely one neighbor of an arriving online vertex is picked, and at most one is matched. (That is, they are only matched as a first pick.) Therefore, by the 0-1 Principle (Proposition 2.4) and closure properties of NA distributions (propositions 2.5 and 2.6), the indicators for a vertex to be matched when it has low fractional degree are NA, and hence are negatively correlated. (2) On the other hand, the probability of a node to be matched when it has high degree is low, since each edge is matched with probability $(1/2+c) \cdot x_{u,v}$, and the residual fractional degree when v has high degree is $1 - \frac{1/2-c}{1/2+c} = \frac{2c}{1/2+c} \leq 4c$. Putting (1) and (2) together, we find that the matched statuses of any two offline vertices have small correlation.

Unfortunately, under interleaved arrivals, the above is no longer true. In particular, if a vertex v has low fractional degree upon arrival, it may still be matched as a second pick upon arrival (due to its high-degree neighbors). Consequently, the indicators for vertices on the same side of the bipartition being matched when they have low fractional degree are no longer negatively associated, thus undoing the entire argument used to bound $\mathrm{Cov}(F_{u,v}, F_{w,v})$ for vertices $u, w < v$ on the same side of the bipartition.

To overcome this problem, we have each arriving vertex v with low fractional degree upon arrival only pick once, and rely on its low fractional degree to pick each neighbor with higher probability. In particular, when such a vertex v arrives, we pick at most one neighbor with probability $x_{u,v} \cdot \frac{1/2+c}{1/2-c}$. (Since v has low fractional degree on arrival, $\sum_{u<v} x_{u,v} \leq \frac{1/2-c}{1/2+c}$, this is well-defined.) Then, if this picked vertex u is free, we match (u,v) with probability $\frac{1/2-c}{\Pr[F_{u,v}]} = \frac{1/2-c}{g(u,v)} (\leq 1)$, resulting in the edge (u,v) being matched with probability $x_{u,v} \cdot (1/2+c)$. Crucially for our analysis, this now allows us to show that the indicators for vertices (in the same side of the graph) to be matched when they have low

fractional degree is again negatively associated. This then results in the matched status of vertices again being decomposable into two variables, with the first being negatively correlated, and the second having low probability, from which we obtain that vertices on the same side of the bipartition have low correlation.[2]

This discussion gives rise to Algorithm 1, which we prove in this section provides the guarantees of Theorem 3.1.

Algorithm 1 Online rounding scheme.

1: **Init:** $\mathcal{M} \leftarrow \emptyset$
2: **for** all vertices v, on arrival **do**
3: read $\{x_{u,v} \mid u < v\}$
4: **if** $\sum_{u<v} x_{u,v} \leq \frac{1/2-c}{1/2+c}$ **then**
5: pick at most one $u < v$ with probability $x_{u,v} \cdot \frac{1/2+c}{1/2-c}$
6: **if** $u \neq$ nil and u is unmatched in $\mathcal{M}$ **then**
7: **with probability** $\frac{1/2-c}{g(u,v)}$ **do**
8: $\mathcal{M} \leftarrow \mathcal{M} \cup \{(u,v)\}$
9: **else**
10: pick at most one $u < v$ with probability $x_{u,v}$
11: **if** $u \neq$ nil and u is unmatched in $\mathcal{M}$ **then**
12: **with probability** $\min\left(1, \frac{1/2+c}{g(u,v)}\right)$ **do**
13: $\mathcal{M} \leftarrow \mathcal{M} \cup \{(u,v)\}$
14: **if** v is still unmatched in $\mathcal{M}$ **then**
15: pick at most one $u < v$ with probability $x_{u,v}$
16: **if** $u \neq$ nil and u is unmatched in $\mathcal{M}$ **then**
17: **with probability** $p_{u,v}$ guaranteeing $\Pr[(u,v) \in \mathcal{M}] = (1/2+c) \cdot x_{u,v}$ **do**
18: $\mathcal{M} \leftarrow \mathcal{M} \cup \{(u,v)\}$
19: **Output** $\mathcal{M}$

3.2 High-Level Analysis

For our analysis and proof of Theorem 3.1, we will assume, by way of an inductive proof, that Equation (4) holds for all edges (u,w) with $u,w < v$ and therefore that for each $u < v$ we have $\Pr[F_{u,v}] = g(u,v)$, as stated in Equation (5).

Given the inductive hypothesis, it is easy to verify that Algorithm 1 guarantees marginal probabilities of each edge to be matched to be precisely $(1/2+c) \cdot x_e$. Indeed, for an arriving vertex v with low fractional degree, $\sum_{u<v} x_{u,v} \leq \frac{1/2-c}{1/2+c}$ (lines 4-8), since by the inductive hypothesis u is free at time v with probability $\Pr[F_{u,v}] = g(u,v)$, we have that

$$\Pr[(u,v) \in \mathcal{M}] = x_{u,v} \cdot \frac{1/2+c}{1/2-c} \cdot g(u,v) \cdot \frac{1/2-c}{g(u,v)} = (1/2+c) \cdot x_{u,v}.$$

[2] We note that Gamlath et al. [19] followed a superficially similar rounding approach, using two choices. As they only required bounds on the (unweighted) matching's size, their analysis relied on showing that *globally* positive correlation is low. As we desire high matching probability on an edge-by-edge (or at least vertex-by-vertex) basis, we must follow a more delicate approach.

In the alternative case of lines 9-18, we trivially have that each edge (u,v) with $u < v$ is matched with probability precisely $\Pr[(u,v) \in \mathcal{M}] = (1/2+c) \cdot x_e$, due to lines 17-18. The crux of the analysis, then, is in proving that this algorithm is well-defined, and in particular that there exists some probabilities $p_{u,v}$ as stated in Line 17.

We note that all probabilistic lines in the algorithm except for Line 17 are trivially well-defined. First, if v has low fractional degree before time v, i.e., $\sum_{u<v} x_{u,v} \leq \frac{1/2-c}{1/2+c}$, then the probability of any neighbor to be picked in Line 5 is at most $\sum_{u<v} x_{u,v} \cdot \frac{1/2+c}{1/2-c} \leq 1$, and so this line is well-defined. Next, by the fractional matching constraint, we have that $\sum_{u<v} x_{u,v} \leq 1$, and consequently lines 10 and 15 are well-defined. Finally, by the fractional matching constraint, we have that $\sum_{w<v}(1/2+c) \cdot x_{u,w} \leq 1/2+c$, and therefore

$$\Pr[F_{u,v}] = g(u,v) \geq 1/2 - c. \tag{6}$$

Consequently, the term $\frac{1/2-c}{g(u,v)}$ in Line 7 is indeed a probability, by our choice of $c = 0.027 \leq 1/2$. We now turn to proving that probabilities $p_{u,v}$ as stated in Line 17 do indeed exist.

First, to show that $p_{u,v} \geq 0$, we must show that the probability of edge (u,v) to be matched as a first pick in Line 13 does not on its own exceed $(1/2+c) \cdot x_{u,v}$.

▶ **Observation 3.2.** *The probability of an edge (u,v) to be matched in Line 13 is at most*

$$\Pr[(u,v) \text{ added to } \mathcal{M} \text{ in Line 13}] \leq (1/2+c) \cdot x_{u,v}.$$

Proof. By the inductive hypothesis, we have that $\Pr[F_{u,v}] = g(u,v)$. Consequently,

$$\Pr[(u,v) \text{ added to } \mathcal{M} \text{ in Line 13}] = x_{u,v} \cdot \min\left(1, \frac{1/2+c}{g(u,v)}\right) \cdot g(u,v) \leq (1/2+c) \cdot x_{u,v}. \quad \blacktriangleleft$$

▶ **Corollary 3.3.** *The parameter $p_{u,v}$ in Line 17 satisfies $p_{u,v} \geq 0$.*

The core of the analysis will then be in proving that $p_{u,v} \leq 1$. For this, we will need to argue that a second pick in lines 14-18 is likely to result in (u,v) being matched, provided we set $p_{u,v} \leq 1$ high enough. We prove as much in the next section.

3.3 Core of the Analysis

In this section we prove that the second pick is likely to result in a match. To this end, we prove that the matched statuses of neighbors of an arriving vertex v have low positive correlation (if any). More formally, if $G = (V_1, V_2, E)$ is our bipartite graph, we will prove the following.

▶ **Lemma 3.4.** *For any $i = 1, 2$, vertex v and vertices $u, w < v$ with $u, w \in V_i$,*

$$\mathrm{Cov}(F_{u,v}, F_{w,v}) \leq 6c.$$

Since the covariance of two binary variables A and B is equal to that of their complements, $\mathrm{Cov}(A,B) = \mathrm{Cov}(1-A, 1-B)$, we will concern ourselves with bounding $\mathrm{Cov}(M_{u,v}, M_{w,v})$, where $M_{u,v} := 1 - F_{u,v}$ is an indicator for u being matched in $\mathcal{M}$ before v arrives.

For this proof, we write $M_{u,v}$ as the sum of two Bernoulli variables, $M_{u,v} = M^L_{u,v} + M^H_{u,v}$. The indicators $M^L_{u,v}$ and $M^H_{u,v}$ correspond to u being matched to some neighbor w at a time z when u had low or high fractional degree, respectively. That is,

$$M^L_{u,v} := \mathbb{I}\left[(u,w) \in \mathcal{M} \text{ for some } w < v \text{ with } \sum_{z<\min\{u,w\}} x_{u,z} \leq \frac{1/2-c}{1/2+c}\right],$$

with $M^H_{u,v} = M_{u,v} - M^L_{u,v}$ defined analogously.

In what follows, we will show that for any vertex v and index $i = 1, 2$, the variables $\{M^L_{u,v} \mid u \in V_i\}$ are negatively correlated, while the variables $\{M^H_{u,v} \mid u \in V_i\}$ have low probability, which implies that they have low positive correlation with any other binary variable. These bounds will allow us to bound the correlation of the sums $M_{u,v} = M^L_{u,v} + M^H_{u,v}$.

We start by proving the negative correlation between $M^L_{u,v}$ variables, and indeed proving negative association of these variables.

▶ **Lemma 3.5.** *For any $i = 1, 2$ and vertex v, the variables $\{M^L_{u,v} \mid u < v,\ u \in V_i\}$ are NA.*

By Proposition 2.7, this implies that the above variables are negatively correlated.

▶ **Corollary 3.6.** *For any $i = 1, 2$, vertex v and earlier vertices $u, w < v$ with $u, w \in V_i$,*

$$\mathrm{Cov}(M^L_{u,v}, M^L_{w,v}) \leq 0.$$

Proof of Lemma 3.5. Recall that $M^L_{u,v}$ is an indicator for u being matched before arrival of v before it has high fractional degree. By definition of Algorithm 1, this implies that a matching event accounted for by $M^L_{u,v}$ can only occur in lines 8 or 13. Such matches occur due to u picking a neighbor or being picked as a neighbor in line 5 or 10, and the probabilistic test in line 7 or 12 (respectively), passing, if the picked vertex was previously unmatched in $\mathcal{M}$. We imagine we perform the probabilistic tests in lines 7 and 12 *before* testing whether the picked vertex was unmatched in $\mathcal{M}$.

For vertices $w < z$, let $A_{w,z}$ be an indicator for z picking w in line 5 or 10, and the probabilistic test in line 7 or 12 (respectively) passing. Then, by the 0-1 Principle (Proposition 2.4), we have that for any vertex z, the variables $\{A_{w,z} \mid w < z\}$ are NA. Moreover, the families of variables $\{A_{w,z} \mid w < z\}$ for distinct z are NA. Therefore, by closure of NA under independent union (Proposition 2.5), the variables $\{A_{w,z} \mid z,\ w < z\}$ are NA. For notational simplicity, letting $A_{z,w} := A_{w,z}$ for $z > w$ (recall that we only defined $A_{w,z}$ for $w < z$), we find that if z' is the smaller of $v - 1$ and the first time z that u has high fractional degree, the variables $M^L_{u,v}$ are precisely equal to

$$M^L_{u,v} := \bigvee_{w \leq z'} A_{w,u}.$$

Indeed, this is due to u being matched while it has low fractional degree upon the first time that it is picked by a neighbor (or it picks a neighbor) in line 5 or 10, and the corresponding probabilistic test in line 7 or 12 passes. Therefore, by closure of NA under monotone function composition (Proposition 2.6), the variables $\{M^L_{u,v} \mid u \in V_i\}$, which are monotone nondecreasing functions of disjoint subsets of the variables $A_{w,u}$ by bipartiteness, are NA.[3] ◀

We now turn to upper bounding the probability of the event $M^H_{u,v}$.

[3] This is the only place in our analysis where we use bipartiteness.

▶ **Lemma 3.7.** *For any edge (u,v) with $u<v$, we have that* $\Pr[M^H_{u,v}] \le 2c$.

Proof. Recall that by the inductive hypothesis, $\Pr[(u,w)\in\mathcal{M}] = (1/2+c)\cdot x_{u,w}$. On the other hand, by the fractional matching constraint, we have that $\sum_{w<v} x_{u,v} \le 1$, and therefore $\Pr[M_{u,v}] \le 1/2+c$. On the other hand, if we denote by z_u the first time u has high fractional degree, then either $z_u \ge v$, in which case $\Pr[M^H_{u,v}]=0$, or

$$\Pr[M^L_{u,v}] \ge \sum_{w<z_u} x_{u,w}\cdot(1/2+c) \ge \frac{1/2-c}{1/2+c}\cdot(1/2+c) = 1/2-c,$$

in which case we have

$$\Pr[M^H_{u,v}] = \Pr[M_{u,v}] - \Pr[M^L_{u,v}] \le 2c. \qquad \blacktriangleleft$$

We are now ready to prove Lemma 3.4, whereby vertices u,w on the same side of the bipartition have weakly correlated matched statuses, namely $\mathrm{Cov}(F_{u,v},F_{w,v}) \le 6c$.

Proof. By definition of covariance, the binary variables $F_{u,v}$ and $F_{w,v}$ satisfy $\mathrm{Cov}(F_{u,v},F_{w,v}) = \mathrm{Cov}(1-F_{u,v},1-F_{w,v}) = \mathrm{Cov}(M_{u,v},M_{w,v})$ (see Proposition 2.10). We therefore turn to upper bounding the covariance of the variables $M_{u,v}$ and $M_{w,v}$.

By the additive law of covariance, the covariance of the variables $M_{u,v} = M^L_{u,v}+M^H_{u,v}$ and $M_{w,v} = M^L_{w,v}+M^H_{w,v}$, denoted by $(\star)=\mathrm{Cov}(M_{u,v},M_{w,v})$, satisfies

$$\begin{aligned}
(\star) &= \mathrm{Cov}(M^L_{u,v}+M^H_{u,v}\,,\,M^L_{w,v}+M^H_{w,v})\\
&= \mathrm{Cov}(M^L_{u,v},M^L_{w,v}) + \mathrm{Cov}(M^L_{u,v},M^H_{w,v}) + \mathrm{Cov}(M^H_{u,v},M^L_{w,v}) + \mathrm{Cov}(M^H_{u,v},M^H_{w,v})\\
&\le 0 + \Pr[M^L_{u,v},M^H_{w,v}] + \Pr[M^H_{u,v},M^L_{w,v}] + \Pr[M^H_{u,v},M^H_{w,v}]\\
&\le 0 + \Pr[M^H_{w,v}] + \Pr[M^H_{u,v}] + \Pr[M^H_{u,v}]\\
&\le 6c.
\end{aligned}$$

Here, the first inequality follows from Corollary 3.6, the second inequality follows from the trivial bound on covariance of Bernoulli variables A and B given by $\mathrm{Cov}(A,B) = \Pr[A,B] - \Pr[A]\cdot\Pr[B] \le \Pr[A,B] \le \Pr[A]$, and the final inequality follows from Lemma 3.7. ◀

Lemma 3.4 now allows us to argue that if u has high degree upon arrival of v, then $F_{u,v}$ is nearly independent of the event R_v, whereby v is rejected (not matched) after its first pick of u_1 (possibly $u_1=\text{nil}$). In particular, we have the following.

▶ **Lemma 3.8.** *Let $u<v$ be a vertex of high fractional degree, $\sum_{w<v} x_{u,w}$, upon arrival of v. Then, for all $w\neq u$ (including possibly $w=$ nil), we have*

$$\Pr[F_{u,v},R_v,u_1=w] \ge \Pr[F_{u,v}]\cdot\Pr[R_v,u_1=w]\cdot\left(1-\frac{6c}{(1/2-c)^2}\right).$$

Proof. For $w=\text{nil}$ the claim follows from the event $u_1=\text{nil}$ implying R_v, and being independent of $F_{u,v}$.

$$\Pr[F_{u,v},R_v,u_1=\text{nil}] = \Pr[F_{u,v}]\cdot\Pr[R_v,u_1=\text{nil}].$$

Next, let $w<v$ be some neighbor of v. If we denote by $q_{w,v} := \min\left(1,\frac{1/2+c}{g(w,v)}\right)$ the probability that w does not reject v if it is picked first and is free, then the probability that $u_1=w$ and v gets rejected in its first pick is

$$\Pr[R_v,u_1=w] = x_{w,v}\cdot(1-q_{w,v}\cdot\Pr[F_{w,v}]) \ge x_{w,v}\cdot(1/2-c), \tag{7}$$

where the inequality follows from $\Pr[F_{w,v}] = g(w,v)$ by Equation (6), which implies that $q_{w,v} \cdot \Pr[F_{w,v}] \leq 1/2 + c$. Similarly, the probability u is free, $u_1 = w$ and v gets rejected in its first pick is

$$\begin{aligned}
\Pr[F_{u,v}, R_v, u_1 = w] &= x_{w,v} \cdot (\Pr[F_{u,v}] - q_{w,v} \cdot \Pr[F_{w,v}, F_{u,v}]) \\
&\geq x_{w,v} \cdot (\Pr[F_{u,v}] - q_{w,v} \cdot (\Pr[F_{w,v}] \cdot \Pr[F_{u,v}] + 6c)), \\
&\geq x_{w,v} \cdot (\Pr[F_{u,v}] - q_{w,v} \cdot (\Pr[F_{w,v}] \cdot \Pr[F_{u,v}]) - 6c) \\
&\geq x_{w,v} \cdot \Pr[F_{u,v}] \cdot \left(1 - q_{wv} \cdot \Pr[F_{w,v}] - \frac{6c}{1/2 - c}\right) \\
&\geq \Pr[F_{u,v}] \cdot \Pr[R_v, u_1 = w] \cdot \left(1 - \frac{6c}{(1/2-c)^2}\right),
\end{aligned}$$

where the first inequality follows from Lemma 3.4, the second inequality follows from the trivial bound $q_{wv} \leq 1$, the third inequality follows from $\Pr[F_{u,v}] = g(u,v) \geq 1/2 - c$ by Equation (5), and the final inequality follows from Equation (7). ◀

In what follows we denote by $x_{\text{nil},v} := 1 - \sum_{w<v} x_{w,v}$ the probability with which $u_1 = \text{nil}$. From Lemma 3.8 and Equation (7), as well as $\Pr[R_v, u_1 = \text{nil}] = \Pr[u_1 = \text{nil}] = x_{\text{nil},v}$, we obtain the following lower bound on $\Pr[F_{u,v}, R_v, u_1 = w]$ in terms of $x_{w,v}$.

▶ **Corollary 3.9.** *For any vertex v and w (possibly $w = \text{nil}$), we have that*

$$\Pr[F_{u,v}, R_v, u_1 = w] \geq \Pr[F_{u,v}] \cdot x_{w,v} \cdot \left(1/2 - c - \frac{6c}{1/2 - c}\right).$$

Finally, we are ready to prove that $p_{u,v}$ is a probability, and in particular $p_{u,v} \leq 1$.

▶ **Lemma 3.10.** *The parameter $p_{u,v}$ in Line 17 satisfies $p_{u,v} \in [0,1]$.*

Proof. Non-negativity of $p_{u,v}$ was proven in Corollary 3.3. We turn to proving that $p_{u,v} \leq 1$ suffices to guarantee $\Pr[(u,v) \in \mathcal{M}] \geq (1/2 + c) \cdot x_{u,v}$, from which we obtain that there exists some $p_{u,v} \in [0,1]$ which results in $\Pr[(u,v) \in \mathcal{M}] = (1/2 + c) \cdot x_{u,v}$.

By Equation (6) we have that $\Pr[F_{u,v}] = g(u,v) \geq 1/2 - c$, and therefore

$$\Pr[(u,v) \in \mathcal{M} \text{ in Line 13}] = x_{u,v} \cdot \min\left(1, \frac{1/2 + c}{g(u,v)}\right) \cdot g(u,v) \geq (1/2 - c) \cdot x_{u,v}. \tag{8}$$

We therefore wish to prove that the probability of (u,v) being matched in Line 18 is at least $2c \cdot x_{u,v}$, for some choice of $p_{u,v} \leq 1$. And indeed,

$$\begin{aligned}
\Pr[(u,v) \text{ added to } \mathcal{M} \text{ in Line 18}] &= x_{u,v} \cdot \sum_{w \neq u} \Pr[F_{u,v}, R_v, u_1 = w] \cdot p_{u,v} \\
&\geq x_{u,v} \cdot \Pr[F_{u,v}] \cdot \sum_{w \neq u} x_{w,v} \cdot \left(1/2 - c - \frac{6c}{1/2 - c}\right) \cdot p_{u,v} \\
&\geq x_{u,v} \cdot (1/2 - c) \cdot (1 - 4c) \cdot \left(1/2 - c - \frac{6c}{1/2 - c}\right) \cdot p_{u,v} \\
&\geq 2c \cdot x_{u,v},
\end{aligned}$$

where the first inequality follows from Corollary 3.9 and Equation (3). The second inequality holds due to Equation (5) implying $\Pr[F_{u,v}] \geq 1/2 - c$ and due to vertex u having high degree at time v, and therefore by the fractional matching constraint $x_{u,v} \leq 1 - \frac{1/2-c}{1/2+c} = \frac{2c}{1/2+c} \leq 4c$, and hence $\sum_{w \neq u} x_{w,v} \geq 1 - 4c \geq 0$ (again using Equation (3)). The final inequality holds for $p_{u,v} = 1$ and for our choice of c, by Equation (2).

Consequently, combining the above with Equation (8), we find that setting $p_{u,v} = 1$ results in (u, v) being matched in either Line 13 or Line 18 with probability at least

$$\Pr[(u, v) \in \mathcal{M}] \geq (1/2 + c) \cdot x_{u,v}. \tag{9}$$

As the probability of (u, v) being added to $\mathcal{M}$ in Line 18 is monotone increasing in $p_{u,v}$, we conclude that there exists some $p_{u,v} \in [0, 1]$ for which Equation (9) holds with equality. ◀

Conclusion of Algorithm 1's analysis. To conclude, Algorithm 1 is well-defined, and this algorithm outputs a random matching $\mathcal{M}$ which matches each edge e with probability precisely $\Pr[e \in \mathcal{M}] = (1/2 + c) \cdot x_e$. Theorem 3.1 follows.

▶ Remark 3.11. **Computational Aspects:** As described, the only way we are aware of to implement Algorithm 1 exactly (and in particular, computing all $p_{u,v}$ exactly) is using an exponential-time algorithm maintaining the joint distributions as they evolve. However, a simple modification of the algorithm, resulting in a polynomial-time algorithm with a $(1 + o(1))$ additional multiplicative loss in each edge's matching probability, can be readily obtained by approximately estimating the above $p_{u,v}$ up to $(1 \pm o(1))$ multiplicative errors, by standard monte carlo methods. As this results in rather cumbersome descriptions and subsequent calculations, and since running time is not our focus, we do not expand on this.

4 Putting it all Together

In this section we prove our main result, Theorem 1.2, restated below for ease of reference.

▶ **Theorem 1.2.** *There exists an online edge coloring algorithm which is* $(1.897 + o(1))$*-competitive w.h.p. on general* n*-node graphs with maximum degree* $\Delta = \omega(\log n)$ *under vertex arrivals.*

Proof. For a graph of maximum degree at most Δ, assigning x-value $1/\Delta$ to each edge yields a fractional matching. Applying Algorithm 1 to this fractional matching in a bipartite graph under vertex arrivals results in each edge being matched with probability $0.527/\Delta$, by Theorem 3.1. Therefore, by Lemma 2.2, there exists an online edge coloring algorithm whose competitive ratio is $(1/0.527 + o(1)) \approx 1.897 + o(1)$ w.h.p. on bipartite graphs of maximum degree $\Delta = \omega(\log n)$ under (interleaved) vertex arrivals. Finally, Lemma 2.1 together with union bound implies that the same competitive ratio (up to $o(1)$ terms) carries over to general graphs under vertex arrivals. ◀

▶ Remark 4.1. Our analysis extends to prove the slightly tighter result, whereby there exist constants $c_1, c_2 > 0$ and a $(2 - c_1)$-competitive online algorithm for n-node graphs of maximum degree at least $c_2 \cdot \log n$ under vertex arrivals. (See Remark 1.3.) For brevity's sake, we omit the details.

5 The Karloff-Shmoys Approach: Online

Here we substantiate our earlier assertion that α-competitive online edge coloring on high-degree graphs is equivalent (up to $o(1)$ terms) to the same task on high-degree *bipartite* graphs. That is, we outline the proof of Lemma 2.1, restated below for ease of reference.

▶ **Lemma 2.1** (Implied by [25]). *Given an online edge coloring algorithm which is* α*-competitive w.h.p. on bipartite graphs of maximum degree* $\Delta = \omega(\log n)$ *under interleaved vertex arrivals, there exists an online edge coloring algorithm which is* $(\alpha + o(1))$*-competitive w.h.p. on* general *graphs of maximum degree* $\Delta = \omega(\log n)$ *under vertex arrivals.*

Proof. The general graph edge coloring algorithm relies on the following subroutine for sampling balanced random subgraphs in subgraphs of maximum degree $\Delta' \geq 18 \cdot \sqrt{\Delta \log n}$. (Note that $\Delta \geq 18\sqrt{\Delta \log n}$, by the hypothesis, whereby $\Delta = \omega(\log n)$.) Assign each vertex to a set $V_i \subseteq V$ with $i = 1, 2$ chosen uniformly at random. For any vertex $v \in V$, let $d(v)$ denotes the degree of v in G, and D_v denotes the (random) degree of v in the random bipartite subgraph $H = H(V_1, V_2, E \cap (V_1 \times V_2))$. Then, we have that $\mathbb{E}[D_v] = d(v)/2 \leq \Delta'/2$. By Chernoff's Bound (Proposition 2.8), since D_v is the sum of independent Bernoulli(1/2) variables, we have that, for $\epsilon = \sqrt[4]{\log n/\Delta} = o(1)$,

$$\Pr[D_v \geq (\Delta'/2) \cdot (1+\epsilon)] \leq \exp\left(\frac{-(\Delta'/2) \cdot \epsilon^2}{3}\right) \leq \frac{1}{n^3}, \tag{10}$$

using $\Delta' \geq 18 \cdot \sqrt{\Delta \cdot \log n}$, and consequently $\Delta \cdot \epsilon^2 \geq 18 \log n$. The same high-probability bound holds for $d(v) - D_v$, which is identically distributed to D_v.

To achieve an online edge coloring algorithm for G from the above, we apply the α-competitive edge coloring algorithm to the random bipartite H, and recursively apply the same approach to the random subgraph induced by the edges outside of H, namely $G \setminus H = G[E \setminus (V_1 \times V_2)]$, until H is guaranteed to have degree at most $18 \cdot \sqrt{\Delta \cdot \log n}$ w.h.p. We note that this approach can be applied online, by assigning to each vertex v on arrival a side of each of the recursive random bipartitions. Moreover, the colors of each recursive level number ℓ can be associated with a contiguous set of integers of cardinality $\alpha \cdot \Delta \cdot ((1+\epsilon)/2)^\ell$, which is the high probability upper bound on the number of colors used in this recursive call. Repeating the above recursively for $t := \log_{2/(1+\epsilon)}(18\sqrt{\Delta/\cdot \log n}) \leq \log n$ levels results in a random uncolored subgraph of maximum degree at most $18\sqrt{\Delta \cdot \log n} = o(\Delta)$ w.h.p., which we color greedily.

Taking union bound over the $O(n^2)$ bad events (some vertex degree D_v exceeding $\Delta' \cdot ((1+\epsilon)/2)$ in a random bipartite subgraph or its complement in a subgraph whose maximum degree is $\Delta' \geq 18\sqrt{\Delta \cdot \log n}$, or any of the bipartite edge coloring algorithms failing to be α competitive on the subgraph it is applied to), we have that w.h.p., the number of colors C used is, as desired, at most

$$\begin{aligned} C &\leq \alpha \cdot \Delta \cdot \frac{1+\epsilon}{2} + \alpha \cdot \Delta \cdot \left(\frac{1+\epsilon}{2}\right)^2 + \cdots + \alpha \cdot \Delta \cdot \left(\frac{1+\epsilon}{2}\right)^t + 36 \cdot \sqrt{\Delta \cdot \log n} \\ &\leq \alpha \cdot \sum_{i \geq 1} \Delta \cdot \left(\frac{1+\epsilon}{2}\right)^i + 36 \cdot \sqrt{\Delta \cdot \log n} \\ &= \alpha \cdot \Delta \cdot \frac{1+\epsilon}{1-\epsilon} + o(\Delta) \\ &= (\alpha + o(1)) \cdot \Delta. \end{aligned}$$

◀

▶ Remark 5.1. As stated in the introduction, we note that the above reduction from general to bipartite graphs results in bipartite graphs with *interleaved* vertex arrivals.

6 Edge Coloring from Random Matchings

In this section, we show how to reduce edge coloring in (bipartite) graphs under vertex arrivals to computing a random matching which matches each edge with probability $\Omega(1/\Delta)$.

▶ **Lemma 2.2** (Implied by [9]). *Let $\mathcal{A}$ be an online matching algorithm which on any (bipartite) graph of maximum degree $\Delta \leq \Delta'$ under vertex arrivals, matches each edge with probability at least $1/(\alpha\Delta')$. Then, there exists an online edge coloring algorithm $\mathcal{A}'$ which is $(\alpha + o(1))$-competitive w.h.p. for (bipartite) graphs of maximum degree $\Delta = \omega(\log n)$ under vertex arrivals.*

Proof. If $\alpha > 2$, then the claim follows trivially from the greedy algorithm's 2-competitiveness. We therefore assume $\alpha \leq 2$. We give a subroutine which decreases the uncolored degree of a subgraph of maximum degree $\Delta' \geq 48 \cdot \sqrt[4]{\Delta^3 \log n}$ at a rate of one per $\alpha + o(1)$ colors w.h.p. (Note that $\Delta \geq 48\sqrt[4]{\Delta^3 \log n}$, by the hypothesis, whereby $\Delta = \omega(\log n)$.)

Our subroutine is as follows. Let $L := 12\sqrt{\Delta \log n}$ and $\epsilon := \sqrt[4]{(\log n)/\Delta} (= o(1) \leq 1/2)$. We note that by our choice of L and ϵ and our lower bound on Δ', we have that

$$4L/\Delta' \leq 48\sqrt{\Delta \log n}/48\sqrt[4]{\Delta^3 \log n} = \sqrt[4]{(\log n)/\Delta} = \epsilon. \tag{11}$$

For $i = 1, \ldots, \lceil \alpha \cdot L \rceil$, we run Algorithm $\mathcal{A}$, which matches each edge with probability at least $(1/\alpha)/\Delta'$, and color all previously-uncolored matched edges in this run of $\mathcal{A}$ using a new (common) color. Fix a vertex v whose degree in the subgraph is at least $d(v) \geq \Delta' - \lceil \alpha \cdot L \rceil$ and let $X_1, \ldots, X_L$ be indicators of v having an edge colored during application $i = 1, \ldots, \lceil \alpha \cdot L \rceil$ of Algorithm $\mathcal{A}$. Since vertex v can have at most $\lceil \alpha \cdot L \rceil \leq 2 \cdot L$ edges colored during these L applications of Algorithm $\mathcal{A}$, we find that the number of uncolored edges of v at any point during this subroutine is at least $\Delta' - 2\lceil \alpha \cdot L \rceil \geq \Delta' - 4L$, independently of previous random choices. On the other hand, since each uncolored edge is matched (and hence colored) with probability at least $(1/\alpha)/\Delta'$, we have that for any history $\mathcal{H}$ of random choices in applications $1, 2, \ldots, i-1$ of $\mathcal{A}$, application i of $\mathcal{A}$ results in one of the (at least) $\Delta' - 4L$ uncolored edges of v being colored with probability at least

$$\Pr[X_i \mid \mathcal{H}] \geq (1/\alpha) \cdot (\Delta' - 4L)/\Delta' = (1/\alpha) \cdot (1 - 4L/\Delta') \geq (1/\alpha) \cdot (1 - \epsilon), \tag{12}$$

where the last inequality relied on Equation (11). Combining Equation (12) with standard coupling arguments (Proposition 2.9) together with a Chernoff Bound (Proposition 2.8), we find that the number of colored edges of v, denoted by $X := \sum_i X_i$ satisfies

$$\Pr[X \leq L \cdot (1-\epsilon)^2] \leq \exp\left(\frac{-L \cdot (1-\epsilon) \cdot \epsilon^2}{2}\right) \leq \exp\left(\frac{-L \cdot \epsilon^2}{4}\right) = \frac{1}{n^3},$$

where the second inequality follows from $\epsilon \leq 1/2$ and the equality follows from choice of L and ϵ. Union bounding over the n vertices, we obtain the following high probability bound on the maximum degree of the uncolored subgraph H after the $\lceil \alpha \cdot L \rceil$ applications of $\mathcal{A}$:

$$\Pr[\Delta(H) \geq \Delta' - L \cdot (1-\epsilon)^2] \leq \frac{1}{n^2}. \tag{13}$$

We now describe how to make use of this subroutine. For $r = 1, \ldots, \Delta/L$ *phases*, let $\Delta_i := \Delta - (i-1) \cdot L \cdot (1-\epsilon)^2$. If $\Delta_i < 48\sqrt[4]{\Delta^3 \log n}$, apply the greedy coloring. Otherwise, apply the above subroutine with $\Delta' = \Delta_i$. A simple inductive argument together with union bound, relying on Equation (13), shows that for $i = 1, 2, \ldots, \Delta/L (\leq n)$, the uncolored subgraph after the first $i-1$ phases has maximum degree at most $\Delta' \leq \Delta_i$ w.h.p., or alternatively it has maximum degree at most $\Delta' \leq 48 \cdot \sqrt[4]{\Delta^3 \log n} = o(\Delta)$. Moreover, each of these Δ/L phases requires at most $\lceil \alpha \cdot L \rceil \leq \alpha \cdot L + 1$ colors, by definition, and therefore these Δ/L phases require at most $\alpha \cdot \Delta + \Delta/L = (\alpha + o(1)) \cdot \Delta$ colors in total. Finally, after these phases we are guaranteed that the maximum degree of the uncolored subgraph is at most $\min\{48 \cdot \sqrt[4]{\Delta^3 \log n}, \Delta - (\Delta/L) \cdot L \cdot (1-\epsilon)^2\} = o(\Delta)$. Applying the greedy algorithm to this uncolored subgraph after the Δ/L phases thus requires a further $2 \cdot o(\Delta) = o(\Delta)$ colors. This results in a proper edge coloring using $(\alpha + o(1)) \cdot \Delta$ colors w.h.p.

Finally, we note that the above algorithm can be implemented online under vertex arrivals, since $\mathcal{A}$ works under vertex arrivals. In particular, when a vertex arrives, we perform the next steps of the different copies of Algorithm $\mathcal{A}$ (with the different settings of Δ_i) on the

uncolored subgraphs obtained from each phase, simulating the arrival of a vertex in each such uncolored subgraph. Combined with the above, this yields the desired result: an edge coloring algorithm which is $(\alpha + o(1))$-competitive on general n-node graphs of maximum degree $\Delta = \omega(\log n)$ under vertex arrivals. ◀

▶ Remark 6.1. Lemma 2.2 naturally extends to edge arrivals. Unfortunately, no algorithm matching each edge with probability $(1/\alpha)/\Delta$ subject to edge arrivals is currently known for any constant $\alpha < 2$.

▶ Remark 6.2. The approach of Lemma 2.2 only requires matching algorithms which match each edge with probability $(1/\alpha)/\Delta$ for *subgraphs* of the input graph. Consequently, improved matching algorithms, with smaller $\alpha \geq 1$, for any downward-closed family of graphs $\mathcal{F}$ imply a similar improved $(\alpha + o(1))$-competitive edge coloring algorithm for the same family.

7 Conclusion

In this work we resolve the longstanding conjecture of Bar-Noy, Motwani and Naor, namely Conjecture 1.1. That is, we show that, while for bounded-degree graphs the greedy algorithm's competitive ratio of 2 is optimal among online algorithms, for high-degree graphs this is not the case.

Some natural questions remain. What is the best achievable competitive ratio? Is a ratio of $1 + o(1)$ possible, as for one-sided arrivals in bipartite graphs and random-order edge arrivals [5, 9]? Can the same be achieved under adversarial *edge* arrivals? Bar-Noy et al. [4] suggested a candidate algorithm for this latter model, but its analysis seems challenging. Finally, does the online rounding Algorithm 1 have more applications beyond edge coloring?

References

1 Gagan Aggarwal, Rajeev Motwani, Devavrat Shah, and An Zhu. Switch scheduling via randomized edge coloring. In *Proceedings of the 44th Symposium on Foundations of Computer Science (FOCS)*, pages 502–512, 2003.

2 Itai Ashlagi, Maximilien Burq, Chinmoy Dutta, Patrick Jaillet, Amin Saberi, and Chris Sholley. Edge weighted online windowed matching. In *Proceedings of the 20th ACM Conference on Economics and Computation (EC)*, pages 729–742, 2019.

3 Bahman Bahmani, Aranyak Mehta, and Rajeev Motwani. Online graph edge-coloring in the random-order arrival model. *Theory of Computing*, 8(1):567–595, 2012.

4 Amotz Bar-Noy, Rajeev Motwani, and Joseph Naor. The greedy algorithm is optimal for on-line edge coloring. *Information Processing Letters (IPL)*, 44(5):251–253, 1992.

5 Sayan Bhattacharya, Fabrizio Grandoni, and David Wajc. Online edge coloring algorithms via the nibble method. In *Proceedings of the 32nd Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 2830–2842, 2021.

6 Niv Buchbinder, Kamal Jain, and Joseph (Seffi) Naor. Online primal-dual algorithms for maximizing ad-auctions revenue. In *Proceedings of the 15th Annual European Symposium on Algorithms (ESA)*, pages 253–264. Springer, 2007.

7 Yi-Jun Chang, Qizheng He, Wenzheng Li, Seth Pettie, and Jara Uitto. The complexity of distributed edge coloring with small palettes. In *Proceedings of the 29th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 2633–2652, 2018.

8 Moses Charikar and Paul Liu. Improved algorithms for edge colouring in the w-streaming model. In *Proceedings of the 4th Symposium on Simplicity in Algorithms (SOSA)*, pages 181–183, 2021.

9 Ilan Reuven Cohen, Binghui Peng, and David Wajc. Tight bounds for online edge coloring. In *Proceedings of the 60th Symposium on Foundations of Computer Science (FOCS)*, pages 1–25, 2019.

10 Ilan Reuven Cohen and David Wajc. Randomized online matching in regular graphs. In *Proceedings of the 29th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 960–979, 2018.

11 Richard Cole, Kirstin Ost, and Stefan Schirra. Edge-coloring bipartite multigraphs in $O(E \log D)$ time. *Combinatorica*, 21(1):5–12, 2001.

12 Nikhil R Devanur, Kamal Jain, and Robert D Kleinberg. Randomized primal-dual analysis of ranking for online bipartite matching. In *Proceedings of the 24th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 101–107, 2013.

13 Ran Duan, Haoqing He, and Tianyi Zhang. Dynamic edge coloring with improved approximation. In *Proceedings of the 30th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1937–1945, 2019.

14 Devdatt Dubhashi and Desh Ranjan. Balls and bins: A study in negative dependence. *BRICS Report Series*, 3(25), 1996.

15 Tomer Ezra, Michal Feldman, Nick Gravin, and Zhihao Gavin Tang. Online stochastic max-weight matching: prophet inequality for vertex and edge arrival models. In *Proceedings of the 21st ACM Conference on Economics and Computation (EC)*, pages 769–787, 2020.

16 Matthew Fahrbach, Zhiyi Huang, Runzhou Tao, and Morteza Zadimoghaddam. Edge-weighted online bipartite matching. In *Proceedings of the 61st Symposium on Foundations of Computer Science (FOCS)*, pages 412–423, 2020.

17 Jon Feldman, Nitish Korula, Vahab Mirrokni, S Muthukrishnan, and Martin Pál. Online ad assignment with free disposal. In *Proceedings of the 5th Conference on Web and Internet Economics (WINE)*, pages 374–385, 2009.

18 Yiding Feng, Rad Niazadeh, and Amin Saberi. Two-stage stochastic matching with application to ride hailing. In *Proceedings of the 32nd Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 2862–2877, 2021.

19 Buddhima Gamlath, Michael Kapralov, Andreas Maggiori, Ola Svensson, and David Wajc. Online matching with general arrivals. In *Proceedings of the 60th Symposium on Foundations of Computer Science (FOCS)*, pages 26–37, 2019.

20 Ian Holyer. The np-completeness of edge-coloring. *SIAM Journal on Computing (SICOMP)*, 10(4):718–720, 1981.

21 Zhiyi Huang, Ning Kang, Zhihao Gavin Tang, Xiaowei Wu, Yuhao Zhang, and Xue Zhu. Fully online matching. *Journal of the ACM (JACM)*, 67(3):1–25, 2020.

22 Zhiyi Huang, Zhihao Gavin Tang, Xiaowei Wu, and Yuhao Zhang. Fully online matching ii: Beating ranking and water-filling. In *Proceedings of the 61st Symposium on Foundations of Computer Science (FOCS)*, pages 1380–1391, 2020.

23 Kumar Joag-Dev and Frank Proschan. Negative association of random variables with applications. *The Annals of Statistics*, pages 286–295, 1983.

24 Bala Kalyanasundaram and Kirk R Pruhs. An optimal deterministic algorithm for online b-matching. *Theoretical Computer Science (TCS)*, 233(1):319–325, 2000.

25 Howard J. Karloff and David B. Shmoys. Efficient parallel algorithms for edge coloring problems. *J. Algorithms*, 8(1):39–52, 1987.

26 Richard M Karp, Umesh V Vazirani, and Vijay V Vazirani. An optimal algorithm for online bipartite matching. In *Proceedings of the 22nd Annual ACM Symposium on Theory of Computing (STOC)*, pages 352–358, 1990.

27 Alam Khursheed and KM Lai Saxena. Positive dependence in multivariate distributions. *Communications in Statistics - Theory and Methods*, 10(12):1183–1196, 1981.

28 Dénes König. Über graphen und ihre anwendung auf determinantentheorie und mengenlehre. *Mathematische Annalen*, 77(4):453–465, 1916.

29 Aranyak Mehta. Online matching and ad allocation. *Foundations and Trends® in Theoretical Computer Science*, 8(4):265–368, 2013.

30 Aranyak Mehta, Amin Saberi, Umesh Vazirani, and Vijay Vazirani. Adwords and generalized online matching. *Journal of the ACM (JACM)*, 54(5):22, 2007.

31 Rajeev Motwani, Joseph (Seffi) Naor, and Moni Naor. The probabilistic method yields deterministic parallel algorithms. *Journal of Computer and System Sciences*, 49(3):478–516, 1994.

32 Joseph Seffi Naor and David Wajc. Near-optimum online ad allocation for targeted advertising. *ACM Transactions on Economics and Computation (TEAC)*, 6(3-4):16:1–16:20, 2018.

33 Chrisots Papadimitriou, Tristan Pollner, Amin Saberi, and David Wajc. Online stochastic max-weight bipartite matching: Beyond prophet inequalities. In *Proceedings of the 22nd ACM Conference on Economics and Computation (EC)*, 2021. To appear.

34 Hsin-Hao Su and Hoa T. Vu. Towards the locality of vizing's theorem. In *Proceedings of the 51st Annual ACM Symposium on Theory of Computing (STOC)*, pages 355–364, 2019.

35 Vadim G Vizing. On an estimate of the chromatic class of a p-graph. *Diskret analiz*, 3:25–30, 1964.

36 David Wajc. Rounding dynamic matchings against an adaptive adversary. In *Proceedings of the 52nd Annual ACM Symposium on Theory of Computing (STOC)*, pages 194–207, 2020.

37 Yajun Wang and Sam Chiu-wai Wong. Two-sided online bipartite matching and vertex cover: Beating the greedy algorithm. In *Proceedings of the 42nd International Colloquium on Automata, Languages and Programming (ICALP)*, pages 1070–1081, 2015.

Quantum Algorithms for Matrix Scaling and Matrix Balancing

Joran van Apeldoorn
Institute for Information Law and QuSoft, University of Amsterdam, The Netherlands

Sander Gribling
IRIF, Université de Paris, CNRS, Paris, France

Yinan Li
Graduate School of Mathematics, Nagoya University, Japan

Harold Nieuwboer
Korteweg–de Vries Institute for Mathematics and QuSoft,
University of Amsterdam, The Netherlands

Michael Walter
KdVI, ITFA, ILLC, and QuSoft, University of Amsterdam, The Netherlands

Ronald de Wolf
QuSoft, CWI, Amsterdam, The Netherlands
University of Amsterdam, The Netherlands

Abstract

Matrix scaling and matrix balancing are two basic linear-algebraic problems with a wide variety of applications, such as approximating the permanent, and pre-conditioning linear systems to make them more numerically stable. We study the power and limitations of quantum algorithms for these problems. We provide quantum implementations of two classical (in both senses of the word) methods: Sinkhorn's algorithm for matrix scaling and Osborne's algorithm for matrix balancing. Using amplitude estimation as our main tool, our quantum implementations both run in time $\widetilde{O}(\sqrt{mn}/\varepsilon^4)$ for scaling or balancing an $n \times n$ matrix (given by an oracle) with m non-zero entries to within ℓ_1-error ε. Their classical analogs use time $\widetilde{O}(m/\varepsilon^2)$, and every classical algorithm for scaling or balancing with small constant ε requires $\Omega(m)$ queries to the entries of the input matrix. We thus achieve a polynomial speed-up in terms of n, at the expense of a worse polynomial dependence on the obtained ℓ_1-error ε. Even for constant ε these problems are already non-trivial (and relevant in applications). Along the way, we extend the classical analysis of Sinkhorn's and Osborne's algorithm to allow for errors in the computation of marginals. We also adapt an improved analysis of Sinkhorn's algorithm for entrywise-positive matrices to the ℓ_1-setting, obtaining an $\widetilde{O}(n^{1.5}/\varepsilon^3)$-time quantum algorithm for ε-ℓ_1-scaling. We also prove a lower bound, showing our quantum algorithm for matrix scaling is essentially optimal for constant ε: every quantum algorithm for matrix scaling that achieves a constant ℓ_1-error w.r.t. uniform marginals needs $\Omega(\sqrt{mn})$ queries.

2012 ACM Subject Classification Theory of computation → Design and analysis of algorithms; Theory of computation → Quantum computation theory

Keywords and phrases Matrix scaling, matrix balancing, quantum algorithms

Digital Object Identifier 10.4230/LIPIcs.ICALP.2021.110

Category Track A: Algorithms, Complexity and Games

Related Version *Full Version*: `https://arxiv.org/abs/2011.12823v1` [9]

Funding *Joran van Apeldoorn*: Dutch Research Council (NWO/OCW), as part of QSC (024.003.037).
Sander Gribling: Partially supported by SIRTEQ-grant QuIPP.
Yinan Li: MEXT Quantum Leap Flagship Program grant no. JPMXS0120319794.
Harold Nieuwboer: NWO grant no. OCENW.KLEIN.267.
Michael Walter: NWO Veni grant no. 680-47-459 and grant no. OCENW.KLEIN.267.
Ronald de Wolf: NWO/OCW through QSC (024.003.037) and QuantAlgo (QuantERA 680-91-034).

Acknowledgements We thank the ICALP referees for some very helpful feedback.

1 Introduction

1.1 Matrix scaling and matrix balancing

Matrix scaling is a basic linear-algebraic problem with many applications. A *scaling* of an $n \times n$ matrix $\mathbf{A}$ with non-negative entries is a matrix $\mathbf{B} = \mathbf{XAY}$ where $\mathbf{X}$ and $\mathbf{Y}$ are positive diagonal matrices (everything straightforwardly extends to non-square $\mathbf{A}$). In other words, we multiply the i-th row with X_{ii} and the j-th column with Y_{jj}. We say $\mathbf{A}$ is *exactly scalable* to marginals $\mathbf{r} \in \mathbb{R}^n_+$ and $\mathbf{c} \in \mathbb{R}^n_+$ if there exist $\mathbf{X}$ and $\mathbf{Y}$ such that the vector $\mathbf{r}(\mathbf{B}) = (\sum_{j=1}^n B_{ij})_{i\in[n]}$ of row sums of the scaled matrix $\mathbf{B}$ equals $\mathbf{r}$, and its vector $\mathbf{c}(\mathbf{B})$ of column sums equals $\mathbf{c}$. One typical example would be if $\mathbf{r}$ and $\mathbf{c}$ are the all-1 vectors, which means we want $\mathbf{B}$ to be doubly stochastic: the rows and columns of $\mathbf{B}$ would be probability distributions. In many cases it suffices to find *approximate* scalings. Different applications use different notions of approximation. We could for instance require $\mathbf{r}(\mathbf{B})$ to be ε-close to $\mathbf{r}$ in ℓ_1- or ℓ_2-norm, or in relative entropy (Kullback-Leibler divergence), for some parameter ε of our choice, and similarly require $\mathbf{c}(\mathbf{B})$ to be ε-close to $\mathbf{c}$.

A related problem is *matrix balancing*. Here we do not prescribe desired marginals, but the goal is to find a diagonal $\mathbf{X}$ such that the row and column marginals of $\mathbf{B} = \mathbf{XAX^{-1}}$ are close *to each other*. Again, different notions of closeness $\mathbf{r}(\mathbf{B}) \approx \mathbf{c}(\mathbf{B})$ are possible.

An important application, used in theory as well as in practical linear-algebra software (e.g. LAPACK [6] and MATLAB [40]), is in improving the numerical stability of linear-system solving. Suppose we are given matrix $\mathbf{A}$ and vector $\mathbf{b}$, and we want to find a solution to the linear system $\mathbf{Av} = \mathbf{b}$. Note that $\mathbf{v}$ is a solution iff $\mathbf{Bv'} = \mathbf{b'}$ for $\mathbf{v'} = \mathbf{Xv}$ and $\mathbf{b'} = \mathbf{Xb}$. An appropriately balanced matrix $\mathbf{B}$ will typically be more numerically stable than the original $\mathbf{A}$, so solving the linear system $\mathbf{Bv'} = \mathbf{b'}$ and then computing $\mathbf{v} = \mathbf{X}^{-1}\mathbf{v'}$, is often a better way to solve the linear system $\mathbf{Av} = \mathbf{b}$ than directly computing $\mathbf{A}^{-1}\mathbf{b}$.

Matrix scaling and balancing have surprisingly many and wide-ranging applications. Matrix scaling was introduced by Kruithof for Dutch telephone traffic computation [37], and has also been used in other areas of economics [50]. In theoretical computer science it has been used for instance to approximate the permanent of a given matrix [38], as a tool to get lower bounds on unbounded-error communication complexity [25], and for approximating optimal transport distances [2]. In mathematics, it has been used as a common tool in practical linear algebra computations [39, 12, 46, 42], but also in statistics [49], optimization [47], and for strengthening the Sylvester-Gallai theorem [11]. Matrix balancing has a similarly wide variety of applications, including pre-conditioning to make practical matrix computations more stable (as mentioned above), and approximating the min-mean-cycle in a weighted graph [3]. Many more applications of matrix scaling and balancing are mentioned in [38, 31, 28]. Related scaling problems have applications to algorithmic non-commutative algebra [27, 17], functional analysis [26], and quantum information [29, 18, 16].

1.2 Known (classical) algorithms

Given the importance of good matrix scalings and balancings, how efficiently can we actually find them? For concreteness, let us first focus on scaling. Note that left-multiplying $\mathbf{A}$ with a diagonal matrix $\mathbf{X}$ corresponds to multiplying the i-th row of $\mathbf{A}$ with X_{ii}. Hence it is very easy to get the desired row sums: just compute all row sums $r_i(\mathbf{A})$ of $\mathbf{A}$ and define $\mathbf{X}$ by $X_{ii} = r_i/r_i(\mathbf{A})$, then $\mathbf{XA}$ has exactly the right row sums. Subsequently, it is easy to get the desired column sums: just right-multiply the current matrix $\mathbf{XA}$ with diagonal matrix $\mathbf{Y}$ where $Y_{jj} = c_j/c_j(\mathbf{XA})$, then $\mathbf{XAY}$ will have the right column sums.

The problem with this approach, of course, is that the second step is likely to undo the good work of the first step, changing the row sums away from the desired values; it is not at all obvious how to *simultaneously* get the row sums and column sums right. Nevertheless, the approach of alternating row-normalizations with column-normalizations turns out to work. This alternating algorithm is known as *Sinkhorn's algorithm* [49], and has actually been (re)discovered independently in several different contexts.

For matrix balancing there is a similar method called *Osborne's algorithm* [43, 45]. In each iteration this chooses a row index i and defines X_{ii} such that the i-th row sum and the i-th column sum become equal. Again, because each iteration can undo the good work of earlier iterations, convergence to a balancing of $\mathbf{A}$ is not at all obvious. Remarkably, even though Osborne's algorithm was proposed more than six decades ago and is widely used in linear algebra software, an explicit convergence-rate bound was only proved recently [48, 44]!

At the same time there have been other, more sophisticated algorithmic approaches for scaling and balancing. Just to mention one: we can parametrize $\mathbf{X} = \mathrm{diag}(e^{\mathbf{x}})$ and $\mathbf{Y} = \mathrm{diag}(e^{\mathbf{y}})$ by vectors $\mathbf{x}, \mathbf{y} \in \mathbb{R}^n$ and consider the following convex potential function:

$$f(\mathbf{x}, \mathbf{y}) = \sum_{i,j=1}^{n} A_{ij} e^{x_i + y_j} - \sum_{i=1}^{n} r_i x_i - \sum_{j=1}^{n} c_j y_j.$$

Note that the partial derivative of this f w.r.t. the variable x_i is $\sum_{j=1}^{n} A_{ij} e^{x_i+y_j} - r_i = r_i(\mathbf{XAY}) - r_i$, and the partial derivative w.r.t. y_j is $c_j(\mathbf{XAY}) - c_j$. A minimizer $\mathbf{x}, \mathbf{y}$ of f will have the property that all these $2n$ partial derivatives are equal to 0, which means $\mathbf{XAY}$ *is exactly scaled!* Accordingly, (approximate) scalings can be obtained by finding (approximate) minimizers using methods from convex optimization. In fact, Sinkhorn's original algorithm can be interpreted as block coordinate descent on this f, and Osborne's algorithm can similarly be derived by slightly modifying f. More advanced methods from convex optimization have also been applied, such as ellipsoid methods [36, 33, 41], box-constrained Newton methods [1, 21] and interior-point methods [21, 19].

Historically, research on matrix scaling and matrix balancing (and generalizations such as operator scaling) has focused on finding ε-ℓ_2-scalings. More recently also algorithms for finding ε-ℓ_1-scalings have been extensively studied, due to their close connection with permanents and finding perfect matchings in bipartite graphs [38, 20], and because the ℓ_1-distance is an important error measure for statistical problems such as computing the optimal transport distance between distributions [22, 2], even already for constant ε. By the Cauchy-Schwarz inequality, an $(\varepsilon/\sqrt{n})$-ℓ_2-scaling for $\mathbf{A}$ is also an ε-ℓ_1-scaling, but often more direct methods work better for finding an ε-ℓ_1-scaling.

Below in Table 1 we tabulate the best known algorithms for finding ε-scalings in ℓ_1-norm for entrywise-positive matrices and general non-negative matrices. We make the standard assumptions that every entry of the target marginals $\mathbf{r}, \mathbf{c}$ is non-zero, and that $\mathbf{A}$ is *asymptotically scalable*: for every $\varepsilon > 0$, there exist $\mathbf{X}$ and $\mathbf{Y}$ such that

$$\|\mathbf{r}(\mathbf{B}) - \mathbf{r}\|_1 \leq \varepsilon \text{ and } \|\mathbf{c}(\mathbf{B}) - \mathbf{c}\|_1 \leq \varepsilon,$$

where $\mathbf{B} = \mathbf{XAY}$ (this implies that the matrix has at least one non-zero entry in every row and column). A sufficient condition for this is that the matrix is entrywise-positive. To state the complexity results, let m be the number of non-zero entries in $\mathbf{A}$ (note that $m \geq n$), assume $\sum_{i,j=1}^{n} A_{ij} = 1$, that its non-zero entries lie in $[\mu, 1]$, and $\|\mathbf{r}\|_1 = \|\mathbf{c}\|_1 = 1$ (so uniform marginal is $\mathbf{1}/n$). We will assume $\varepsilon \in (0, 1)$. The input numbers to the algorithm are all assumed to be rational, with bit size bounded by $\mathrm{polylog}(n)$, unless specified otherwise.[1]

[1] The complexity of our algorithms depends polylogarithmically on the magnitude of the entries; the assumption on the bit size of the entries is made to simplify the presentation.

Note that the table contains both first-order and second-order methods; the former just use the gradient of the potential (or a related potential), whereas the latter also use its Hessian. The second-order methods typically have a polylogarithmic dependence on the inverse of the desired precision ε, whereas the first-order methods have inverse polynomial dependence on ε. For entrywise-positive matrices, second-order methods theoretically outperform the *classical* first-order methods in any parameter regime. However, they depend on non-trivial results for graph sparsification and Laplacian system solving which are relatively complicated to implement, in contrast to the eminently practical (first-order) Sinkhorn and Osborne.

For matrix balancing, Osborne's algorithm has very recently been shown to produce an ε-ℓ_1-balancing in time $\widetilde{O}(m/\varepsilon^2)$ when in each iteration the update is chosen randomly [4]. Algorithms based on box-constrained Newton methods and interior-point methods can find ε-balancings in time $\widetilde{O}(m \log \kappa)$ and $\widetilde{O}(m^{1.5})$, respectively, where κ denotes the ratio between the largest and the smallest entries of the optimal balancing.

Table 1 State-of-the-art time complexity of first- and second-order methods for finding an ε-ℓ_1-scaling, both to uniform marginals and to arbitrary marginals. The boldface lines are from this paper, and the only quantum algorithms for scaling we are aware of. h is the smallest integer such that $h\mathbf{r}$, $h\mathbf{c}$ are integer vectors; m upper bounds the number of non-zero entries of $\mathbf{A}$; κ is the ratio between largest and smallest entries of the optimal scalings $\mathbf{X}$ and $\mathbf{Y}$, which can be exponential in n. Many references use a different error model (e.g., ℓ_2 or Kullback-Leibler), which we convert to guarantees in ℓ_1-norm for comparison. $\widetilde{O}$-notation hides polylogarithmic factors in n, $1/\varepsilon$, $1/\mu$.

	$(\mathbf{1}/n, \mathbf{1}/n)$	$(\mathbf{r}, \mathbf{c})$	References and remarks
General non-negative	$\widetilde{O}(m/\varepsilon^2)$	$\widetilde{O}(m/\varepsilon^2)$	Sinkhorn, via KL [20][2]
	$\widetilde{O}(mn^{2/3}/\varepsilon^{2/3})$	$\widetilde{O}(mn/(h^{1/3}\varepsilon^{2/3}))$	first-order, via ℓ_2 [1]
	$\widetilde{O}(m \log \kappa)$	$\widetilde{O}(m \log \kappa)$	box-constrained method, via ℓ_2 [21]
	$\widetilde{O}(m^{1.5})$	$\widetilde{O}(m^{1.5})$	interior-point method, via ℓ_2 [21]
	$\widetilde{O}(\sqrt{mn}/\varepsilon^4)$	$\widetilde{O}(\sqrt{mn}/\varepsilon^4)$	**Sinkhorn, quantum, Corollary 5**
Entrywise positive	$\widetilde{O}(n^2/\varepsilon)$	$\widetilde{O}(n^3/\varepsilon)$	Sinkhorn, via ℓ_2 [35, 34], $h\varepsilon \leq \sqrt{2n}$
	$\widetilde{O}(n^2/\varepsilon^2)$	$\widetilde{O}(n^2/\varepsilon^2)$	Sinkhorn, via KL [2, 20]
	$\widetilde{O}(n^2)$	$\widetilde{O}(n^2)$	box-constrained, via ℓ_2 [1, 21]
	$\widetilde{O}(n^{1.5}/\varepsilon^3)$	$\widetilde{O}(n^{1.5}/\varepsilon^3)$	**Sinkhorn, quantum, Corollary 7**

1.3 First contribution: quantum algorithms for ℓ_1-scaling and balancing

Because a classical scaling algorithm has to look at each non-zero matrix entry (at least with large probability), it is clear that $\Omega(m)$ is a classical lower bound. This is $\Omega(n^2)$ in the case of a dense or even entrywise-positive matrix $\mathbf{A}$. As can be seen from Table 1, the best classical algorithms also achieve this m lower bound up to logarithmic factors, with various dependencies on ε. The same is true for balancing: $\Omega(m)$ queries are necessary, and this is achievable in different ways, with different dependencies on ε and other parameters.

Our first contribution is to give (in Section 3) quantum algorithms for scaling and balancing that beat the best-possible classical algorithms, at least for relatively large $\varepsilon \in (0, 1)$:

[2] Their proofs work only for input matrices that are exactly scalable. However, with our potential gap bound we can generalize their analysis to work for arbitrary asymptotically-scalable matrices.

Scaling: We give a quantum algorithm that (with probability $\geq 2/3$) finds an ε-ℓ_1-scaling for an asymptotically-scalable $n \times n$ matrix $\mathbf{A}$ with m non-zero entries (given by an oracle) to desired positive marginals $\mathbf{r}$ and $\mathbf{c}$ in time $\widetilde{O}(\sqrt{mn}/\varepsilon^4)$. When $\mathbf{A}$ is entrywise positive (so $m = n^2$), the upper bound can be improved to $\widetilde{O}(n^{1.5}/\varepsilon^3)$.

Balancing: We give a quantum algorithm that (with probability $\geq 2/3$) finds an ε-ℓ_1-balancing for an asymptotically-balanceable $n \times n$ matrix $\mathbf{A}$ with m non-zero entries (given by an oracle) in time $\widetilde{O}(\sqrt{mn}/\varepsilon^4)$.

Our scaling algorithms in fact achieve closeness measured in terms of the relative entropy, and then use Pinsker's inequality ($\|\mathbf{p}-\mathbf{q}\|_1^2 = O(D(\mathbf{p}||\mathbf{q}))$) to convert this to an upper bound on the ℓ_1-error. Our algorithms achieve a *sublinear* dependence on the input size m.

Note that compared to the classical algorithms we have polynomially better dependence on n and m, at the expense of a worse dependence on ε. There have recently been a number of new quantum algorithms with a similar tradeoff: they are better than classical in terms of the main size parameter but worse in terms of the precision parameter. Examples are the quantum algorithms for solving linear and semidefinite programs [14, 8, 13, 7] and for boosting of weak learning algorithms [10, 30, 32].

Conceptually our algorithms are quite simple: we implement the Sinkhorn and Osborne algorithms but replace the exact computation of each row and column sum by *quantum amplitude estimation*; this allows us to approximate the sum of n numbers up to some small multiplicative error δ (with high probability) at the expense of roughly $\sqrt{n}/\delta$ queries to those numbers, and a similar number of other operations.

Our analysis is based on a potential argument (for Sinkhorn we use the above-mentioned potential f). The error δ causes us to make less progress in each iteration compared to an "exact" version of Sinkhorn or Osborne. If δ is too large we may even make backwards progress, while if δ is very small there is no quantum speed-up! We show there is a choice of δ for which the negative contribution due to the approximation errors is of the same order as the progress in the "exact" version, and that choice results in a speed-up. We should caution, however, that it is quite complicated to actually implement this idea precisely and to keep track of and control the various approximation errors and error probabilities induced by the quantum estimation algorithms, as well as by the fact that we cannot represent the numbers involved with infinite precision (this issue of precision is sometimes swept under the rug in classical research on scaling algorithms). Finally, we note that due to the error δ our potential need not decrease monotonically. The standard analysis of Sinkhorn still applies if we can *test* whether the current scaling is an ε-scaling after each full Sinkhorn iteration. We show how to do so efficiently in the quantum setting. However, in Osborne's algorithm one updates only a single (random) row/column per iteration, and the quantum cost of our testing procedure is higher than the cost of updating (classically, this problem is overcome by simply keeping track of the row and column marginals). To circumvent the need for testing every iteration, we give a novel analysis of Osborne's algorithm (and of a randomized version of Sinkhorn) showing uniformly random iterates provide an ε-balancing with high probability.

1.4 Second contribution: quantum lower bound for scaling

A natural question would be whether our upper bounds for the time complexity of matrix scaling and balancing can be improved further. Since the output has length roughly n, there is an obvious lower bound of n even for quantum algorithms. An $\widetilde{O}(n)$ quantum algorithm would, however, still be an improvement over our algorithms, and it would be a quadratic

speed-up over the best classical algorithm. In Section 4 we dash this hope for matrix scaling by showing that our algorithm is essentially optimal for constant ε, even for the special case of $\mathbf{A}$ that is exactly scalable to uniform marginals:

> There exists a constant $\varepsilon > 0$ such that every quantum algorithm that (with probability $\geq 2/3$) finds an ε-ℓ_1-scaling for a given $n \times n$ matrix $\mathbf{A}$ that is exactly scalable to uniform and has m potentially non-zero entries, has to make $\Omega(\sqrt{mn})$ queries to $\mathbf{A}$.

Our proof constructs instances $\mathbf{A}$ that hide permutations, shows how approximate scalings of $\mathbf{A}$ give us information about the hidden permutation, and then uses the adversary method [5] to lower bound the number of quantum queries to the matrix needed to find that information. In particular, we show that for a permutation $\sigma \in S_n$, it takes $\Omega(n\sqrt{n})$ queries to the entries of the associated permutation matrix to learn $\sigma(i) \bmod 2$ for each $i \in [n]$.

2 Preliminaries

2.1 Matrix scaling and balancing

Throughout we use $\mathbf{r}, \mathbf{c} \in \mathbb{R}^n$ as the desired row and column marginals. Unambiguously, we also use $\mathbf{r}\colon \mathbb{R}^{n\times n} \to \mathbb{R}^n$ (resp. $\mathbf{c}\colon \mathbb{R}^{n\times n} \to \mathbb{R}^n$) as the function that sends an $n \times n$-matrix to its row (resp. column) marginal: $\mathbf{r}(\mathbf{A})$ (resp. $\mathbf{c}(\mathbf{A})$) is the vector whose i-th entry equals $r_i(\mathbf{A}) = \sum_{j=1}^n A_{ij}$ (resp. $c_i(\mathbf{A}) = \sum_{j=1}^n A_{ji}$). We use $\mathbf{A}(\mathbf{x}, \mathbf{y}) = (A_{ij}e^{x_i+y_j})_{i,j\in[n]}$ to denote the rescaled matrix $\mathbf{A}$ with scalings given by $e^{\mathbf{x}}$ and $e^{\mathbf{y}}$. We say a non-negative matrix $\mathbf{A} \in \mathbb{R}_+^{n\times n}$ is *exactly scalable* to $(\mathbf{r}, \mathbf{c}) \in \mathbb{R}_+^n \times \mathbb{R}_+^n$, if there exist $\mathbf{x}, \mathbf{y} \in \mathbb{R}^n$ such that $\mathbf{r}(\mathbf{A}(\mathbf{x}, \mathbf{y})) = \mathbf{r}$ and $\mathbf{c}(\mathbf{A}(\mathbf{x}, \mathbf{y})) = \mathbf{c}$. For an $\varepsilon > 0$, we say $\mathbf{A} \in \mathbb{R}_+^{n\times n}$ is ε-ℓ_1-*scalable* to $(\mathbf{r}, \mathbf{c}) \in \mathbb{R}_+^n \times \mathbb{R}_+^n$, if there exist $\mathbf{x}, \mathbf{y} \in \mathbb{R}^n$ such that $\|\mathbf{r}(\mathbf{A}(\mathbf{x}, \mathbf{y})) - \mathbf{r}\|_1 \leq \varepsilon$ and $\|\mathbf{c}(\mathbf{A}(\mathbf{x}, \mathbf{y})) - \mathbf{c}\|_1 \leq \varepsilon$. We say $\mathbf{A} \in \mathbb{R}_+^{n\times n}$ is *asymptotically scalable* to $(\mathbf{r}, \mathbf{c}) \in \mathbb{R}_+^n \times \mathbb{R}_+^n$ if it is ε-ℓ_1-scalable to $(\mathbf{r}, \mathbf{c})$ for every $\varepsilon > 0$. In the matrix-balancing setting we require $\mathbf{y} = -\mathbf{x}$, and the marginals are compared to each other. We abbreviate $\mathbf{A}(\mathbf{x}) = \mathbf{A}(\mathbf{x}, -\mathbf{x})$. We say a non-negative matrix $\mathbf{A} \in \mathbb{R}_+^{n\times n}$ is *exactly balanceable*, if there exists a vector $\mathbf{x} \in \mathbb{R}^n$ such that $\mathbf{r}(\mathbf{A}(\mathbf{x})) = \mathbf{c}(\mathbf{A}(\mathbf{x}))$. For an $\varepsilon > 0$, we say $\mathbf{A} \in \mathbb{R}_+^{n\times n}$ is ε-ℓ_1-*balanceable*, if there exists an $\mathbf{x} \in \mathbb{R}^n$ such that $\frac{\|\mathbf{r}(\mathbf{A}(\mathbf{x})) - \mathbf{c}(\mathbf{A}(\mathbf{x}))\|_1}{\|\mathbf{A}(\mathbf{x})\|_1} \leq \varepsilon$. We say $\mathbf{A} \in \mathbb{R}_+^{n\times n}$ is *asymptotically balanceable* if it is ε-ℓ_1-balanceable for every $\varepsilon > 0$. The associated optimization problems are as follows.

▶ **Problem 1** (ε-ℓ_1-scaling problem). *Given $\mathbf{A} \in \mathbb{R}_+^{n\times n}$ and desired marginals $\mathbf{r}, \mathbf{c} \in \mathbb{R}_+^n$ with $\|\mathbf{r}\|_1 = \|\mathbf{c}\|_1 = 1$, find $\mathbf{x}, \mathbf{y} \in \mathbb{R}^n$ s.t. $\|\mathbf{r}(\mathbf{A}(\mathbf{x}, \mathbf{y})) - \mathbf{r}\|_1 \leq \varepsilon$ and $\|\mathbf{c}(\mathbf{A}(\mathbf{x}, \mathbf{y})) - \mathbf{c}\|_1 \leq \varepsilon$.*

▶ **Problem 2** (ε-ℓ_1-balancing problem). *Given $\mathbf{A} \in \mathbb{R}_+^{n\times n}$, find $\mathbf{x} \in \mathbb{R}^n$ s.t. $\|\mathbf{r}(\mathbf{A}(\mathbf{x})) - \mathbf{c}(\mathbf{A}(\mathbf{x}))\|_1 / \|\mathbf{A}(\mathbf{x})\|_1 \leq \varepsilon$.*

For matrix scaling, our algorithm is most naturally analyzed with the error measured by the relative entropy, which can be converted to give an upper bound on ℓ_1-distance using (a generalized version of) Pinsker's inequality. We therefore also consider the following problem:

▶ **Problem 3** (ε-relative-entropy-scaling problem). *Given $\mathbf{A} \in \mathbb{R}_+^{n\times n}$ and desired marginals $\mathbf{r}, \mathbf{c} \in \mathbb{R}_+^n$ with $\|\mathbf{r}\|_1 = \|\mathbf{c}\|_1 = 1$, find $\mathbf{x}, \mathbf{y} \in \mathbb{R}^n$ such that $D(\mathbf{r}\|\mathbf{r}(\mathbf{A}(\mathbf{x}, \mathbf{y}))) \leq \varepsilon$ and $D(\mathbf{c}\|\mathbf{c}(\mathbf{A}(\mathbf{x}, \mathbf{y}))) \leq \varepsilon$.*

2.2 Computational model

We assume *sparse black-box access* to the elements of $\mathbf{A}$ via lists of the potentially non-zero entries for each row and each column. A quantum algorithm can make such queries also in superposition. We also assume (classical) black-box access to the target marginals $\mathbf{r}, \mathbf{c}$. Our computational model is of a classical computer (say, a Random Access Machine for concreteness) that can invoke a quantum computer as a subroutine. The classical computer can write to a classical-write quantum-read memory ("QCRAM")[3], and send a description of a quantum circuit that consists of one- and two-qubit gates from some fixed discrete universal gate set (say, the H and T gates, Controlled-NOT, and 2-qubit controlled rotations over angles $2\pi/2^s$ for positive integers s; these controlled rotations are used in the circuit for the quantum Fourier transform (QFT), which we invoke later), queries to the input oracles, and queries to the QCRAM to the quantum computer. The quantum computer runs the circuit, measures the full final state in the computational basis, and returns the measurement outcome to the classical computer. See [9, Sec. 2] for details.

3 A Sinkhorn algorithm with approximate updates

We state and analyze Algorithm 1, a variant of the well-known Sinkhorn algorithm. Here we give an overview of its analysis, we refer to [9, Sec. 3] for the proofs. The algorithm's objective is to find scaling vectors $\mathbf{x}, \mathbf{y} \in \mathbb{R}^n$ such that the matrix $\mathbf{A}(\mathbf{x}, \mathbf{y}) = (A_{ij} e^{x_i + y_j})_{i,j \in [n]}$ has row and column marginals $\mathbf{r}$ and $\mathbf{c}$, respectively. Sinkhorn-type algorithms do so in the following iterative way. Starting from the rational matrix $\mathbf{A} \in [0,1]^{n \times n}$, find a vector $\mathbf{x}$ such that the row marginals of $(A_{ij} e^{x_i})_{i,j \in [n]}$ are $\mathbf{r}$, and then find a $\mathbf{y}$ such that the column marginals of $\mathbf{A}(\mathbf{x}, \mathbf{y})$ are $\mathbf{c}$. The second step may have changed the row marginals, so we repeat the procedure. We can view this as updating the coordinates of $\mathbf{x}$ and $\mathbf{y}$ one at a time, starting from the all-0 vectors. To update the row scaling vectors, we wish to find $\hat{\mathbf{x}} = \mathbf{x} + \mathbf{\Delta}$ such that $\mathbf{r}(\mathbf{A}(\hat{\mathbf{x}}, \mathbf{y})) = \mathbf{r}$. Expanding this equation yields $e^{\Delta_\ell} \cdot r_\ell(\mathbf{A}(\mathbf{x}, \mathbf{y})) = r_\ell$, for $\ell \in [n]$. Every row and column contains at least one non-zero entry, so this has a unique solution:

$$\hat{x}_\ell = x_\ell + \Delta_\ell = x_\ell + \ln\left(\frac{r_\ell}{r_\ell(\mathbf{A}(\mathbf{x}, \mathbf{y}))}\right) = \ln\left(\frac{r_\ell}{\sum_{j=1}^n A_{\ell j} e^{y_j}}\right). \tag{3.1}$$

Similar formulas can be derived for the column-updates. We use the term "one Sinkhorn iteration" to refer to the process of updating all n row scaling vectors, or updating all n column scaling vectors. We state the Sinkhorn algorithm in terms of two subroutines, `ApproxScalingFactor` and `TestScaling`. For both subroutines we provide both classical and quantum implementations in [9, Sec. 4]. A key ingredient of both subroutines is a procedure that computes the logarithm of a sum of exponentials, see Section 3.2 for a high-level explanation of the quantum subroutine. For the analysis of Algorithm 1, we only use the guarantees of the subroutines as stated, and do not refer to their actual implementation.

We study a version of Sinkhorn's algorithm where, instead of computing row and column marginals in each iteration exactly, we use a *multiplicative* approximation of the marginals to compute δ-additive approximations of Equation (3.1) and similar for column-updates. In the classical literature, δ can be chosen to be very small, since the cost per iteration scales as $\mathrm{polylog}(1/\delta)$, and hence that error is essentially a minor technical detail. In the quantum setting, we obtain better dependence in terms of n at the cost of allowing a $\mathrm{poly}(1/\delta)$-dependence, so the required precision δ merits detailed attention in the analysis.

[3] Note that we do not require a full QRAM that can hold qubits as well. We believe QCRAM is a natural assumption since it simply amounts to classical RAM that can be read in superposition.

Algorithm 1 Full Sinkhorn with finite precision and failure probability.

Input: Oracle access to $\mathbf{A} \in [0,1]^{n\times n}$ with $\|\mathbf{A}\|_1 \leq 1$ and non-zero entries at least $\mu > 0$, target marginals $\mathbf{r}, \mathbf{c} \in (0,1]^n$ with $\|\mathbf{r}\|_1 = \|\mathbf{c}\|_1 = 1$, iteration count $T \in \mathbb{N}$, bit counts $b_1, b_2 \in \mathbb{N}$, estimation precision $0 < \delta < 1$, test precision $0 < \delta' < 1$ and subroutine failure probability $\eta \in [0,1]$

Output: Vectors $\mathbf{x}, \mathbf{y} \in \mathbb{R}^n$ with entries encoded in (b_1, b_2) fixed-point format

Guarantee: For $\varepsilon \in (0,1]$, with parameters chosen as in Proposition 13, $(\mathbf{x}, \mathbf{y})$ form an ε-relative-entropy-scaling of $\mathbf{A}$ to $(\mathbf{r}, \mathbf{c})$ with probability $\geq 2/3$

$\mathbf{x}^{(0)}, \mathbf{y}^{(0)} \leftarrow \mathbf{0}$; `// entries in` (b_1, b_2) `fixed-point format`
for $t \leftarrow 1, 2, \ldots, T$ **do**
  if *t is odd* **then**
    for $\ell \leftarrow 1, 2, \ldots, n$ **do**
      $x_\ell^{(t)} \leftarrow$ `ApproxScalingFactor`$(\mathbf{A}_{\ell\bullet}, r_\ell, \mathbf{y}^{(t-1)}, \delta, b_1, b_2, \eta, \mu)$;
    end for
    $\mathbf{y}^{(t)} \leftarrow \mathbf{y}^{(t-1)}$;
  else if *t is even* **then**
    for $\ell \leftarrow 1, 2, \ldots, n$ **do**
      $y_\ell^{(t)} \leftarrow$ `ApproxScalingFactor`$(\mathbf{A}_{\bullet\ell}, c_\ell, \mathbf{x}^{(t-1)}, \delta, b_1, b_2, \eta, \mu)$;
    end for
    $\mathbf{x}^{(t)} \leftarrow \mathbf{x}^{(t-1)}$;
  end if
  if `TestScaling`$(\mathbf{A}, \mathbf{r}, \mathbf{c}, \mathbf{x}^{(t)}, \mathbf{y}^{(t)}, \delta', b_1, b_2, \eta, \mu)$ **then**
    return $(\mathbf{x}^{(t)}, \mathbf{y}^{(t)})$;
  end if
end for
return $(\mathbf{x}^{(T)}, \mathbf{y}^{(T)})$;

The Sinkhorn algorithm thus has a number of tunable parameters (precision, error parameters, iteration count). We show how to choose them in such a way that the resulting quantum algorithm obtains an ε-relative-entropy-scaling in time $\widetilde{O}(\sqrt{mn}/\varepsilon^2)$.

▶ **Theorem 4.** *Let $\mathbf{A} \in [0,1]^{n\times n}$ be a matrix with $\|\mathbf{A}\|_1 \leq 1$ and m non-zero entries, each rational and at least $\mu > 0$, let $\mathbf{r}, \mathbf{c} \in (0,1]^n$ with $\|\mathbf{r}\|_1 = \|\mathbf{c}\|_1 = 1$, and let $\varepsilon \in (0,1]$. Assume $\mathbf{A}$ is asymptotically scalable to $(\mathbf{r}, \mathbf{c})$. Then there exists a quantum algorithm that, given sparse oracle access to $\mathbf{A}$, with probability $\geq 2/3$, computes $(\mathbf{x}, \mathbf{y}) \in \mathbb{R}^n \times \mathbb{R}^n$ such that $\mathbf{A}(\mathbf{x}, \mathbf{y})$ is ε-relative-entropy-scaled to $(\mathbf{r}, \mathbf{c})$, for a total time complexity of $\widetilde{O}(\sqrt{mn}/\varepsilon^2)$.*

A generalization of Pinsker's inequality (cf. [9, Lem. 2.1]) implies the following corollary.

▶ **Corollary 5.** *Let $\mathbf{A}, \mathbf{r}, \mathbf{c}$ and ε be as in Theorem 4. Then there exists a quantum algorithm that with probability $\geq 2/3$ computes $(\mathbf{x}, \mathbf{y}) \in \mathbb{R}^n \times \mathbb{R}^n$ such that $\mathbf{A}(\mathbf{x}, \mathbf{y})$ is ε-ℓ_1-scaled to $(\mathbf{r}, \mathbf{c})$, for a total time complexity of $\widetilde{O}(\sqrt{mn}/\varepsilon^4)$.*

In [9, Thm. C.6], we show that if the matrix $\mathbf{A}$ is entrywise-positive, then the number of iterations to obtain an ε-relative-entropy-scaling can be reduced to roughly $1/\sqrt{\varepsilon}$ rather than roughly $1/\varepsilon$, leading to the following theorem.

Procedure ApproxScalingFactor($\mathbf{a}, r, \mathbf{y}, \delta, b_1, b_2, \eta, \mu$).

Input: Oracle access to rational $\mathbf{a} \in [0,1]^n$, rational $r \in (0,1]$, oracle access to $\mathbf{y} \in \mathbb{R}^n$ encoded in (b_1, b_2) fixed-point format, desired precision $\delta \in (0,1]$, desired failure prob. $\eta \in [0,1]$, lower bound $\mu > 0$ on non-zero entries of $\mathbf{a}$

Output: A number x encoded in (b_1, b_2) fixed-point format

Guarantee: If $b_1 \geq \lceil \log_2(|\ln(r/\sum_{j=1}^n a_j e^{y_j})|) \rceil$ and $b_2 \geq \lceil \log_2(1/\delta) \rceil$, then with prob. $\geq 1-\eta$, x is a δ-additive approximation of $\ln(r/\sum_{j=1}^n a_j e^{y_j})$

Procedure TestScaling($\mathbf{A}, \mathbf{r}, \mathbf{c}, \mathbf{x}, \mathbf{y}, \delta, b_1, b_2, \eta, \mu$).

Input: Oracle access to rational $\mathbf{A} \in [0,1]^{n \times n}$ with $\|\mathbf{A}\|_1 \leq 1$, rational $\mathbf{r}, \mathbf{c} \in (0,1]^n$, oracle access to $\mathbf{x}, \mathbf{y} \in \mathbb{R}^n$ encoded in (b_1, b_2) fixed-point format, test precision $\delta \in (0,1)$, desired failure probability $\eta \in [0,1]$, lower bound $\mu > 0$ on non-zero entries of $\mathbf{A}$

Output: A bit indicating whether $\mathbf{x}, \mathbf{y}$ forms a δ-relative-entropy-scaling of $\mathbf{A}$ to target marginals $\mathbf{r}, \mathbf{c}$.

Guarantee: If $b_1 \geq \log_2(|\ln(r_\ell/\sum_{j=1}^n A_{\ell j} e^{y_j})|)$ for all $\ell \in [n]$, and similarly for the columns, and furthermore $b_2 \geq \lceil \log_2(1/\delta) \rceil$, then with probability at least $1-\eta$: outputs **False** if $D(\mathbf{r} \| \mathbf{r}(\mathbf{A}(\mathbf{x}, \mathbf{y}))) \geq 2\delta$ or $D(\mathbf{c} \| \mathbf{c}(\mathbf{A}(\mathbf{x}, \mathbf{y}))) \geq 2\delta$, outputs **True** if both are at most δ

▶ **Theorem 6.** *Let $\mathbf{A} \in [\mu, 1]^{n \times n}$ be a matrix with $\|\mathbf{A}\|_1 \leq 1$, each entry rational and at least $\mu > 0$, let $\mathbf{r}, \mathbf{c} \in (0,1]^n$ with $\|\mathbf{r}\|_1 = \|\mathbf{c}\|_1 = 1$, and let $\varepsilon \in (0,1]$. Then there exists a quantum algorithm that, given sparse oracle access to $\mathbf{A}$, with probability $\geq 2/3$, computes $(\mathbf{x}, \mathbf{y}) \in \mathbb{R}^n \times \mathbb{R}^n$ such that $\mathbf{A}(\mathbf{x}, \mathbf{y})$ is ε-relative-entropy-scaled to $(\mathbf{r}, \mathbf{c})$, at a total time complexity of $\widetilde{O}(n^{1.5}/\varepsilon^{1.5})$.*

The next corollary also follows from the generalization of Pinsker's inequality.

▶ **Corollary 7.** *Let $\mathbf{A}, \mathbf{r}, \mathbf{c}$ and ε be as in Theorem 6. Then there exists a quantum algorithm that with probability $\geq 2/3$ computes $(\mathbf{x}, \mathbf{y}) \in \mathbb{R}^n \times \mathbb{R}^n$ such that $\mathbf{A}(\mathbf{x}, \mathbf{y})$ is ε-ℓ_1-scaled to $(\mathbf{r}, \mathbf{c})$, for a total time complexity of $\widetilde{O}(\sqrt{mn}/\varepsilon^3)$.*

3.1 Potential argument

The analysis uses the convex potential function

$$f(\mathbf{x}, \mathbf{y}) = \sum_{i,j=1}^n A_{ij} e^{x_i + y_j} - \sum_{i=1}^n r_i x_i - \sum_{j=1}^n c_j y_j.$$

This function (already mentioned in the introduction) is often used in the context of matrix scaling, as its gradient is the difference between the current and desired marginals. Many of the more sophisticated algorithms for matrix scaling minimize this function directly. For our purposes, we first bound the potential gap $f(\mathbf{0}, \mathbf{0}) - \inf_{\mathbf{x}, \mathbf{y} \in \mathbb{R}^n} f(\mathbf{x}, \mathbf{y})$ (see [9, App. A]).

▶ **Lemma 8** (Potential gap). *Assume $\mathbf{A} \in [0,1]^{n \times n}$ with $\|\mathbf{A}\|_1 \leq 1$ and non-zero entries at least $\mu > 0$. If $\mathbf{A}$ is asymptotically $(\mathbf{r}, \mathbf{c})$-scalable, then $f(\mathbf{0}, \mathbf{0}) - \inf_{\mathbf{x}, \mathbf{y} \in \mathbb{R}^n} f(\mathbf{x}, \mathbf{y}) \leq \ln(1/\mu)$.*

For matrices $\mathbf{A}$ that are *exactly* $(\mathbf{r}, \mathbf{c})$-scalable, this bound is well-known (see e.g. [34, 20]), but to the best of our knowledge, it has not yet appeared in the literature when $\mathbf{A}$ is only assumed to be asymptotically scalable to $(\mathbf{r}, \mathbf{c})$.

One can show that, for a Sinkhorn iteration in which we update the rows exactly, i.e., $\hat{x}_\ell = \ln(r_\ell / \sum_{j=1}^n A_{\ell j} e^{y_j})$ for $\ell \in [n]$, the potential decreases by exactly the relative entropy:

$$f(\mathbf{x}, \mathbf{y}) - f(\hat{\mathbf{x}}, \mathbf{y}) = D(\mathbf{r} \| \mathbf{r}(\mathbf{A}(\mathbf{x}, \mathbf{y}))), \tag{3.2}$$

and similarly for exact column updates. The next lemma generalizes this to allow for error in the update; it shows that we can lower bound the decrease of the potential function in every iteration in terms of the relative entropy between the target marginal and the current marginal, assuming every call to the subroutine `ApproxScalingFactor` succeeds.

▶ **Lemma 9.** *Let $\mathbf{A} \in \mathbb{R}_+^{n \times n}$, let $\mathbf{x}, \mathbf{y} \in \mathbb{R}^n$, let $\delta \in [0, 1]$, and let $\hat{\mathbf{x}} \in \mathbb{R}^n$ be a vector such that for every $\ell \in [n]$, we have $|\hat{x}_\ell - \ln(r_\ell / \sum_{j=1}^n A_{\ell j} e^{y_j})| \leq \delta$. Then*

$$f(\mathbf{x}, \mathbf{y}) - f(\hat{\mathbf{x}}, \mathbf{y}) \geq D\big(\mathbf{r} \big\| \mathbf{r}(\mathbf{A}(\mathbf{x}, \mathbf{y}))\big) - 2\delta.$$

A similar statement holds for an update of $\mathbf{y}$ (using $\mathbf{c}$ instead of $\mathbf{r}$ in the relative entropy).

When $\delta = 0$, the inequality becomes an equality which is well-known, see e.g. [2] or [20]. The lemma shows that updating the scaling vectors with additive precision δ suffices to make progress in minimizing the potential function f, as long as we are still $\Omega(\delta)$ away from the desired marginals (in relative entropy distance).

We thus wish to store the entries of $\mathbf{x}$ and $\mathbf{y}$ with additive precision $\delta > 0$. We want to do so using a (b_1, b_2) fixed-point format, so we need $b_2 \geq \lceil \log_2(1/\delta) \rceil$. The guarantees of `ApproxScalingFactor` and `TestScaling` assert that this choice of b_2 is also sufficient. Lemma 10 shows how large we need to take b_1 to ensure the requirements of `ApproxScalingFactor` and `TestScaling` are satisfied in every iteration. The algorithm returns as soon as `TestScaling` returns **True**, or after T iterations. However, to simplify the analysis, we always assume that $\mathbf{x}^{(t)}$ and $\mathbf{y}^{(t)}$ are defined for $t = 0, \ldots, T$.

▶ **Lemma 10** (Bounding the scalings). *Let $\mathbf{A} \in [0, 1]^{n \times n}$ with $\|\mathbf{A}\|_1 \leq 1$ and non-zero entries at least $\mu > 0$. Let $T \geq 1$ and $\delta \in [0, 1]$. Denote $\sigma = \max(|\ln r_{\min}|, |\ln c_{\min}|)$. Let $b_2 = \lceil \log_2(1/\delta) \rceil$ and choose $b_1 = \lceil \log_2(T) + \log_2(\ln(\frac{1}{\mu}) + 1 + \sigma) \rceil$. If for all $t \in [T]$ the subroutine `ApproxScalingFactor` succeeds, then for all $t \in [T]$ and $\ell \in [n]$ we have*

$$\left| \ln \left(\frac{r_\ell}{\sum_{j=1}^n A_{\ell j} e^{y_j^{(t)}}} \right) \right| \leq 2^{b_1}, \quad \left| \ln \left(\frac{c_\ell}{\sum_{i=1}^n A_{i \ell} e^{x_i^{(t)}}} \right) \right| \leq 2^{b_1}$$

and $\|(\mathbf{x}^{(t)}, \mathbf{y}^{(t)})\|_\infty \leq t \left(\ln \left(\frac{1}{\mu} \right) + \delta + \sigma \right) \leq t \left(\ln \left(\frac{1}{\mu} \right) + 1 + \sigma \right)$.

To formally analyze the expected progress it is convenient to define the following events.

▶ **Definition 11** (Important events). *For $t = 1, \ldots, T$, we define the following events:*

- *Let S_t be the event that all n calls to `ApproxScalingFactor` succeed in the t-th iteration.*
- *Define S to be the intersection of the events S_t, i.e., $S = \cap_{t=1}^T S_t$.*

To give some intuition, we note below that the event S is the "good" event where a row-update makes the relative entropy between $\mathbf{r}$ and the updated row-marginals at most δ (and similarly for the columns). We only use Lemma 12 in [9, App. C].

▶ **Lemma 12.** *If S holds and $\delta \leq 1$, then the following holds for all $t \in [T]$:*

- *If t is odd, then $D(\mathbf{r} \| \mathbf{r}(\mathbf{A}(\mathbf{x}^{(t)}, \mathbf{y}^{(t)}))) \leq \delta$.*
- *If t is even, then $D(\mathbf{c} \| \mathbf{c}(\mathbf{A}(\mathbf{x}^{(t)}, \mathbf{y}^{(t)}))) \leq \delta$.*

We can combine Lemmas 8 and 9 to show Algorithm 1 returns, with high probability, an ε-relative-entropy-scaling to $(\mathbf{r}, \mathbf{c})$ by choosing $\delta = O(\varepsilon)$.

▶ **Proposition 13.** *Let $\mathbf{A} \in [0,1]^{n\times n}$ with $\|\mathbf{A}\|_1 \leq 1$ and non-zero entries at least $\mu > 0$ and rational, and let $\mathbf{r}, \mathbf{c} \in (0,1]^n$ with $\|\mathbf{r}\|_1 = \|\mathbf{c}\|_1 = 1$. Assume $\mathbf{A}$ is asymptotically scalable to $(\mathbf{r}, \mathbf{c})$. For $\varepsilon \in (0,1]$, choose $T = \left\lceil \frac{8}{\varepsilon} \ln\left(\frac{1}{\mu}\right) \right\rceil + 1$, $\delta = \frac{\varepsilon}{16}$, $\delta' = \frac{\varepsilon}{2}$, $\eta = \frac{1}{3(n+1)T}$, $b_2 = \lceil \log_2(\frac{1}{\delta}) \rceil$, and $b_1 = \lceil \log_2(T) + \log_2(\ln(\frac{1}{\mu}) + \sigma + 1) \rceil$, where $\sigma = \max(|\ln r_{\min}|, |\ln c_{\min}|)$. Then, Algorithm 1 returns $(\mathbf{x}, \mathbf{y})$ s.t. $D(\mathbf{r} \| \mathbf{r}(\mathbf{A}(\mathbf{x}, \mathbf{y}))) \leq \varepsilon$ and $D(\mathbf{c} \| \mathbf{c}(\mathbf{A}(\mathbf{x}, \mathbf{y}))) \leq \varepsilon$ with probability $\geq 2/3$.*

3.2 Quantum approximate summing

`ApproxScalingFactor` and `TestScaling` both rely on the computation of additive approximations to numbers of the form $\ln(\sum_{i=1}^n a_i e^{y_i})$. Here we sketch our approach and mention some complications; see [9, Sec. 4] for details. If we assume the numbers $b_i = a_i e^{y_i}$ can be queried at unit cost, then we can efficiently compute $\ln(\sum_{i=1}^n b_i)$ up to additive error δ using amplitude estimation, as follows. After pre-processing (using *quantum maximum finding* [24]) one may assume that $b_i \in [0,1]$ and $\max_i b_i \geq 1/2$. With 2 queries to the b_is and a small number of other gates we prepare $\frac{1}{\sqrt{n}} \sum_{i=1}^n |i\rangle \left(\sqrt{b_i}\,|0\rangle + \sqrt{1-b_i}\,|1\rangle\right)$. The squared norm of the part ending in $|0\rangle$ equals $p = \frac{1}{n}\sum_{i=1}^n b_i \in [1/2n, 1]$. Let $\delta \in (0, 1/2]$ be an error parameter that we instantiate later. Using amplitude amplification we estimate p up to multiplicative error $1 \pm \delta$ using $O(\frac{1}{\delta}\sqrt{1/p}) = O(\frac{1}{\delta}\sqrt{n})$ queries to the b_is, and $\widetilde{O}(\frac{1}{\delta}\sqrt{n})$ gates, with error probability $\leq 1/3$ [15, Theorem 12]. We can reduce the error probability to a small $\eta > 0$, by running this $O(\log(1/\eta))$ times and outputting the median outcome. Naturally, multiplicative approximation of $\sum_{i=1}^n b_i$ yields additive approximation of $\ln(\sum_{i=1}^n b_i) = \ln(\sum_{i=1}^n a_i e^{y_i})$.

One obstacle to efficiently implementing the above is that one cannot simply compute all numbers to sufficient precision. For `ApproxScalingFactor` for instance, we aim to compute a number $\ln(r/\sum_{j=1}^n a_j e^{y_j})$ where the y_j can (and typically do) grow linearly with n, so we cannot compute e^{y_j} with sufficiently high precision in time sublinear in n. Instead we compute additive approximations of relative quantities such as $e^{y_i - y_j} \leq 1$ for $i, j \in [n]$ with $y_j \geq y_i$, and use properties of the log to relate this to the original desired quantity. This approach is widely used in practice, e.g., [4]. Note that these issues concern both the classical and quantum setting, but are particularly important for the latter, since we aim for a better dependence on m and n for the time complexity. We implement everything such that the fixed-point format (b_1, b_2) for both the input and output of the oracles is the same, avoiding the need to change the encoding format in every Sinkhorn or Osborne iteration.

3.3 Time complexity

Combining the above, we prove one of our main results (already stated earlier), bounding the time complexity of computing an ε-relative-entropy-scaling of $\mathbf{A}$ to marginals $(\mathbf{r}, \mathbf{c})$.

▶ **Theorem 4.** *Let $\mathbf{A} \in [0,1]^{n\times n}$ be a matrix with $\|\mathbf{A}\|_1 \leq 1$ and m non-zero entries, each rational and at least $\mu > 0$, let $\mathbf{r}, \mathbf{c} \in (0,1]^n$ with $\|\mathbf{r}\|_1 = \|\mathbf{c}\|_1 = 1$, and let $\varepsilon \in (0,1]$. Assume $\mathbf{A}$ is asymptotically scalable to $(\mathbf{r}, \mathbf{c})$. Then there exists a quantum algorithm that, given sparse oracle access to $\mathbf{A}$, with probability $\geq 2/3$, computes $(\mathbf{x}, \mathbf{y}) \in \mathbb{R}^n \times \mathbb{R}^n$ such that $\mathbf{A}(\mathbf{x}, \mathbf{y})$ is ε-relative-entropy-scaled to $(\mathbf{r}, \mathbf{c})$, for a total time complexity of $\widetilde{O}(\sqrt{mn}/\varepsilon^2)$.*

Proof. We show that Algorithm 1 with the parameters chosen as in Proposition 13 has the stated time complexity. Note that the cost of computing these parameters from the input will be dominated by the runtime of the algorithm. Proposition 13 shows that Algorithm 1 runs for at most $O(\ln(1/\mu)/\varepsilon)$ iterations. Next we show the time complexity per iteration is $\widetilde{O}(\sqrt{mn}/\varepsilon)$, which implies the claimed total time complexity of $\widetilde{O}(\sqrt{mn}/\varepsilon^2)$.

Theorem 4.5 in [9] formalizes the discussion of Section 3.2: using `ApproxScalingFactor` with precision $\delta = \Theta(\varepsilon)$ on a row containing s potentially non-zero entries incurs a cost $\widetilde{O}(\sqrt{s}/\varepsilon)$, where we suppress a polylogarithmic dependence on n. One iteration of Algorithm 1 applies `ApproxScalingFactor` once to each row or once to each column, so by Cauchy–Schwarz the total cost of the calls to `ApproxScalingFactor` in one iteration is

$$\widetilde{O}\Big(\sum_{i=1}^{n}\sqrt{s_i^r}/\varepsilon + \sum_{j=1}^{n}\sqrt{s_j^c}/\varepsilon\Big) \subseteq \widetilde{O}(\sqrt{mn}/\varepsilon),$$

where we recall that s_i^r and s_j^c are the numbers of potentially non-zero entries in the i-th row and j-th column of $\mathbf{A}$, respectively, and m is the total number of potentially non-zero entries in $\mathbf{A}$ (i.e., $\sum_{i=1}^n s_i^r = m = \sum_{j=1}^n s_j^c$). Similarly, [9, Thm. 4.7] shows invoking `TestScaling` with precision $\delta' = \Theta(\varepsilon)$ incurs a cost of order $\widetilde{O}(\sqrt{mn}/\varepsilon)$. Finally we observe that compiling the quantum circuits (and preparing their inputs) for the calls to `ApproxScalingFactor` and `TestScaling` can be done with at most a polylogarithmic overhead. ◀

Note that the dependency on $\ln(1/\mu)$ is suppressed by the $\widetilde{O}$, since we assume the numerator and denominators of the rational inputs are bounded by a polynomial in n.

3.4 Complications in Osborne's algorithm

For the matrix balancing problem one can use a similar approach as for the matrix scaling problem. The idea is to fix the requirement of being ε-ℓ_1-balanced for individual coordinates, one at a time. More precisely, given an index $\ell \in [n]$, the update is given by $\mathbf{x}' = \mathbf{x} + \Delta_\ell \mathbf{e}_\ell$, where Δ_ℓ is chosen such that $r_\ell(\mathbf{A}(\mathbf{x}')) = c_\ell(\mathbf{A}(\mathbf{x}'))$. Expanding this and using $A_{\ell\ell} = 0$ yields $e^{\Delta_\ell} \cdot r_\ell(\mathbf{A}(\mathbf{x})) = e^{-\Delta_\ell} \cdot c_\ell(\mathbf{A}(\mathbf{x}))$. Since we assume every row and column contains at least one non-zero entry, the above equation has a unique solution, given by

$$\Delta_\ell = \ln\left(\sqrt{c_\ell(\mathbf{A}(\mathbf{x}))/r_\ell(\mathbf{A}(\mathbf{x}))}\right). \tag{3.3}$$

Note that the updates of multiple coordinates *cannot* be done simultaneously, since each coordinate can potentially affect all row and column marginals. This is in contrast with the Sinkhorn algorithm for matrix scaling, where all rows or all columns can be updated at the same time. This provides a significant challenge in the analysis of the algorithm since we can no longer test whether we have found an ε-balancing in between each iteration. We give a novel analysis of Osborne's algorithm [9, Sec. 6] and of a randomized version of Sinkhorn's algorithm [9, Sec. 5] that shows a uniformly random iterate provides an ε-balancing/scaling with high probability. For matrix balancing this yields the following.

▶ **Theorem 14.** *Let $\mathbf{A} \in [0,1]^{n\times n}$ be a matrix whose non-zero entries are rational, at least $\mu > 0$, with zeroes on the diagonal, each row and column having at least one non-zero element, and let $\varepsilon \in (0,1]$. Assume $\mathbf{A}$ is asymptotically balanceable. Then there exists a quantum algorithm that, given sparse oracle access to $\mathbf{A}$, returns with probability $\geq 2/3$ a vector $\mathbf{x} \in \mathbb{R}^n$ such that $\mathbf{A}(\mathbf{x})$ is ε-ℓ_1-balanced, with expected time complexity $\widetilde{O}(\sqrt{mn}/\varepsilon^4)$.*

4 Matching lower bound for matrix scaling with constant ε

We show that our algorithm for matrix scaling is in fact optimal with respect to the dependence on n and m, for constant $\varepsilon > 0$. We prove an $\Omega(\sqrt{mn})$ lower bound on the query complexity of quantum algorithms for $\Theta(1)$-ℓ_1-scaling to the uniform marginals $(\mathbf{1}/n, \mathbf{1}/n)$. Here we sketch the case $m = n^2$; Section 7 in [9] gives the full proof also for the sparse case.

We consider the problem of learning a permutation "modulo two" in the following sense.[4]

▶ **Definition 15** (Single-bit descriptor). *Let $\sigma \in S_n$ be a permutation. The* single-bit descriptor *of σ is the bit string $z \in \{0,1\}^n$ with entries $z_i \equiv \sigma(i) \bmod 2$.*

We first use the adversary method [5] to prove an $\Omega(n\sqrt{n})$ bound for recovering the single-bit descriptor, given (dense) oracle access to the permutation matrix. This is tight, since one can use Grover on each column to fully recover the permutation matrix. We follow a similar proof structure as in the $\Omega(\sqrt{n})$-query lower bound given in [5] for finding $\sigma^{-1}(1)$, and the $\Omega(n\sqrt{n})$-query lower bound for graph connectivity [23].

▶ **Lemma 16.** *Let n be a positive multiple of 4. Given an $n \times n$ permutation matrix $\mathbf{P}$ corresponding to a permutation σ (i.e., $P_{ij} = \delta_{i,\sigma(j)}$ for $i, j \in [n]$), recovering the single-bit descriptor z of σ, with success probability at least $2/3$, requires $\Omega(n\sqrt{n})$ quantum queries to a dense matrix oracle for $\mathbf{P}$.*

One can then boost this lower bound to show that even learning a (certain) constant fraction of the entries of the single-bit descriptor of a permutation requires $\Omega(n\sqrt{n})$ quantum queries. (The precise constant depends on those in Lemma 16 and in Grover search.) We then reduce the problem of learning the single-bit descriptor to the scaling problem, by replacing each 1-entry of the permutation matrix by one of two 2×2 gadget matrices (and each 0-entry by the 2×2 all-0 matrix). These gadgets are such that we can determine (most of) the single-bit descriptor from the column-scaling vectors $\mathbf{y}$ of an $\Theta(1)$-ℓ_1-scaling to uniform marginals. Explicitly, the gadget matrices are as follows:

$$\mathbf{B}_0 = \begin{bmatrix} \frac{2}{9} & \frac{4}{9} \\ \frac{1}{9} & \frac{2}{9} \end{bmatrix}, \qquad \mathbf{B}_1 = \begin{bmatrix} \frac{4}{9} & \frac{2}{9} \\ \frac{2}{9} & \frac{1}{9} \end{bmatrix},$$

Note that the two matrices have the same columns, but in reverse order. We show in the next lemma that from an approximate scaling of $\mathbf{B}_i$ to uniform marginals, one can recover the bit i.

▶ **Lemma 17.** *The matrices $\mathbf{B}_0, \mathbf{B}_1 \in \{\frac{1}{9}, \frac{2}{9}, \frac{4}{9}\}^{2\times 2}$ are entrywise positive, with entries summing to one, and they are exactly scalable to uniform marginals. For $i \in \{0,1\}$, let $(\mathbf{x}, \mathbf{y})$ be $\frac{1}{8}$-ℓ_1-scaling vectors for $\mathbf{B}_i$ to uniform marginals. If $i = 0$ then $y_1 - y_2 > 0.18$, while if $i = 1$ then $y_1 - y_2 < -0.18$. Moreover, $(\mathbf{x}, (y_2, y_1))$ are $\frac{1}{8}$-ℓ_1-scaling vectors for $\mathbf{B}_{1-i}$ to uniform marginals.*

In other words, the matrices can be distinguished just by learning the column-scaling vectors, but they have the same set of possible row-scaling vectors.

[4] Alternatively, one could consider the problem of learning an entire permutation, which would simplify the notation and proofs slightly. However, for the reduction to matrix scaling, this seems to require gadget matrices of size roughly $\log_2 n \times \log_2 n$, leading to a slightly weaker lower bound.

Proof. Since one matrix is obtained by swapping the columns of the other, the last claim is immediately clear, and it suffices to prove the remaining claims for $\mathbf{B}_0$.

First, we note that $\mathbf{B}_0$ is exactly scalable, since

$$\begin{bmatrix} \frac{3}{4} & 0 \\ 0 & \frac{3}{2} \end{bmatrix} \begin{bmatrix} \frac{2}{9} & \frac{4}{9} \\ \frac{1}{9} & \frac{2}{9} \end{bmatrix} \begin{bmatrix} \frac{3}{2} & 0 \\ 0 & \frac{3}{4} \end{bmatrix} = \begin{bmatrix} \frac{1}{4} & \frac{1}{4} \\ \frac{1}{4} & \frac{1}{4} \end{bmatrix}$$

has uniform marginals. Now suppose that $(\mathbf{x}, \mathbf{y})$ is an $\frac{1}{8}$-ℓ_1-scaling of $\mathbf{B}_0$ to uniform marginals. By the requirement on the column marginals, we have

$$\left(\frac{2}{9}e^{x_1} + \frac{1}{9}e^{x_2}\right) e^{y_1} \geq \frac{1}{2} - \frac{1}{8} \quad \text{and} \quad \left(\frac{4}{9}e^{x_1} + \frac{2}{9}e^{x_2}\right) e^{y_2} \leq \frac{1}{2} + \frac{1}{8}.$$

By dividing the first inequality by the second one we get

$$\frac{1}{2} \cdot \frac{e^{y_1}}{e^{y_2}} \geq \frac{3}{5},$$

and so $y_1 - y_2 \geq \ln \frac{6}{5} > 0.18$. ◀

Together with Lemma 16 this leads to the following lower bound.

▶ **Theorem 18.** *There exists a constant $\varepsilon \in (0, 1)$ such that any quantum algorithm which, given a sparse oracle for an $n \times n$-matrix that is exactly scalable to uniform marginals and has m potentially non-zero entries which sum to 1, returns an ε-ℓ_1-scaling with probability $\geq 2/3$, requires $\Omega(\sqrt{mn})$ quantum queries to the oracle.*

References

1. Zeyuan Allen-Zhu, Yuanzhi Li, Rafael Oliveira, and Avi Wigderson. Much faster algorithms for matrix scaling. In *Proceedings of IEEE 58th Annual Symposium on Foundations of Computer Science (FOCS'17)*, pages 890–901, 2017.
2. Jason Altschuler, Jonathan Niles-Weed, and Philippe Rigollet. Near-linear time approximation algorithms for optimal transport via Sinkhorn iteration. In *Advances in Neural Information Processing Systems*, volume 30, pages 1964–1974, 2017.
3. Jason M. Altschuler and Pablo A. Parrilo. Approximating Min-Mean-Cycle for low-diameter graphs in near-optimal time and memory, 2020. `arXiv:2004.03114`.
4. Jason M. Altschuler and Pablo A. Parrilo. Random Osborne: A simple, practical algorithm for matrix balancing in near-linear time, 2020. `arXiv:2004.02837`.
5. Andris Ambainis. Quantum lower bounds by quantum arguments. *Journal of Computer and System Sciences*, 64(4):750–767, 2002. Earlier version in STOC'00. quant-ph/0002066.
6. E. Anderson, Z. Bai, C. Bischof, L. S. Blackford, J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenney, and D. Sorensen. *LAPACK Users' Guide*. SIAM, 1999. `doi:10.1137/1.9780898719604`.
7. Joran van Apeldoorn and András Gilyén. Improvements in quantum SDP-solving with applications. In *Proceedings of 46th International Colloquium on Automata, Languages, and Programming (ICALP 2019)*, volume 132 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 99:1–99:15, 2019. `doi:10.4230/LIPIcs.ICALP.2019.99`.
8. Joran van Apeldoorn, András Gilyén, Sander Gribling, and Ronald de Wolf. Quantum SDP-solvers: Better upper and lower bounds. *Quantum*, 4(230), 2020. Earlier version in FOCS'17. `arXiv:1705.01843`.
9. Joran van Apeldoorn, Sander Gribling, Yinan Li, Harold Nieuwboer, Michael Walter, and Ronald de Wolf. Quantum algorithms for matrix scaling and matrix balancing, 2020. `arXiv:2011.12823v1`.

10 Srinivasan Arunachalam and Reevu Maity. Quantum boosting. In *Proceedings of 37th International Conference on Machine Learning (ICML'20)*, 2020. arXiv:2002.05056.

11 Boaz Barak, Zeev Dvir, Amir Yehudayoff, and Avi Wigderson. Rank bounds for design matrices with applications to combinatorial geometry and locally correctable codes. In *Proceedings of 43rd Symposium on Theory of Computing (STOC'11)*, pages 519–528. ACM, 2011.

12 Andrew Michael Bradley. *Algorithms for the equilibration of matrices and their application to limited-memory Quasi-Newton methods*. PhD thesis, Stanford University, 2010.

13 Fernando G. S. L. Brandão, Amir Kalev, Tongyang Li, Cedric Yen-Yu Lin, Krysta M. Svore, and Xiaodi Wu. Quantum SDP solvers: Large speed-ups, optimality, and applications to quantum learning. In *Proceedings of 46th International Colloquium on Automata, Languages, and Programming (ICALP 2019)*, volume 132 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 27:1–27:14, 2019. doi:10.4230/LIPIcs.ICALP.2019.27.

14 Fernando G. S. L. Brandão and Krysta M. Svore. Quantum speed-ups for solving semidefinite programs. In *Proceedings of IEEE 58th Annual Symposium on Foundations of Computer Science (FOCS'17)*, pages 415–426, 2017. doi:10.1109/FOCS.2017.45.

15 Gilles Brassard, Peter Høyer, Michele Mosca, and Alain Tapp. Quantum amplitude amplification and estimation. In *Quantum Computation and Quantum Information: A Millennium Volume*, volume 305 of *Contemporary Mathematics*, pages 53–74. American Mathematical Society, 2002. arXiv:quant-ph/0005055.

16 Peter Bürgisser, Cole Franks, Ankit Garg, Rafael Oliveira, Michael Walter, and Avi Wigderson. Efficient algorithms for tensor scaling, quantum marginals, and moment polytopes. In *Proceedings of 59th IEEE Annual Symposium on Foundations of Computer Science (FOCS'18)*, pages 883–897, 2018. doi:10.1109/FOCS.2018.00088.

17 Peter Bürgisser, Cole Franks, Ankit Garg, Rafael Oliveira, Michael Walter, and Avi Wigderson. Towards a theory of non-commutative optimization: geodesic 1st and 2nd order methods for moment maps and polytopes. In *Proceedings of 60th IEEE Annual Symposium on Foundations of Computer Science (FOCS'19)*, pages 845–861. IEEE, 2019.

18 Peter Bürgisser, Ankit Garg, Rafael Oliveira, Michael Walter, and Avi Wigderson. Alternating minimization, scaling algorithms, and the null-cone problem from invariant theory. In *Proceedings of 9th Innovations in Theoretical Computer Science Conference (ITCS 2018)*, volume 94 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 24:1–24:20, 2018. doi:10.4230/LIPIcs.ITCS.2018.24.

19 Peter Bürgisser, Yinan Li, Harold Nieuwboer, and Michael Walter. Interior-point methods for unconstrained geometric programming and scaling problems, 2020. arXiv:2008.12110.

20 Deeparnab Chakrabarty and Sanjeev Khanna. Better and simpler error analysis of the Sinkhorn–Knopp algorithm for matrix scaling. *Mathematical Programming*, pages 1–13, 2020.

21 Michael B. Cohen, Aleksander Madry, Dimitris Tsipras, and Adrian Vladu. Matrix scaling and balancing via box constrained Newton's method and interior point methods. In *Proceedings of IEEE 58th Annual Symposium on Foundations of Computer Science (FOCS'17)*, pages 902–913, 2017. doi:10.1109/FOCS.2017.88.

22 Marco Cuturi. Sinkhorn distances: Lightspeed computation of optimal transport. In *Advances in Neural Information Processing Systems*, volume 26, pages 2292–2300, 2013.

23 Christoph Dürr, Mark Heiligman, Peter Høyer, and Mehdi Mhalla. Quantum query complexity of some graph problems. *SIAM Journal on Computing*, 35(6):1310–1328, 2006. doi:10.1137/050644719.

24 Christoph Dürr and Peter Høyer. A quantum algorithm for finding the minimum, 1996. arXiv:quant-ph/9607014.

25 Jürgen Forster. A linear lower bound on the unbounded error probabilistic communication complexity. In *Proceedings of 16th Annual IEEE Conference on Computational Complexity*, pages 100–106, 2001. doi:10.1109/CCC.2001.933877.

26 Ankit Garg, Leonid Gurvits, Rafael Oliveira, and Avi Wigderson. Algorithmic and optimization aspects of Brascamp-Lieb inequalities, via operator scaling. *Geometric and Functional Analysis*, 28(1):100–145, 2018. Earlier version in STOC'17.

27 Ankit Garg, Leonid Gurvits, Rafael Oliveira, and Avi Wigderson. Operator scaling: theory and applications. *Foundations of Computational Mathematics*, pages 1–68, 2019. Earlier version in FOCS'16.

28 Ankit Garg and Rafael Oliveira. Recent progress on scaling algorithms and applications. *Bulletin of the EATCS, Computational Complexity Column*, 125, 2018. arXiv:1808.09669.

29 Leonid Gurvits. Classical complexity and quantum entanglement. *Journal of Computer and System Sciences*, 69(3):448–484, 2004. doi:10.1016/j.jcss.2004.06.003.

30 Yassine Hamoudi, Maharshi Ray, Patrick Rebentrost, Miklos Santha, Xin Wang, and Siyi Yang. Quantum algorithms for hedging and the Sparsitron, 2020. arXiv:2002.06003.

31 Martin Idel. A review of matrix scaling and Sinkhorn's normal form for matrices and positive maps, 2016. arXiv:1609.06349.

32 Adam Izdebski and Ronald de Wolf. Improved quantum boosting, 2020. arXiv:2009.08360.

33 B. Kalantari, L. Khachiyan, and A. Shokoufandeh. On the complexity of matrix balancing. *SIAM Journal on Matrix Analysis and Applications*, 18(2):450–463, 1997. doi:10.1137/S0895479895289765.

34 B. Kalantari, I. Lari, F. Ricca, and B. Simeone. On the complexity of general matrix scaling and entropy minimization via the RAS algorithm. *Mathematical Programming*, 112:371–401, 2008.

35 Bahman Kalantari and Leonid Khachiyan. On the rate of convergence of deterministic and randomized RAS matrix scaling algorithms. *Operations Research Letters*, 14(5):237–244, 1993. doi:10.1016/0167-6377(93)90087-W.

36 Bahman Kalantari and Leonid Khachiyan. On the complexity of nonnegative-matrix scaling. *Linear Algebra and its Applications*, 240:87–103, 1996. doi:10.1016/0024-3795(94)00188-X.

37 J. Kruithof. Telefoonverkeersrekening. *De Ingenieur*, 52:E15–E25, 1937.

38 Nathan Linial, Alex Samorodnitsky, and Avi Wigderson. A deterministic strongly polynomial algorithm for matrix scaling and approximate permanents. *Combinatorica*, 20(4):545–568, 2000. URL: https://www.math.ias.edu/~avi/PUBLICATIONS/MYPAPERS/LSW98/lsw00.pdf.

39 Oren E. Livne and Gene H. Golub. Scaling by binormalization. *Numerical Algorithms*, 35(1):97–120, 2004.

40 Mathworks. balance: diagonal scaling to improve eigenvalue accuracy. URL: https://www.mathworks.com/help/matlab/ref/balance.html.

41 Arkadi Nemirovski and Uriel Rothblum. On complexity of matrix scaling. *Linear Algebra and its Applications*, 302-303:435–460, 1999. doi:10.1016/S0024-3795(99)00212-8.

42 Brendan O'Donoghue, Eric Chu, Neal Parikh, and Stephen Boyd. A primal-dual operator splitting method for conic optimization. *Journal of Optimization Theory and Applications*, 169(3):1042–1068, 2016. arXiv:1312.3039.

43 E. E. Osborne. On pre-conditioning of matrices. *Journal of ACM*, 7(4), 1960. doi:10.1145/321043.321048.

44 Rafail Ostrovsky, Yuval Rabani, and Arman Yousefi. Matrix balancing in L_p norms: Bounding the convergence rate of Osborne's iteration. In *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA'17)*, pages 154–169, 2017. doi:10.1137/1.9781611974782.11.

45 B. N. Parlett and C. Reinsch. Balancing a matrix for calculation of eigenvalues and eigenvectors. *Numerische Mathematik*, 13:293–304, 1969.

46 Thomas Pock and Antonin Chambolle. Diagonal preconditioning for first order primal-dual algorithms in convex optimization. In *Proceedings of IEEE International Conference on Computer Vision (ICCV)*, pages 1762–1769, 2011.

47 Uriel G. Rothblum and Hans Schneider. Scalings of matrices which have prespecified row sums and column sums via optimization. *Linear Algebra and its Applications*, 114:737–764, 1989.

48 Leonard J. Schulman and Alistair Sinclair. Analysis of a classical matrix preconditioning algorithm. In *Proceedings of 47th Annual ACM Symposium on Theory of Computing (STOC'15)*, pages 831–840, 2015.

49 Richard Sinkhorn. A relationship between arbitrary positive matrices and doubly stochastic matrices. *The Annals of Mathematical Statistics*, 35(2):876–879, 1964.

50 Richard Stone. *Multiple classifications in social accounting*. University of Cambridge, Department of Applied Economics, 1964.

Fourier Conjectures, Correlation Bounds, and Majority

Emanuele Viola
Khoury College of Computer Sciences, Northeastern University, Boston, MA, USA

Abstract

Recently several conjectures were made regarding the Fourier spectrum of low-degree polynomials. We show that these conjectures imply new correlation bounds for functions related to Majority. Then we prove several new results on correlation bounds which aim to, but don't, resolve the conjectures. In particular, we prove several new results on Majority which are of independent interest and complement Smolensky's classic result.

2012 ACM Subject Classification Theory of computation → Circuit complexity

Keywords and phrases Fourier analysis, polynomials, Majority, correlation, lower bound, conjectures

Digital Object Identifier 10.4230/LIPIcs.ICALP.2021.111

Category Track A: Algorithms, Complexity and Games

Related Version *Full Version*: `https://eccc.weizmann.ac.il/report/2020/175`

Funding *Emanuele Viola*: Supported by NSF CCF award 1813930.

Acknowledgements This paper includes the results in [28].
I am grateful to Chin Ho Lee for pointing out the work [5] to me, and to an anonymous reviewer for suggesting the use of hypercontractivity to bound $\mathbb{E}|g_k(x)|$ in the proof of Theorem 1 (alternatively one can reason along the lines of the proof of Theorem 4).
A preliminary version of this paper had Theorem 7 only for $d \geq \Omega(n^{1/3})$, and the degree bound was $O(d\sqrt{\log n})$. Jarosław Błasiok pointed out to us how to improve the proof to obtain Theorem 7. The proof in the preliminary version was similar, but rather than performing a case analysis, detected the two cases explicitly with an auxiliary polynomial, which led to $d \geq \Omega(n^{1/3})$. It also used the polynomials for Maj with polynomially-small error, as opposed to constant, which led to the extra $\sqrt{\log n}$ factor. Following these ideas, we also improved the results on the coin problem and h_2. We are very grateful to Jarosław Błasiok for letting us include the improved results!

1 Introduction and our results

The recent "polarizing random walks" paradigm [6, 8, 7, 5] constructs new pseudorandom generators against classes of functions with "bounded Fourier tails." For a function $f : \{0,1\}^n \to \{-1,1\}$ define

$$L_k(f) := \sum_{S \subseteq \{1,2,\ldots,n\} : |S|=k} \left|\hat{f}(S)\right|,$$
$$M_k(f) := \sum_{S \subseteq \{1,2,\ldots,n\} : |S|=k} \hat{f}(S),$$

where $\hat{f}(S) := \mathbb{E}_x f(x)\chi_S(x)$ for $\chi_S(x) := (-1)^{\sum_{i \in S} x_i}$ is the Fourier transform of f [16]. These papers construct pseudorandom generators for functions with small L_k or M_k for several settings of parameters.

In an effort to use this framework to improve the state of pseudorandom generators against *low-degree polynomials* over $\mathbb{F}_2 = \{0,1\}$ [3, 14, 26, 10], several conjectures have been put forth about polynomials. Let p be a degree-d polynomial over $\mathbb{F}_2$ in n variables. For $f := (-1)^p$ it has been conjectured (see [6, 8, 5]):

$$L_k(f) \leq 2^{O(dk)} \qquad \forall k. \tag{1}$$

$$L_2(f) \leq O(d^2), \tag{2}$$

$$M_k(f) \leq 2^{o(dk)+O(k \log\log n)} \qquad \forall k \leq O(\log n). \tag{3}$$

Conjecture (1) would not imply new pseudorandom generators, but would come close to matching the state-of-the-art using this framework – something which was eventually achieved in [5]. But conjectures (2) and (3) would imply new generators, improving on long-standing open problems. One interesting feature of this approach is that, unlike the influential approach by Nisan [15], it is not based on *correlation bounds*. In particular, Conjecture (2) is not known to imply such bounds. Still, correlation bounds where shown to be *sufficient* for this approach in [7].

We show that in fact correlation bounds are also *necessary*. That is, we show that this approach requires proving new correlation bounds for polynomials. This is new information about Conjecture (2). Conjecture (3) was shown in [5] to imply new pseudorandom generators with good dependence on the error, and the latter are known to imply new correlation bounds for a function in NP [26]. We give a direct proof of this implication which yields a function in P (and other parameter improvements). In fact, we show that even weaker versions of the conjectures, such as $M_2 \leq o(\sqrt{n})$ for polynomials of degree $\log_2 n$, already imply new correlation bounds.

Correlation bounds

We say that a function $f : \{0,1\}^n \to \{-1,1\}$ has *δ-advantage (or $(1-\delta)$-error) (probabilistic) degree d* if there is a distribution P on polynomials $p : \{0,1\}^n \to \{0,1\}$ over $\mathbb{F}_2$ of degree d such that for every input x we have $\mathbb{P}[(-1)^{P(x)} = f(x)] \geq \delta$. By Yao's min-max argument [30], a function f has δ-advantage degree d iff for every distribution D on $\{0,1\}^n$ it has δ-advantage degree d under D, meaning there exists a polynomial p over $\mathbb{F}_2$ of degree d such that $\mathbb{P}[(-1)^{p(D)} = f(D)] \geq \delta$. If f has range $\{0,1\}$ instead of $\{-1,1\}$ we use the same notation except $(-1)^{P(x)}$ is replaced simply by $P(x)$.

For two functions f and g from $\{0,1\}^n$ to $\{-1,1\}$ we define their *correlation* under a distribution D by $\mathbb{E}[f(D)g(D)]$, which we note equals $2(\mathbb{P}[f(D) = g(D)] - 1/2)$ and so it is (twice) the distance of $1/2$ from the advantage.

Since the classical works by Razborov and Smolensky [17, 19] the best-available explicit probabilistic-degree lower bound for degree $d \geq \log_2 n$ gives error at best

$$1/2 - \Omega(d/\sqrt{n}) \tag{4}$$

which holds for the Majority function on n bits. In particular, it is consistent with our knowledge that every explicit function has $(1/2 + 1/\sqrt{n})$-advantage degree $\log_2 n$ (while non-constructively there exist functions which do not even have advantage exponentially close to $1/2$ for polynomial degree). For recent progress on functions computable in exponential-time classes see [29].

Proving correlation bounds is a fundamental open problem whose solution stands in the way of progress on a striking variety of fronts, including: circuit lower bounds, multiparty communication complexity, and matrix rigidity. For more on this long-standing challenge and a discussion of the just-mentioned implications, we refer the reader to [25, 27, 29].

The conjectures imply new correlation bounds

We show that bounds on M_k imply new probabilistic-degree lower bounds for an explicit function h_k. We now define h_k and state our results.

Let $g_k : \{0,1\}^n \to \mathbb{Z}$ and $h_k : \{0,1\}^n \to \{-1,1\}$ be defined as

$$g_k(x) := \sum_{S:|S|=k} \chi_S(x),$$

$$h_k(x) := \mathrm{Sign}(g_k(x)),$$

where $\mathrm{Sign}(i) = 1$ if $i > 0$ and -1 otherwise (the value on $i = 0$ is arbitrary).

▶ **Theorem 1.** *Let F be a distribution on functions from $\{0,1\}^n$ to $\{-1,1\}$ such that $\mathbb{P}[F(x) = h_k(x)] \geq 1/2 + \epsilon$ for every x. Then there is an outcome f of F such that $M_k(f) \geq 2\epsilon \cdot e^{-k}\sqrt{\binom{n}{k}}$.*

To illustrate the theorem, consider first $k = 2$, in which case the conclusion becomes $M_2(f) \geq \Omega(\epsilon n)$. This means that showing even just $M_2(p) \leq o(\sqrt{n})$ for every degree-d polynomial requires showing that h_2 does not have $(1/2 + \Omega(1/\sqrt{n}))$-advantage degree d. This would improve the tradeoff (4) mentioned above when $d \geq \log_2 n$. Conjecture (2) implies the stronger bound $M_2(p) \leq O(d^2)$ for every degree-d polynomial p. This would mean that h_2 does not even have $(1/2 + cd^2/n)$-advantage degree d for a constant c, a quadratic improvement on the tradeoff (4). Consider now the case of larger k. Assuming that h_k has $(1/2+\epsilon)$-advantage degree d, and assuming Conjecture (3) and using the bound $\binom{n}{k} \geq (n/k)^k$ we obtain

$$2\epsilon \cdot e^{-k}\left(\frac{n}{k}\right)^{k/2} \leq 2\epsilon \cdot e^{-k}\sqrt{\binom{n}{k}} \leq 2^{o(dk)+O(k\log\log n)}.$$

This implies $\epsilon \leq 2^{k(o(d)+O(\log\log n)-0.5\log_2(n/k))}$. For $k = \log_2 n$ this yields new correlation bounds. Indeed, let $d := \log_2 n$. Then because $o(d)$, $\log\log n$, and $\log(k)$ are all $o(\log n)$ we obtain

$$\epsilon \leq 2^{-\Omega(k\log n)} = 2^{-\Omega(\log^2 n)}$$

which improves on the tradeoff (4).

Proof. Note that for any function f, by linearity of expectation, we have

$$M_k(f) = \mathbb{E}_x f(x)g_k(x).$$

Fix any x and let $\mathbb{P}[F(x) = h_k(x)]$ be equal to $1/2 + \epsilon_x \geq 1/2 + \epsilon$. We can write

$$\mathbb{E}_F[F(x)g_k(x)] = (1/2 + \epsilon_x) \cdot \mathrm{Sign}(g_k(x)) \cdot g_k(x) + (1/2 - \epsilon_x) \cdot (-\mathrm{Sign}(g_k(x))) \cdot g_k(x),$$

holding even if $g_k(x) = 0$. Note that $\mathrm{Sign}(g_k(x)) \cdot g_k(x) = |g_k(x)|$. Hence

$$\mathbb{E}_F[F(x)g_k(x)] = (1/2 + \epsilon_x)|g_k(x)| + (1/2 - \epsilon_x)(-|g_k(x)|) = 2\epsilon_x|g_k(x)| \geq 2\epsilon|g_k(x)|.$$

This gives $\mathbb{E}_{x,F}F(x)g_k(x) \geq \mathbb{E}_x 2\epsilon|g_k(x)|$. In particular, there exists an outcome f such that

$$\mathbb{E}_x f(x)g_k(x) \geq 2\epsilon\mathbb{E}_x|g_k(x)|.$$

There remains to bound $\mathbb{E}_x|g_k(x)|$. We make use of *hypercontractivity* from the analysis of Boolean functions. Because g_k is a polynomial of degree k, by Theorem 9.22 in [16] we have

$$\mathbb{E}_x|g_k(x)| \geq e^{-k}\sqrt{\mathbb{E}_x|g_k(x)|^2}.$$

Now observe that

$$\mathbb{E}_x|g_k(x)|^2 = \mathbb{E}_x \sum_{S,T:|S|=|T|=k} \chi_S(x)\chi_T(x) = \mathbb{E}_x \sum_{S,T:|S|=|T|=k} \chi_{S\oplus T}(x) = \binom{n}{k},$$

where $\oplus$ is symmetric difference. The last equality holds because the terms where $S \neq T$ have expectation zero, and the others have expectation one. The result follows. ◀

A natural question is whether Theorem 1 holds even for functions that correlate with h_k under the uniform distribution. We show that it does not.

▶ **Theorem 2.** *Let n be a power of 2. For any integer s between 0 and $\sqrt{n}/2$ there is a function $f : \{0,1\}^n \to \{-1,1\}$ such that $\mathbb{P}[f(x) = h_2(x)] \geq 1/2 + \Omega(s/\sqrt{n})$ but $M_2(f) \leq O(s^2)$.*

To get a sense of the parameters let $\mathbb{P}[f(x) = h_2(x)] = 1/2 + \epsilon$. Then $M_2(f)$ is only $O(\epsilon^2 n)$ as opposed to $\Omega(\epsilon n)$ in Theorem 1. In particular, if $s = O(1)$ and $\epsilon = \Theta(1/\sqrt{n})$ we get $M_2(f) = O(1)$ as opposed to $\Omega(\sqrt{n})$ in Theorem 1.

We have shown that understanding the probabilistic degree of the functions h_k is also important for the feasibility of recent approaches to pseudorandom generators against polynomials. We obtain new bounds on the probabilistic degree of the functions h_k which however fall short of resolving whether the correlation bounds in the conclusion of Theorem 1 hold or not. We begin with studying h_1 which is essentially the majority function Maj. The results are of independent interest, and a natural step to tackle h_k for larger k. Indeed, below we use techniques developed for Maj to give new results on h_2.

We point out that the probabilistic degree tradeoff of Majority is not known. Given the tremendous interest in this function, this may come as a surprise. One might be tempted to think that Smolensky's tradeoff (4) is tight. We can show that it is indeed tight *under the uniform distribution.*

▶ **Theorem 3.** *Majority has $(1/2 + \Omega(d/\sqrt{n}))$-advantage degree d under the uniform distribution.*

Recall this means that there are degree-d polynomials p over $\mathbb{F}_2$ such that $\mathbb{P}_x[p(x) = \mathrm{Maj}(x)] \geq 1/2 + \Omega(d/\sqrt{n})$, where x is uniform in $\{0,1\}^n$. Such a result was only known for $d = O(1)$ or $d = \Omega(\sqrt{n})$, see [25].

However, there are harder distributions. We beat Smolensky's bound for degree one. While such polynomials are simple, in light of Theorem 3 this result already requires a non-uniform distribution.

▶ **Theorem 4.** *Majority does not have $(1/2 + c/n)$-advantage degree one, for some constant c. This bound is tight up to the value of c.*

We now turn to constructions of probabilistic polynomials for majority. This problem is related to the so-called *coin problem,* defined next.

▶ **Definition 5.** *For $\delta \in [0,1]$ we denote by N_δ^t the distribution over $\{0,1\}^t$ where the bits are i.i.d. and each comes up 1 with probability δ. We say that a distribution F on boolean functions on t bits $(1/2+\alpha)$*-solves the δ-coin problem with advantage α *if the following is true:*
(1) $\mathbb{P}[F(N_\delta^t) = 1] \geq 1/2 + \alpha$; *and*
(2) $\mathbb{P}[F(N_{1-\delta}^t) = 0] \geq 1/2 + \alpha$.

The study of the coin problem for low-degree polynomials goes back to [18] (see also the thesis [24]) and has been the subject of several recent works including [13, 11, 21]. This problem has also been studied in a variety of other models; the terminology "coin problem" was coined in [4].

However, these works consider large advantage $\alpha = \Omega(1)$. By contrast, we are interested in the setting where α is close to 0. We give tight bounds in this setting, showing that with degree d the best we can do is to boost the bias by d.

▶ **Theorem 6.** *There is a distribution on polynomials of degree $O(d)$ that $(1/2 + d\epsilon)$-solves the $(1/2+\epsilon)$-coin problem, whenever $d\epsilon < c$ for an absolute constant c. Moreover, this is tight up to the constant in the $O(.)$.*

Computing Majority on n bits for odd n can be randomly reduced to solving the $(1/2+1/n)$-coin problem, simply by selecting uniform bits from the input. Hence, Theorem 6 shows that Majority has $(1/2+d/n)$-advantage degree $\leq O(d)$. We improve the advantage to $\Omega(d^2/n)$, and conjecture that this is tight.

▶ **Theorem 7.** *Majority on n bits, for odd n, has $(1/2 + d^2/n)$-advantage degree $\leq O(d)$.*

▶ **Conjecture 8.** *Theorem 7 is tight. A "hard" distribution can be uniform on the inputs of Hamming weights $n/2 + 2^{\ell-1}$ and $n/2 - 2^{\ell-1}$ where $d < 2^\ell$.*

To understand the choice of the hard distribution, recall that *symmetric* polynomials of degree $d < 2^\ell$ only depend on the weight of the input modulo 2^ℓ (see Lemma 11). For example, for $\ell = 1$ symmetric polynomials of degree $1 < 2$ only depend on the input weight modulo 2. The two Hamming weights in the conjecture are congruent modulo 2^ℓ; hence any symmetric polynomial of degree $< 2^\ell$ has correlation zero.

Finally, we turn to h_2. One can reduce h_2 to a majority on $\binom{n}{2}$ bits, and then apply Theorem 7 to obtain advantage $1/2 + \Omega(d^2/n^2)$. We improve this to $1/2 + \Omega(d^2/n^{3/2})$, under a condition on n.

▶ **Theorem 9.** *Let ℓ be the smallest integer such that $d \leq 2^\ell$. Suppose that the remainder of $\sqrt{n}$ divided by $2^{\ell+100}$ is not in $[0, 2d] \cup [2^{\ell+100} - 2d, 2^{\ell+100}]$.*
Then h_2 has $(1/2 + d^2/n^{3/2})$-advantage degree $O(d)$.

This result is not strong enough to disprove Conjecture (2). For that we require advantage $1/2 + \omega(d^2/n)$.

The rest of the paper is organized as follows. After some preliminaries in Section 2 we prove the statements in the same order in which we discussed them, except that the proof of Theorem 2 is in Section 8.

2 Preliminaries

In this section we collect several results which are used in later proofs.

The following lemma shows that the majority of several i.i.d. Bernoulli random variables increases their bias, even in the regime where the bias is very small to start with.

▶ **Lemma 10.** $\mathbb{P}[\mathrm{Maj}(N^t_{1/2+\alpha}) = 1] \geq 1/2 + \Omega(\alpha\sqrt{t})$, *whenever* $\sqrt{t}\alpha < c$ *for an absolute constant* c.

We are not aware of a source from which this result can be easily extracted, so we provide a proof. But Jarosław Błasiok let us know that this lemma appears as Lemma 8 in [23].

Proof. We prove $\mathbb{P}[\mathrm{Maj}(N^t_{1/2+\alpha}) = 1] - \mathbb{P}[\mathrm{Maj}(N^t_{1/2+\alpha}) = 0] \geq \Omega(\alpha\sqrt{t})$. The former difference can be written as

$$\sum_{i=1/2}^{t/2} \binom{t}{t/2+i} \left((1/2+\alpha)^{t/2+i}(1/2-\alpha)^{t/2-i} - (1/2-\alpha)^{t/2+i}(1/2+\alpha)^{t/2-i}\right),$$

where the sum is for $i = 1/2, 1+1/2, 2+1/2, \ldots, t/2$.

Collecting a 2^t factor and writing z for 2α this equals

$$2^{-t} \sum_{i=1/2}^{t/2} \binom{t}{t/2+i} \left((1+z)^{t/2+i}(1-z)^{t/2-i} - (1-z)^{t/2+i}(1+z)^{t/2-i}\right).$$

Further collecting $(1-z)^{t/2}(1+z)^{t/2} = (1-z^2)^{t/2}$ we rewrite it as

$$2^{-t}(1-z^2)^{t/2} \sum_{i=1/2}^{t/2} \binom{t}{t/2+i} \left(\left(\frac{1+z}{1-z}\right)^i - \left(\frac{1-z}{1+z}\right)^i\right).$$

Note that $\left(\frac{1+z}{1-z}\right) > 1$ and so $\left(\frac{1+z}{1-z}\right)^i - \left(\frac{1-z}{1+z}\right)^i$ is positive and increasing with i. Hence for any s we can bound below the expression by

$$2^{-t}(1-z^2)^{t/2} \sum_{i=s}^{t/2} \binom{t}{t/2+i} \left(\left(\frac{1+z}{1-z}\right)^s - \left(\frac{1-z}{1+z}\right)^s\right).$$

Moreover, let us write

$$\left(\frac{1+z}{1-z}\right)^s - \left(\frac{1-z}{1+z}\right)^s = (1+x)^s - (1-y)^s$$

where $x = 2z/(1-z)$ and $y = 2z/(1+z)$. We bound below the right-hand side by

$$1 + xs - e^{-ys} \geq 1 + xs - (1 - ys + (ys)^2) = s(x+y) - y^2s^2.$$

We pick $s = \sqrt{t}/100 + 1/2$. The above expression is $\Omega(\sqrt{t}\alpha)$ as long as $\sqrt{t}\alpha = \Theta(st)$ is sufficiently small. Moreover, we have

$$2^{-t}(1-z^2)^{t/2} \sum_{i=s}^{t/2} \binom{t}{t/2+i} \geq \Omega(1).$$

This holds because $(1-z^2)^{t/2} \geq \Omega(1)$ and the sum of binomial coefficients is also $\Omega(2^{-t})$ using Stirling's approximation to the binomial coefficient. ◀

We use the following characterization of symmetric polynomials which is Theorem 2.4 in [2] and follows from Lucas' theorem.

▶ **Lemma 11.** *Let $f : \{0,1\}^n \to \{0,1\}$ be a symmetric function that only depends on the input Hamming weight modulo 2^ℓ. Then f is computable by a symmetric $\mathbb{F}_2$ polynomial of degree $< 2^\ell$. Conversely, any function $f : \{0,1\}^n \to \{0,1\}$ computable by a symmetric $\mathbb{F}_2$ polynomial of degree $< 2^\ell$ only depends on the input Hamming weight modulo 2^ℓ.*

Then we need constructions of probabilistic polynomials for symmetric functions, obtained in [1]. The bounds in the earlier paper [20] would also suffice for the main points in this paper. See also [22] for a recent characterization.

▶ **Lemma 12** ([1])**.** *Let $f : \{0,1\}^n \to \{0,1\}$ be symmetric. Then f has $(1-\epsilon)$-advantage degree $O(\sqrt{n \log(1/\epsilon)})$, for any ϵ.*

3 Proof of Theorem 3

The main proof is for odd n. If n is even we can use the polynomial $p'(x_0, x_1, \ldots, x_{n-1}) := p(x_0, x_1, \ldots, x_{n-2})(1 - x_{n-1})$ where p is the polynomial with the highest correlation γ with majority on input length $n-1$. The correlation of p' with majority is $> \gamma/2$.

We now proceed with the main proof. We can assume without loss of generality that d is a power of 2 and $\leq 0.1\sqrt{n}$. The polynomial witnessing the correlation will be *symmetric.* For a symmetric function $f : \{0,1\}^n \to \{0,1\}$ write $f_w : \{0,1,\ldots,n\} \to \{0,1\}$ for $f(x) = f_w(|x|)$ where $|x|$ is the Hamming weight of x. The correlation between a symmetric polynomial p and $(-1)^{\mathrm{Maj}}$ can be written as

$$2^{-n} \sum_{i=0}^{n} \binom{n}{i} (-1)^{p_w(i)} (-1)^{Maj_w(i)}.$$

To construct p we use Lemma 11 for $\ell = \log_2(2d)$. That shows that for any $f_w : \{0,1,\ldots,n\} \to \{0,1\}$ that depends only on the input modulo 2^ℓ there is a symmetric polynomial $p : \{0,1\}^n \to \{0,1\}$ of degree 2^ℓ such that $p_w = f_w$.

The definition of f_w and hence p is as follows. Define Block i to be the $2d$ integers $2di + 0, 2di + 1, \ldots, 2di + 2d - 1$. Let i^* be the smallest i such that Block i contains an integer larger than $n/2$. Let t be the number of integers less than $n/2$ in Block i^*. (If $n+1$ is a power of 2 we have $t = 0$, and below there is no residual chunk.) Define f_w to be 1 on the smallest t inputs, 0 on the next t, 0 on the next $d - t$, and finally 1 on the next $d - t$. Here's an example for $n = 17, d = 2, t = 1, i^* = 2$; the last row shows the division in blocks:

weight	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
	-	-	-	-	-	-	-	-	-	+	+	+	+	+	+	+	+	+
$(-1)^{p_w}$	-	+	+	-	-	+	+	-	-	+	+	-	-	+	+	-	-	+

Note that p_w is by construction anti-symmetric in the sense, different from above, that: $p_w(i) = 1 - p_w(n-i)$. The same is true for Maj_w. Therefore $g(i) := (-1)^{p_w(i)}(-1)^{Maj_w(i)}$ is symmetric, that is $g(i) = g(n-i)$. Hence we only need to consider the bigger half of the Hamming weights. Majority is always 1, and so we can rewrite the correlation as

$$2^{-n} \cdot 2 \cdot \sum_{i=0}^{(n-1)/2} \binom{n}{(n+1)/2+i} (-1)^{p_w((n+1)/2+i)}.$$

Enumerate the above binomial coefficients starting from the biggest one for $i = 0$. The term $(-1)^{p_w((n+1)/2+i)}$ will be $+1$ on the first $t + (d-t) = d$, then -1 on the next d, then again $+1$ on the next d, and so on. We group the coefficients in chunks of length $2d$; in each chunk

the term is $+1$ for the first half and -1 for the second half. The number of coefficients is $(n+1)/2$. Hence we have $\lfloor (n+1)/4d \rfloor$ chunks, plus a residual truncated chunk of length $\ell < 2d$.

Hence we can write the correlation as follows.

$$2^{-n} \cdot 2 \cdot \sum_{i=0}^{\lfloor (n+1)/4d \rfloor - 1} \sum_{j=0}^{d-1} \left(\binom{n}{(n+1)/2 + 2di + j} - \binom{n}{(n+1)/2 + 2di + j + d} \right) + 2^{-n} \cdot 2 \cdot \sum_{i=0}^{\ell-1} \binom{n}{n-i} (-1)^{p_w((n+1)/2+i)}.$$

By, say, a Chernoff bound the absolute value of the latter summand $+2^{-n} \cdots$ is at most $2^{-\Omega(n)}$, using that $\ell < 2d = O(\sqrt{n})$. Now consider the first summand. Because the binomials are decreasing in size, each difference is positive. Hence we obtain a lower bound if we reduce the range of i. We reduce it to $\lfloor \sqrt{n}/d \rfloor$. So the correlation is at least

$$2^{-n} \cdot 2 \cdot \sum_{i=0}^{\lfloor \sqrt{n}/d \rfloor} \sum_{j=0}^{d-1} \left(\binom{n}{(n+1)/2 + 2di + j} - \binom{n}{(n+1)/2 + 2di + j + d} \right) - 2^{-\Omega(n)}.$$

The next lemma bounds below the difference of two such binomial coefficients.

▶ **Lemma 13.** *For $s \leq 4\sqrt{n}$ and $d \leq 0.1\sqrt{n}$ we have: $2^{-n} \left(\binom{n}{n/2+s} - \binom{n}{n/2+s+d} \right) \geq \Omega(sd/n^{3/2})$.*

We apply the lemma with $s = 1/2 + 2di + j$ which note is $\leq 1/2 + 2\sqrt{n} + 0.1\sqrt{n} \leq 3\sqrt{n}$. The correlation is at least

$$\sum_{i=0}^{\lfloor \sqrt{n}/d \rfloor} \sum_{j=0}^{d-1} \Omega((1/2 + 2di + j)d/n^{3/2}) - 2^{-\Omega(n)} \geq \sum_{k=0}^{\Omega(\sqrt{n})} \Omega(kd/n^{3/2}) - 2^{-\Omega(n)} \geq \Omega(d/\sqrt{n}).$$

To justify the first inequality we use $1/2 + 2di + j \geq di + j$ and then do the change of variable $k = di + j$. For the second we use that the sum of all k up to $\Omega(\sqrt{n})$ is $\Omega(n)$. This concludes the proof except for the lemma.

Proof of lemma

We have

$$\begin{aligned} &\binom{n}{n/2+s} - \binom{n}{n/2+s+d} \\ =&\frac{n!}{(n/2+s)!(n/2-s)!} - \frac{n!}{(n/2+s+d)!(n/2-s-d)!} \\ =&\frac{n!}{(n/2+s)!(n/2-s)!} \left[1 - \frac{(n/2-s)(n/2-s-1)\cdots(n/2-s-d+1)}{(n/2+s+d)(n/2+s+d-1)\cdots(n/2+s+1)} \right]. \end{aligned}$$

The ratio inside the square bracket is at most

$$\frac{(n/2-s)^d}{(n/2)^d} = (1-2s/n)^d \leq e^{-2sd/n} \leq 1 - sd/n,$$

where the last inequality holds because $2sd/n \leq 1$.

The binomial coefficient outside of the square bracket is

$$\binom{n}{n/2+s} \geq \frac{2^{nh(1/2+s/n)}}{\sqrt{8n(1/2+s/n)(1/2-sn)}} \geq \Omega\left(\frac{2^{n(1-O(s^2/n^2))}}{\sqrt{n}}\right) \geq \Omega\left(\frac{2^n}{\sqrt{n}}\right).$$

Here h is the binary entropy function, and the first inequality can be found as Lemma 17.5.1 in [9]. The second and third inequalities follow from the approximation $h(1/2+x) \geq 1-4x^2$, valid for every x, and $s = O(\sqrt{n})$.

The lemma follows by combining the two bounds.

4 Proof of Theorem 4

First let us discuss tightness. To show tightness for odd n we simply output a uniformly selected bit. For even n this works for all inputs except those of Hamming weight $= n/2$. To fix this, we modify the distribution on polynomials to equal 1 with probability $1/n$. On input of weight $= n/2$ we get the right value with probability $1/n + (1-1/n)(1/2) \geq 1/2+\Omega(1/n)$. On inputs of Hamming weight $\neq n/2$ we also get the right value with probability $(1-1/n)(1/2+1/n) \geq 1/2+\Omega(1/n)$.

We now move to negative results. First we note that we can reduce the case of even n to that of odd n: simply append a bit whose value is that of majority on balanced inputs. This does not change the value of majority, and has negligible effect on the advantage. Hence it suffices to prove a negative result for even n, and we do so in the rest of this section.

We select as the hard distribution the distribution D which is uniform on inputs of Hamming weight $n/2+1$ and $n/2-1$. Our goal is to show that for every fixed degree-one polynomial f we have $\mathbb{P}[f(D) = \mathrm{Maj}(D)] \leq 1/2 + O(1/n)$. Using *generating functions* we obtain a proof which is nearly calculation-free, requiring only elementary bounds on binomials. Let $m = n/2$ and $f = x_1 + x_2 + \cdots + x_k$ for a parameter k. Let

$$b(n,m,k) = \sum_{i=0}^{k} (-1)^i \binom{m}{i}\binom{n-m}{k-i}.$$

Note that $b(n,m,k)/\binom{n}{k}$ is the probability that a uniform set of size k has odd intersection with a fixed set of size m, minus the probability that it has even intersection. By the definition of D and f one obtains that $|\mathbb{P}[f(D) = \mathrm{Maj}(D)] - 1/2|$ is at most big-Oh of

$$\alpha(n, n/2-1, k) := \left|\frac{1}{\binom{n}{k}}(b(n,n/2-1,k) - b(n,n/2+1,k))\right|.$$

Note that we can assume that f has no constant term because we are taking absolute values in the expression $|\mathbb{P}[f(D) = \mathrm{Maj}(D)] - 1/2|$.

First we use generating functions to obtain a closed form for $b(n,m,k)$. Recall the generating functions (see e.g. [12] for background on this technique)

$$(1+z)^n = \sum_{i\geq 0}\binom{n}{i} z^i,$$

$$(1-z)^n = \sum_{i\geq 0}\binom{n}{i}(-1)^i z^i.$$

We have

$$(1-z)^m(1+z)^{n-m} = \sum_{i\geq 0, j\geq 0}\binom{m}{i}\binom{n-m}{j}(-1)^i z^{i+j} = \sum_{k\geq 0} b(n,m,k) z^k.$$

If $m = n/2 - t$ the left-hand side can be written as

$$\begin{aligned}&(1-z)^{n/2-t}(1+z)^{n/2-t}(1+z)^{2t}\\ =&(1-z^2)^{n/2-t}(1+z)^{2t}\\ =&\sum_{i\geq 0}(-1)^i\binom{n/2-t}{i}z^{2i}(1+z)^{2t}.\end{aligned}$$

Similarly, if $m = n/2 + t$ then it can be written as

$$\begin{aligned}&(1-z)^{n/2-t}(1+z)^{n/2-t}(1-z)^{2t}\\ =&\sum_{i\geq 0}(-1)^i\binom{n/2-t}{i}z^{2i}(1-z)^{2t}.\end{aligned}$$

Specializing to $t = 1$ we obtain

$$\begin{aligned}&\sum_{k\geq 0}(b(n,n/2-1,k)-b(n,n/2+1,k))z^k\\ =&\sum_{i\geq 0}(-1)^i\binom{n/2-1}{i}z^{2i}\left((1+z)^2-(1-z)^2\right)\\ =&\sum_{i\geq 0}(-1)^i\binom{n/2-1}{i}z^{2i}\cdot 4z\\ =&4\sum_{i\geq 0}(-1)^i\binom{n/2-1}{i}z^{2i+1}.\end{aligned}$$

Equating coefficients of z^k yields

$$b(n,n/2-1,k)-b(n,n/2+1,k) = 4(-1)^{(k-1)/2}\binom{n/2-1}{(k-1)/2}$$

if k is odd, otherwise the left-hand side is zero.

Hence we get

$$\alpha = 4\binom{n/2-1}{(k-1)/2}\Big/\binom{n}{k}$$

if k is odd, and $\alpha = 0$ if k is even.

There remains to bound the right-hand side. First, we can assume that $k \leq n/2$ because replacing k with $n-k$ does not change the value of α. If $k = 0, 1$ we readily have $\alpha = O(1/n)$, using that n is even. Otherwise we can use the bounds

$$(n/k)^k \leq \binom{n}{k} \leq (en/k)^k$$

to again show $\alpha = O(1/n)$. We have

$$\alpha \leq 4\left(\frac{n}{(k-1)}\right)^{(k-1)/2}\left(\frac{k}{n}\right)^k = 4\sqrt{\frac{k}{n}}\left(\sqrt{\frac{1}{k-1}}\cdot\frac{k}{\sqrt{n}}\right)^k.$$

We can conclude by noticing that if $k \leq 100\log_2 n$ then this is at most $\operatorname{poly}\log n/n^{1.5} \leq O(1/n)$, using $k \geq 2$; while if $k \geq 100\log_2 n$ using that $k \leq n/2$ and $k - 1 \geq 0.99k$ we have

$$\alpha \leq O(1)\cdot\left(\frac{\sqrt{k}}{\sqrt{0.99n}}\right)^k \leq O(1)(\sqrt{0.5/0.99})^k \leq O(1)(3/4)^k \leq 1/n.$$

5 Proof of Theorem 6

The theorem follows immediately from the following more general lemma, which we will also use later.

▶ **Lemma 14.** *There is a distribution P on polynomials on $s = O(d^2)$ bits of degree $O(d)$ such that for every $\epsilon \in [-1/2, 1/2]$, $\epsilon \leq 1/d$, we have $\mathbb{P}[P(N^s_{1/2+\epsilon}) = \mathrm{Sign}(\epsilon)] \geq 1/2 + \Omega(d\epsilon)$.*

Proof. Let $P'' : \{0,1\}^s \to \{0,1\}$ be the probabilistic polynomial of degree $O(d)$ from Theorem 12 which computes Maj on every input with probability 0.99, with input length $s = O(d^2)$ which is assumed to be odd without loss of generality.

We modify P'' so that the probability that it makes a mistake on input x only depends on $||x| - n/2|$. That is, it is the same on every two inputs of weights $n/2 + i$ and $n/2 - i$. First, let P' pick a random permutation of the input bits, and then apply P''. The probability that P' makes a mistake only depends on $|x|$. Second, define P that on input x tosses a coin, and if it is heads it outputs $P'(x)$, and if it is tails it complements x to obtain $\neg x$, runs $P'(\neg x)$, and flips the answer. Because $\mathrm{Maj}(x) = 1 - \mathrm{Maj}(\neg x)$ on inputs of odd length, the probability that it makes a mistake on input x only depends on $||x| - n/2|$

For an input y of Hamming weight i, denote

$$m_i := \mathbb{P}[P(y) \neq \mathrm{Maj}(y)].$$

We conclude the proof assuming $\epsilon \geq 0$. This will cover the case $\epsilon < 0$ as well, since $\mathbb{P}[P(N^s_{1/2-\epsilon}) = 0] = \mathbb{P}[P(N^s_{1/2+\epsilon}) = 1]$.

Let $p_i := \mathbb{P}[|N^s_{1/2+\epsilon}| = i]$. We can write

$$\begin{aligned}
\mathbb{P}[P(N^s_{1/2+\epsilon}) = 1] &= \sum_{i>s/2} p_i \cdot (1 - m_i) + \sum_{i<s/2} p_i \cdot m_i \\
&= \sum_{i>s/2} (p_i \cdot (1 - m_i) + p_{s-i} \cdot m_{s-i}) \\
&= \sum_{i>s/2} (p_i - m_i(p_i - p_{s-i}))\,.
\end{aligned}$$

Where the last equality holds because by construction $m_i = m_{n-i}$ for every i.

Because $\epsilon \geq 0$ and $i > s/2$, the factor $(p_i - p_{s-i})$ is positive. Hence we bound the sum below if we replace m_i with its maximum value 0.01, obtaining

$$\sum_{i>s/2} (p_i - 0.01(p_i - p_{s-i})) = \mathbb{P}[\mathrm{Maj}(N^s_{1/2+\epsilon}) = 1](1 - 0.01) + 0.01 \cdot \mathbb{P}[\mathrm{Maj}(N^s_{1/2+\epsilon}) = 0].$$

Writing $\mathbb{P}[\mathrm{Maj}(N^s_{1/2+\epsilon}) = 0] = 1 - \mathbb{P}[\mathrm{Maj}(N^s_{1/2+\epsilon}) = 1]$ this becomes

$$\mathbb{P}[\mathrm{Maj}(N^s_{1/2+\epsilon}) = 1](1 - 2 \cdot 0.01) + 0.01.$$

By Lemma 10, $\mathbb{P}[\mathrm{Maj}(N^s_{1/2+\epsilon}) = 1] \geq 1/2 + \Omega(d\epsilon)$. Hence we conclude

$$\mathbb{P}[P(N^s_{1/2+\epsilon}) = 1] \geq (1/2 + \Omega(d\epsilon))(1 - 2 \cdot 0.01) + 0.01 = 1/2 + \Omega(d\epsilon).$$ ◀

At first sight, it may seem suspicious that we can tolerate constant error in the polynomials for majority. Some intuition why this might be OK follows. If $\mathbb{P}[P(N^s_{1/2+\epsilon}) = 1]$ is close to 1, constant error won't bother us, since we are only aiming for advantage close to 1/2.

On the other hand, if that probability is close to 1/2, the loss will recouped thanks to the symmetrization. That is, mistakes will be made on $N^s_{1/2-\epsilon}$ with the same probability, boosting the correctness.

To prove that this result is tight, suppose there is a distribution on degree-d polynomials that solves the $(1/2+\epsilon)$-coin problem with advantage $1/2+\alpha$. If we sample $O(1/\alpha)^2$ times independently these polynomials, and compute the majority, a Chernoff bound shows that we obtain advantage 0.99. By Lemma 12 the majority computation can be done with error 1/100 by a probabilistic polynomial of degree $O(1/\alpha)$. Composing this with the degree-d polynomial we obtain a probabilistic polynomial of degree $O(d/\alpha)$ which solves the $(1/2+\epsilon)$-coin problem with advantage 0.98. By averaging we can fix the polynomial and still maintain advantage 0.96. Now we can appeal to a result proved in [13] which shows that any such polynomial has degree $\Omega(1/\epsilon)$. Hence, $d/\alpha \geq \Omega(1/\epsilon)$. In other words, $\alpha \leq O(d\epsilon)$, as desired.

6 Proof of Theorem 7

By Yao's argument mentioned in the introduction, it suffices to show that for every distribution Z on $\{0,1\}^n$ there exists a polynomial which computes Maj correctly with probability $1/2+\Omega(d^2/n)$ over Z. By averaging, it suffices to give, for any Z, a distribution $P=P(Z)$ on polynomials that computes Maj correctly with the same probability over both the input drawn from Z and P. Our polynomials will depend only on the Hamming weight $|Z|$ of Z.

Case: $\mathbb{P}[||\boldsymbol{Z}| - \boldsymbol{n/2}| \geq \boldsymbol{d}] \geq \mathbf{0.01}$

Let $M : \{0,1\}^{O(d^2)} \to \{0,1\}$ be the probabilistic polynomial of degree $O(d)$ from Lemma 14. Define $P(x)$ to compute M on an odd number $s := O(d^2)$ bits y selected uniformly at random from x. We first analyze the performance of this polynomial on any fixed input x of Hamming weight $w = n(1/2+\epsilon)$. Note that y has the distribution $N^s_{1/2+\epsilon}$

We have

$$\mathbb{P}[P(x) = \mathrm{Maj}(x)] = \mathbb{P}[M(N^s_{1/2+\epsilon}) = \mathrm{Sign}(\epsilon)] \geq 1/2 + \Omega(d\epsilon),$$

By Lemma 14.

Now we use the assumption on Z. With probability $\Omega(1)$, we have $|\epsilon| \geq d/n$, in which case the probability is $\geq 1/2+\Omega(d^2/n)$. In every other case, the probability is at least 1/2. Overall, $\mathbb{P}[P(Z) = \mathrm{Maj}(Z)] \geq 1/2+\Omega(d^2/n)$, concluding this case.

Case: $\mathbb{P}[||\boldsymbol{Z}| - \boldsymbol{n/2}| \leq \boldsymbol{d}] \geq \mathbf{0.99}$

Let P be the polynomial of degree $O(d)$ from Lemma 11 that computes Maj on every input whose Hamming weight w has distance $\leq d$ from $n/2$. In this case, we have $\mathbb{P}[P(Z) = \mathrm{Maj}(Z)] \geq \mathbb{P}[||Z| - n/2| \leq d] \geq 0.99$.

7 Proof of Theorem 9

As in the proof of Theorem 7, it suffices to show that for every distribution Z on $\{0,1\}^n$ there exists a distribution on polynomials which computes h_2 well over Z. Our polynomials will again depend only on the Hamming weight $|Z|$ of Z.

From the definition of g_2 we have that on inputs x with $n/2+t$ zeroes and $n/2-t$ ones we have

$$g_2(x) = 2t^2 - n/2.$$

As a function of t, this is a parabola which roots at $t = \pm\sqrt{n/4} = \pm n \cdot r$ where $r := 1/\sqrt{4n}$. Let $L := [-nr-d, -nr+d] \bigcap \mathbb{Z}$ and $R := [nr-d, nr+d] \bigcap \mathbb{Z}$ be the integers at distance $\leq d$ from either root.

Case: $\mathbb{P}[|Z| - n/2 \in L \bigcup R|] \geq 0.99$

In this case we use polynomials of degree $O(d)$ from Lemma 11 to compute h_2 correctly on $L \bigcup R$. This definition is possible if the elements in L and R are not congruent modulo $2^{\ell+100}$. That is, we require that for every x, y of absolute value at most d the values $-nr+x$ and $nr+y$ are not congruent modulo $2^{\ell+100}$. For this it suffices that the remainder of $2nr = \sqrt{n}$ divided by $2^{\ell+100}$ is not in $[0, 2d] \cup [2^{\ell+100} - 2d, 2^{\ell+100}]$, given by assumption.

Case: $\mathbb{P}[|Z| - n/2 \in L \bigcup R|] < 0.01$

Consider the following process. With probability $1/(1+4r^2)$ pick two uniform elements from the input and output their XOR; otherwise output zero. On any input with weight $1/2+\alpha$ the probability the process outputs 1 is

$$1/2 + \epsilon := \frac{1/2 + 2\alpha^2}{1+4r^2} = \frac{1/2 + 2(r+\alpha-r)^2}{1+4r^2} = 1/2 + \frac{2\left(\alpha^2 - r^2\right)}{1+4r^2}.$$

Note $\epsilon = 0$ exactly when $\alpha = \pm r$, and $\epsilon < 0$ exactly when α is between these two roots.

Now repeat the process s times to generate $N^s_{1/2+\epsilon}$, and run the polynomial from Lemma 14 on them.

On any input, we compute correctly with probability $\geq 1/2$.

Assume now the input weight is not in $L \bigcup R$. Let $c := 2/(1+4r^2)$.

If $|\alpha| \geq r + d/n$ then $\epsilon \geq c(d^2/n^2 + 2rd/n) = \Omega(rd/n)$.

If $|\alpha| \leq r - d/n$ then $\epsilon \leq c(d^2/n^2 - 2rd/n) = -\Omega(rd/n)$.

In either case, by Lemma 14 we compute h_2 correctly with probability $1/2 + d \cdot \Omega(rd/n) = 1/2 + \Omega(d^2/n^{3/2})$.

8 Proof of Theorem 2

We essentially define f to have correlation zero with h_2 on every Hamming weight, except for s Hamming weights where the value of g_2 is as small as possible. Let $M := \{n/2 + \sqrt{n}/2, n/2 + \sqrt{n}/2 - 1, \ldots, n/2 + \sqrt{n}/2 - s + 1\}$ and let Z_i be the inputs with i zeroes. For $x \in Z_i$ and $i \in M$ let $f(x) = h_2(x) = -1$. For $x \in Z_0$ let, say, $f(x) = 1$ and for $x \in Z_n$ let $f(x) = -1$. For any other Z_i, divide the inputs in Z_i in two equals parts, which is possible by Lucas' theorem because n is a power of 2. Let f be 1 on one part and -1 on the other.

Consider $\mathbb{E}_x[f(x)h_2(x)]$. We have $\mathbb{E}_x[f(x)h_2(x)|x \in Z_0 \cup Z_n] = 0$, and $\mathbb{E}_x[f(x)h_2(x)|x \in Z_i] = 0$ if $i \notin M$ and $i \neq 0$ and $i \neq n$, by definition. Otherwise the expectation is 1. Hence $\mathbb{E}_x[f(x)h(x)]$ is the probability that $x \in Z_i$ for some $i \in M$. Assuming $s \leq \sqrt{n}/2$ this probability is $\geq \Omega(s) \cdot \mathbb{P}[x \in Z_{n/2+\sqrt{n}/2}]$. The latter probability is $\Omega(1/\sqrt{n})$ using the standard bound $\binom{n}{n/2+\sqrt{n}/2} = \Theta(2^n/\sqrt{n})$ which can be verified using Stirling's approximation. Hence $\mathbb{E}_x[f(x)h_2(x)] \geq \Omega(s/\sqrt{n})$, and so $\mathbb{P}[f(x) = h_2(x)] \geq 1/2 + \Omega(s/\sqrt{n})$.

Now consider $\mathbb{E}_x[f(x)g_2(x)]$. Again, this is zero unless the number of zeroes of x lies in M. Note that $g_2(x) = 2t^2 - n/2$ on inputs in $Z_{n/2+t}$. The maximum value of $|g_2(x)|$ for inputs with weights in M is for $t = \sqrt{n}/2 - s + 1$ which yields value $|2(\sqrt{n}/2 - s + 1)^2 - n/2| = |2(-s+1)^2 + (-s+1)\sqrt{n}| \leq O(s^2 + s\sqrt{n})$. For $s \leq \sqrt{n}/2$ the latter is $O(s\sqrt{n})$. The chance that the number of zeroes of x lies in M is $\Theta(s/\sqrt{n})$ as noted before. Hence we get $M_2(f) \leq O(s\sqrt{n} \cdot s/\sqrt{n}) \leq O(s^2)$.

References

1 Josh Alman and Ryan Williams. Probabilistic polynomials and hamming nearest neighbors. In *IEEE Symp. on Foundations of Computer Science (FOCS)*, pages 136–150, 2015. doi:10.1109/FOCS.2015.18.

2 Nayantara Bhatnagar, Parikshit Gopalan, and Richard J. Lipton. Symmetric polynomials over Z_m and simultaneous communication protocols. *J. of Computer and System Sciences*, 72(2):252–285, 2006.

3 Andrej Bogdanov and Emanuele Viola. Pseudorandom bits for polynomials. *SIAM J. on Computing*, 39(6):2464–2486, 2010.

4 Joshua Brody and Elad Verbin. The coin problem, and pseudorandomness for branching programs. In *51th IEEE Symp. on Foundations of Computer Science (FOCS)*, 2010.

5 Eshan Chattopadhyay, Jason Gaitonde, Chin Ho Lee, Shachar Lovett, and Abhishek Shetty. Fractional pseudorandom generators from any fourier level. *CoRR*, abs/2008.01316, 2020. arXiv:2008.01316.

6 Eshan Chattopadhyay, Pooya Hatami, Kaave Hosseini, and Shachar Lovett. Pseudorandom generators from polarizing random walks. In *CCC*, volume 102 of *LIPIcs*, pages 1:1–1:21. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2018.

7 Eshan Chattopadhyay, Pooya Hatami, Kaave Hosseini, Shachar Lovett, and David Zuckerman. XOR lemmas for resilient functions against polynomials. In Konstantin Makarychev, Yury Makarychev, Madhur Tulsiani, Gautam Kamath, and Julia Chuzhoy, editors, *ACM Symp. on the Theory of Computing (STOC)*, pages 234–246. ACM, 2020. doi:10.1145/3357713.3384242.

8 Eshan Chattopadhyay, Pooya Hatami, Shachar Lovett, and Avishay Tal. Pseudorandom generators from the second fourier level and applications to AC0 with parity gates. In *ACM Innovations in Theoretical Computer Science conf. (ITCS)*, pages 22:1–22:15, 2019. doi:10.4230/LIPIcs.ITCS.2019.22.

9 Thomas Cover and Joy Thomas. *Elements of Information Theory (Wiley Series in Telecommunications and Signal Processing)*. Wiley-Interscience, 2006.

10 Bill Fefferman, Ronen Shaltiel, Christopher Umans, and Emanuele Viola. On beating the hybrid argument. *Theory of Computing*, 9:809–843, 2013.

11 Alexander Golovnev, Rahul Ilango, Russell Impagliazzo, Valentine Kabanets, Antonina Kolokolova, and Avishay Tal. $AC^0[p]$ lower bounds against MCSP via the coin problem. In Christel Baier, Ioannis Chatzigiannakis, Paola Flocchini, and Stefano Leonardi, editors, *46th International Colloquium on Automata, Languages, and Programming, ICALP 2019, July 9-12, 2019, Patras, Greece*, volume 132 of *LIPIcs*, pages 66:1–66:15. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019. doi:10.4230/LIPIcs.ICALP.2019.66.

12 Ronald L. Graham, Donald E. Knuth, and Oren Patashnik. *Concrete Mathematics: A Foundation for Computer Science, 2nd Ed.* Addison-Wesley, 1994. URL: https://www-cs-faculty.stanford.edu/%7Eknuth/gkp.html.

13 Nutan Limaye, Karteek Sreenivasaiah, Srikanth Srinivasan, Utkarsh Tripathi, and S. Venkitesh. A fixed-depth size-hierarchy theorem for $AC^0[\oplus]$ via the coin problem. In *ACM Symp. on the Theory of Computing (STOC)*, pages 442–453. ACM, 2019.

14 Shachar Lovett. Unconditional pseudorandom generators for low degree polynomials. *Theory of Computing*, 5(1):69–82, 2009. arXiv:toc:v005/a003.

15 Noam Nisan. Pseudorandom bits for constant depth circuits. *Combinatorica. An Journal on Combinatorics and the Theory of Computing*, 11(1):63–70, 1991.

16 Ryan O'Donnell. *Analysis of Boolean Functions*. Cambridge University Press, 2014.

17 Alexander Razborov. Lower bounds on the dimension of schemes of bounded depth in a complete basis containing the logical addition function. *Akademiya Nauk SSSR. Matematicheskie Zametki*, 41(4):598–607, 1987. English translation in Mathematical Notes of the Academy of Sci. of the USSR, 41(4):333-338, 1987.

18 Ronen Shaltiel and Emanuele Viola. Hardness amplification proofs require majority. *SIAM J. on Computing*, 39(7):3122–3154, 2010.

19 Roman Smolensky. Algebraic methods in the theory of lower bounds for Boolean circuit complexity. In *19th ACM Symp. on the Theory of Computing (STOC)*, pages 77–82. ACM, 1987.

20 Srikanth Srinivasan. On improved degree lower bounds for polynomial approximation. In *FSTTCS*, volume 24 of *LIPIcs*, pages 201–212. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2013.

21 Srikanth Srinivasan. A robust version of hegedus's lemma, with applications. In Konstantin Makarychev, Yury Makarychev, Madhur Tulsiani, Gautam Kamath, and Julia Chuzhoy, editors, *Proccedings of the 52nd Annual ACM SIGACT Symposium on Theory of Computing, STOC 2020, Chicago, IL, USA, June 22-26, 2020*, pages 1349–1362. ACM, 2020. doi:10.1145/3357713.3384328.

22 Srikanth Srinivasan, Utkarsh Tripathi, and S. Venkitesh. On the probabilistic degrees of symmetric boolean functions, 2019. arXiv:1910.02465.

23 Charalampos E. Tsourakakis, Michael Mitzenmacher, Jaroslaw Blasiok, Ben Lawson, Preetum Nakkiran, and Vasileios Nakos. Predicting positive and negative links with noisy queries: Theory & practice. *CoRR*, abs/1709.07308, 2017. arXiv:1709.07308.

24 Emanuele Viola. The complexity of hardness amplification and derandomization. *Ph.D. thesis, Harvard University*, 2006.

25 Emanuele Viola. On the power of small-depth computation. *Foundations and Trends in Theoretical Computer Science*, 5(1):1–72, 2009.

26 Emanuele Viola. The sum of d small-bias generators fools polynomials of degree d. *Computational Complexity*, 18(2):209–217, 2009.

27 Emanuele Viola. Challenges in computational lower bounds. *SIGACT News, Open Problems Column*, 48(1), 2017.

28 Emanuele Viola. Matching Smolensky's correlation bound with majority, 2019. URL: https://eccc.weizmann.ac.il/report/2019/175/.

29 Emanuele Viola. New lower bounds for probabilistic degree and AC0 with parity gates. *Theory of Computing*, 2020. URL: https://eccc.weizmann.ac.il/report/2020/015/.

30 Andrew Chi-Chih Yao. Probabilistic computations: Toward a unified measure of complexity (extended abstract). In *IEEE Symp. on Foundations of Computer Science (FOCS)*, pages 222–227. IEEE Computer Society, 1977.

Separations for Estimating Large Frequency Moments on Data Streams

David P. Woodruff
Carnegie Mellon University, Pittsburgh, PA, USA

Samson Zhou
Carnegie Mellon University, Pittsburgh, PA, USA

Abstract

We study the classical problem of moment estimation of an underlying vector whose n coordinates are implicitly defined through a series of updates in a data stream. We show that if the updates to the vector arrive in the random-order insertion-only model, then there exist space efficient algorithms with improved dependencies on the approximation parameter ε. In particular, for any real $p > 2$, we first obtain an algorithm for F_p moment estimation using $\tilde{\mathcal{O}}\left(\frac{1}{\varepsilon^{4/p}} \cdot n^{1-2/p}\right)$ bits of memory. Our techniques also give algorithms for F_p moment estimation with $p > 2$ on arbitrary order insertion-only and turnstile streams, using $\tilde{\mathcal{O}}\left(\frac{1}{\varepsilon^{4/p}} \cdot n^{1-2/p}\right)$ bits of space and two passes, which is the first optimal multi-pass F_p estimation algorithm up to $\log n$ factors. Finally, we give an improved lower bound of $\Omega\left(\frac{1}{\varepsilon^2} \cdot n^{1-2/p}\right)$ for one-pass insertion-only streams. Our results separate the complexity of this problem both between random and non-random orders, as well as one-pass and multi-pass streams.

2012 ACM Subject Classification Theory of computation → Streaming, sublinear and near linear time algorithms

Keywords and phrases streaming algorithms, frequency moments, random order, lower bounds

Digital Object Identifier 10.4230/LIPIcs.ICALP.2021.112

Category Track A: Algorithms, Complexity and Games

Related Version *Full Version*: https://arxiv.org/abs/2105.03773

Funding The authors would like to thank support from NSF grant No. CCF-181584 and a Simons Investigator Award.

1 Introduction

The efficient computation of statistics has emerged as an increasingly important goal for large databases storing information generated from financial markets, internet traffic, IoT sensors, scientific observations, etc. The one-pass streaming model formally defines an implicit and underlying dataset through sequential updates that arrive one at a time and describe the evolution of the dataset over time. The goal is to aggregate or approximate some statistic of the input data using space that is sublinear in the size of the input.

The frequency moment estimation problem is fundamental to data streams. Since the celebrated paper of Alon, Matias, and Szegedy [1], the frequency moment estimation problem has been a central problem in the streaming model; more than two decades of research [1, 14, 4, 33, 24, 23, 29, 27, 26, 17, 10, 9, 6, 11, 19, 35] has studied the space or time complexity of this problem. We first consider the *insertion-only* model, where updates take the form $u_1, \ldots, u_m$ in a stream of length m and each update u_t is in $[n] = \{1, 2, \ldots, n\}$ for $t \in [m]$. We assume for simplicity that $m \leq \text{poly}(n)$. The updates implicitly define a frequency vector $f \in \mathbb{R}^n$ so that each update effectively increases a coordinate of f in the sense that $f_i = |\{t : u_t = i\}|$ for each $i \in [n]$. Given $p, \varepsilon > 0$, the F_p moment estimation problem is to approximate $F_p = \sum_{i \in [n]} (f_i)^p$ within a $(1 \pm \varepsilon)$ factor. The complexity of

this problem differs greatly for the range of p. For $p > 2$, [4, 14] showed that even for the streaming model, the space usage for F_p-estimation requires polynomial factors in n and m, whereas polylogarithmic factors are achievable for $p \leq 2$ [1, 23, 29, 27, 26, 6].

We initially focus on the frequency moment estimation problem in the *random-order* model, where the set of stream updates to the underlying frequency vector is worst case, but the order of their arrival is uniformly random. As usual, the algorithms we initially consider are only permitted a single pass over the stream, though we later relax this constraint to permit adversarial ordering of updates, as well as both positive and negative integer updates (with magnitude bounded by $\text{poly}(n)$) to the coordinates of the frequency vector in the *turnstile* model. Random-order streams have been shown to be a natural assumption for problems of sorting and selecting in limited space [32] and many other real-world applications [21, 22, 16, 13]. Interestingly, there has sometimes been a significant qualitative difference between random-order streams and adversarial (or arbitrary) order streams [28, 9, 11]. In particular for F_p-moment estimation, the best known lower bound is $\Omega\left(\frac{1}{\varepsilon^2}\right)$ [12]. While the best known upper bound for $p \in (0, 2]$ on arbitrary order insertion-only streams is $\mathcal{O}\left(\frac{1}{\varepsilon^2}\log n\right)$ [1, 27, 26], Braverman *et. al.* [11] gave an algorithm for $p \in (0, 2)$ on random-order streams that only used $\tilde{\mathcal{O}}\left(\frac{1}{\varepsilon^2} + \log n\right)$ space, where for a function $g(n, \varepsilon)$, we use $\tilde{\mathcal{O}}\left(g(n, \varepsilon)\right)$ to denote a function bounded by $g(n, \varepsilon) \cdot \text{polylog}(g(n, \varepsilon))$. They also gave an algorithm for F_2-moment estimation that only uses $\tilde{\mathcal{O}}\left(\frac{1}{\varepsilon^2} + \log n\right)$ space, but requires the assumption that $F_2 \geq F_1 \cdot \log n$, i.e., the second moment must be a logarithmic factor larger than the length of the stream.

The above works raise a number of important questions. A tantalizing open question, studied extensively in [25], and dating back to the original work of Alon, Matias, and Szegedy [1], is the exact space complexity of F_p-moment estimation in insertion-only streams. For $p > 2$, the best known upper bound for F_p-moment estimation on arbitrary order insertion-only streams is the minimum of $\mathcal{O}\left(\frac{n^{1-2/p}}{\varepsilon^{20}}\right)$ [9] and $\tilde{\mathcal{O}}\left(\frac{1}{\varepsilon^2} \cdot n^{1-2/p}\right)$ [17, 19]. For insertion-only streams, the best known lower bound is $\tilde{\Omega}\left(\frac{n^{1-2/p}}{\varepsilon^2 \log n}\right)$ [18]. We summarize these results in Figure 1. There are also major gaps in our understanding for F_p-moment estimation

Reference	Space Complexity	Order
[24, 31, 2]	$\tilde{\mathcal{O}}\left(n^{1-2/p}\varepsilon^{-\mathcal{O}(1)}\right)$	Arbitrary
[9]	$\mathcal{O}\left(n^{1-2/p}\varepsilon^{-20}\right)$	Arbitrary
[3]	$\tilde{\mathcal{O}}\left(n^{1-2/p}\varepsilon^{-2-6/p}\right)$	Arbitrary
[5]	$\tilde{\mathcal{O}}\left(n^{1-2/p}\varepsilon^{-2-4/p}\right)$	Arbitrary
[17, 19]	$\tilde{\mathcal{O}}\left(n^{1-2/p}\varepsilon^{-2}\right)$	Arbitrary
This work	$\tilde{\mathcal{O}}\left(n^{1-2/p}\varepsilon^{-4/p}\right)$	Random or Two-Pass Arbitrary
[1, 33]	$\Omega\left(n^{1-5/p} + \varepsilon^{-2}\right)$	Arbitrary
[14]	$\Omega\left(n^{1-2/p}\varepsilon^{-2/p}\right)$	Arbitrary
[34]	$\Omega\left(n^{1-2/p}\varepsilon^{-4/p}/\log^{\mathcal{O}(1)} n\right)$	Arbitrary, $\mathcal{O}(1)$-Passes
[18]	$\Omega\left(n^{1-2/p}\varepsilon^{-2}/\log n\right)$	Arbitrary
[12]	$\Omega(n^{1-2.5/p} + \varepsilon^{-2})$	Random
This work	$\Omega\left(n^{1-2/p}\varepsilon^{-2}\right)$	Arbitrary

Figure 1 Summary of recent work for large frequency moment estimation.

for $p > 2$ in random order streams. The best known[1] lower bound is $\Omega(n^{1-2.5/p} + \varepsilon^{-2})$ [12], while no upper bounds that do better in random order streams than in arbitrary insertion streams are known.

1.1 Our Results

In this paper, we show a separation not only between random-order and arbitrary insertion-only streams for the F_p moment estimation problem, but also one-pass and multi-pass streams. We first show improved bounds for the F_p moment estimation problem for every $p > 2$ in the random-order insertion-only streaming model.

▶ **Theorem 1.1.** *For $p > 2$, there exists an algorithm that outputs a $(1+\varepsilon)$-approximation to the F_p moment of a random-order insertion-only stream with probability at least $\frac{2}{3}$, while using total space (in bits) $\tilde{\mathcal{O}}\left(\frac{1}{\varepsilon^{4/p}} \cdot n^{1-2/p}\right)$.*

Theorem 1.1 utilizes the random-order model to improve the algorithm on arbitrary-order insertion-only streams using $\tilde{\mathcal{O}}\left(\frac{1}{\varepsilon^2} \cdot n^{1-2/p}\right)$ space [17, 19], in terms of the dependence on $\frac{1}{\varepsilon}$. We then give an algorithm that uses roughly the same bounds even for the two-pass streaming model, even if the order of the updates is adversarial.

▶ **Theorem 1.2.** *For $p > 2$, there exists a two-pass streaming algorithm that outputs a $(1+\varepsilon)$-approximation to the F_p moment with probability at least $\frac{2}{3}$.*

- *If the stream is insertion-only, the algorithm uses $\tilde{\mathcal{O}}\left(\frac{1}{\varepsilon^{4/p}} \cdot n^{1-2/p}\right)$ bits of space (see Theorem 3.6).*
- *If the steam has turnstile updates, the algorithm uses $\tilde{\mathcal{O}}\left(\frac{1}{\varepsilon^{4/p}} \cdot n^{1-2/p}\right)$ bits of space (see Theorem 3.5).*

Theorem 1.2 is the first algorithm to match the lower bound of $\tilde{\Omega}\left(\frac{1}{\varepsilon^{4/p}} \cdot n^{1-2/p}\right)$ for multi-pass frequency moment estimation [34] up to $\log n$ factors.

By contrast, we give a lower bound for F_p estimation in the one-pass insertion-only streaming model when the order of the updates is adversarial.

▶ **Theorem 1.3.** *For any constant $p > 2$ and parameter $\varepsilon = \Omega\left(\frac{1}{n^{1/p}}\right)$, any one-pass insertion-only streaming algorithm that outputs a $(1+\varepsilon)$-approximation to the F_p moment of an underlying frequency vector with probability at least $\frac{9}{10}$ requires $\Omega\left(\frac{1}{\varepsilon^2} \cdot n^{1-2/p}\right)$ bits of space.*

Theorem 1.3 improves the lower bound of $\Omega\left(\frac{1}{\varepsilon^2 \log n} \cdot n^{1-2/p}\right)$ for general insertion-only streams for $p > 2$ by [17]. Together, Theorem 1.1 and Theorem 1.3 for small enough $\varepsilon > 0$ show a somewhat surprising result that random-order streams are strictly easier than arbitrary insertion-only streams. Similarly, Theorem 1.2 and Theorem 1.3 together show that multiple passes are strictly easier than a single pass on arbitrary insertion-only streams.

1.2 Our Techniques

We first describe our random-order insertion-only algorithm. At a high level, our algorithm interleaves the recent heavy-hitters algorithm of [7] with a subsampling procedure to estimate the contributions of various level sets toward the frequency moment.

[1] After discussion with the authors, there appears to be an error in [20], which claims a stronger lower bound.

Approximate frequencies of heavy-hitters. The heavy-hitters algorithm of [7] partitions the updates of a random-order stream into blocks of updates. It then randomly chooses a number of coordinates from the universe $[n]$ to test in each block. The heavy-hitters will pass the test and subsequently be tracked across a number of following blocks before estimates for their frequencies are output. Due to the uniformity properties of the random-order stream, the heavy-hitters are sufficiently "spread out", so the algorithm crucially outputs a $(1+\varepsilon)$-approximation to the frequency of each heavy-hitter; algorithms with the same guarantee and space complexity for arbitrary-order streams do not exist [18].

Using level sets to estimate frequency moments. Given the subroutine for finding $(1+\varepsilon)$-approximations to the frequencies of heavy-hitters, we now build upon a standard subsampling approach to estimate the frequency moment. Informally, we conceptually define the level sets so that level set Λ_i contains the coordinates $k \in [n]$ such that $f_k^p \in \left[\frac{F_p}{2^i}, \frac{2F_p}{2^i}\right]$. Since the level sets partition the universe, it is easy to see that if we define the contribution C_i of a level set Λ_i by the sum of the contributions of all their coordinates, $C_i := \sum_{k\in\Lambda_i} f_k^p$, then F_p is just the sum of all the contributions of the level sets, $F_p = \sum_i C_i$.

[24] showed that the contributions of each "significant" level set can be estimated by subsampling at exponentially smaller rates and considering the approximate frequencies of the heavy-hitters in each of the subsamples. For example, a single item with contribution F_p will be detected at the top level, while n items with contribution 1 will be detected at a subsampling level where there are roughly $\Theta\left(\frac{1}{\varepsilon^p}\right)$ survivors in expectation. Crucially, $(1+\varepsilon)$-approximations to the contribution of the surviving heavy-hitters in each subsampling level can then be rescaled by the sampling rate to obtain "good" approximations to the contributions of each significant level set; these very good estimates are not available in standard subsampling schemes for arbitrary order streams.

We adapt this approach for $p > 2$ to obtain Theorem 1.1. The first observation is that an item i with $f_i^p \geq \varepsilon^2 F_p$ should be identified to control the variance but also satisfies $f_i^2 \geq \varepsilon^{4/p}/n^{1-2/p} \cdot F_2$, so we can identify these items using the heavy-hitter algorithm of [7] with the corresponding threshold; this induces the overall $n^{1-2/p}/\varepsilon^{4/p}$ dependency. Moreover as we subsample, the space of the universe decreases in expectation from n to $n/2$ to $n/4$ and so forth. Thus determining the L_p-heavy hitters at lower subsampling rates can be done using significantly less space. We can take advantage of this by requiring that our heavy-hitter algorithm aggressively seeks heavy-hitters with lower thresholds at lower subsampling rates. We can thus achieve a geometric series and avoid an additional $\mathcal{O}(\log n)$ factor in our space.

From random-order to two-pass arbitrary order. As stated, our algorithm necessitates the random-order model so that the heavy-hitter subroutine can output $(1+\varepsilon)$-approximations to the frequencies of heavy-hitters across different subsampling levels; these approximations are then used to obtain $(1+\varepsilon)$-approximations to the contributions of each level set. The state-of-the-art heavy-hitter algorithms in insertion-only [8] or turnstile [15] streams with arbitrary arrival order do not give a $(1+\varepsilon)$-approximation to the frequency of each heavy-hitter while still using space dependency $\frac{1}{\varepsilon^2}$. Fortunately, our approach can be remedied in two-passes over the data stream by first using a pass to identify each of the heavy-hitters across the different subsampling levels and then using the second pass to exactly count their frequencies; note that since $n^{1-2/p}/\varepsilon^{4/p} \geq \frac{1}{\varepsilon^2}$ for $n \geq \frac{1}{\varepsilon^2}$, we are still permitted space to track the frequencies of $\min\left(n, \frac{1}{\varepsilon^2}\right)$ items. We can then obtain $(1+\varepsilon)$-approximations to the contributions of each significant level set, thus obtaining a $(1+\varepsilon)$-approximation to the F_p frequency moment.

Source of the separation. In summary, our improved upper bounds in both the random-order and multi-pass models exploit each model to obtain $(1+\varepsilon)$-approximate frequencies of the heavy-hitters in each subsampling level. Due to the uniformity of heavy-hitters across the stream in the random-order model, we are able to obtain $(1+\varepsilon)$-approximate frequencies by simply tracking the frequency of the heavy-hitters within a small block of the stream and then scaling by the entire length of the stream. For multi-pass models, we are able to identify heavy-hitters in the first pass and exactly track their frequencies in the second pass. By contrast, obtaining $(1+\varepsilon)$-approximate frequencies in adversarially ordered insertion-only streams with the same space guarantees in a single pass cannot be done [18].

Lower bound. To prove our improved lower bound, we first recall the standard approach for showing F_p moment estimation lower bounds in insertion-only streams for $p > 2$ that uses the multiplayer set disjointness problem, e.g., [4, 18]. In this problem, t players have binary vectors of length n and the promise is that the largest coordinate in the sum of all vectors is either (1) at most 1 or (2) exactly t. For $t = \varepsilon^{1/p} n^{1/p}$, the F_p frequency moment of the sum of the binary vectors differs by a factor of $(1+\varepsilon)$ between the two cases, so that a $(1+\varepsilon)$-approximation to F_p solves the multiplayer set disjointness problem. The total communication complexity of the multiplayer set disjointness problem is $\Omega\left(\frac{n}{t}\right)$ so one of the t players communicates $\Omega\left(\frac{n}{t^2}\right)$ bits, and thus a lower bound of $\Omega\left(\frac{1}{\varepsilon^{2/p}} \cdot n^{1-2/p}\right)$ follows.

To improve the ε dependency in the lower bound to $\frac{1}{\varepsilon^2}$, we define the (t, ε, n)-player set disjointness estimation problem so that there are $t+1$ players $P_1, \ldots, P_{t+1}$ with private coins in the standard blackboard model. The first t players each receive a vector $\mathbf{v}_s \in \{0,1\}^n$ for $s \in [t]$, while player P_{t+1} receives an index $j \in [n]$ and a bit $c \in \{0,1\}$. For $\mathbf{u} = \sum_{s\in[t]} \mathbf{v}_s$, the inputs are promised to satisfy $u_i \leq 1$ for each $i \neq j$ and either $u_j = 1$ or $u_j = t$, similar to the multiparty set disjointness problem. With probability at least $\frac{9}{10}$, P_{t+1} must differentiate between the three possible input cases (1) $u_j + \frac{ct}{\varepsilon} \leq t$, (2) $u_j + \frac{ct}{\varepsilon} \in \left\{\frac{t}{\varepsilon}, \frac{t}{\varepsilon}+1\right\}$, or (3) $u_j + \frac{ct}{\varepsilon} = (1+\varepsilon)\frac{t}{\varepsilon}$, where $\varepsilon \in (0,1)$. We call coordinate $j \in [n]$ the *spike* location and show that the (t, ε, n)-player set disjointness estimation problem requires $\Omega\left(\frac{n}{t}\right)$ total communication by using a direct sum embedding to decompose the conditional information complexity into a sum of n single coordinate problems. The intuition is that the first t players do not know the spike location, so they must effectively solve the problem on each coordinate. We then bound the conditional information complexity of each single coordinate problem by the Hellinger distances between inputs for which the outputs differ. We thus apply the same reduction and set $t = \Theta\left(\frac{1}{\varepsilon} \cdot n^{1/p}\right)$ to obtain the desired lower bound.

We remark that the (t, ε, n)-player set disjointness estimation problem can be seen as a generalization of the augmented L_∞ promise problem introduced by [30]. In the augmented L_∞ promise problem, there are only three players, but each coordinate in the first two players' input vectors can be as large as εk for some fixed parameter k. [30] used the augmented L_∞ promise problem to give a lower bound of $\Omega\left(\frac{\log n}{\varepsilon^2} \cdot n^{1-2/p}\right)$ for F_p moment estimation on turnstile streams. However, since their reduction crucially allows players to use turnstile updates, we cannot adapt their techniques to obtain a reduction for insertion-only streams.

1.3 Preliminaries

For a positive integer n, we use the notation $[n]$ to denote the set of integers $\{1, 2, \ldots, n\}$. For a frequency vector f with dimension n and $p \geq 2$, we define the F_p moment function by $F_p(f) := \sum_{k\in[n]} |f_k|^p$ and the L_p norm of f by $L_p(f) = (F_p(f))^{1/p}$. When f is defined through the updates of a insertion-only data stream, we simply use F_p to denote $F_p(f)$.

Thus for a subset $I \subseteq [n]$, the notation $F_p(I)$ is understood to mean $\sum_{k \in I} f_k^p$. We also use the notation $\|f\|_p$ to denote the L_p norm of f. The F_p moment estimation problem and the norm estimation problem are often used interchangeably, as a $(1+\varepsilon)$-approximation algorithm to one of these problems can be modified to output a $(1+\varepsilon)$-approximation to the other using a rescaling of ε, for constant $p > 0$.

We use log and ln to denote the base two logarithm and natural logarithms, respectively. We use $\text{poly}(n)$ to denote a fixed constant degree polynomial in n and $\frac{1}{\text{poly}(n)}$ to denote some arbitrary degree polynomial in n corresponding to the choice of constants in the algorithms. We use $\text{polylog}(n)$ to denote polylogarithmic factors of n.

The L_p-heavy hitters problem is to output all coordinates i such that $f_i \geq \varepsilon \cdot L_p$; such a coordinate is called a *heavy-hitter*. The problem allows coordinates j with $f_j \leq \varepsilon \cdot L_p$ to be output, provided that $f_j \geq \frac{\varepsilon}{2} \cdot L_p$. Moreover, each coordinate output by the algorithm must also have a frequency estimation with additive error at most $\frac{\varepsilon}{2} \cdot L_p$.

2 F_p Estimation for $p > 2$ in Random-Order Streams

In this section, we give our F_p estimation algorithm for $p > 2$. We first introduce an algorithm CountHH on random-order insertion-only streams that outputs an approximate frequency for each L_2 heavy-hitter.

▶ **Theorem 2.1** (Theorem 28 in [7]). *There exists a one-pass algorithm CountHH on random-order insertion-only streams that outputs a list H of ordered pairs $(j, \widehat{f_j})$ such that $\widehat{f_j} = (1 \pm 2\varepsilon) f_j$, for each $j \in H$, where f is the underlying frequency vector. Moreover, we have that $j \in H$ for each $j \in [n]$ with $f_j^2 \geq \varepsilon^2 \cdot F_2$ and $j \notin H$ for each j with $f_j^2 \leq \frac{\varepsilon^2}{2} \cdot F_2$. The algorithm uses $\mathcal{O}\left(\frac{1}{\varepsilon^2}\left(\log^2 \frac{1}{\varepsilon} + \log^2 \log m + \log n\right) + \log m\right)$ bits of space and succeeds with probability at least $38/39$.*

A standard heavy-hitter algorithm such as CountSketch [15] or BPTree [8] outputs each $j \in [n]$ with $f_j^2 \geq \frac{\varepsilon^2}{2} \cdot F_2$ but only finds a constant-factor approximation to the frequency of each heavy-hitter. By comparison, CountHH crucially uses the random-order stream to output a $(1+\varepsilon)$-approximation to the frequency of each heavy-hitter; we defer the full details of the algorithm to the full version of the paper [36]. We use CountHH in a subsampling approach to approximate the contributions of each of the level sets, defined as follows:

▶ **Definition 2.2** (Level sets and contribution). *Given $\widehat{F_p}$ such that $\widehat{F_p} \leq F_p \leq 1.01 \cdot \widehat{F_p}$ and a uniformly random $\zeta \in [1, 2]$, we define the* level set Λ_i *for each $i \in [\log 4n]$ so that*

$$\Lambda_i := \left\{ k \,|\, f_k^p \in \left[\frac{\zeta \cdot \widehat{F_p}}{2^i}, \frac{2\zeta \cdot \widehat{F_p}}{2^i} \right] \right\}.$$

Then we define the contribution C_i *of level set Λ_i to be $C_i := \sum_{k \in \Lambda_i} f_k^p$. We define the* fractional contribution ϕ_i *of level set Λ_i to be the ratio $\phi_i := \frac{C_i}{\zeta \widehat{F_p}}$, so that $\phi_i \in [0, 1.01]$.*

For a stream of length $m = \text{poly}(n)$, let α be an integer such that $2^\alpha > m^p$. We say a level set Λ_i is significant *if its fractional contribution ϕ_i is at least $\frac{\varepsilon}{2\alpha \log n}$. Otherwise, we say the level set is* insignificant.

Observe that it suffices to obtain a multiplicative $\left(1 + \frac{\varepsilon}{2}\right)$-approximation to the contribution of each significant level set and estimate the contributions of the insignificant level sets to be zero, since there are at most $\alpha \log n$ level sets and thus the total additive error from the insignificant level sets is at most $\frac{\varepsilon}{2\alpha \log n} \cdot F_p \cdot \alpha \log n = \frac{\varepsilon}{2} \cdot F_p$.

To estimate the contribution of each level set, we use a combination of CountHH and subsampling to approximate the frequencies of a number of heavy-hitters. The main idea is that subsampling induces a separate substream with a frequency vector with a smaller F_p. Thus the items in the level sets Λ_i with small i will be heavy-hitters of the frequency vector while the items in the level sets Λ_i with larger i will be heavy-hitters of the substreams with smaller sampling rates (if the items are subsampled), due to the lower F_p of the substreams. We can then use a $(1+\varepsilon)$-approximation to the frequency of each sampled heavy-hitter to estimate a $(1+\varepsilon)$-approximation to the contribution of the level set, due to uniformity properties of random-order streams. Note that this is where we crucially use CountHH over other possible heavy-hitter algorithms, due to its advantage of providing $(1+\varepsilon)$-approximate frequencies in the random-order model. We describe our algorithm for F_p estimation with $p > 2$ for random-order insertion-only streams in Algorithm 1.

Algorithm 1 F_p Estimation in the Random-Order Insertion-Only Model, $p > 2$.

Input: Accuracy parameter $\varepsilon \in (0,1)$, $\widehat{F_p} \le F_p \le 1.01 \cdot \widehat{F_p}$
Output: $(1+\varepsilon)$-approximation to F_p.

1: Let $\zeta \in [1,2]$ be chosen uniformly at random and α be a positive integer such that $2^\alpha > m^p$.
2: Let $\eta > \sum_{j \ge 0} 2^{-j(p/16-1/8)}$ be a sufficiently large constant.
3: $\gamma \leftarrow 2^{11}$, $n_i \leftarrow \min\left(\left(\frac{16\alpha p \log n}{\varepsilon^{1-2/p}}\right)^{(2p)/(p-2)}, \frac{10\gamma n}{2^i}\right)$, $\varepsilon_i = \frac{\varepsilon}{16\eta \cdot 2^{i(p/16-1/8)} \log \frac{1}{\varepsilon^2}}$
4: Let I_i^r be a (nested) subset of $[n]$ subsampled at rate $p_i := \min(1, 2^{-i}\gamma)$.
5: Let H_i^r be the output of CountHH with threshold parameter $\frac{(\varepsilon_i)^{2/p}}{80\gamma} \cdot \left(\frac{1}{n_i}\right)^{1/2-1/p}$ on the substream induced by I_i^r.
6: **for** $i \in [\alpha \log n]$, $r \le \mathcal{O}(\log \log n)$ **do**
7: $\quad \ell_i := \min\{k : 2^k > 2^i \cdot \varepsilon_k^2\}$
8: $\quad$ Let S_i^r be the set of ordered pairs $(j, \widehat{f_j})$ in $H_{\ell_i}^r$ with $\left(\widehat{f_j}\right)^p \in \left[\frac{\zeta \widehat{F_p}}{2^i}, \frac{2\zeta \widehat{F_p}}{2^i}\right]$.
9: $\quad \widehat{C_i} \leftarrow \text{median}_r \frac{1}{p_{\ell_i}} \cdot \left(\sum_{(j,\widehat{f_j}) \in S_i^r} \left(\widehat{f_j}\right)^p\right)$
10: **return** $\tilde{F_p} := \sum_i \widehat{C_i}$

▶ Remark 2.3. We remark that Algorithm 1 is written requiring an input $\widehat{F_p}$ such that $\widehat{F_p} \le F_p \le 1.01 \cdot \widehat{F_p}$ in order to correctly index the level sets defined by Definition 2.2. Algorithm 1 can be rewritten by setting $X = (F_1)^p$ to be an upper bound on F_p and redefining the level sets in Definition 2.2 so that $\Lambda_i := \left\{k \,|\, f_k^p \in \left[\frac{\zeta \cdot X}{2^i}, \frac{2\zeta \cdot X}{2^i}\right]\right\}$. Then we again have $F_p = \sum_i C_i$ across the contributions of all the level sets.

We first show that there exists a subsampling rate such that items in each level set will be reported as a heavy-hitter if they are successfully subsampled. Moreover, each item reported as a heavy-hitter will also be reported with an "accurate" estimate of its frequency.

▶ **Lemma 2.4.** *Let $\varepsilon \in (0,1)$, Λ_i be a fixed level set, and let $\ell := \ell_i := \min\{k : 2^k > 2^i \cdot \varepsilon_k^2\}$. For a fixed r, let $\mathcal{E}_1$ be the event that $|I_\ell^r| \le \frac{10\gamma n}{2^\ell}$ and let $\mathcal{E}_2$ be the event that $F_p(I_\ell^r) \le \frac{10\gamma F_p}{2^\ell}$. Conditioned on $\mathcal{E}_1$ and $\mathcal{E}_2$, there exists a $(j, \widehat{f_j})$ in S_i^r for each $j \in \Lambda_i \cap I_\ell^r$ such that with probability at least $\frac{7}{8}$,*

$$\left(1 - \frac{\varepsilon}{8\alpha \log n}\right) \cdot f_j^p \le \left(\widehat{f_j}\right)^p \le \left(1 + \frac{\varepsilon}{8\alpha \log n}\right) \cdot f_j^p.$$

Proof. Recall that Algorithm 1 considers H_ℓ^r across $r \le \mathcal{O}(\log\log n)$ independent subsamples of $[n]$, to construct $\widehat{C_i}$ by subsampling at rate $p_\ell := \min(1, 2^{-\ell}\gamma)$.

Suppose $2^i \cdot \varepsilon^2 < \gamma$, so that $\ell := \ell_i = \min\{k : 2^k > 2^i \cdot \varepsilon_k^2\} \le \log\gamma$ since $\varepsilon_k \le \varepsilon$ for all k. Then $p_\ell = \min(1, 2^{-\ell}\gamma) = 1$ and all items are subsampled, i.e., $H_\ell^r = [n]$. By Definition 2.2 of the level sets, each item $j \in \Lambda_i$ satisfies $f_j^p \in \left[\frac{\zeta\cdot\widehat{F_p}}{2^i}, \frac{2\zeta\cdot\widehat{F_p}}{2^i}\right]$. Since $\varepsilon^2 \ge \frac{1}{2^i}$, then we have $f_j^p \ge \varepsilon^2 \cdot \widehat{F_p}$. We also have $\widehat{F_p} \le F_p \le 1.01 \cdot \widehat{F_p}$ and $n^{1/2-1/p} \cdot F_p^{1/p} \ge F_2^{1/2}$. Thus, each item in $j \in \Lambda_i$ satisfies

$$f_j \ge \frac{\varepsilon^{2/p}}{1.01^{1/p}} \cdot F_p^{1/p} \ge \frac{\varepsilon^{2/p}}{1.01^{1/p} n^{1/2-1/p}} \cdot F_2^{1/2},$$

so that $f_j^2 \ge \frac{\varepsilon^{4/p}}{1.01 n^{1-2/p}} F_2$. Then by Theorem 2.1, each item $j \in \Lambda_i$ corresponds to some estimate $\left(\widehat{f_j}\right)^2$ reported by H_ℓ^r output by CountHH with threshold $\frac{(\varepsilon_\ell)^{2/p}}{80\gamma}$, as $\varepsilon_\ell \le \varepsilon$ and $n_\ell = n$. Hence, $\widehat{f_j^p}$ is a $\left(1 + \frac{\varepsilon}{8\alpha\log n}\right)$-approximation to f_j^p. Furthermore, CountHH rounds the estimate of the frequency of each heavy-hitter to the nearest power of $\left(1 + \frac{\varepsilon}{16p\alpha\log n}\right)$. Hence, $\widehat{f_j^p}$ is a $\left(1 + \frac{\varepsilon}{8\alpha\log n}\right)$-approximation to f_j^p.

Now suppose $2^i \cdot \varepsilon^2 \ge \gamma$, so that $\ell := \ell_i = \min\{k : 2^k > 2^i \cdot \varepsilon_k^2\} \ge \log\gamma$ and $p_\ell := \min(1, 2^{-\ell}\gamma) \ge \frac{\gamma}{2\cdot 2^i \varepsilon_\ell^2}$. Again by Definition 2.2 of the level sets, each item $j \in \Lambda_i$ satisfies $f_j^p \in \left[\frac{\zeta\cdot\widehat{F_p}}{2^i}, \frac{2\zeta\cdot\widehat{F_p}}{2^i}\right]$. Now if $j \notin I_\ell^r$, then it will not be reported by CountHH. Thus we assume that $j \in I_\ell^r$.

Conditioning on the event $\mathcal{E}_2$, we have that

$$F_p(I_\ell^r) \le \frac{10\gamma F_p}{2^\ell} \le \frac{20\gamma F_p}{2^i\varepsilon^2}.$$

Since $\widehat{F_p} \le F_p \le 1.01 \cdot \widehat{F_p}$, then we have that $f_j^p \ge \frac{\varepsilon^2}{80\gamma} \cdot F_p(I_\ell^r)$ and thus

$$f_j \ge \frac{\varepsilon^{2/p}}{(80\gamma)^{1/p}} \cdot F_p^{1/p}(I_\ell^r).$$

Instead of applying the inequality $n^{1/2-1/p} \cdot F_p^{1/p} \ge F_2^{1/2}$, we note that conditioning on the event $\mathcal{E}_1$, we have that $|I_\ell^r| \le n_\ell = \frac{10\gamma n}{2^\ell}$. Hence, the frequency vector defined by the substream I_ℓ^r potentially has much smaller support size than n. Thus we have

$$f_j \ge \frac{\varepsilon^{2/p}}{(80\gamma)^{1/p}} \cdot \left(\frac{1}{n_\ell}\right)^{1/2-1/p} \cdot F_2^{1/2}(I_\ell^r).$$

By Theorem 2.1, each item $j \in \Lambda_i \cap I_\ell^r$ corresponds to some estimate $\widehat{f_j}$ reported by H_ℓ^r output by CountHH with threshold $\frac{(\varepsilon_\ell)^{2/p}}{80\gamma} \cdot \left(\frac{1}{n_\ell}\right)^{1/2-1/p}$, since $\varepsilon_\ell \le \varepsilon$. Moreover, the estimate of the frequency of each heavy-hitter reported by CountHH is within a factor of $\left(1 + \frac{\varepsilon}{16p\alpha\log n}\right)$ of the true frequency, since $n_i \ge \left(\frac{16\alpha p\log n}{\varepsilon^{1-2/p}}\right)^{(2p)/(p-2)}$ implies that the threshold is at most $\frac{\varepsilon}{16p\alpha\log n}$. Hence for sufficiently small ε, $\left(\widehat{f_j}\right)^p$ is a $\left(1 + \frac{\varepsilon}{8\alpha\log n}\right)$-approximation to f_j^p. ◀

In summary, an item $k \in \Lambda_i$ may not always be sampled by I_ℓ^r, but CountHH outputs a quantity $\widehat{f_k}$ such that $\left(\widehat{f_k}\right)^p$ is a $\left(1 + \frac{\varepsilon}{8\alpha\log n}\right)$-approximation to f_k^p if $k \in I_\ell^r$.

These approximations allow us to recover a $(1+\varepsilon)$-approximation to the F_p moment through Lemma 2.5, which is our main correctness statement and which we now show. Lemma 2.5 claims that the output $\tilde{F}_p$ of our algorithm serves as a $(1+\varepsilon)$-approximation to the F_p moment of the underlying frequency vector. The proof of Lemma 2.5 first considers an idealized process and shows that we obtain "good" approximations to the contributions of each significant level set. Since the contributions of the insignificant level sets can be ignored, we thereby obtain a $(1+\varepsilon)$-approximation to the F_p moment. We then show that when the process is not idealized, the estimate $\tilde{F}_p$ only occurs a small error and thus still guarantees a $(1+\varepsilon)$-approximation to the F_p moment.

▶ **Lemma 2.5.** *With probability at least $\frac{2}{3}$, we have that $|\tilde{F}_p - F_p| \leq \varepsilon \cdot F_p$.*

Proof. Let Λ_i be a fixed level set and let $\ell := \ell_i := \min\{k : 2^k > 2^i \cdot \varepsilon_k^2\}$. Let $k \in \Lambda_i \cap S_\ell^r$, so that $f_k^p \in \left[\frac{\zeta \cdot \widehat{F_p}}{2^i}, \frac{2\zeta \cdot \widehat{F_p}}{2^i}\right]$ and k is subsampled at the level in which its estimated frequency $\left(\widehat{f_k}\right)^p$ is used to estimate the contribution C_i of level set Λ_i. Then Lemma 2.4 shows that we obtain an estimate $\left(\widehat{f_k}\right)^p$ such that

$$\left(1 - \frac{\varepsilon}{8\alpha \log n}\right) \cdot f_k^p \leq \left(\widehat{f_k}\right)^p \leq \left(1 + \frac{\varepsilon}{8\alpha \log n}\right) \cdot f_k^p,$$

with constant probability. In an idealized process, we would have

$$\frac{\zeta \cdot \widehat{F_p}}{2^i} \leq \left(\widehat{f_k}\right)^p \leq \frac{2\zeta \cdot \widehat{F_p}}{2^i},$$

so that the estimate $\left(\widehat{f_k}\right)^p$ is used toward the estimation $\widehat{C_i}$ of contribution C_i of level set Λ_i. However, this may not always be the case because the value of f_k^p may be near the boundary of the interval $\left[\frac{\zeta \cdot \widehat{F_p}}{2^i}, \frac{2\zeta \cdot \widehat{F_p}}{2^i}\right]$ and the value of the estimate $\left(\widehat{f_k}\right)^p$ may lie outside of the interval, so that the estimate $\left(\widehat{f_k}\right)^p$ is used toward the estimation of some other level set $\Lambda_{i'}$. We first analyze an idealized process, so that $\left(\widehat{f_k}\right)^p$ is correctly classified for all k across all level sets, and show that the output is a $(1+\mathcal{O}(\varepsilon))$-approximation to F_p. We then argue that because we randomize the boundaries of each interval due to the selection of ζ, the overall guarantee is only slightly worsened but remains a $(1+\varepsilon)$-approximation to F_p.

Idealized process. We first show that for an idealized process where $\left(\widehat{f_k}\right)^p$ is correctly classified for all k across all level sets, then for a fixed level set i, we have $|\widehat{C_i} - C_i| \leq \frac{\varepsilon_\ell}{2} \cdot F_p$ with probability at least $1 - \frac{1}{\text{polylog}(n)}$. Let $\mathcal{E}_1$ be the event that $|I_\ell^r| \leq \frac{10\gamma n}{2^\ell}$ and let $\mathcal{E}_2$ be the event that $F_p(I_\ell^r) \leq \frac{10\gamma F_p}{2^\ell}$. Lemma 2.4 shows that conditioned on $\mathcal{E}_1$ and $\mathcal{E}_2$, then CountHH outputs a $\left(1 + \frac{\varepsilon}{8\alpha \log n}\right)$-approximation to f_k^p if $k \in I_\ell^r$. We thus analyze the approximation $\widehat{C_i^r}$ to C_i for a given set of subsamples, where we define

$$\widehat{C_i^r} := \frac{1}{p_{\ell_i}} \cdot \sum_{(k, \widehat{f_k}) \in S_i^r} \left(\widehat{f_k}\right)^p,$$

so that $\widehat{C_i} = \text{median}_r \widehat{C_i^r}$. Conditioned on $\mathcal{E}_1$ and $\mathcal{E}_2$, we note that $\widehat{C_i^r}$ is a $\left(1 + \frac{\varepsilon}{8\alpha \log n}\right)$-approximation to

$$D_i^r := \frac{1}{p_{\ell_i}} \cdot \sum_{k \in S_i^r \cap \Lambda_i} f_k^p.$$

We thus analyze the expectation and variance of D_i^r.

We first analyze the expectation of D_i^r. Note that

$$\mathbb{E}\left[D_i^r\right] = \frac{1}{p_\ell} \cdot \sum_{k \in \Lambda_i} p_\ell \cdot f_k^p = C_i.$$

We next analyze the variance of D_i^r, which results from whether items $k \in \Lambda_i$ are sampled by I_ℓ^r. Since $p_\ell \geq \frac{\gamma}{2 \cdot 2^i \varepsilon_\ell^2}$, we have

$$\text{Var}(D_i^r) = \frac{1}{p_\ell^2} \cdot \sum_{k \in \Lambda_i} p_\ell f_k^{2p} \leq \sum_{k \in \Lambda_i} \frac{2 \cdot 2^i \varepsilon_\ell^2}{\gamma} \cdot f_k^{2p}.$$

Observe that for each $k \in \Lambda_i$, we have $f_k^{2p} \leq \frac{16(F_p)^2}{2^{2i}}$ and $\frac{|\Lambda_i|}{2^i} \leq \phi_i \leq 1$. Thus for $\gamma = 2^{11}$,

$$\text{Var}(D_i^r) \leq |\Lambda_i| \cdot \frac{2 \cdot 2^i \varepsilon_\ell^2}{\gamma} \cdot \frac{16(F_p)^2}{2^{2i}} \leq \phi_i \varepsilon_\ell^2 (F_p)^2 \leq \frac{\varepsilon_\ell^2}{64}(F_p)^2.$$

Hence, by Chebyshev's inequality, we have that

$$\mathbf{Pr}\left[|D_i^r - C_i| \geq \frac{\varepsilon_\ell}{2} \cdot F_p\right] \leq \frac{1}{16}.$$

Since $C_i \leq F_p$ and $\widehat{C_i^r}$ is a $\left(1 + \frac{\varepsilon}{8\alpha \log n}\right)$-approximation to D_i^r, then $\widehat{C_i^r}$ gives an approximation to C_i with additive error at most $\left(\frac{\varepsilon}{4\alpha \log n} + \varepsilon_\ell\right) \cdot F_p$ with probability at least $\frac{15}{16}$, conditioned on the events $\mathcal{E}_1$ and $\mathcal{E}_2$, and the correctness of the subroutine CountHH. By Theorem 2.1, the correctness of CountHH occurs with probability at least $\frac{38}{39}$. By a union bound, we have that conditioned on $\mathcal{E}_1$ and $\mathcal{E}_2$, then

$$\mathbf{Pr}\left[|\widehat{C_i^r} - C_i| \leq \left(\frac{\varepsilon}{4\alpha \log n} + \varepsilon_\ell\right) \cdot F_p\right] \geq \frac{7}{8}.$$

For a fixed r, let $\mathcal{E}_1$ be the event that $|I_\ell^r| \leq \frac{10\gamma n}{2^\ell}$ and let $\mathcal{E}_2$ be the event that $F_p(I_\ell^r) \leq \frac{10\gamma F_p}{2^\ell}$. Recall that I_ℓ^r is a nested subset of $[n]$ subsampled at rate $p_\ell := \min(1, 2^{-\ell}\gamma)$. Thus we have $\mathbb{E}\left[|I_\ell^r|\right] \leq \frac{\gamma n}{2^\ell}$, so that by Markov's inequality, we have that $\mathbf{Pr}\left[\mathcal{E}_1\right] \geq \frac{9}{10}$. Similarly, we have $\mathbb{E}\left[F_p(I_\ell^r)\right] \leq \frac{\gamma F_p}{2^\ell}$, so that by Markov's inequality, we have that $\mathbf{Pr}\left[\mathcal{E}_2\right] \geq \frac{9}{10}$. Hence by a union bound, $\mathbf{Pr}\left[\mathcal{E}_1 \wedge \mathcal{E}_2\right] \geq \frac{8}{10}$.

By Lemma 2.4, conditioned on the events $\mathcal{E}_1$ and $\mathcal{E}_2$, we have that $|\widehat{C_i^r} - C_i| \leq \left(\frac{\varepsilon}{4\alpha \log n} + \varepsilon_\ell\right) \cdot F_p$ for a fixed r, with probability at least $\frac{7}{8}$. Thus by a union bound, we have that $|\widehat{C_i^r} - C_i| \leq \left(\frac{\varepsilon}{4\alpha \log n} + \varepsilon_\ell\right) \cdot F_p$ for a fixed r, with probability at least $\frac{5}{8}$. Since $\widehat{C_i} = \text{median}_r \widehat{C_i^r}$ across $r \leq \mathcal{O}\left(\log \log n\right)$ iterations, then we have that $|\widehat{C_i} - C_i| \leq \left(\frac{\varepsilon}{4\alpha \log n} + \varepsilon_\ell\right) \cdot F_p$ with probability at least $1 - \frac{1}{\text{polylog}(n)}$.

By a union bound over the $\alpha \log n$ level sets, then with probability at least $1 - \frac{1}{\text{polylog}(n)}$, we have that $|\widehat{C_i} - C_i| \leq \left(\frac{\varepsilon}{4\alpha \log n} + \varepsilon_{\ell_i}\right) \cdot F_p$ simultaneously for all $i \in [\alpha \log n]$, where $\ell_i := \min\{k : 2^k > 2^i \cdot \varepsilon_k^2\}$. We form our estimate $\tilde{F}_p$ to F_p by setting $\tilde{F}_p := \sum_i \widehat{C_i}$ and we have $F_p = \sum_i C_i$. Since $i \in [\alpha \log n]$, $\ell_i := \min\{k : 2^k > 2^i \cdot \varepsilon_k^2\}$, and $\varepsilon_{\ell_i} = \frac{\varepsilon}{16\eta \cdot 2^{\ell_i(p/16-1/8)} \log \frac{1}{\varepsilon^2}}$, then

$$
\begin{aligned}
|\tilde{F}_p - F_p| &= \left|\sum_i \widehat{C_i} - \sum_i C_i\right| \leq \sum_i |\widehat{C_i} - C_i| \leq \sum_i \left(\frac{\varepsilon}{4\alpha \log n} + \varepsilon_{\ell_i}\right) \cdot F_p \\
&\leq \sum_{i \in [\alpha \log n]} \frac{\varepsilon}{4\alpha \log n} \cdot F_p + \sum_{i : \ell_i \leq \log \frac{1}{\varepsilon^2}} \varepsilon_{\ell_i} \cdot F_p + \sum_{i : \ell_i > \log \frac{1}{\varepsilon^2}} \varepsilon_{\ell_i} \cdot F_p \\
&\leq \frac{\varepsilon}{4} \cdot F_p + \log \frac{1}{\varepsilon^2} \cdot \frac{\varepsilon}{16 \log \frac{1}{\varepsilon^2}} \cdot F_p + \frac{\varepsilon}{16} \cdot F_p,
\end{aligned}
$$

where the last bound on the last term follows from $\eta > \sum_{j \geq 0} 2^{-j(p/16-1/8)}$. Thus we have that $|\tilde{F}_p - F_p| \leq \frac{\varepsilon}{2} \cdot F_p$ with probability at least $1 - \frac{1}{\text{polylog}(n)}$ in an idealized process.

Effects of randomized boundaries. We say that for a fixed r, an item $k \in [n]$ is *misclassified* if there exists a level set Λ_i such that

$$\frac{\zeta \cdot F_p}{2^i} \leq f_k^p \leq \frac{2\zeta \cdot F_p}{2^i},$$

but for an estimate $\left(\widehat{f_k}\right)^p$ output by CountHH on the set $S_{\ell_i}^r$, we have

$$\frac{\zeta \cdot F_p}{2^i} \leq \left(\widehat{f_k}\right)^p \leq \frac{2\zeta \cdot F_p}{2^i}.$$

Recall that by Lemma 2.4, we have for any fixed value of ζ that

$$\left(1 - \frac{\varepsilon}{8\alpha \log n}\right) \cdot f_k^p \leq \left(\widehat{f_k}\right)^p \leq \left(1 + \frac{\varepsilon}{8\alpha \log n}\right) \cdot f_k^p.$$

Since $\zeta \in [1, 2]$, then the probability that item $k \in [n]$ is misclassified is at most $\frac{\varepsilon}{2\alpha \log n}$. Moreover, in the event that item $k \in \Lambda_i$ is misclassified, it can only be misclassified into either level set Λ_{i+1} or level set Λ_{i-1}, since $\left(\widehat{f_k}\right)^p$ is a $\left(1 \pm \frac{\varepsilon}{8\alpha \log n}\right)$ multiplicative approximation to f_k^p.

Thus in the event that item $k \in [n]$ is misclassified, then $\left(\widehat{f_k}\right)^p$ will be rescaled by an incorrect probability, but only by at most a factor of two. Hence the error in the computation of the contribution of f_k^p to some level set Λ_i is at most $2f_k^p$. Then in expectation across all $k \in [n]$, the error due to the misclassification is at most $2F_p \cdot \frac{\varepsilon}{2\alpha \log n} = \frac{\varepsilon}{\alpha \log n} \cdot F_p$. Hence by Markov's inequality for sufficiently large n, the misclassification error is at most an additive $\frac{\varepsilon}{2} \cdot F_p$ with probability at least $\frac{3}{4}$. Therefore in total, we have that $|\tilde{F}_p - F_p| \leq \varepsilon \cdot F_p$ with probability at least $\frac{2}{3}$. ◀

It remains to analyze the space complexity of the algorithm as well as remove some additional unnecessary assumptions.

▶ **Theorem 1.1.** *For $p > 2$, there exists an algorithm that outputs a $(1 + \varepsilon)$-approximation to the F_p moment of a random-order insertion-only stream with probability at least $\frac{2}{3}$, while using total space (in bits) $\tilde{\mathcal{O}}\left(\frac{1}{\varepsilon^{4/p}} \cdot n^{1-2/p}\right)$.*

Proof. We first observe that we only require a 1.01-approximation $\widehat{F_p}$ to F_p as the input of Algorithm 1 to create the level sets and analyze accordingly. However, we can instead define the level sets using any upper bound X on $\widehat{F_p}$ by setting level set Λ_i to be the indices $k \in [n]$ for which $f_k^p \in \left[\frac{\zeta X}{2^i}, \frac{2\zeta X}{2^i}\right]$, rather than $\left[\frac{\zeta \widehat{F_p}}{2^i}, \frac{2\zeta \widehat{F_p}}{2^i}\right]$, and the same analysis will follow (see Remark 2.3). Intuitively, the fluidity of the definition of the level sets can be seen from the fact that we randomize the boundaries by going through a multiplicative ζ chosen randomly from $[1, 2]$ anyway, and additional empty level sets will not change the approximation guarantee. Thus by Lemma 2.5, there exists an algorithm that outputs a $(1+\varepsilon)$-approximation to the F_p moment.

It remains to analyze the space complexity. By Theorem 2.1, CountHH with threshold ε requires $\mathcal{O}\left(\frac{1}{\varepsilon^2}\left(\log^2\frac{1}{\varepsilon} + \log^2\log m + \log n\right) + \log m\right)$ bits of space. Algorithm 1 runs a separate instance of CountHH with threshold $\frac{(\varepsilon_i)^{2/p}}{80\gamma} \cdot \left(\frac{1}{n_i}\right)^{1/2-1/p}$ to output a set H_i^r for $r \le \mathcal{O}(\log\log n)$, where γ is a sufficiently large constant. Thus the total space for the CountHH instances across all i for a fixed r is at most

$$C_1 \log n + \left(C_1^2 \log n\right) \cdot \sum_{i \in [\alpha \log n]} \frac{C_1 (n_i)^{1-2/p}}{(\varepsilon_i)^{4/p}},$$

for some positive constants $C_1, \alpha > 0$. In particular, we have $\varepsilon_i = \frac{\varepsilon}{16\eta \cdot 2^{i(p/16-1/8)} \log\frac{1}{\varepsilon^2}}$ and $n_i = \min\left(\left(\frac{16\alpha p \log n}{\varepsilon^{1-2/p}}\right)^{(2p)/(p-2)}, \frac{10\gamma n}{2^i}\right)$ for a sufficiently large constant η. Then the total space for the CountHH instances across all i for a fixed r is at most

$$C_1 \log n + \left(C_1 \log^2 n\right) \cdot \sum_{i=1}^{\alpha \log n} \left(\frac{C_2 n^{1-2/p}}{\varepsilon^{4/p}} \cdot \frac{2^{i(1/4-1/(2p))}}{2^{i(1-2/p)}} \log^2\frac{1}{\varepsilon} + \frac{C_2 \log^2 n}{\varepsilon^2}\right),$$

for some positive constants $C_1, C_2 > 0$. Observe that $\sum_{i=1}^{\infty} \frac{2^{i(1/4-1/(2p))}}{2^{i(1-2/p)}} = \sum_{i=1}^{\infty} \frac{1}{2^{3i(1-2/p)/4}}$ is a geometric series that is upper bounded by an absolute constant. Hence, the total space across all i for a fixed r is at most $\mathcal{O}\left(\frac{1}{\varepsilon^{4/p}} \cdot n^{1-2/p} \log^2 n \log^2\frac{1}{\varepsilon} + \frac{1}{\varepsilon^2}\log^5 n\right)$ and the total space across all $r \le \mathcal{O}(\log\log n)$ is

$$\mathcal{O}\left(\frac{1}{\varepsilon^{4/p}} \cdot n^{1-2/p} \log^2 n \log^2\frac{1}{\varepsilon} \log\log n + \frac{1}{\varepsilon^2}\log^5 n \log\log n\right),$$

which is $\tilde{\mathcal{O}}\left(\frac{1}{\varepsilon^{4/p}} \cdot n^{1-2/p}\right)$ space in total, since $n \ge \frac{1}{\varepsilon^2}$ implies $\frac{1}{\varepsilon^{4/p}} \cdot n^{1-2/p} \ge \frac{1}{\varepsilon^2}$. ◀

3 F_p Estimation for $p > 2$ in Two-Pass Streams

In this section, we consider two-pass algorithms for F_p estimation, with $p > 2$. For turnstile streams, we require the guarantees of the well-known CountSketch algorithm for finding heavy-hitters.

▶ **Theorem 3.1** ([15]). *There exists an algorithm CountSketch that reports all items $i \in [n]$ such that $f_i \ge \varepsilon L_2$ and no items $j \in [n]$ such that $f_j \le \frac{\varepsilon}{2} \cdot L_2$ in the turnstile streaming algorithm. The algorithm uses $\mathcal{O}\left(\frac{1}{\varepsilon^2}\log^2 n\right)$ bits of space and succeeds with probability $1 - \frac{1}{\mathrm{poly}(n)}$.*

For insertion-only streams, we use the more space-efficient BPTree algorithm for finding heavy-hitters.

▶ **Theorem 3.2** ([8]). *There exists an algorithm BPTree that reports all items $i \in [n]$ such that $f_i \geq \varepsilon L_2$ and no items $j \in [n]$ such that $f_j \leq \frac{\varepsilon}{2} \cdot L_2$ in the turnstile streaming algorithm. The algorithm uses $\mathcal{O}\left(\frac{1}{\varepsilon^2} \log n\right)$ bits of space and succeeds with probability 0.99.*

Recall that the main idea of our one-pass algorithm in the random-order insertion-only streaming model was to subsample at different rates and find $(1+\varepsilon)$-approximations to the frequencies of the heavy-hitters in each subsampling rate, which are then used to form estimates of the contributions of each level set and ultimately estimate of the frequency moment. The random-order setting crucially allowed us to obtain $(1+\varepsilon)$-approximations to the frequencies of the heavy-hitters. By contrast, in an adversarial-order setting, by the time a possible heavy-hitter is detected, a significant fraction of its frequency may have already appeared in the stream if we use a heavy-hitter algorithm with space dependency $\frac{1}{\varepsilon^2}$. Fortunately, in a two-pass setting, we can use the first pass to identify possible heavy-hitters and the second pass to find their frequencies. We give the full details in Algorithm 2, where the subroutine HeavyHitters denotes the algorithm CountSketch of Theorem 3.1 for turnstile streams and BPTree of Theorem 3.2 for insertion-only streams.

Algorithm 2 F_p Estimation on Two-Pass Streams, $p > 2$.

Input: Accuracy parameter $\varepsilon \in (0,1)$, $\widehat{F_p} \leq F_p \leq 1.01 \cdot \widehat{F_p}$
Output: $(1+\varepsilon)$-approximation to F_p

Let $\eta > \sum_{j \geq 0} 2^{-j(p/16-1/8)}$ be a sufficiently large constant.
$\gamma \leftarrow 2^{11}$, $n_i \leftarrow \frac{10\gamma n}{2^i}$, $\varepsilon_i = \frac{\varepsilon}{16\eta \cdot 2^{i(p/16-1/8)} \log \frac{1}{\varepsilon^2}}$
Let I_i^r be a (nested) subset of $[n]$ subsampled at rate $p_i := \min(1, 2^{-i}\gamma)$.
for first pass $i \in [\alpha \log n]$, $r \leq \mathcal{O}(\log \log n)$: **do**
    Let H_i^r be the output of HeavyHitters with threshold parameter $\frac{(\varepsilon_i)^{2/p}}{80\gamma} \cdot \left(\frac{1}{n_i}\right)^{1/2-1/p}$ on the substream induced by I_i^r.
for second pass do
    Track the frequency f_k for any coordinate $k \in \cup H_i^r$.
    Let S_i^r be the items $k \in [n]$ with $f_k^p \in \left[\frac{\widehat{F_p}}{2^i}, \frac{2\widehat{F_p}}{2^i}\right]$.
    $\ell_i \leftarrow \min\{k : 2^k > 2^i \cdot \varepsilon_k^2\}$
    $D_i^r \leftarrow \frac{1}{p_{\ell_i}} \cdot \left(\sum_{k \in S_i^r} f_k^p\right)$
    $\widehat{C_i} \leftarrow \text{median}_r\, D_i^r$
return $\tilde{F}_p := \sum_i \widehat{C_i}$

Using CountSketch as the subroutine for HeavyHitters on two-pass turnstile streams and BPTree as the subroutine for HeavyHitters on two-pass insertion-only streams, we obtain the following guarantees for Algorithm 2. We first show that there exists a subsampling rate such that items in each level set will be reported as a heavy-hitter. The proof is similar to the proof of Lemma 2.4, but we no longer require each reported item to also be reported with a $(1+\varepsilon)$-approximation to their frequency. In fact, this cannot be done in a single pass; we will instead track their frequencies in the second pass.

▶ **Lemma 3.3.** *Let $\varepsilon \in (0,1)$, Λ_i be a fixed level set, and let $\ell := \ell_i := \min\{k : 2^k > 2^i \cdot \varepsilon_k^2\}$. For a fixed r, let $\mathcal{E}_1$ be the event that $|I_\ell^r| \leq \frac{10\gamma n}{2^\ell}$ and let $\mathcal{E}_2$ be the event that $F_p(I_\ell^r) \leq \frac{10\gamma F_p}{2^\ell}$. Conditioned on $\mathcal{E}_1$ and $\mathcal{E}_2$, then HeavyHitters reports $j \in S_i^r$ for each $j \in \Lambda_i \cap I_\ell^r$ with probability at least $1 - \frac{1}{\text{poly}(n)}$.*

Proof. We consider casework on the value of i. First suppose $2^i \cdot \varepsilon^2 < \gamma$, so that $\ell := \ell_i = \min\{k : 2^k > 2^i \cdot \varepsilon_k^2\} \leq \log \gamma$ since $\varepsilon_k \leq \varepsilon$ for all k. By Definition 2.2 of the level sets, each item $j \in \Lambda_i$ satisfies $f_j^p \in \left[\frac{\zeta \cdot \widehat{F_p}}{2^i}, \frac{2\zeta \cdot \widehat{F_p}}{2^i}\right]$. We also have $\widehat{F_p} \leq F_p \leq 1.01 \cdot \widehat{F_p}$ and $n^{1/2-1/p} \cdot F_p^{1/p} \geq F_2^{1/2}$. Thus, each item in $j \in \Lambda_i$ satisfies

$$f_j \geq \frac{\varepsilon^{2/p}}{1.01^{1/p}} \cdot F_p^{1/p} \geq \frac{\varepsilon^{2/p}}{1.01^{1/p} n^{1/2-1/p}} \cdot F_2^{1/2},$$

so that $f_j^2 \geq \frac{\varepsilon^{4/p}}{1.01 n^{1/2-1/p}} F_2$. Then by Theorem 3.1 or Theorem 3.2, each item $j \in \Lambda_i$ is reported by CountSketch or BPTree with threshold $\frac{(\varepsilon_\ell)^{2/p}}{80\gamma}$, as $\varepsilon_\ell \leq \varepsilon$ and $n_\ell = n$.

Otherwise, suppose $2^i \cdot \varepsilon^2 \geq \gamma$, so that $\ell := \ell_i = \min\{k : 2^k > 2^i \cdot \varepsilon_k^2\} \geq \log \gamma$ and $p_\ell := \min(1, 2^{-\ell}\gamma) \geq \frac{\gamma}{2 \cdot 2^i \varepsilon_\ell^2}$. Again by Definition 2.2 of the level sets, each item $j \in \Lambda_i$ satisfies $f_j^p \in \left[\frac{\zeta \cdot \widehat{F_p}}{2^i}, \frac{2\zeta \cdot \widehat{F_p}}{2^i}\right]$. If $j \notin I_\ell^r$, then j certainly will not be reported by HeavyHitters (regardless of whether HeavyHitters is CountSketch or BPTree). Hence, we assume that $j \in I_\ell^r$.

Conditioning on the event $\mathcal{E}_2$,

$$F_p(I_\ell^r) \leq \frac{10\gamma F_p}{2^\ell} \leq \frac{20\gamma F_p}{2^i \varepsilon^2}.$$

Given $\widehat{F_p} \leq F_p \leq 1.01 \cdot \widehat{F_p}$, then $f_j^p \geq \frac{\varepsilon^2}{80\gamma} \cdot F_p(I_\ell^r)$. Therefore,

$$f_j \geq \frac{\varepsilon^{2/p}}{(80\gamma)^{1/p}} \cdot F_p^{1/p}(I_\ell^r).$$

Rather than applying the inequality $n^{1/2-1/p} \cdot F_p^{1/p} \geq F_2^{1/2}$, we observe that conditioning on the event $\mathcal{E}_1$, it follows that $|I_\ell^r| \leq n_\ell = \frac{10\gamma n}{2^\ell}$. Thus, the frequency vector defined by the substream I_ℓ^r has significantly smaller support size than n, which we can leverage to use a heavy-hitter algorithm with a lower threshold. We have

$$f_j \geq \frac{\varepsilon^{2/p}}{(80\gamma)^{1/p}} \cdot \left(\frac{1}{n_\ell}\right)^{1/2-1/p} \cdot F_2^{1/2}(I_\ell^r).$$

Therefore by Theorem 3.1 or Theorem 3.2, each item $j \in \Lambda_i \cap I_\ell^r$ will be reported by H_ℓ^r output by HeavyHitters with threshold $\frac{(\varepsilon_\ell)^{2/p}}{80\gamma} \cdot \left(\frac{1}{n_\ell}\right)^{1/2-1/p}$, since $\varepsilon_\ell \leq \varepsilon$. ◀

We now show our main correctness statement for our two-pass algorithms. Lemma 3.4 proves that the output $\tilde{F}_p$ of our algorithm gives a $(1+\varepsilon)$-approximation to the F_p moment of the underlying frequency vector. Although the guarantees of Lemma 3.4 are similar to the guarantees of Lemma 2.5 are similar, the proof of Lemma 3.4 is much simpler. We show that we obtain $(1+\varepsilon)$-approximations to the contributions of each significant level set, thus obtaining a $(1+\varepsilon)$-approximation to the F_p moment, since the contributions of the insignificant level sets can be ignored. Unlike Lemma 2.5, we need not concern about an idealized process since the frequency of each heavy-hitter is exactly tracked in the second pass of the algorithm over the data stream, so no heavy-hitters can be accidentally misclassified into an incorrect level set.

▶ **Lemma 3.4.** *With probability at least $\frac{2}{3}$, we have that $|\tilde{F}_p - F_p| \leq \varepsilon \cdot F_p$.*

Proof. Let Λ_i be a fixed level set and $\ell := \min\{k : 2^k > 2^i \cdot \varepsilon_k^2\}$. Let $\mathcal{E}_1$ be the event that $|I_\ell^r| \leq \frac{10\gamma n}{2^\ell}$ and let $\mathcal{E}_2$ be the event that $F_p(I_\ell^r) \leq \frac{10\gamma F_p}{2^\ell}$. By Lemma 3.3, CountSketch returns each $k \in \Lambda_i \cap I_\ell^r$ in S_i^r, conditioned on $\mathcal{E}_1$ and $\mathcal{E}_2$. Thus in the second pass, Algorithm 2 tracks the contribution f_k^p explicitly, by tracking f_k. We first analyze the approximation $\widehat{C_i^r}$ to C_i for a given set of subsamples, where we define

$$D_i^r := \frac{1}{p_\ell} \cdot \sum_{k \in \Lambda_i \cap I_\ell^r} f_k^p.$$

so that $\widehat{C_i} = \text{median}_r\, D_i^r$. Conditioned on $\mathcal{E}_1$ and $\mathcal{E}_2$, we note that

$$\mathbb{E}\left[D_i^r\right] = \frac{1}{p_\ell} \cdot \sum_{k \in \Lambda_i} p_\ell \cdot f_k^p = C_i.$$

We also have

$$\text{Var}(D_i^r) = \frac{1}{p_\ell^2} \cdot \sum_{k \in \Lambda_i} p_\ell f_k^{2p} \leq \sum_{k \in \Lambda_i} \frac{2 \cdot 2^i \varepsilon_\ell^2}{\gamma} \cdot f_k^{2p},$$

since $p_\ell \geq \frac{\gamma}{2 \cdot 2^i \varepsilon_\ell^2}$. Observe that for each $k \in \Lambda_i$, we have $f_k^{2p} \leq \frac{16(F_p)^2}{2^{2i}}$ and $\frac{|\Lambda_i|}{2^i} \leq \phi_i \leq 1$. Thus for $\gamma = 2^{11}$,

$$\text{Var}(D_i^r) \leq |\Lambda_i| \cdot \frac{2 \cdot 2^i \varepsilon_\ell^2}{\gamma} \cdot \frac{16(F_p)^2}{2^{2i}} \leq \phi_i \varepsilon_\ell^2 (F_p)^2 \leq \frac{\varepsilon_\ell^2}{64}(F_p)^2.$$

Hence, by Chebyshev's inequality, we have that

$$\mathbf{Pr}\left[|D_i^r - C_i| \geq \frac{\varepsilon_\ell}{2} \cdot F_p\right] \leq \frac{1}{16}.$$

In summary, D_i^r gives an approximation to C_i with additive error at most $\frac{\varepsilon_\ell}{2} \cdot F_p$ with probability at least $\frac{15}{16}$, conditioned on the events $\mathcal{E}_1$ and $\mathcal{E}_2$, and the correctness of the subroutine CountSketch. Since CountSketch fails with probability at most $1 - \frac{1}{\text{poly}(n)}$ by Theorem 3.1, then by a union bound, we have that conditioned on $\mathcal{E}_1$ and $\mathcal{E}_2$,

$$\mathbf{Pr}\left[|D_i^r - C_i| \leq \left(\frac{\varepsilon}{4\alpha \log n} + \varepsilon_\ell\right) \cdot F_p\right] \geq \frac{7}{8}.$$

For a fixed r, let $\mathcal{E}_1$ be the event that $|I_\ell^r| \leq \frac{10\gamma n}{2^\ell}$ and let $\mathcal{E}_2$ be the event that $F_p(I_\ell^r) \leq \frac{10\gamma F_p}{2^\ell}$. Recall that I_ℓ^r is a nested subset of $[n]$ subsampled at rate $p_\ell := \min(1, 2^{-\ell}\gamma)$, so that $\mathbb{E}\left[|I_\ell^r|\right] \leq \frac{\gamma n}{2^\ell}$. Thus by Markov's inequality, $\mathbf{Pr}\left[\mathcal{E}_1\right] \geq \frac{9}{10}$. Similarly, $\mathbb{E}\left[F_p(I_\ell^r)\right] \leq \frac{\gamma F_p}{2^\ell}$. Thus by Markov's inequality, $\mathbf{Pr}\left[\mathcal{E}_2\right] \geq \frac{9}{10}$. By a union bound, we first have $\mathbf{Pr}\left[\mathcal{E}_1 \wedge \mathcal{E}_2\right] \geq \frac{8}{10}$. Applying another union bound, we have that $|D_i^r - C_i| \leq \frac{\varepsilon_\ell}{2} \cdot F_p$ for a fixed r, with probability at least $\frac{5}{8}$. Since $\widehat{C_i} = \text{median}_r\, D_i^r$ across $r \leq \mathcal{O}\left(\log \log n\right)$ iterations, then we have that $|\widehat{C_i} - C_i| \leq \frac{\varepsilon_\ell}{2} \cdot F_p$ with probability at least $1 - \frac{1}{\text{polylog}(n)}$.

By a union bound over the indices $i \in [\alpha \log n]$, corresponding to the $\alpha \log n$ level sets, then with probability at least $1 - \frac{1}{\text{polylog}(n)}$, we have that $|\widehat{C_i} - C_i| \leq \frac{\varepsilon_\ell}{2} \cdot F_p$ simultaneously for all $i \in [\alpha \log n]$, where $\ell_i := \min\{k : 2^k > 2^i \cdot \varepsilon_k^2\}$. We form our estimate $\tilde{F}_p$ to F_p by setting $\tilde{F}_p := \sum_i \widehat{C_i}$ and we have $F_p = \sum_i C_i$. Since $i \in [\alpha \log n]$, $\ell_i := \min\{k : 2^k > 2^i \cdot \varepsilon_k^2\}$, and $\varepsilon_{\ell_i} = \frac{\varepsilon}{16\eta \cdot 2^{\ell_i(p/16 - 1/8)} \log \frac{1}{\varepsilon^2}}$, then

$$|\tilde{F}_p - F_p| = \left|\sum_i \widehat{C_i} - \sum_i C_i\right| \le \sum_i |\widehat{C_i} - C_i| \le \sum_i \left(\frac{\varepsilon}{4\alpha \log n} + \varepsilon_{\ell_i}\right) \cdot F_p$$
$$\le \sum_{i \in [\alpha \log n]} \frac{\varepsilon}{4\alpha \log n} \cdot F_p + \sum_{i:\ell_i \le \log \frac{1}{\varepsilon^2}} \varepsilon_{\ell_i} \cdot F_p + \sum_{i:\ell_i > \log \frac{1}{\varepsilon^2}} \varepsilon_{\ell_i} \cdot F_p$$
$$\le \frac{\varepsilon}{4} \cdot F_p + \log \frac{1}{\varepsilon^2} \cdot \frac{\varepsilon}{16 \log \frac{1}{\varepsilon^2}} \cdot F_p + \frac{\varepsilon}{16} \cdot F_p,$$

where the last bound on the last term follows from $\eta > \sum_{j \ge 0} 2^{-j(p/16 - 1/8)}$. Therefore with probability at least $1 - \frac{1}{\text{polylog}(n)}$, we have that $|\tilde{F}_p - F_p| \le \frac{\varepsilon}{2} \cdot F_p$. ◀

We now justify the full guarantees claimed by Theorem 1.2. We first handle two passes over a turnstile stream.

▶ **Theorem 3.5.** *For $p > 2$, there exists a two-pass turnstile streaming algorithm that outputs a $(1+\varepsilon)$-approximation to the F_p moment with probability at least $\frac{2}{3}$, while using $\mathcal{O}\left(\frac{1}{\varepsilon^{4/p}} \cdot n^{1-2/p} \log^2 n \log\log n\right)$ bits of space.*

Proof. Our analysis is similar to the proof of Theorem 1.1. We again observe that by redefining the level sets according to any upper bound on F_p, we do not require a 1.01-approximation $\widehat{F_p}$ to F_p as the input of Algorithm 2. Thus by Lemma 3.4, there exists an algorithm that outputs a $(1+\varepsilon)$-approximation to the F_p moment on two pass turnstile streams and it remains to analyze the space complexity. By Theorem 3.1, CountSketch with threshold ε requires $\mathcal{O}\left(\frac{1}{\varepsilon^2} \log^2 n\right)$ bits of space to output the indices of the heavy-hitters. Algorithm 2 runs a separate instance of CountSketch with threshold $\frac{(\varepsilon_i)^{2/p}}{80\gamma} \cdot \left(\frac{1}{n_i}\right)^{1/2-1/p}$ to output a set H_i^r for $r \le \mathcal{O}(\log\log n)$, where γ is a sufficiently large constant. Thus the total space in the first pass across all indices i for a fixed r is at most

$$\left(C_1 \log^2 n\right) \cdot \sum_{i \in [\alpha \log n]} \frac{(n_i)^{1-2/p}}{(\varepsilon_i)^{4/p}},$$

for some positive constants $C_1, \alpha > 0$. In particular, we have $n_i = \frac{10\gamma n}{2^i}$ and $\varepsilon_i = \frac{\varepsilon}{16\eta \cdot 2^{i(p/16-1/8)} \log \frac{1}{\varepsilon^2}}$ for a sufficiently large constant η. Then the total space in the first pass across all indices i for a fixed r is at most

$$\left(C_1 \log^2 n\right) \cdot \sum_{i=1}^{\infty} \frac{C_2 n^{1-2/p}}{\varepsilon^{4/p}} \cdot \frac{2^{i(1/4-1/(2p))}}{2^{i(1-2/p)}},$$

for some positive constants $C_1, C_2 > 0$. Since $\sum_{i=1}^{\infty} \frac{2^{i(1/4-1/(2p))}}{2^{i(1-2/p)}} = \sum_{i=1}^{\infty} \frac{1}{2^{3i(1-2/p)/4}}$ is a geometric series that is upper bounded by a fixed constant, the total space in the first pass across all i for a fixed r is $\mathcal{O}\left(\frac{1}{\varepsilon^{4/p}} \cdot n^{1-2/p} \log^2 n\right)$. Because $r \in \mathcal{O}(\log\log n)$, then the total space is $\mathcal{O}\left(\frac{1}{\varepsilon^{4/p}} \cdot n^{1-2/p} \log^2 n \log\log n\right)$.

In the second pass, we track the frequencies of each item reported by some instance of CountSketch. Since at most $\mathcal{O}\left(\frac{1}{\varepsilon^{4/p}} \cdot n^{1-2/p} \log\log n\right)$ indices can be reported across all instances of CountSketch and $\mathcal{O}(\log n)$ bits of space can be used to track the frequency of each reported index, then the total space for the second pass is at most $\mathcal{O}\left(\frac{1}{\varepsilon^{4/p}} \cdot n^{1-2/p} \log n \log\log n\right)$. Thus, the total space is $\mathcal{O}\left(\frac{1}{\varepsilon^{4/p}} \cdot n^{1-2/p} \log^2 n \log\log n\right)$. ◀

Finally, we note that to report at most $\mathcal{O}\left(\frac{1}{\varepsilon^{4/p}} \cdot n^{1-2/p} \log\log n\right)$ indices of possible heavy-hitters in turnstile streams, CountSketch uses at most $\mathcal{O}\left(\frac{1}{\varepsilon^{4/p}} \cdot n^{1-2/p} \log^2 n \log\log n\right)$ space. By the same reasoning, we use space $\mathcal{O}\left(\frac{1}{\varepsilon^{4/p}} \cdot n^{1-2/p} \log n \log\log n\right)$ in insertion-only streams by using the more space efficient BPTree.

▶ **Theorem 3.6.** *For $p > 2$, there exists a two-pass insertion-only streaming algorithm that outputs a $(1+\varepsilon)$-approximation to the F_p moment with probability at least $\frac{2}{3}$, while using $\mathcal{O}\left(\frac{1}{\varepsilon^{4/p}} \cdot n^{1-2/p} \log n \log\log n\right)$ bits of space.*

Proof. The proof of correctness is exactly the same as that of Theorem 3.5 since using BPTree as the subroutine for HeavyHitters rather than CountSketch offers the same guarantee for insertion-only streams. By Theorem 3.2, BPTree with threshold ε requires $\mathcal{O}\left(\frac{1}{\varepsilon^2} \log n\right)$ bits of space to output the indices of the heavy-hitters. To analyze the space complexity, note that Algorithm 2 runs a separate instance of BPTree with threshold $\frac{(\varepsilon_i)^{2/p}}{80\gamma} \cdot \left(\frac{1}{n_i}\right)^{1/2-1/p}$ to output a set H_i^r for $r \leq \mathcal{O}(\log\log n)$, where γ is a sufficiently large constant. Hence, the first pass across all indices i for a fixed r uses space at most

$$(C_1 \log n) \cdot \sum_{i \in [\alpha \log n]} \frac{(n_i)^{1-2/p}}{(\varepsilon_i)^{4/p}},$$

for some absolute constants $C_1, \alpha > 0$. Since $n_i = \frac{10\gamma n}{2^i}$ and $\varepsilon_i = \frac{\varepsilon}{16\eta \cdot 2^{i(p/16-1/8)} \log \frac{1}{\varepsilon^2}}$ for a sufficiently large constant η, the first pass uses space at most

$$(C_1 \log n) \cdot \sum_{i=1}^{\infty} \frac{C_2 n^{1-2/p}}{\varepsilon^{4/p}} \cdot \frac{2^{i(1/4-1/(2p))}}{2^{i(1-2/p)}},$$

for some absolute constants $C_1, C_2 > 0$, across all indices i for a fixed r. As $\sum_{i=1}^{\infty} \frac{2^{i(1/4-1/(2p))}}{2^{i(1-2/p)}} = \sum_{i=1}^{\infty} \frac{1}{2^{3i(1-2/p)/4}}$ is a geometric series that is upper bounded by some constant, then the total space in the first pass is $\mathcal{O}\left(\frac{1}{\varepsilon^{4/p}} \cdot n^{1-2/p} \log n\right)$ across all indices i for a fixed r. Since $r \leq \mathcal{O}(\log\log n)$, then the total space in the first pass is $\mathcal{O}\left(\frac{1}{\varepsilon^{4/p}} \cdot n^{1-2/p} \log n \log\log n\right)$.

The second pass tracks the frequencies of each item reported by some instance of BPTree. Since at most $\mathcal{O}\left(\frac{1}{\varepsilon^{4/p}} \cdot n^{1-2/p} \log\log n\right)$ indices can be reported across all instances of BPTree and $\mathcal{O}(\log n)$ bits of space can be used to track the frequency of each reported index, then the total space for the second pass is at most $\mathcal{O}\left(\frac{1}{\varepsilon^{4/p}} \cdot n^{1-2/p} \log n \log\log n\right)$. Thus, the total space is $\mathcal{O}\left(\frac{1}{\varepsilon^{4/p}} \cdot n^{1-2/p} \log n \log\log n\right)$. ◀

4 Lower Bounds

In this section, we first consider the standard *blackboard* communication model, where a number of players each have a local input and the goal is to solve some predetermined communication problem by sending messages to a shared medium. Each player is assumed to have access to an unlimited amount of private randomness. The sequence of messages on the shared blackboard is called the *transcript* and the maximum length of the transcript over all inputs is the *communication cost* of a given protocol. The communication complexity of f, denoted by $R_\delta(f)$, is the minimal communication cost of all protocols that succeed with probability at least $1-\delta$ for all legal inputs to f.

▶ **Definition 4.1.** *In the (t, ε, n)-player set disjointness estimation problem (t, ε, n) − DisjInfty, there are $t+1$ players $P_1, \ldots, P_{t+1}$ with private coins in the standard blackboard model. For $s \in [t]$, each player P_s receives a vector $\mathbf{v}_s \in \{0,1\}^n$ and player P_{t+1} receives both an index $j \in [n]$ and a bit $c \in \{0,1\}$. For $\mathbf{u} = \sum_{s\in[t]} \mathbf{v}_s$, the inputs are promised to satisfy $u_i \leq 1$ for each $i \neq j$ and either $u_j = 1$ or $u_j = t$. With probability at least $\frac{9}{10}$, P_{t+1} must differentiate between the three possible input cases:*
(1) $u_j + \frac{ct}{\varepsilon} \leq t$
(2) $u_j + \frac{ct}{\varepsilon} \in \left\{\frac{t}{\varepsilon}, \frac{t}{\varepsilon} + 1\right\}$
(3) $u_j + \frac{ct}{\varepsilon} = (1+\varepsilon)\frac{t}{\varepsilon}$,
where $\varepsilon \in (0,1)$. We call coordinate $j \in [n]$ the spike *location.*

Direct sum embedding. We use the direct sum technique by showing that even in the case where $u_i \leq 1$ for all $i \in [n]$, the information cost of the players is sufficiently high. We describe the embedding performed by each player to sample coordinates independently conditioned on the auxiliary variable D. We define the auxiliary variable $D = (D_1, \ldots, D_n)$ by first defining a distribution μ for D_i and $v_{s,i}$, for each fixed coordinate $i \in [n]$ and across all players $s \in [t]$:
(1) $D_i \sim [t]$ uniformly at random, so that D_i determines a player whose input bit will be randomized while the remaining players have input bit zero.
(2) Conditioned on $D_i = s$, then each player P_r with $r \neq s$ sets $v_{r,i} = 0$ while player P_s independently and uniformly sets $v_{s,i} \in \{0,1\}$.
We define the distribution $\zeta := \mu^n$ so that the auxiliary random variable $D = (D_1, \ldots, D_n)$ is the vector whose i-th coordinate determines the player P_{D_i} whose i-th bit $v_{D_i,i}$ in their input vector will vary, while the other players P_s have i-th bit $v_{s,i}$ zero in their vectors, for $s \neq D_i$. Observe that under the distribution ζ, we indeed have $u_i \leq 1$ for each coordinate $i \in [n]$ of $\mathbf{u} = \sum_{s\in[t]} \mathbf{v}_s$. Finally, we fix the input $c = 0$ to player P_{t+1} and pick $j \in [n]$ according to any arbitrary distribution.

▶ **Theorem 4.2.** *The total communication complexity for the (t, ε, n)-player set disjointness estimation problem is $\Omega\left(\frac{n}{t}\right)$.*

We remark that the communication complexity of Theorem 4.2 matches the communication complexity of t-player set disjointness. However, since the problem requires correctness on all inputs, then we can distinguish between the possible input cases by focusing on the specific coordinate j given to player P_{t+1}. By contrast, the reduction of [18] from t-player set disjointness requires an algorithm to "test" all coordinates $i \in [n]$ for the spike location. Thus the reduction requires that an F_p moment estimation algorithm succeeds with probability $1 - \frac{1}{\mathrm{poly}(n)}$, thereby losing a multiplicative $\mathcal{O}(\log n)$ factor and achieving $\Omega\left(\frac{n^{1-2/p}}{\varepsilon^2 \log n}\right)$ in the space lower bound for F_p moment estimation. Since our communication problem gives the specific spike location as input, our reduction only requires an F_p moment estimation algorithm that succeeds with constant probability. We remark that a similar technique was used in [30] for the L_∞ promise problem.

Reduction. Let $t = \Theta\left(\frac{1}{\varepsilon} \cdot n^{1/p}\right)$. We reduce $(1+\varepsilon)$-approximate F_p moment estimation to an instance of (t, ε) − DisjInfty as follows. Let $\mathcal{A}$ be any fixed randomized one-pass insertion-only streaming algorithm that outputs a $\left(1 + \frac{\varepsilon}{3}\right)$-approximation to the F_p moment of the underlying frequency vector with probability at least $\frac{2}{3}$. Recall that the first t players each receive a vector $\mathbf{v}_s$ with $s \in [t]$ and player P_{t+1} receives both a special index $j \in [n]$ and a bit $c \in \{0,1\}$.

- For each $s \in [t]$, player P_s takes the state of the algorithm $\mathcal{A}$, inserts the coordinates of vector $\mathbf{v}_s$, and passes the state of the algorithm to player P_{s+1}.
- Player P_{t+1} takes the state of $\mathcal{A}$, adds the vector $\frac{ct}{\varepsilon} \cdot \mathbf{e}_j$, where $\mathbf{e}_j$ is the elementary vector with a one in position j and zeros elsewhere, and then queries the state of $\mathcal{A}$ to obtain a $\left(1+\frac{\varepsilon}{3}\right)$-approximation to the F_p moment of underlying frequency vector.

▶ **Theorem 4.3.** *For any constant $p > 2$ and parameter $\varepsilon = \Omega\left(\frac{1}{n^{1/p}}\right)$, any one-pass insertion-only streaming algorithm that outputs a $(1+\varepsilon)$-approximation to the F_p moment of an underlying frequency vector with probability at least $\frac{9}{10}$ requires $\Omega\left(\frac{1}{\varepsilon^2} \cdot n^{1-2/p}\right)$ bits of space.*

Proof. Player P_{t+1} receives the state of $\mathcal{A}$ on the input $\mathbf{u} = \sum_{s\in[t]} \mathbf{v}_s$ and induces a frequency vector $\mathbf{x} := \mathbf{u} + \frac{ct}{\varepsilon} \cdot \mathbf{e}_j$. Recall that the inputs are promised to satisfy $u_i \leq 1$ for each $i \neq j$ and either $u_j = 1$ or $u_j = t$. If $c = 0$, then $\mathbf{x} = \mathbf{u}$ so that for a constant $C > 0$ and $t = \frac{C}{\varepsilon} \cdot n^{1/p}$,

$$\|\mathbf{x}\|_p^p \leq F_0 + t^p \leq n + \frac{C^p}{\varepsilon^p} \cdot n.$$

If $c = 1$ and $u_j \leq 1$, then for $t = \frac{C}{\varepsilon} \cdot n^{1/p}$, we have $\|\mathbf{x}\|_p^p \geq \left(\frac{t}{\varepsilon}\right)^p = \frac{C^p}{\varepsilon^{2p}} \cdot n$ and thus

$$\|\mathbf{x}\|_p^p \leq F_0 + \left(1+\frac{t}{\varepsilon}\right)^p \leq n + p + \frac{p \cdot C^p}{\varepsilon^{2p}} \cdot n.$$

Finally, if $c = 1$ and $u_j = t$, then

$$\|\mathbf{x}\|_p^p \geq \left(t+\frac{t}{\varepsilon}\right)^p = \left(1+\frac{1}{\varepsilon}\right)^p \cdot \frac{C^p}{\varepsilon^{2p}} \cdot n.$$

For sufficiently small $\varepsilon \in (0,1)$ with $\varepsilon = \Omega\left(\frac{1}{n^{1/p}}\right)$ and constant $p > 2$, there exists a constant $C > 0$ such that these three cases are separated by a multiplicative $(1+\varepsilon)$. Since $\mathcal{A}$ outputs a $\left(1+\frac{\varepsilon}{3}\right)$-approximation to the F_p moment of the underlying frequency vector, then player P_{t+1} obtains a $\left(1+\frac{\varepsilon}{3}\right)$-approximation to $\|\mathbf{x}\|_p^p$ and can differentiate between the three cases, thus solving $(t,\varepsilon) - \textsc{DisjInfty}$ with probability at least $\frac{9}{10}$. By Theorem 4.2, $\mathcal{A}$ uses $\Omega\left(\frac{n}{t}\right)$ total communication across the $t+1$ players. Therefore, the space complexity of $\mathcal{A}$ is $\Omega\left(\frac{n}{t^2}\right) = \Omega\left(\frac{1}{\varepsilon^2} \cdot n^{1-2/p}\right)$ for $t = \Theta\left(\frac{1}{\varepsilon} \cdot n^{1/p}\right)$. The result then follows from rescaling ε. ◀

References

1. Noga Alon, Yossi Matias, and Mario Szegedy. The space complexity of approximating the frequency moments. *J. Comput. Syst. Sci.*, 58(1):137–147, 1999.
2. Alexandr Andoni. High frequency moments via max-stability. In *IEEE International Conference on Acoustics, Speech and Signal Processing, ICASSP*, pages 6364–6368, 2017.
3. Alexandr Andoni, Robert Krauthgamer, and Krzysztof Onak. Streaming algorithms via precision sampling. In *IEEE 52nd Annual Symposium on Foundations of Computer Science, FOCS*, pages 363–372, 2011.
4. Ziv Bar-Yossef, T. S. Jayram, Ravi Kumar, and D. Sivakumar. An information statistics approach to data stream and communication complexity. *J. Comput. Syst. Sci.*, 68(4):702–732, 2004.
5. Lakshminath Bhuvanagiri, Sumit Ganguly, Deepanjan Kesh, and Chandan Saha. Simpler algorithm for estimating frequency moments of data streams. In *Proceedings of the Seventeenth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA*, pages 708–713, 2006.
6. Jaroslaw Blasiok, Jian Ding, and Jelani Nelson. Continuous monitoring of ℓ_p norms in data streams. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques, APPROX/RANDOM*, volume 81, pages 32:1–32:13, 2017.

7 Mark Braverman, Sumegha Garg, and David P. Woodruff. The coin problem with applications to data streams. *Electron. Colloquium Comput. Complex.*, 27:139, 2020.

8 Vladimir Braverman, Stephen R. Chestnut, Nikita Ivkin, Jelani Nelson, Zhengyu Wang, and David P. Woodruff. Bptree: An ℓ_2 heavy hitters algorithm using constant memory. In *Proceedings of the 36th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems, PODS*, pages 361–376, 2017.

9 Vladimir Braverman, Jonathan Katzman, Charles Seidell, and Gregory Vorsanger. An optimal algorithm for large frequency moments using $O(n^{1-2/k})$ bits. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques, APPROX/RANDOM*, pages 531–544, 2014.

10 Vladimir Braverman and Rafail Ostrovsky. Approximating large frequency moments with pick-and-drop sampling. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques - 16th International Workshop, APPROX, and 17th International Workshop, RANDOM. Proceedings*, 2013.

11 Vladimir Braverman, Emanuele Viola, David P. Woodruff, and Lin F. Yang. Revisiting frequency moment estimation in random order streams. In *45th International Colloquium on Automata, Languages, and Programming, ICALP*, pages 25:1–25:14, 2018.

12 Amit Chakrabarti, Graham Cormode, and Andrew McGregor. Robust lower bounds for communication and stream computation. *Theory Comput.*, 12(1):1–35, 2016.

13 Amit Chakrabarti, T. S. Jayram, and Mihai Patrascu. Tight lower bounds for selection in randomly ordered streams. In Shang-Hua Teng, editor, *Proceedings of the Nineteenth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA*, pages 720–729, 2008.

14 Amit Chakrabarti, Subhash Khot, and Xiaodong Sun. Near-optimal lower bounds on the multi-party communication complexity of set disjointness. In *18th Annual IEEE Conference on Computational Complexity*, pages 107–117, 2003.

15 Moses Charikar, Kevin C. Chen, and Martin Farach-Colton. Finding frequent items in data streams. *Theor. Comput. Sci.*, 312(1):3–15, 2004.

16 Erik D. Demaine, Alejandro López-Ortiz, and J. Ian Munro. Frequency estimation of internet packet streams with limited space. In *Algorithms - ESA, 10th Annual European Symposium, Proceedings*, pages 348–360, 2002.

17 Sumit Ganguly. Polynomial estimators for high frequency moments. *CoRR*, abs/1104.4552, 2011. `arXiv:1104.4552`.

18 Sumit Ganguly. A lower bound for estimating high moments of a data stream. *CoRR*, abs/1201.0253, 2012. `arXiv:1201.0253`.

19 Sumit Ganguly and David P. Woodruff. High probability frequency moment sketches. In *45th International Colloquium on Automata, Languages, and Programming, ICALP*, 2018.

20 Sudipto Guha and Zhiyi Huang. Revisiting the direct sum theorem and space lower bounds in random order streams. In *International Colloquium on Automata, Languages, and Programming*, pages 513–524. Springer, 2009.

21 Sudipto Guha and Andrew McGregor. Approximate quantiles and the order of the stream. In *Proceedings of the Twenty-Fifth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, pages 273–279, 2006.

22 Sudipto Guha and Andrew McGregor. Lower bounds for quantile estimation in random-order and multi-pass streaming. In *Automata, Languages and Programming, 34th International Colloquium, ICALP, Proceedings*, pages 704–715, 2007.

23 Piotr Indyk. Stable distributions, pseudorandom generators, embeddings, and data stream computation. *J. ACM*, 53(3):307–323, 2006.

24 Piotr Indyk and David P. Woodruff. Optimal approximations of the frequency moments of data streams. In *Proceedings of the 37th Annual ACM Symposium on Theory of Computing (STOC)*, pages 202–208, 2005.

25 Rajesh Jayaram and David P. Woodruff. Towards optimal moment estimation in streaming and distributed models. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques, APPROX/RANDOM*, pages 29:1–29:21, 2019.

26 Daniel M. Kane, Jelani Nelson, Ely Porat, and David P. Woodruff. Fast moment estimation in data streams in optimal space. In *Proceedings of the 43rd ACM Symposium on Theory of Computing, STOC*, pages 745–754, 2011.

27 Daniel M. Kane, Jelani Nelson, and David P. Woodruff. On the exact space complexity of sketching and streaming small norms. In *Proceedings of the Twenty-First Annual ACM-SIAM Symposium on Discrete Algorithms, SODA*, pages 1161–1178, 2010.

28 Christian Konrad, Frédéric Magniez, and Claire Mathieu. Maximum matching in semi-streaming with few passes. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques - 15th International Workshop, APPROX, and 16th International Workshop, RANDOM*, pages 231–242, 2012.

29 Ping Li. Estimators and tail bounds for dimension reduction in $\ell_\alpha (0 < \alpha \leq 2)$ using stable random projections. In *Proceedings of the Nineteenth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA*, pages 10–19, 2008.

30 Yi Li and David P. Woodruff. A tight lower bound for high frequency moment estimation with small error. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques - 16th International Workshop, APPROX, and 17th International Workshop, RANDOM. Proceedings*, pages 623–638, 2013.

31 Morteza Monemizadeh and David P. Woodruff. 1-pass relative-error L_p-sampling with applications. In *Proceedings of the Twenty-First Annual ACM-SIAM Symposium on Discrete Algorithms, SODA*, pages 1143–1160, 2010.

32 J. Ian Munro and Mike Paterson. Selection and sorting with limited storage. *Theor. Comput. Sci.*, 12:315–323, 1980.

33 David P. Woodruff. Optimal space lower bounds for all frequency moments. In *Proceedings of the Fifteenth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA*, pages 167–175, 2004.

34 David P. Woodruff and Qin Zhang. Tight bounds for distributed functional monitoring. In *Proceedings of the 44th Symposium on Theory of Computing Conference, STOC*, pages 941–960, 2012.

35 David P. Woodruff and Samson Zhou. Tight bounds for adversarially robust streams and sliding windows via difference estimators. *CoRR*, abs/2011.07471, 2020. `arXiv:2011.07471`.

36 David P. Woodruff and Samson Zhou. Separations for estimating large frequency moments on data streams. *CoRR*, abs/2105.03773, 2021. `arXiv:2105.03773`.

Breaking the 2^n Barrier for 5-Coloring and 6-Coloring

Or Zamir ✉
Blavatnik School of Computer Science, Tel Aviv University, Israel

Abstract

The coloring problem (i.e., computing the chromatic number of a graph) can be solved in $O^*(2^n)$ time, as shown by Björklund, Husfeldt and Koivisto in 2009. For $k = 3, 4$, better algorithms are known for the k-coloring problem. 3-coloring can be solved in $O(1.33^n)$ time (Beigel and Eppstein, 2005) and 4-coloring can be solved in $O(1.73^n)$ time (Fomin, Gaspers and Saurabh, 2007). Surprisingly, for $k > 4$ no improvements over the general $O^*(2^n)$ are known. We show that both 5-coloring and 6-coloring can also be solved in $O\left((2-\varepsilon)^n\right)$ time for some $\varepsilon > 0$. As a crucial step, we obtain an exponential improvement for computing the chromatic number of a very large family of graphs. In particular, for any constants $\Delta, \alpha > 0$, the chromatic number of graphs with at least $\alpha \cdot n$ vertices of degree at most Δ can be computed in $O\left((2-\varepsilon)^n\right)$ time, for some $\varepsilon = \varepsilon_{\Delta,\alpha} > 0$. This statement generalizes previous results for bounded-degree graphs (Björklund, Husfeldt, Kaski, and Koivisto, 2010) and graphs with bounded average degree (Golovnev, Kulikov and Mihajlin, 2016). We generalize the aforementioned statement to List Coloring, for which no previous improvements are known even for the case of bounded-degree graphs.

2012 ACM Subject Classification Mathematics of computing → Combinatorial algorithms; Theory of computation → Graph algorithms analysis

Keywords and phrases Algorithms, Graph Algorithms, Graph Coloring

Digital Object Identifier 10.4230/LIPIcs.ICALP.2021.113

Category Track A: Algorithms, Complexity and Games

Related Version *Full Version*: `https://arxiv.org/abs/2007.10790`

Acknowledgements The author would like to deeply thank Noga Alon for important discussions and insights regarding the subset removal lemma, and Haim Kaplan and Uri Zwick for many helpful discussions and comments on the paper. The author would also like to thank anonymous reviewers for helpful comments.

1 Introduction

The problem of k-coloring a graph, or determining the *chromatic number* of a graph (i.e., finding the smallest k for which the graph is k-colorable) is one of the most classic and well studied NP-Complete problems. Computing the chromatic number is listed as one of the first NP-Complete problems in Karp's paper from 1972 [17]. In a similar fashion to k-SAT, the problem of 2-coloring is polynomial, yet k-coloring is NP-complete for every $k \geqslant 3$ (proven independently by Lovász [22] and Stockmeyer [30]). An algorithm solving 3-coloring in sub-exponential time would imply, via the mentioned reductions, that 3-SAT can also be solved in sub-exponential time. It is strongly believed that this is not possible (as stated in a widely believed conjecture called *The Exponential Time Hypothesis* [15]), and thus it is believed that exact algorithms solving k-coloring must be exponential.

There is a substantial and ever-growing body of work exploring exponential-time worst-case algorithms for NP-Complete problems. A 2003 survey of Woeginger [31] covers and refers to dozens of papers exploring such algorithms for many problems including satisfiability,

graph coloring, knapsack, TSP, maximum independent sets and more. Subsequent review article of Fomin and Kaski [10] and book of Fomin and Kratsch [11] further cover the topic of exact exponential-time algorithms.

For satisfiability (commonly abbreviated as SAT), the running time of the trivial algorithm enumerating over all possible assignments is $O^*(2^n)$. No algorithms solving SAT in time $O^*\left((2-\varepsilon)^n\right)$ for any $\varepsilon > 0$ are known, and a popular conjecture called The *Strong* Exponential Time Hypothesis [5] states that no such algorithm exists. On the other hand, it is known that for every fixed k there exists a constant $\varepsilon_k > 0$ such that k-SAT can be solved in $O^*\left((2-\varepsilon_k)^n\right)$ time. A result of this type was first published by Monien and Speckenmeyer in 1985 [23]. A long list of improvements for the values of ε_k were published since, including the celebrated 1998 PPSZ algorithm of Paturi, Pudlák, Saks and Zane [25] and the recent improvement over it by Hansen, Kaplan, Zamir and Zwick [13].

For coloring, on the other hand, the situation is less understood. The trivial algorithm solving k-coloring by enumerating over all possible colorings takes $O^*(k^n)$ time. Thus, it is not even immediately clear that computing the chromatic number of a graph can be done in $O^*(c^n)$ time for a constant c independent of k. In 1976, Lawler [21] introduced the idea of using dynamic-programming to find the minimal number of independent sets covering the graph. The trivial implementation of this idea results in an $O^*(3^n)$ algorithm. More sophisticated bounds on the number of maximal independent sets in a graph and fast algorithms to enumerate over them (Moon and Moser [24], Paull and Unger [26]) resulted in an $O^*(2.4422^n)$ algorithm. This was improved several times (including Eppstein [8] and Byskov [4]), until finally an algorithm computing the chromatic number in $O^*(2^n)$ time was devised by Björklund, Husfeldt and Koivisto in 2009 [3]. This settled an open problem of Woeginger [31]. A relatively recent survey of Husfeldt [14] covers the progress on graph coloring algorithms.

For $k = 3, 4$, better algorithms are known for the k-coloring problem. Schiermeyer [28] showed that 3-coloring can be solved in $O^*(1.415^n)$ time. Beigel and Eppstein [1] gave algorithms solving 3-coloring in $O^*(1.3289^n)$ time and 4-coloring in $O^*(1.8072^n)$ time in 2005. Fomin, Gaspers and Saurabh [9] have improved the running time of 4-coloring to $O^*(1.7272^n)$ in 2007. Unlike the situation in k-SAT, for every $k > 4$ the best known running time for k-coloring is $O^*(2^n)$, the same as computing the chromatic number. Thus, a very fundamental question was left wide open.

▶ **Open Problem 1.** *Can 5-coloring be solved in $O^*\left((2-\varepsilon)^n\right)$ time, for some $\varepsilon > 0$?*

More generally,

▶ **Open Problem 2.** *Can k-coloring be solved in $O^*\left((2-\varepsilon_k)^n\right)$ time, for some $\varepsilon_k > 0$, for every k?*

In our work, we answer Problem 1 affirmatively, the answer extends to 6-coloring as well. We also make steps towards settling Problem 2.

The main technical theorem of our paper follows.

▶ **Definition 3.** *For $0 \leqslant \alpha \leqslant 1$ and $\Delta > 0$ we say that a graph $G = (V(G), E(G))$ is* (α, Δ)-bounded *if it contains at least $\alpha \cdot |V(G)|$ vertices of degree at most Δ.*

▶ **Theorem 4.** *For every $\Delta, \alpha > 0$ there exists $\varepsilon_{\Delta,\alpha} > 0$ such that we can compute the chromatic number of (α, Δ)-bounded graphs in $O\left((2-\varepsilon_{\Delta,\alpha})^n\right)$ time.*

In other words, we can answer Problem 2 affirmatively unless the graph has almost only vertices of super-constant degrees. This theorem generalizes a few previous results. A similar statement for the restricted case of bounded degree graphs was obtained by Björklund et

al. in [2]. Golovnev, Kulikov and Mihajlin [12] used a variant of FFT to get an algorithmic improvement for computing the chromatic number of graphs with bounded average degree. Prior to that, Cygan and Pilipczuk [7] obtained an improvement for the running time required for the Traveling Salesman Problem for graphs with bounded average degree.

It is important to stress that Theorem 4 is much more general than the mentioned results. In particular, (α, Δ)-bounded graphs may have $\Omega(n^2)$ edges (and in turn average degree $\Omega(n)$). The techniques used for graphs with bounded average degree do not extend to this case. Another important difference is that our algorithms extend to finding a coloring of the graph while the mentioned ones solve the decision problem, as later discussed in Section 2.2. Moreover, the generality of Theorem 4 is crucial for our derivation of the reductions resulting in the improvements for 5 and 6 coloring.

In the List Coloring problem (defined formally in Section 2), we are given a graph and lists C_v of colors for each vertex v, and are asked to find a valid coloring of the graph such that each vertex v is colored by some color appearing in its list C_v. In the k-list-coloring problem each list C_v is of size at most k.

List Coloring can also be solved in $O^*(2^n)$ time [3], yet no improvements were known even for the bounded-degree case. We extend Theorem 4 to k-list-coloring, for any constant k, as follows.

▶ **Theorem 5.** *For every $k, \Delta, \alpha > 0$ there exists $\varepsilon_{k,\Delta,\alpha} > 0$ such that we can solve k-list-coloring for (α, Δ)-bounded graphs in $O\left((2 - \varepsilon_{k,\Delta,\alpha})^n\right)$ time, regardless of the size of the universe of colors.*

Using Theorem 4 as a crucial component, we devise the following reductions and corollaries.

▶ **Theorem 6.** *Given an algorithm solving $(k-1)$-list-coloring in time $O\left((2-\varepsilon)^n\right)$ for some constant $\varepsilon > 0$, we can construct an algorithm solving k-coloring in time $O\left((2-\varepsilon')^n\right)$ for some (other) constant $\varepsilon' > 0$. Furthermore, the reduction is deterministic.*

▶ **Theorem 7.** *Given an algorithm solving $(k-2)$-list-coloring in time $O\left((2-\varepsilon)^n\right)$ for some constant $\varepsilon > 0$, we can construct an algorithm solving k-coloring with high probability in time $O\left((2-\varepsilon')^n\right)$ for some (other) constant $\varepsilon' > 0$.*

From which we finally conclude the following, answering Problem 1 affirmatively.

▶ **Theorem 8.** *5-coloring can be solved in time $O\left((2-\varepsilon)^n\right)$ for some constant $\varepsilon > 0$.*

▶ **Theorem 9.** *6-coloring can be solved with high probability in time $O\left((2-\varepsilon)^n\right)$ for some constant $\varepsilon > 0$.*

We note that our 5-coloring algorithm is deterministic, while our 6-coloring algorithm is randomized with an exponentially small one-sided error probability.

As part of our work, we develop a new removal lemma for small subsets. This could be of independent interest. Very roughly, it states that every collection of small sets must have a large sub-collection that can be made pairwise-disjoint by the removal of a small subset of the universe. The exact statement follows.

▶ **Theorem 10.** *Let $\mathcal{F}$ be a collection of subsets of a universe U such that every set $F \in \mathcal{F}$ is of size $|F| \leqslant \Delta$. Let $C > 0$ be any constant. Then, there exist subsets $\mathcal{F}' \subseteq \mathcal{F}$ and $U' \subseteq U$, such that*

- *$|\mathcal{F}'| > \rho(\Delta, C) \cdot |\mathcal{F}| + C \cdot |U'|$, where $\rho(\Delta, C) > 0$ depends only on Δ, C.*
- *The sets in $\mathcal{F}'$ are disjoint when restricted to $U \backslash U'$, i.e., for every $F_1, F_2 \in \mathcal{F}'$ we have $F_1 \cap F_2 \subseteq U'$.*

In the full version of the paper we present an upper bound for the function ρ appearing in Theorem 10. This upper bound implies that the constant ε we can obtain using our technique must be very small.

2 Preliminaries

The terminology used throughout the paper is standard. For a graph G we denote by $V(G)$ and $E(G)$ its vertex-set and edge-set, respectively. Throughout the paper, n is used to denote $|V(G)|$. For a subset $V' \subseteq V(G)$ we denote by $G[V']$ the sub-graph of G induced by V'. For $v \in V$ we denote by $\deg(v)$ the degree of v in G, by $N(v)$ the set of neighbours of v, and by $N[v] := N(V) \cup \{v\}$.

For $0 \leqslant \alpha \leqslant 1$ and $\Delta > 0$ we say that a graph $G = (V(G), E(G))$ is (α, Δ)-*bounded* if it contains at least $\alpha \cdot |V(G)|$ vertices of degree at most Δ. Note that if $\alpha = 1$ this definition coincides with the standard definition of a bounded degree graph.

In the *k-coloring problem*, we are given a graph G and need to decide whether there exists a k-coloring $c : V(G) \to [k]$ of G, such that for every $(u, v) \in E(G)$ we have $c(u) \neq c(v)$. If a graph has a k-coloring, we say that it is k-colorable. In the *chromatic number problem*, we are given a graph G and need to compute $\chi(G)$, the minimal integer k for which G is k-colorable.

In the *k-list-coloring problem*, we are given a graph G and a set $C_v \subseteq U$ of size $|C_v| \leqslant k$ for every $v \in V(G)$, where U is some arbitrary universe. We need to decide whether there exists a coloring $c : V(G) \to U$ such that for every $v \in V(G)$ we have $c(v) \in C_v$ and for every $(u, v) \in E(G)$ we have $c(u) \neq c(v)$.

In a general (a, b)-CSP (Constraint Satisfaction Problem, see [20] or [29] for a complete definition and discussions) we are given a list of *constraints*[1] on the values of subsets of size b of n a-ary variables, and need to decide whether there exists an assignment of values to the variables for which all constraints are satisfied. k-coloring and k-list-coloring are examples of $(k, 2)$-CSP problems. k-SAT is an example of a $(2, k)$-CSP problem.

2.1 Inverse Möbius Transform

Let U be an n-element set. The *Inverse Möbius* transform (sometimes also called the *Zeta* transform) [27] maps a function $f : P(U) \to \mathbb{R}$ from the power-set of U into another function $\hat{f} : P(U) \to \mathbb{R}$ defined as

$$\hat{f}(X) = \sum_{Y \subseteq X} f(Y).$$

Naively, $\hat{f}(X)$ is computed using $2^{|X|}$ additions. Thus, we can compute all values of $\hat{f}$ in a straightforward manner with $O(3^n)$ operations. Yates' method from 1937 ([19, 32]) improves on the above and computes all values of $\hat{f}$ using just $O(n2^n)$ operations. The resulting algorithm is usually called *the fast möbius transform* or *the fast zeta transform* ([3, 18]). The authors of [2] and [3] use the fast Inverse Möbius Transform to devise algorithms for combinatorial optimization problems such as computing the chromatic and the domatic numbers of a graph. The algorithm of [3] is summarized in Section 3.

A description of Yates' method follows.

---

[1] A general constraint on a set $x_1, \ldots, x_b$ of a-ary variables is a subset T of the a^b possible assignments in $\{x_1, \ldots x_b\} \to [a]$. The constraint is satisfied by an assignment c, possibly on more variables, if $c|_{\{x_1, \ldots x_b\}} \in T$.

▶ **Lemma 11.** *The Inverse Möbius Tranform $\hat{f}$ for some function $f : P(U) \to \mathbb{R}$ can be computed in $O(n2^n)$ time, where $n := |U|$.*

Proof. Denote by $U = \{u_1, \ldots, u_n\}$ some enumeration of U's elements. Denote by $f_0 := f$. We preform n iterations for $i = 1, \ldots, n$, in which we compute all values of the function $f_i : P(U) \to \mathbb{R}$ defined using f_{i-1} as follows.

$$f_i(X) = \begin{cases} f_{i-1}(X) + f_{i-1}(X \backslash \{u_i\}) & \text{if } u_i \in X \\ f_{i-1}(X) & \text{otherwise} \end{cases}$$

Namely, in the i-th iteration we add the values the function gets in the sub-cube defined by $u_i = 0$ to the corresponding values in the sub-cube defined by $u_i = 1$.

A simple induction on i shows that $f_i(X) = \sum_{Y \in S_i(X)} f(Y)$ where $S_i(X)$ is the set of all subsets $Y \subseteq X$ such that

$$\{u_j \in Y \mid j > i\} = \{u_j \in X \mid j > i\}$$

In particular, by the end of the algorithm $f_n = \hat{f}$. ◀

2.2 Decision versus Search

The k-coloring problem can be stated in two natural ways. In the first, given a graph G decide whether it can be colored using k colors. The second, given a graph G return a k-coloring for it if one exists, or say that no such coloring exists. A few folklore reductions show that the two problems have the same running time up to polynomial factors. We state one for completeness. Others appear in the survey of [14].

▶ **Lemma 12.** *Let $\mathcal{A}$ be an algorithm deciding whether a graph is k-colorable in $O(T(n))$ time. Then, there exists an algorithm $\mathcal{A}'$ that finds a k-coloring for G, if one exists, in $O^*(T(n))$ time.*

Proof. We describe $\mathcal{A}'$. First, use $\mathcal{A}(G)$ to decide whether G is k-colorable, if it returns *False* we return that no k-coloring exists. Otherwise, repeat the following iterative process. For every pair of distinct vertices $(u, v) \notin E(G)$ that is not an edge of G, use $\mathcal{A}\left(G' := (V(G), E(G) \cup \{(u, v)\})\right)$ to check whether G stays k-colorable after adding (u, v) as an edge. If it does, add (u, v) to $E(G)$. We stop when no such pair (u, v) exists.

The reader can verify that the resulting graph must be a complement of k disjoint cliques, and thus we can easily construct a k-coloring. ◀

A problem comes up while trying to use this type of reductions in the settings of this paper. The aforementioned reduction adds edges to the graph, and therefore increases the degrees of vertices. In particular, we cannot use it (or other similar reductions) in a black-box manner for statements like Theorem 4. The algorithm of [2] solves the decision version of k-coloring for bounded degree graphs, and cannot be trivially converted into an algorithm that finds a coloring. The algorithms presented in this paper, on the other hand, can be easily converted into algorithms that find a k-coloring. This is briefly discussed later in Section 4.4.

3 Overview of the $O^*(2^n)$ algorithm

In this section we present a summary of Björklund, Husfeldt and Koivisto's algorithm from [3]. We present a concise variant of their work that applies specifically to the coloring problem. The original paper covers a larger variety of set partitioning problems and thus the description in this section is simpler.

We begin by making the following very simple observation, yielding an equivalent phrasing of the coloring problem.

▶ **Observation 13.** *A graph G is k-colorable if and only if its vertex set $V(G)$ can be* covered *by k independent sets.*

A short outline of the algorithm follows, complete details appear below. We need to decide whether $V(G)$ can be covered by k independent sets. In order to do so, we compute the number of independent sets in every induced sub-graph and then use a simple inclusion-exclusion argument in order to compute the number of (ordered) covers of $V(G)$ by k independent sets. We are interested in whether this number is positive.

▶ **Definition 14.** *For a subset $V' \subseteq V(G)$ of vertices, let $i(G[V'])$ denote the number of independent sets in the induced sub-graph $G[V']$.*

We next show that using dynamic programming, we can quickly compute these values.

▶ **Lemma 15.** *We can compute the values of $i(G[V'])$ for all $V' \subseteq V$ in $O^*(2^n)$ time.*

Proof. Let $v \in V'$ be an arbitrary vertex contained in V'. The number of independent sets in V' that do not contain v is exactly $i(G[V' \backslash \{v\}])$. On the other hand, the number of independent sets in V' that do contain v is exactly $i(G[V' \backslash N[v]])$. Thus, we have

$$i(G[V']) = i(G[V' \backslash \{v\}]) + i(G[V' \backslash N[v]]).$$

We note that both $V' \backslash \{v\}$ and $V' \backslash N[v]$ are of size strictly less than $|V'|$. Thus, we can compute all 2^n values of $i(G[\cdot])$ using dynamic programming processing the sets in non-decreasing order of size. ◀

Consider the expression

$$F(G) = \sum_{V' \subseteq V(G)} (-1)^{|V(G)|-|V'|} \cdot i(G[V'])^k.$$

Using the values of $i(G[\cdot])$ computed in Lemma 15, we can easily compute the value of $F(G)$ by directly evaluating the above expression in $O^*(2^n)$ time.

▶ **Lemma 16.** *Let $S_1 \subseteq S_2$ be sets. It holds that*

$$\sum_{S_1 \subseteq S \subseteq S_2} (-1)^{|S|} = \begin{cases} 0 & \text{if } S_1 \neq S_2 \\ (-1)^{|S_2|} & \text{if } S_1 = S_2 \end{cases}$$

Proof. If $S_1 \subsetneq S_2$ then there exists a vertex $v \in S_2 \backslash S_1$. We can pair each set $S_1 \subseteq S \subseteq S_2$ with $S \triangle \{v\}$, its symmetric difference with $\{v\}$. Clearly, in each pair of sets one is of odd size and one is of even size, and thus their signs cancel each other. Therefore, the sum is zero. In the second case, the claim is straightforward. ◀

▶ **Lemma 17.** *$F(G)$ equals the number of k-tuples $(I_0, \ldots, I_{k-1})$ of independent sets in G such that $V(G) = I_0 \cup \ldots \cup I_{k-1}$.*

Proof. As $i(G[V'])$ counts the number of independent sets in $G[V']$, raising it to the k-th power (namely, $i(G[V'])^k$) counts the number of k-tuples of independent sets in $G[V']$.

Let $(I_0, \ldots, I_{k-1})$ be a k-tuple of independent sets in G. It appears exactly in terms of the sum corresponding to sets V' such that $I_0 \cup \ldots \cup I_{k-1} \subseteq V' \subseteq V(G)$. Each time this k-tuple is counted, it is counted with a sign determined by the parity of V'. By Lemma 16, the sum of the signs corresponding to sets $I_0 \cup \ldots \cup I_{k-1} \subseteq V' \subseteq V(G)$ is zero if $I_0 \cup \ldots \cup I_{k-1} \neq V(G)$ and one if $I_0 \cup \ldots \cup I_{k-1} = V(G)$. ◀

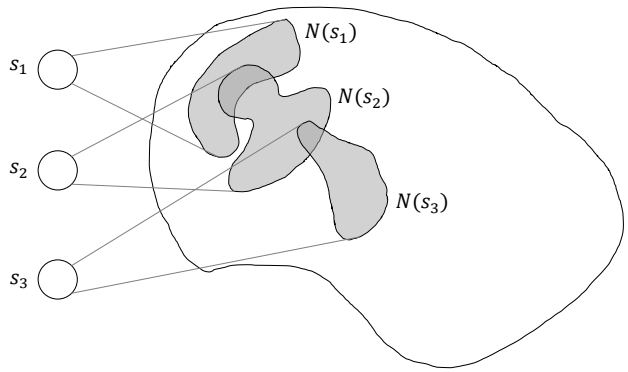
We conclude with

▶ **Corollary 18.** *$F(G)$ can be computed in time $O^*(2^n)$, and G is k-colorable if and only if $F(G) > 0$.*

4 Faster Coloring Algorithms for (α, Δ)-bounded Graphs

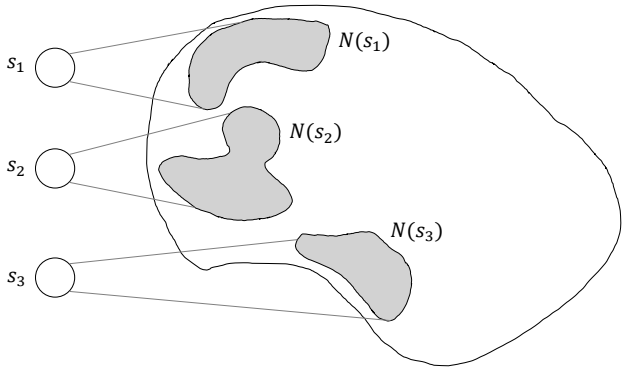
The main purpose of this section is proving Theorem 4.

We first outline our approach. Let G be a graph with a constant chromatic number $\chi(G) \leqslant k$. It is well known that G must contain a large independent set. Let S be an independent set in G. We think of $|S|$ as a constant fraction of $|V(G)|$, when we consider k as a constant. Let $c : (V(G) \backslash S) \to [k]$ be a k-coloring of the induced sub-graph $G[V(G) \backslash S]$. We say that c can be *extended* to a k-coloring of G if there exists a proper k-coloring $c' : V(G) \to [k]$ such that $c'|_{V(G)\backslash S} = c$. For a subset $V' \subseteq V(G) \backslash S$ of vertices, we say that *c does not use the full palette on V'* if $|c(V')| < k$, namely, if c does not use all k colors on the vertices of V'. Clearly, a proper k-coloring c of $V(G) \backslash S$ can be extended to a proper k-coloring of G if and only if $|c(N(s))| < k$ for every $s \in S$. Our approach, on a high-level, is to construct an algorithm that finds an extendable k-coloring of $V(G) \backslash S$. We aim to do so in $O\left(2^{|V(G)\backslash S|} (2 - \varepsilon)^{|S|}\right)$ time.



In Section 4.1 we consider a restricted version of the problem in which the independent set S has the following two additional properties. First, we assume that every vertex $s \in S$ is of degree $\deg(s) \leqslant \Delta$, where Δ is some constant. Second, we assume that no pair of vertices $s_1, s_2 \in S$ share a neighbor in G. Equivalently, the neighborhoods $N(s)$ for every $s \in S$ are all

disjoint. Under these conditions, we present an algorithm that runs in $O\left(2^{|V(G)\setminus S|}\left(2-\varepsilon\right)^{|S|}\right)$ time, where ε depends only on Δ. As ε does not depend on k, we can in fact compute the chromatic number of G exponentially faster than $O^*(2^n)$ if G contains an independent set S with these properties. We also observe that if G is of maximum degree Δ then it contains a large such independent set S. Our algorithm is based on methods that generalize Section 3, and on a simple approach to implicitly compute values of the Inverse Möbius Transform.



In Section 4.2 we modify the algorithm of Section 4.1 and remove the second assumption on S. Namely, we now only assume that S is an independent set and that for every $s \in S$ we have $\deg(s) \leqslant \Delta$. Our algorithm still runs in $O\left(2^{|V(G)\setminus S|}\left(2-\varepsilon\right)^{|S|}\right)$ time. A main ingredient in the modification is a new removal lemma for small subsets. The proof of this combinatorial lemma is given in the full version of the paper and its statement is used in a black-box manner in this section.

In Section 4.3 we extend the result to List Coloring.

4.1 k-coloring bounded-degree graphs

In this subsection we begin illustrating the ideas leading towards proving Theorem 4. We also prove the following (much) weaker statement.

▶ **Theorem 19.** *For every k, Δ there exists $\varepsilon_{k,\Delta} > 0$ such that we can solve k-coloring for graphs with maximum degree Δ in $O\left((2-\varepsilon_{k,\Delta})^n\right)$ time.*

In fact, as a graph G with maximum degree Δ has chromatic number $\chi(G) \leqslant \Delta + 1$, we can compute the chromatic number of a graph with degrees bounded by Δ in time $O\left((2-\varepsilon_{\Delta+1,\Delta})^n\right)$.

As outlined in the beginning of this section, our approach begins by finding a large independent set with some additional properties. We show that a graph with bounded degrees must contain a very large independent set S such that the distance between each pair of vertices in S is at least three. In other words, S is an independent set, and no pair of vertices in S share a neighbor. In particular, the neighborhoods $N(s)$ for $s \in S$ are all disjoint. The core theorem of this subsection is

▶ **Theorem 20.** *Let G be a graph and $S \subseteq V(G)$ a set of vertices such that the distance between each two vertices in S is at least three and the degree of each vertex in S is at most Δ. For any k, we can solve k-coloring for G in $O^*\left(2^{|V(G)|-|S|} \cdot (2-2^{-\Delta})^{|S|}\right)$ time.*

It is important to note that the existence of such a set S is our sole use of the bound on the graph degrees. Note that the bound of Theorem 20 does not depend on k. Thus, we get an exponential improvement for computing the chromatic number of a graph G that contains a large enough set S with the stated properties.

Before proving Theorem 20, we describe a simple algorithm for finding a set S with the required properties in bounded-degree graphs.

▶ **Lemma 21.** *Let G be a graph with maximum degree at most Δ. There exists a set $S \subseteq V(G)$ of at least $\frac{1}{1+\Delta^2} \cdot |V(G)|$ vertices such that the distance between every distinct pair $s_1, s_2 \in S$ is at least three. Furthermore, we can find such S efficiently.*

Proof. We construct S in a greedy manner. We begin with $S = \varnothing$ and $V' = V(G)$. As long as V' is not empty we pick an arbitrary vertex $v \in V'$ and add it to S. We then remove from V' the vertex v and every vertex of distance at most two from it.

By construction, the minimum distance between a pair of vertices in S is at least three. The size of the 2-neighborhood of a vertex is bounded by $1 + \Delta + \Delta \cdot (\Delta - 1) = 1 + \Delta^2$ and thus we get the desired lower bound on the size of S. ◀

Theorem 19 now follows from Lemma 21 and Theorem 20.

Proof of Theorem 19. Let G be a graph of maximum degree at most Δ and let k be an integer. By Lemma 21, we can construct a set S of size $|S| \geqslant \frac{1}{1+\Delta^2} \cdot |V(G)|$ satisfying the conditions of Theorem 20. Thus, by Theorem 20, we can solve k-coloring for G in time

$$O^*\left(2^{n-\frac{1}{1+\Delta^2}\,n} \cdot (2-2^{-\Delta})^{\frac{1}{1+\Delta^2}\,n}\right) = O^*\left(\left(2 \cdot \left(1-2^{-(\Delta+1)}\right)^{\frac{1}{1+\Delta^2}}\right)^n\right).$$ ◀

In the rest of the subsection we prove Theorem 20.

▶ **Definition 22.** *For subsets $V' \subseteq V(G)\backslash S$ and $S' \subseteq S$ denote by $\beta(V', S')$ the number of independent sets I in $G[V']$ that intersect every neighborhood $N(s)$ of $s \in S'$, that is, $I \cap N(s) \neq \varnothing$ for every $s \in S'$.*

Consider, for a subset $S' \subseteq S$, the following sum

$$h(G, S') := \sum_{V' \subseteq V(G)\backslash S} (-1)^{|V(G)|-|V'|}\, \beta\left(V', S'\right)^k.$$

The following proof is almost identical to the proof of Lemma 17 in Section 3.

▶ **Lemma 23.** *$h(G, S')$ is the number of covers of $V(G)\backslash S$ by k-tuples $(I_0, \ldots, I_{k-1})$ of independent sets in $G[V(G)\backslash S]$ such that $I_i \cap N(s) \neq \varnothing$ for every $s \in S'$ and every $0 \leqslant i \leqslant k-1$.*

Proof. Each value of $\beta(V', S')$ counts independent sets in $G[V']$ that intersect every neighborhood $N(s)$ for $s \in S'$.

Each k-tuple $(I_0, \ldots, I_{k-1})$ of that type is counted in terms corresponding to sets V' such that

$$I_0 \cup \ldots \cup I_{k-1} \subseteq V' \subseteq V(G)\backslash S.$$

By Lemma 16 the multiplicity with which such k-tuple is counted is one if

$$I_0 \cup \ldots \cup I_{k-1} = V(G)\backslash S.$$

and zero otherwise. ◀

Consider the following expression.

$$H(G,S) := \sum_{S' \subseteq S} (-1)^{|S'|}\, h(G,S')$$

$H(G,S)$ is the number of covers of $V(G)\backslash S$ by k-tuples of independent sets that *do not use the full palette* on any neighborhood $N(s)$ for $s \in S$. The precise claim follows.

▶ **Lemma 24.** *$H(G,S)$ is the number of covers of $V(G)\backslash S$ by k-tuples $(I_0,\ldots,I_{k-1})$ of independent sets in $G[V(G)\backslash S]$ such that for every $s \in S$ there exists $0 \leqslant i \leqslant k-1$ such that $I_i \cap N(s) = \varnothing$.*

Proof. In Lemma 23 we showed that $h(G,S')$ counts the number of covers of $V(G)\backslash S$ by k-tuples $(I_0,\ldots,I_{k-1})$ of independent sets in $G[V(G)\backslash S]$ such that for every $s \in S'$ and for every $0 \leqslant i \leqslant k-1$ we have $I_i \cap N(s) \neq \varnothing$.

A covering k-tuple of independent sets $(I_0,\ldots,I_{k-1})$ is counted exactly in terms corresponding to subsets S' such that for every $0 \leqslant i \leqslant k-1$ and every $s \in S'$, the independent set I_i intersects the neighborhood $N(s)$. These are exactly the subsets S' such that

$$S' \subseteq \{s \in S \mid \forall 0 \leqslant i \leqslant k-1,\ I_i \cap N(s) \neq \varnothing\}.$$

Using Lemma 16 with $S_1 = \varnothing$ and $S_2 = \{s \in S \mid \forall 0 \leqslant i \leqslant k-1,\ I_i \cap N(s) \neq \varnothing\}$ we deduce that the multiplicity with which the k-tuple is counted is one if

$$\{s \in S \mid \forall 0 \leqslant i \leqslant k-1,\ I_i \cap N(s) \neq \varnothing\} = \varnothing$$

and zero otherwise. ◀

As outlined at the beginning of the section, we now claim that $H(G,S)$ is positive if and only if G is k-colorable. Note that for the correctness of this lemma we still did not use the disjointness of the neighborhoods $N(s)$. We will need this property to improve the computation time.

▶ **Lemma 25.** *Let G be a graph and S an independent set in it. Then, $H(G,S) > 0$ if and only if G is k-colorable.*

Proof. Assume that there exists a k-coloring $c : V(G) \to [k]$ of G. For $0 \leqslant i \leqslant k-1$ denote by

$$I_i := \{v \in V(G)\backslash S \mid c(v) = i\}$$

the subset of $V(G)\backslash S$ colored by i. Each I_i is an independent set as c is a proper coloring of G. Furthermore, for each $s \in S$, the neighborhood $N(s)$ does not intersect $I_{c(s)}$. Thus, $(I_0,\ldots,I_{k-1})$ is a cover of $V(G)\backslash S$ by k independent sets that do not all intersect any neighborhood $N(s)$ of $s \in S$. By Lemma 24, $H(G,S) \geqslant 1$.

On the other hand, if $H(G,S) > 0$ then by Lemma 24 there exists a cover by independent sets and in particular a k-coloring $c : V(G)\backslash S \to [k]$ of $G[V(G)\backslash S]$ such that the full palette is not used on any neighborhood $N(s)$ for $s \in S$. Thus, we may extend c to a k-coloring $c' : V(G) \to [k]$ of the entire graph by coloring each $s \in S$ with a color that does not appear in $c(N(s))$. As S is an independent set, this coloring is proper. ◀

Up to this point, we have formalized the outline from the beginning of this section, reducing k-coloring to a problem of k-coloring with some restrictions the smaller graph $G[V(G)\backslash S]$ and then to the computation of $H(G,S)$.

Unfortunately, $H(G,S)$ is a sum of $2^{|S|}$ terms, each of the form $h(G,S')$ which is a sum of $2^{|V(G)|-|S|}$ terms by itself. Evidently, there are 2^n different terms of the form $\beta(V',S')$ that are used in the definition of $H(G,S)$. Thus, we cannot hope to compute $H(G,S)$ in less than 2^n steps if we need to explicitly examine 2^n terms of the form $\beta(\cdot,\cdot)$. Moreover, it is also not clear how quickly we can compute the values of $\beta(\cdot,\cdot)$.

We begin by explaining how values of $\beta(\cdot)$ can be computed efficiently. The term $h(G,S')$ is a weighted sum of the values $\beta\left(V',S'\right)$ for all $V' \subseteq V(G)\backslash S$. Denote by $\beta_\mu\left(V',S'\right)$ the indicator function that gets the value 1 if V' is an independent set in $G[V(G)\backslash S]$ and for every $s \in S'$ we have $V' \cap N(s) \neq \varnothing$, and 0 otherwise. We can efficiently compute the value of β_μ for a specific input in a straightforward manner (i.e., checking whether it is an independent set that intersects the relevant sets). We observe that

$$\beta\left(V',S'\right) = \sum_{V''\subseteq V'} \beta_\mu\left(V'',S'\right),$$

thus, $\beta = \hat{\beta_\mu}$ as functions of V', and we can compute the values of $\beta\left(V',S'\right)$ for all $V' \subseteq V(G)\backslash S$ in $O^*(2^{|V(G)|-|S|})$ time using the Inverse Möbius Transform presented in Section 2.1.

An improvement to the running time comes from noticing that for many inputs (V',S') the value of $\beta\left(V',S'\right)$ is zero. In particular, if $V' \cap N(s) = \varnothing$, for some $s \in S'$, then $\beta\left(V',S'\right) = 0$ as no subset (and in particular no independent set) in V' intersects $N(s)$. In the computation of $h(G,S')$ we only need to consider terms corresponding to subsets $V' \subseteq V(G)\backslash S$ in which for every $s \in S'$ the intersection $V' \cap N(s)$ is non-empty, as the values of other terms are all zero. We present a variant of the Inverse Möbius Transform that computes only the non-zero values by implicitly setting the others to zero. We then show that for most subsets $S' \subseteq S$ the number of non-zero entries is exponentially smaller than $2^{|V(G)|-|S|}$.

▶ **Definition 26.** *For any $S' \subseteq S$ denote by $B(S') := \{V' \subseteq V(G)\backslash S \mid \forall s \in S'.\ V' \cap N(s) \neq \varnothing\}$ the set of all subsets of $V(G)\backslash S$ intersecting all neighborhoods of S'.*

As we observed above, for every $V' \notin B(S')$ we have $\beta\left(V',S'\right) = 0$. We conclude that

▶ **Observation 27.** *For every S' we have*

$$h(G,S') = \sum_{V'\in B(S')} (-1)^{|V(G)|-|V'|}\, \beta\left(V',S'\right)^k.$$

▶ **Lemma 28.** *If the neighborhoods $N(s)$ are disjoint for all $s \in S'$, then we can compute $h(G,S')$ in $O^*(|B(S')|)$ time.*

Proof. It suffices to compute $\beta\left(V',S'\right)$ for every $V' \in B(S')$ and then use Observation 27. We do so by introducing a variant of the Inverse Möbius Transform that implicitly sets the value of $\beta\left(V',S'\right)$ to zero for every $V' \notin B(S')$.

We first note that

$$B(S') \cong P\left(V(G)\backslash\left(S \cup \bigcup_{s\in S'} N(s)\right)\right) \times \bigtimes_{s\in S'} \left(P(N(s))\backslash\{\varnothing\}\right).$$

Thus, we can efficiently construct a simple bijection between $[|B(S')|]$ and $B(S')$ as a Cartesian product. We can also efficiently check if a set V' belongs to $B(S')$. Let $index : B(S') \to [|B(S')|]$ be a map from $B(S')$ to indices of $[|B(S')|]$. If $V' \notin B(S')$ we define

$index(V') = -1$. By the observation above, we can define *index* in way for which *index* and $index^{-1}$ are efficiently computable. We also arbitrarily order the vertices of $V(G)\backslash S$ as $v_1, v_2, \ldots, v_{|V(G)\backslash S|}$.

We describe the algorithm in pseudo-code.

Algorithm 1 Algorithm for the proof of Lemma 28.

```
Initialize an array f of size |B(S')|;
for ℓ in [|B(S')|] do
    if index^{-1}(ℓ) is an independent set in G[V(G)\S] then
        f(ℓ) ← 1 ;
    else
        f(ℓ) ← 0 ;
for i in [|V(G)\S|] do
    for ℓ in [|B(S')|] do
        V' ← index^{-1}(ℓ) ;
        if v_i ∈ V' and index(V'\{v_i}) ≠ −1 then
            f(ℓ) ← f(ℓ) + f(index(V'\{v_i}));
```

We view f throughout the algorithm as function $f : B(S') \to \mathbb{N}$. Denote the function represented by f at the end of the first *for* loop by f_0. By definition, $f_0(V') = \beta_\mu(V', S')$ for every $V' \in B(S')$. Denote by f_i the function represented by f at the end of the i-th iteration of the second (outer) *for* loop.

We observe that f_i is defined using f_{i-1} as

$$f_i(V') = \begin{cases} f_{i-1}(V') + f_{i-1}(V'\backslash\{v_i\}) & \text{if } v_i \in V' \\ f_{i-1}(V') & \text{otherwise} \end{cases}$$

where $f_{i-1}(V'\backslash\{v_i\})$ is implicitly defined to be zero if $V'\backslash\{v_i\} \notin B(S')$.

By induction on i, similar to this of Section 2.1, we can show that

$$f_i(V') = \sum_{\substack{V'' \subseteq V' \\ V''\backslash\{v_1,\ldots,v_i\} = V'\backslash\{v_1,\ldots,v_i\}}} f(V'').$$

In particular, by the end of the algorithm $f = \hat{f}_0 = \hat{\beta}_\mu = \beta$ for the entire domain $B(S')$. ◀

After computing $h(G, S')$ for every $S' \subseteq S$ we can compute $H(G, S)$ in $O^*(2^{|S|})$ time. We thus finish the proof of Theorem 20 with the following counting lemma.

▶ **Lemma 29.** *Assume that the neighborhoods $N(s)$ are disjoint for all $s \in S$ and that each neighborhood is of size $|N(s)| \leqslant \Delta$. Then, $\sum_{S' \subseteq S} |B(S')| = O^*\left(2^{|V(G)\backslash S|} \cdot (2 - 2^{-\Delta})^{|S|}\right)$.*

Proof. Denote $n(s) := |N(s)|$. Also denote by $N = \bigcup_{s \in S} N(s)$ all neighbors of vertices of S and by $N^c = (V(G)\backslash S)\backslash N$ their complement in $V(G)\backslash S$. We have

$$\begin{aligned} |B(S')| &= 2^{|N^c|} \cdot \prod_{s \in S'} \left(2^{n(s)} - 1\right) \cdot \prod_{s \in S\backslash S'} 2^{n(s)} \\ &= 2^{|N^c|} \cdot \prod_{s \in S'} \left(1 - 2^{-n(s)}\right) \cdot \prod_{s \in S} 2^{n(s)} \\ &= 2^{|N^c|} \cdot \prod_{s \in S'} \left(1 - 2^{-n(s)}\right) \cdot 2^{|N|} \\ &= 2^{|V(G)\backslash S|} \cdot \prod_{s \in S'} \left(1 - 2^{-n(s)}\right). \end{aligned}$$

For every $s \in S$ we have $n(s) \leqslant \Delta$ and thus $\left(1-2^{-n(s)}\right) \leqslant \left(1-2^{-\Delta}\right)$. Hence,

$$\begin{aligned}|B(S')| &\leqslant 2^{|V(G)\setminus S|} \cdot \prod_{s\in S'} \left(1-2^{-\Delta}\right) \\ &= 2^{|V(G)\setminus S|} \cdot \left(1-2^{-\Delta}\right)^{|S'|}.\end{aligned}$$

Therefore we have

$$\begin{aligned}\sum_{S'\subseteq S} |B(S')| &\leqslant \sum_{S'\subseteq S} 2^{|V(G)\setminus S|} \cdot \left(1-2^{-\Delta}\right)^{|S'|} \\ &= 2^{|V(G)\setminus S|} \cdot \sum_{i=0}^{|S|} \binom{|S|}{i} \left(1-2^{-\Delta}\right)^{i} \\ &= 2^{|V(G)\setminus S|} \cdot \left(2-2^{-\Delta}\right)^{|S|}.\end{aligned}$$

◀

4.2 From bounded-degree graphs to (α, Δ)-bounded graphs

In this section we prove the main technical theorem of the paper.

▶ **Theorem 4.** *For every $\Delta, \alpha > 0$ there exists $\varepsilon_{\Delta,\alpha} > 0$ such that we can compute the chromatic number of (α, Δ)-bounded graphs in $O\left(\left(2-\varepsilon_{\Delta,\alpha}\right)^n\right)$ time.*

We prove the following seemingly weaker statement.

▶ **Theorem 30.** *For every $k, \Delta, \alpha > 0$ there exists $\varepsilon_{k,\Delta,\alpha} > 0$ such that we can solve k-coloring for (α, Δ)-bounded graphs in $O\left(\left(2-\varepsilon_{k,\Delta,\alpha}\right)^n\right)$ time.*

We then note that Theorem 30 in fact implies Theorem 4. Let G be a (α, Δ)-bounded graph. We use Theorem 30 for every $1 \leqslant k \leqslant \Delta$. If we did not find a valid coloring of G, then $\chi(G) \geqslant \Delta + 1$ and we may use a standard argument (fully presented in the full version of the paper) to show that removing all vertices of degree at most Δ does not change $\chi(G)$. By definition of (α, Δ)-bounded graphs, removing these vertices leaves a graph with at most $(1-\alpha)n$ vertices and thus the standard chromatic number algorithm runs in $O^*(2^{(1-\alpha)n})$ time.

As in Section 4.1, we deduce Theorem 30 from the following theorem.

▶ **Theorem 31.** *Let G be a graph and $S \subseteq V(G)$ an independent set in G. Assume that the degree of each vertex in S is at most Δ. Then, we can solve k-coloring for G in $O^*\left(2^{|V(G)|} \cdot (1-\varepsilon_{k,\Delta})^{|S|}\right)$ time, for some constant $\varepsilon_{k,\Delta} > 0$.*

Let G be a graph with a subset $U \subseteq V(G)$ of vertices such that for every $v \in U$ we have $\deg(v) \leqslant \Delta$. In a similar fashion to Lemma 21 of the previous subsection (and even slightly simpler), we can greedily construct a subset $S \subseteq U$ of size $|S| \geqslant \frac{1}{1+\Delta} \cdot |U|$ which is an independent set. Thus, Theorem 31 immediately implies Theorem 4. Unlike the case of Section 4.1, this time the neighborhoods $N(s)$ for $s \in S$ are not necessarily disjoint. Thus, statements comparable to Lemma 29 are not true. Our solution for this problem is surprisingly general. In the full version of the paper we prove Theorem 10. Plugging $\mathcal{F} = \{N(s)\}_{s\in S}$, we get a small set $U' \subseteq V(G)\setminus S$ of graph vertices, and a large subset $S' \subseteq S$ of the independent set, such that the neighborhoods $N(s)$ of $s \in S'$ become pairwise disjoint if we remove the vertices of U' from G. As we want to preserve the correctness of the algorithm, we do not actually remove U' from G, but *enumerate* over the colors they receive in a proper k-coloring, if one exists. The main technical gap is adjusting the algorithm and proofs of Section 4.1 to the case in which some of the graph vertices have fixed colors.

▶ **Theorem 32.** *Let G be a graph, $V_0 \subseteq V(G)$ a subset of its vertices and $c : V_0 \to [k]$ a proper k-coloring of $G[V_0]$. Denote by $V := V(G) \backslash V_0$. Let $S \subseteq V$ be an independent set in G such that the distance in $G[V]$ between each two vertices of S is at least three and the degree in $G[V]$ of each vertex in S is at most Δ. For any k, we can decide whether c can be extended to a k-coloring of the entire graph G in $O^*\left(2^{|V|-|S|} \cdot (2-2^{-\Delta})^{|S|}\right)$ time.*

Throughout the rest of the section, it is important to carefully distinguish $V(G)$ from V. Note that V does not include the vertices of V_0, as their colors are already fixed. For $j \in [k]$, denote by $V_0^j := c^{-1}(j)$ the subset of V_0 colored by j color. Note that $V_0 = \bigcup_{j=1}^{k} V_0^j$. We begin adapting the algorithm by redefining the $\beta(\cdot,\cdot)$ function.

▶ **Definition 33.** *For subsets $V' \subseteq V \backslash S$, $S' \subseteq S$, and a color $j \in [k]$, we denote by $\beta_j(V', S')$ the number of sets $I \subseteq V'$ such that $I \cup V_0^j$ is an independent set in G and that $I \cup V_0^j$ intersects $N(s)$ for every $s \in S'$, that is, for every $s \in S'$ we have $\left(I \cup V_0^j\right) \cap N(s) \neq \varnothing$.*

We also revise the definition of

$$h(G, S') := \sum_{V' \subseteq V \backslash S} (-1)^{|V|-|V'|} \prod_{j=0}^{k-1} \beta_j\left(V', S'\right).$$

The proof of Lemma 23 can be easily revised to show the following.

▶ **Lemma 34.** *$h(G, S')$ is the number of covers of $V \backslash S$ by k-tuples of sets $I_0, \ldots, I_{k-1}$ such that for every $j \in [k]$, $I_j \cup V_0^j$ is an independent set in G and that for every $s \in S'$ and every $j \in [k]$ the set $I_j \cup V_0^j$ intersects the neighborhood $N(s)$.*

Without revising the definition of $H(G, S)$, the proof of Lemma 24 now shows that

▶ **Lemma 35.** *$H(G, S)$ is the number of covers of $V \backslash S$ by k-tuples of sets $I_0, \ldots, I_{k-1}$ such that for every $j \in [k]$, $I_j \cup V_0^j$ is an independent set in G and that for every $s \in S$ the neighborhood $N(s)$ is not intersected by at least one of the k independent sets $\left(I_j \cup V_0^j\right)$ for $j \in [k]$.*

Therefore, we have

▶ **Lemma 36.** *Let G be a graph, $V_0 \subseteq V(G)$ a subset of its vertices and $c : V_0 \to [k]$ a proper k-coloring of $G[V_0]$. Denote by $V := V(G) \backslash V_0$. Let $S \subseteq V$ be an independent set in G. Then, $H(G, S) > 0$ if and only if c can be extended to a k-coloring of G.*

The non-trivial part of the revision and the heart of this subsection, is adjusting the algorithm for computing the values of $h(G, S')$ without increasing the running time.

For $j \in [k]$, denote by

$$S_j := \{s \in S \mid N(s) \cap V_0^j \neq \varnothing\}$$

the set of vertices in S whose neighborhood intersects V_0^j. The key observation of this subsection follows.

▶ **Lemma 37.** *For any $j \in [k]$, $S' \subseteq S$, $V' \subseteq V$, we have*

$$\beta_j\left(V', S'\right) = \beta_j\left(V', S' \cup S_j\right)$$

Proof. For any set $I \subseteq V'$ the set $I \cup V_0^j$ intersects every set in $\{N(s)\}_{s \in S_j}$. In particular, an independent set $I \subseteq V'$ intersects all of $\{N(s)\}_{s \in S'}$ if and only if it intersects all of $\{N(s)\}_{s \in (S' \cup S_j)}$. ◀

Lemma 37 implies that it is enough to compute $\beta_j(V', S')$ only for sets $S' \subseteq S \backslash S_j$, as its other values can be deduced from these as $\beta_j(V', S') = \beta_j(V', S' \backslash S_j)$.

For any $S' \subseteq S$ we again denote by $B(S') := \{V' \subseteq V \backslash S \mid \forall s \in S'.\ V' \cap N(s) \neq \varnothing\}$ the set of all subsets of $V \backslash S$ intersecting all neighborhoods of S'. Note the slight difference from Section 4.1 of considering subsets of $V \backslash S$ and not of $V(G) \backslash S$.

As for every $s \in S \backslash S_j$, $N(s) \cap V_0^j = \varnothing$, we still have that for every $V' \notin B(S')$ the value of $\beta_j(V', S')$ is zero. In particular, we can still use the implicit Inverse Möbius Transform of Lemma 28 and get

▶ **Lemma 38.** *Assume $S' \subseteq S \backslash S_j$. We can compute $\beta_j(V', S')$ for every $V' \in B(S')$ in $O^*(|B(S')|)$ time.*

By Lemma 29 we get

$$\sum_{S' \subseteq S \backslash S_j} |B(S')| = O^*\left(2^{|V \backslash S|} \cdot (2 - 2^{-\Delta})^{|S \backslash S_j|}\right).$$

We can thus compute $\beta_j(V', S')$ for every $S' \subseteq S \backslash S_j$ and every $V' \in B(S')$ in $O^*\left(2^{|V \backslash S|} \cdot (2 - 2^{-\Delta})^{|S \backslash S_j|}\right)$ time. This is the time to emphasise a crucial point. Note that if we would consider every $S' \subseteq S$ instead of $S' \subseteq S \backslash S_j$, then the running time would be $O^*\left(2^{|V \backslash S|} \cdot (2 - 2^{-\Delta})^{|S \backslash S_j|} \cdot 2^{|S_j|}\right)$, as the neighborhoods corresponding to S_j are intersected by V_0^j. This is why we compute every β_j separately, and do so for *all* relevant sets S' before computing even a single value $h(G, S')$. As it always holds that $|S \backslash S_j| \leqslant |S|$, we conclude that

▶ **Corollary 39.** *We can compute $\beta_j(V', S')$ for all $j \in [k]$, $S' \subseteq S \backslash S_j$ and $V' \in B(S')$ in $O^*\left(2^{|V \backslash S|} \cdot (2 - 2^{-\Delta})^{|S|}\right)$ time.*

Note that $k = O^*(1)$.

We are now ready to compute the values of $h(G, S')$. We start by making the following observation.

▶ **Observation 40.** *If $\bigcap_{j=0}^{k-1} S_j \neq \varnothing$ then c cannot be extended to a coloring of G.*

This holds as if some $s \in S$ has neighbors colored in each of the k colors then it cannot be properly colored. We are thus dealing with the case where $\bigcap_{j=0}^{k-1} S_j = \varnothing$.

▶ **Lemma 41.** *For any $S' \subseteq S$ and $V' \subseteq V \backslash S$ such that $V' \notin B(S')$ we have*

$$\prod_{j=0}^{k-1} \beta_j(V', S') = 0.$$

Proof. As $V' \notin B(S')$ there exists some $s \in S$ such that $V' \cap N(s) = \varnothing$. As $\bigcap_{j=0}^{k-1} S_j = \varnothing$, there exists a $j \in [k]$ for which $s \notin S_j$. Thus, $V_0^j \cap N(s) = \varnothing$ as well. We conclude that $\beta_j(V', S') = 0$. ◀

From Lemma 37 and Lemma 41 we conclude that

$$h(G, S') := \sum_{V' \in B(S')} (-1)^{|V| - |V'|} \prod_{j=0}^{k-1} \beta_j\left(V', S' \backslash S_j\right).$$

Thus, we can compute $h(G, S')$ in $O^*(|B(S')|)$ time using the values computed in Corollary 39. Using Lemma 29 once again, we get that

$$\sum_{S' \subseteq S} |B(S')| = O^*\left(2^{|V \setminus S|} \cdot (2 - 2^{-\Delta})^{|S|}\right)$$

which completes the proof of Theorem 32.

We can now prove Theorem 31.

Proof. We apply the removal lemma of Theorem 10 to $\mathcal{F} = \{N(s)\}_{s \in S}$ with C to be chosen later. We thus get a sub-collection $S' \subseteq S$ and a subset of vertices $V_0 \subseteq V(G) \setminus S$ such that $|S'| > \rho(\Delta, C) \cdot |S| + C \cdot |V_0|$ and that for every $s_1, s_2 \in S'$ it holds that $N(s_1) \cap N(s_2) \subseteq V_0$. Denote by $V = V(G) \setminus (S' \cup V_0)$. We enumerate over all k-colorings $c : V_0 \to [k]$. For each coloring c, we check if it is a proper k-coloring of $G[V_0]$ and if so we apply Theorem 32 on G with V_0, c, S'. If any of the applications of Theorem 32 returned that there exists a valid extension of c to a coloring of G, we return that G is k-colorable, and otherwise that it is not.

The running time of the entire algorithm, up to polynomial factors, is

$$\begin{aligned} k^{|V_0|} \cdot \left(2^{|V \setminus S'|} \cdot (2 - 2^{-\Delta})^{|S'|}\right) &= 2^{|V|} \cdot k^{|V_0|} \cdot (1 - 2^{-(\Delta+1)})^{|S'|} \\ &\leqslant 2^{|V|} \cdot k^{|V_0|} \cdot (1 - 2^{-(\Delta+1)})^{\rho(\Delta,C) \cdot |S| + C \cdot |V_0|}. \end{aligned}$$

By picking $C = \frac{\log k}{-\log\left(1 - 2^{-(\Delta+1)}\right)} > 0$ we have

$$k^{|V_0|} \cdot (1 - 2^{-(\Delta+1)})^{C \cdot |V_0|} = 1$$

and thus the running time is bound by

$$2^{|V|} \cdot (1 - 2^{-(\Delta+1)})^{\rho(\Delta,C) \cdot |S|}.$$ ◀

4.3 Generalization to List Coloring

In this Section we deduce Theorem 5.

Proof. Let $G = (V, E)$ be a (α, Δ)-bounded graph with color lists C_v of size at most k for each $v \in V$. Denote by $U = \bigcup_{v \in V} C_v$ the color universe. Note that $|U|$ might be as large as kn, where $n = |V|$. We construct a new graph G' on the set of vertices $V \cup U$ by adding $|U|$ isolated vertices to the graph G and then connecting each node $v \in V$ to every node $u \in U$ such that $u \notin C_v$. If we color each $u \in U$ by the color u, then there is an extension of this coloring to a (regular) $|U|$-coloring for all of G' if and only if G is list-colorable.

We now follow the proof of Theorem 31. We can again find a subset $S \subset V$ of size $|S| \geqslant \frac{\alpha}{1+\Delta} n$ which is an independent set in G that contains only vertices of degree at most Δ. We then apply the removal lemma of Theorem 10 to $\mathcal{F} = \{N(s)\}_{s \in S}$ where the neighbourhoods are within G and C is to be chosen later. Define S' and V_0 as in the proof of Theorem 31. Since every color-list C_v is of size at most k, there are only $k^{|V_0|}$ possible colorings $c : V_0 \to U$. We enumerate over these colorings and for each one which is a proper coloring of $G[V_0]$ we apply Theorem 32 on G' where $U \cup V_0$ are already colored (by their corresponding colors and by c, respectively). The crucial point here is that in Theorem 32 the running time depends on $|V|$, the number of uncolored vertices, and is independent of the number of colored vertices. In particular, the total runtime is thus

$$\begin{aligned} k^{|V_0|} \cdot \left(2^{|V \setminus (S' \cup V_0)|} \cdot (2 - 2^{-\Delta})^{|S'|}\right) &\leqslant 2^{|V|} \cdot k^{|V_0|} \cdot (1 - 2^{-(\Delta+1)})^{|S'|} \\ &\leqslant 2^{|V|} \cdot k^{|V_0|} \cdot (1 - 2^{-(\Delta+1)})^{\rho(\Delta,C) \cdot |S| + C \cdot |V_0|}. \end{aligned}$$

We can thus again pick $C = \frac{\log k}{-\log\left(1-2^{-(\Delta+1)}\right)} > 0$ to have

$$k^{|V_0|} \cdot (1-2^{-(\Delta+1)})^{C\cdot|V_0|} = 1$$

which results in a running time bounded by

$$2^{|V|} \cdot (1-2^{-(\Delta+1)})^{\rho(\Delta,C)\cdot|S|}.$$ ◀

4.4 On finding a coloring

In both previous subsections, we used the bounds on the degrees only in order to construct a *good* independent set S. After doing so, we may apply the self-reduction of Section 2.2 to the graph $G[V(G)\backslash S]$, in which we no longer care about the number of edges nor the degrees. This would result in finding a k-coloring of $G[V(G)\backslash S]$. Such coloring can be extended to a k-coloring of G by the constructive proof of Lemma 25. The exact claim follows.

▶ **Lemma 42.** *In the conditions of Theorem 20 or Theorem 31 we can also* find *a k-coloring of G.*

Proof. Consider the reduction between the decision and search versions of k-coloring of Lemma 12. Since adding edges to vertices whose both endpoints are in $V(G)\backslash S$ does not violate the conditions of the theorems, we may apply the reduction of Lemma 12 to $G[V(G)\backslash S]$. By the end of the reduction, we have a k-coloring of $G[V(G)\backslash S]$ that is a restriction of some k-coloring of G. We can extend this k-coloring to a k-coloring of G using the algorithm of Lemma 25. ◀

As a corollary, in the conditions of Theorem 4 we can also *find* a k-coloring of G.

5 Summary of the full version of this paper

Due to length limitation, several main components of the paper appear only in the full version of the paper. The full version is available on-line.

5.1 Removal Lemma For Small Sets

We show that any collection of small sets must contain a *large* sub-collection of *almost* pairwise-disjoint sets. The precise statement follows.

▶ **Theorem 10.** *Let $\mathcal{F}$ be a collection of subsets of a universe U such that every set $F \in \mathcal{F}$ is of size $|F| \leqslant \Delta$. Let $C > 0$ be any constant. Then, there exist subsets $\mathcal{F}' \subseteq \mathcal{F}$ and $U' \subseteq U$, such that*

- *$|\mathcal{F}'| > \rho(\Delta, C) \cdot |\mathcal{F}| + C \cdot |U'|$, where $\rho(\Delta, C) > 0$ depends only on Δ, C.*
- *The sets in $\mathcal{F}'$ are disjoint when restricted to $U\backslash U'$, i.e., for every $F_1, F_2 \in \mathcal{F}'$ we have $F_1 \cap F_2 \subseteq U'$.*

We should think of the statement of Theorem 10 in the following manner. We interpret an *almost* pairwise-disjoint sub-collection as a sub-collection that would become pairwise-disjoint after the removal of a *small* number of elements of the universe. If Δ is a constant, then the precise meaning of *small* and *large* is that on the one hand, the size of the sub-collection is at least a constant fraction of the size of the entire collection, and on the other hand, its size is arbitrarily larger than the number of removed universe elements. The constant C represents the exact meaning of *arbitrarily larger*.

We also discuss the optimality of Theorem 10. In particular, we show that in the settings of Theorem 10 we must have $\rho(\Delta, C) \leqslant (C+1)^{-\Delta}$.

5.2 Reducing k-coloring to $(k-1)$-list-coloring

We prove the following reduction.

▶ **Theorem 6.** *Given an algorithm solving $(k-1)$-list-coloring in time $O\left((2-\varepsilon)^n\right)$ for some constant $\varepsilon > 0$, we can construct an algorithm solving k-coloring in time $O\left((2-\varepsilon')^n\right)$ for some (other) constant $\varepsilon' > 0$. Furthermore, the reduction is deterministic.*

Beigel and Eppstein [1] show that 4-list-coloring (as a special case of a $(4,2)$-CSP) can be solved in time $O(1.81^n)$. Therefore we conclude the proof of Theorem 8 regarding 5-coloring.

We include a short intuitive description of the reduction. By Theorem 4, it suffices to solve k-coloring for graphs in which most vertices have high degrees. We show that in this case, the graph has a small *dominating set*, this is a subset R of vertices such that every vertex not in R is adjacent to at least one vertex of R. Given a k-coloring of the dominating set, the problem of extending the coloring to a k-coloring of the entire graph becomes a problem of $(k-1)$-list-coloring the rest of the graph. This is because each vertex not in the dominating set has a neighbor in it, and thus has at least one of the k colors which it cannot use. Assuming the dominating set is small enough, we can enumerate over the k-colorings of vertices in it, and then solve the remaining $(k-1)$-list-coloring problem. The complete details appear in the full version.

5.3 Reducing k-coloring to $(k-2)$-list-coloring

We then prove the following much more complicated reduction.

▶ **Theorem 7.** *Given an algorithm solving $(k-2)$-list-coloring in time $O\left((2-\varepsilon)^n\right)$ for some constant $\varepsilon > 0$, we can construct an algorithm solving k-coloring with high probability in time $O\left((2-\varepsilon')^n\right)$ for some (other) constant $\varepsilon' > 0$.*

Once again, we use the 4-list-coloring algorithm of Beigel and Eppstein [1] to conclude the proof of Theorem 9 regarding 6-coloring.

The problem with generalizing the idea used in the proof of Theorem 6 is that even if R contains several neighbors of every graph vertex, it could be that in the correct k-coloring all of these neighbors are colored by the same color. In that case, the size of the list of possible colors would not get smaller than $(k-1)$. Thus, more involved algorithmic ideas are necessary for proving Theorem 7.

6 Conclusions and Open Problems

The main algorithmic contribution of the paper is Theorem 4. We use it in order to answer a few fundamental questions regarding the running time of k-coloring algorithms. In particular, we present the first $O\left((2-\varepsilon)^n\right)$ algorithms solving 5-coloring and 6-coloring, for some $\varepsilon > 0$. While the ε we can get using our tools is very small, this serves as the first proof that 5-coloring can be solved faster than we can currently compute the chromatic number in general. The upper bound on ρ in the full version shows that the magnitude of ε is a necessary consequence of using the removal lemma.

The main open problem that we leave unsettled is

▶ **Open Problem 43.** *Can we solve k-coloring in $O^*\left((2-\varepsilon_k)^n\right)$ time for some $\varepsilon_k > 0$, for every k?*

Theorem 4 makes some progress towards answering it, by giving some additional conditions on the input graph under which the answer is affirmative. In particular, we show that it holds for every graph that does not contain almost only vertices of super-constant degrees. In [12] very different techniques (using modifications of the FFT algorithm) were used to get a statement similar to Theorem 4 for graphs with bounded average degree. It seems like their methods do not extend to the case of (α, Δ)-bounded graphs, nevertheless, it is intriguing to find out whether a combination of their techniques with these presented in this paper can lead to further improvements.

While it is believed that $O^*(2^n)$ is the right bound for computing the chromatic number, we have no strong evidence to support this. There are reductions from popular problems and conjectures (like SETH) to other partitioning problems [6] or other parameterizations of the coloring problem [16]. It is interesting whether it can be showed that an $O^*\left((2-\varepsilon)^n\right)$ algorithm for computing the chromatic number would refute any other popular conjecture. This question was raised several times, including in the book of Fomin and Kratsch [11].

Another technical contribution of the paper is Theorem 10. We believe that the presented removal lemma could serve as a tool in the design of other exponential time algorithms. It would be interesting to find more problems for which it can be used.

References

1. Richard Beigel and David Eppstein. 3-coloring in time $O(1.3289^n)$. *Journal of Algorithms*, 54(2):168–204, 2005.
2. Andreas Björklund, Thore Husfeldt, Petteri Kaski, and Mikko Koivisto. Trimmed moebius inversion and graphs of bounded degree. *Theory of Computing Systems*, 47(3):637–654, 2010.
3. Andreas Björklund, Thore Husfeldt, and Mikko Koivisto. Set partitioning via inclusion-exclusion. *SIAM Journal on Computing*, 39(2):546–563, 2009.
4. Jesper Makholm Byskov. Enumerating maximal independent sets with applications to graph colouring. *Operations Research Letters*, 32(6):547–556, 2004.
5. Chris Calabro, Russell Impagliazzo, and Ramamohan Paturi. The complexity of satisfiability of small depth circuits. In *International Workshop on Parameterized and Exact Computation*, pages 75–85. Springer, 2009.
6. Marek Cygan, Holger Dell, Daniel Lokshtanov, Dániel Marx, Jesper Nederlof, Yoshio Okamoto, Ramamohan Paturi, Saket Saurabh, and Magnus Wahlström. On problems as hard as CNF-SAT. *ACM Transactions on Algorithms (TALG)*, 12(3):1–24, 2016.
7. Marek Cygan and Marcin Pilipczuk. Faster exponential-time algorithms in graphs of bounded average degree. *Information and Computation*, 243:75–85, 2015.
8. David Eppstein. Small maximal independent sets and faster exact graph coloring. In *Workshop on Algorithms and Data Structures*, pages 462–470. Springer, 2001.
9. Fedor V Fomin, Serge Gaspers, and Saket Saurabh. Improved exact algorithms for counting 3-and 4-colorings. In *International Computing and Combinatorics Conference*, pages 65–74. Springer, 2007.
10. Fedor V Fomin and Petteri Kaski. Exact exponential algorithms. *Communications of the ACM*, 56(3):80–88, 2013.
11. F.V. Fomin and D. Kratsch. *Exact Exponential Algorithms*. Texts in Theoretical Computer Science. An EATCS Series. Springer Berlin Heidelberg, 2010.
12. Alexander Golovnev, Alexander S Kulikov, and Ivan Mihajlin. Families with infants: speeding up algorithms for np-hard problems using fft. *ACM Transactions on Algorithms (TALG)*, 12(3):1–17, 2016.
13. Thomas Dueholm Hansen, Haim Kaplan, Or Zamir, and Uri Zwick. Faster k-SAT algorithms using biased-PPSZ. In *Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing*, pages 578–589, 2019.

14 Thore Husfeldt. *Graph colouring algorithms*, page 277–303. Encyclopedia of Mathematics and its Applications. Cambridge University Press, 2015. doi:10.1017/CBO9781139519793.016.

15 Russell Impagliazzo and Ramamohan Paturi. On the complexity of k-SAT. *Journal of Computer and System Sciences*, 62(2):367–375, 2001.

16 Lars Jaffke and Bart MP Jansen. Fine-grained parameterized complexity analysis of graph coloring problems. In *International Conference on Algorithms and Complexity*, pages 345–356. Springer, 2017.

17 Richard M Karp. Reducibility among combinatorial problems. In *Complexity of computer computations*, pages 85–103. Springer, 1972.

18 Robert Kennes. Computational aspects of the mobius transformation of graphs. *IEEE Transactions on Systems, Man, and Cybernetics*, 22(2):201–223, 1992.

19 Donald Ervin Knuth. Seminumerical algorithms. *The art of computer programming*, 2, 1997.

20 Vipin Kumar. Algorithms for constraint-satisfaction problems: A survey. *AI magazine*, 13(1):32–32, 1992.

21 Eugene L Lawler. A note on the complexity of the chromatic number problem, 1976.

22 László Lovász. Coverings and colorings of hypergraphs. In *Proc. 4th Southeastern Conference of Combinatorics, Graph Theory, and Computing*, pages 3–12. Utilitas Mathematica Publishing, 1973.

23 Burkhard Monien and Ewald Speckenmeyer. Solving satisfiability in less than 2n steps. *Discrete Applied Mathematics*, 10(3):287–295, 1985.

24 John W Moon and Leo Moser. On cliques in graphs. *Israel journal of Mathematics*, 3(1):23–28, 1965.

25 Ramamohan Paturi, Pavel Pudlák, Michael E Saks, and Francis Zane. An improved exponential-time algorithm for k-SAT. *Journal of the ACM (JACM)*, 52(3):337–364, 2005.

26 Marvin C Paull and Stephen H Unger. Minimizing the number of states in incompletely specified sequential switching functions. *IRE Transactions on Electronic Computers*, pages 356–367, 1959.

27 Gian-Carlo Rota. On the foundations of combinatorial theory i. theory of möbius functions. *Zeitschrift für Wahrscheinlichkeitstheorie und verwandte Gebiete*, 2(4):340–368, 1964.

28 Ingo Schiermeyer. Deciding 3-colourability in less than $O(1.415^n)$ steps. In *International Workshop on Graph-Theoretic Concepts in Computer Science*, pages 177–188. Springer, 1993.

29 T Schoning. A probabilistic algorithm for k-SAT and constraint satisfaction problems. In *40th Annual Symposium on Foundations of Computer Science (Cat. No. 99CB37039)*, pages 410–414. IEEE, 1999.

30 Larry Stockmeyer. Planar 3-colorability is polynomial complete. *ACM Sigact News*, 5(3):19–25, 1973.

31 Gerhard J Woeginger. Exact algorithms for NP-hard problems: A survey. In *Combinatorial optimization – eureka, you shrink!*, pages 185–207. Springer, 2003.

32 Frank Yates. *The design and analysis of factorial experiments*. Imperial Bureau of Soil Science Harpenden, UK, 1937.

Deterministic Maximum Flows in Simple Graphs

Tianyi Zhang ✉
Tsinghua University, Beijing, China

Abstract

In this paper we are interested in deterministically computing maximum flows in undirected simple graphs where edges have unit capacities. When the input graph has n vertices and m edges, and the maximum flow is known to be upper bounded by τ as prior knowledge, our algorithm has running time [1]$\tilde{O}(m + n^{5/3}\tau^{1/2})$; in the extreme case where $\tau = \Theta(n)$, our algorithm has running time $\tilde{O}(n^{2.17})$. This always improves upon the previous best deterministic upper bound $\tilde{O}(n^{9/4}\tau^{1/8})$ by [Duan, 2013]. Furthermore, when $\tau \geq n^{0.67}$ our algorithm is faster than a classical upper bound of $O(m + n\tau^{3/2})$ by [Karger and Levin, 1998].

2012 ACM Subject Classification Theory of computation → Network flows

Keywords and phrases graph algorithms, maximum flows, dynamic data structures

Digital Object Identifier 10.4230/LIPIcs.ICALP.2021.114

Category Track A: Algorithms, Complexity and Games

Acknowledgements I want to thank helpful discussions with my advisor Ran Duan as well as my colleague Shucheng Chi.

1 Introduction

Let $G = (V, E)$ be an undirected simple graph on n vertices and m edges, where each edge has unit capacity. Fix two special vertices $s, t \in V$ and we are interested in deterministic algorithms that compute an exact maximum flow from s to t in graph G. There has been a long line of literature on the study of maximum flows since the 60's. As one of the pioneering works, Ford and Fulkerson [5] introduced the idea of augmenting paths and proposed an algorithm that runs in time $O(m\tau)$; here τ is an upper bound on the maximum flow value. Subsequent improvements came along in [3, 7], which are known as the blocking flow algorithm and the push-relabel algorithm. For several decades, the best running time was $\tilde{O}(m \min\{m^{1/2}, n^{2/3}\} \log U)$ (U being the largest integer capacity) by [6]. Recently, a new line of developments [17, 14, 18, 16, 15, 13, 19] based on the interior point method surpassed the blocking flow barrier; for large capacities, the best running time is $\tilde{O}((m + n^{1/2}) \log U)$ [19], while for small capacities, the best running time so far is $O(m^{4/3+o(1)}U^{1/3})$ [15, 13].

Another branch of literature focuses on a special case when the maximum flow is known to be small as a prior knowledge. Let τ be a known upper bound on the maximum flow value. Karger proposed the first such kind of upper bound in [10], which is a randomized algorithm running in time $\tilde{O}(m^{2/3}n^{1/3}\tau)$ for simple graphs. This was improved later in [11] to randomized upper bounds of $\tilde{O}(m + n\tau^{5/4})$ and $\tilde{O}(m + n^{11/9}\tau)$ and deterministic upper bound $O(m + n\tau^{3/2})$. This line of works culminated in [12] as a randomized upper bound $\tilde{O}(m + n\tau)$. In a very recent work [1], the authors obtained a deterministic upper bound of $\tilde{O}(m\tau^{2/3})$ time which is even faster when τ is small.

A gap between randomized and deterministic algorithms has remained so far. For many years, the best deterministic algorithm has running time $\tilde{O}(m + n\tau^{3/2})$ [11], so in simple graphs this upper bound could be as large as $\Omega(n^{2.5})$. The basic idea of this algorithm is

[1] $\tilde{O}$ hides poly-logarithmic factors.

to sparsify the residual graph so that it always has $O(n\tau^{1/2})$ directed edges, and it uses connectivity structures to handle undirected edges, and in this way each flow augmentation takes time only proportional to the number of directed edges, yielding a total running time of $O(m + n\tau^{3/2})$. It was explicitly asked by the authors in [11] whether one could achieve a full sparsification of $O(n\tau^{1/2})$ edges so that running the blocking flow algorithm of [6] on the sparsified graph would take time $\tilde{O}(m + n^{5/3}\tau^{1/2})$.

To break through the 2.5 exponent even in simple graphs when $\tau = \Theta(n)$, the author of [4] managed to combine the original approach of [11] with the blocking flow technique and achieved a better running time $\tilde{O}(n^{9/4}\tau^{1/8})$ in simple graphs; this upper bound is always at most $\tilde{O}(n^{2.375})$ so it beats the 2.5 exponent in the worst case when $\tau = \Theta(n)$.

1.1 Our result

In this chapter, we answer the question from [11] in the affirmative for simple graphs.

▶ **Theorem 1.** *Assume $\tau \geq n^{2/3}$ is an upper bound on the value of the maximum s-t flow, then there is a deterministic algorithm that computes s-t maximum flows in undirected simple graphs in $\tilde{O}(n^{5/3}\tau^{1/2})$ running time.*

Our algorithm is always faster than [4] for any choice of $1 \leq \tau < n$ in simple graphs; in the extreme case where $\tau = \Omega(n)$, we have a running time of $O(n^{2.17})$. For a moderately large $\tau \geq n^{0.67}$, our result is also better than the classical algorithm from [11] which has $O(m + n\tau^{3/2})$ running time.

1.2 Technical overview

Generalizing the blocking flows. Our algorithm will be based on [4], so let us first try to summarize the benchmark. Throughout iterations, it maintains a set of disjoint connected components $\mathcal{V}$ in the residual graph, and define a binary edge weight function $\mu : E \to \{0, 1\}$, where inter-component edges have weight 1, and intra-component edges have weight 0. In each iteration, the algorithm searches for a blocking flow whose augmentation would increase the s-t distance in the residual graph under edge weight μ. To search for a blocking flow in the current residual graph, the algorithm performs a depth-first search only on inter-component edges, and use a connectivity data structure to route flows within components. Therefore, in general their algorithm needs to keep the total number of inter-component edges small.

The first obstacle of this approach is that the total number of directed edges would grow during augmentations. As all directed edges are inter-component ones, this immediately affects the time of finding blocking flows. To work around this issue, the algorithm of [4] was to perform a decycle operation between every pair of consecutive levels in terms of distances from s. Although this could keep the number of directed edges small, it merely translates directed edges to inter-component undirected edges, so the total number of inter-component edges remains large. To handle this large number of inter-component edges, their algorithm builds a decremental clustering structure between each pair of consecutive levels that comprises a collection of disjoint clusters and a sparse graph. More specifically, their data structure accepts an integer parameter h and decrementally maintains a set of disjoint star subgraphs, each of size $> h$, such that the number of edges incident tn unclustered vertices is bounded by $O(nh)$. The key motivation of using star subgraphs is that it ensures each flow augmentation uses at most $O(n/h)$ inter-component edges, so that the number of inter-component edges grows slowly.

A better clustering approach. One downside of [4]'s decremental clustering algorithm is that each edge deletion could add $O(n)$ more edges incident on unclustered vertices. This substantially limits the usage of the decremental data structure since it needs frequently rebuilding as the number of edges incident on unclustered vertices increases fast. Our new data structure does not need any rebuilding, and furthermore, this advantage allows us to apply it to maintain the set of disjoint components $\mathcal{V}$ directly, instead of maintaining inter-component edges between consecutive levels. In this way, we can directly upper bound the length of each flow, and so we do not need any decycling step, which simplifies the blocking flow approach of [4] as well.

Comparison with layered core decompositions. Our new decremental clustering data structure is actually inspired by the layered core decompositions devised in a recent paper [2], but ours is much simpler and weaker in some sense. The total update time of our algorithm is at least quadratic, and it also depends on the pattern of updates, while the layered core decomposition has almost linear total update time in the worst case. Furthermore, in our clustering scheme, each cluster is simply a star subgraph, where in the layered core decomposition each cluster is an expander.

One reason we do not directly apply the layered core decomposition in the maximum flow computations is that it is a heavy tool and may probably make an overkill. When computing maximum flows, edge updates to the clustering scheme always have some benign properties, and so even though star graphs are far less robust than expanders, the clustering scheme does not have to work against strong adversaries. Furthermore, a more technical reason is that paths within the same core may have sub-polynomial lengths, while each star only has diameter 2, which contributes to the length of augmenting paths. Therefore, using the layered core decompositions might induce an extra sub-polynomial factor in the running time of max-flows.

2 Preliminaries

Let $G = (V, E)$ be an undirected simple graph with unit-capacities, and let $s, t \in V$ be a source and a sink. For the rest, G will be the input graph where we compute max-flows.

For any graph $H = (X, F)$ and vertex subset $S \subseteq H$, let $H[S]$ be the induced subgraph of H on S. An edge orientation on the edge set is represented as $\mathcal{O} : V \times V \to \{-1, 0, 1\}$, such that for each edge $(u, v) \in E$, $\mathcal{O}(u, v)$ is equal to 1 if the edge (u, v) is oriented from u to v, and -1 if it is from v to u, and 0 if it is not oriented. For each $u \in V$, define $\deg_H(u)$ to be the number of neighbors of u in H. If an orientation $\mathcal{O} : F \to \{-1, 0, -1\}$ of edges is imposed on F, then let $\deg^+_{H,\mathcal{O}}(u), \deg^-_{H,\mathcal{O}}(u)$ be the out/in-degree of u in G. Usually, when H and $\mathcal{O}$ are known from context, we would ignore the subscripts.

For any s-t flow f in G, let $|f|$ denote the value of the flow, and E_f the set of edges carrying the flow. The residual graph of G with respect to f is denoted by G_f, which is defined as following.

▶ **Definition 2** (residual graph). *Given a flow f in G, a residual graph G_f is defined as following: for each edge $(u, v) \in E$ such that $f(u, v) = 1$, there is a directed edge from v to u with capacity 2.*

Here we emphasize that the orientation of edges have nothing to do with the direction of edges in the residual graph.

Next, we state some lemmas regarding flows from previous works.

▶ **Lemma 3** ([11]). *An acyclic flow f in a graph with integer capacities and no parallel edges uses at most $2n\sqrt{|f|}$ edges.*

▶ **Lemma 4** ([11]). *In a simple undirected graph, it is possible to take a flow f and find an acyclic flow f' of the same value in time $\tilde{O}(|E_f|)$.*

▶ **Lemma 5** ([11]). *Given an acyclic flow f, finding k augmenting paths takes $\tilde{O}(m + kn(k + |f|)^{1/2})$ deterministic time.*

We will also be using a dynamic maximum spanning forest algorithm with fast amortized update time.

▶ **Lemma 6** ([8, 9]). *Given an undirected weighted graph on n vertices undergoing edge insertions and deletions, there is a deterministic dynamic algorithm that maintains a maximum spanning forest with $O(\log^4 n)$ amortized update time.*

3 Decremental layered clustering

In this section, let $H = (V, F)$ be an arbitrary undirected simple graph on n vertices that undergoes a sequence of edge deletions.

▶ **Definition 7.** *For any undirected simple graph $H = (X, F)$, a subset $A \subseteq X$ is called **d-pruned** with respect to graph H, if all edges incident on A could be oriented under an orientation $\mathcal{O}$ of F such that*
(1) *All edges between A and $X \setminus A$ are directed outward from A.*
(2) *Out-degree of every vertex in A is less than d.*

▶ **Lemma 8.** *For any simple graph $H = (X, F)$, there exists a d-pruned set A, such that for any $u \in X \setminus A$, $\deg_{H[X\setminus A]}(u) \geq d$; plus such A can be computed in linear time.*

Proof. The set A is constructed in the greedy manner: starting with $A \leftarrow \emptyset$, repeatedly check if there is a vertex $u \in X \setminus A$ such that $\deg_{H[X\setminus A]} < d$; if so, orient all edges incident on u in $H[X \setminus A]$ away from u, and move u to A. So, in this way, the out-degree is small $\deg^+_H(u) < d$. Clearly, the algorithm can be implemented in linear time. By the stopping condition, all vertices in the induced subgraph $H[X \setminus A]$ have degree at least d. ◀

Now, let us define a layering scheme of H.

▶ **Definition 9** (layering). *Let $h > 0$ be an integer parameter. A partition $(A_0, A_1, \cdots, A_{\lceil \log(n/h) \rceil})$ of V, together with an orientation $\mathcal{O}$ of all edges in E, is called a layering scheme of H, if the following requirements are satisfied.*
(1) *$(u, v) \in E$ such that $u \in A_i, v \in A_j, i < j$, this edge is oriented from u to v, namely $\mathcal{O}(u, v) = 1$.*
(2) *For each index $b \geq 0$, $u \in A_b$, we have $\deg^+_{H,\mathcal{O}}(u) < 2^{b+1}h$.*

To initialize a layering of H, consider the following procedure: initialize $X = V$, and for $i = 0, 1, 2, \cdots, \lceil \log(n/h) \rceil - 1$, apply Lemma 8 on $H[X]$ with parameter $d = 2^i h$ to compute a set A_i, and then update $X \leftarrow X \setminus A_i$. After all iterations, define $A_{\lceil \log(n/h) \rceil} = X$ and orient edges in $E \cap (A_{\lceil \log(n/h) \rceil} \times A_{\lceil \log(n/h) \rceil})$ arbitrarily. By construction, for each $u \in A_b$, all its out-neighbors are in $\bigcup_{i \geq b} A_i$, and $\deg^+_{H,\mathcal{O}}(u) \leq 2^b h < 2^{b+1}h$.

On top of the layering scheme, we need to define a clustering scheme of vertices.

▶ **Definition 10** (clustering). *Given a layering* $(A_0, A_1, \cdots, A_{\lceil \log(n/h) \rceil})$ *of* H*, a set of tuples* $\{(\mathcal{C}_b, Y_b, Z_b)\}_{b \geq 1}$ *is called a* cluster structure *on layer* $(A_0, A_1, \cdots)$*, where each layer* $A_b, b \geq 1$ *can be divided into two parts* $A_b = Y_b \cup Z_b$*, if the following properties hold.*

(1) *Y_b is partitioned into a collection of clusters $\mathcal{C}_b = \{C_1, C_2, \cdots\}$ of subsets, where each C_i has size $\geq 2^{b-1}h + 1$ and is spanned by a star subgraph of H. For each C_i, its center is defined to be the center of a star subgraph that spans C_i.*

(2) *Each vertex $z \in Z_b$ is adjacent to some vertices in $Y_b \cup \bigcup_{i>b} A_i$.*

Besides, define edge weight function ω associated with this cluster structure as follows.

(a) *All edges of star graphs in $\mathcal{C}_b$ have weight $2b$.*

(b) *All non-star edges incident on Y_b in $H[A_b]$, and edges between A_b and $\bigcup_{i>b} A_i$ have weight $2b - 1$.*

(c) *All edges in $H[Z_b]$ have weight $2b - 2$.*

▶ **Lemma 11.** *Given any layering* $(A_0, A_1, \cdots, A_{\lceil \log(n/h) \rceil})$ *of* H*, a cluster structure* $\{(\mathcal{C}_b, Y_b, Z_b)\}_{b \geq 1}$ *can be computed in* $\tilde{O}(n^2)$ *time.*

Proof. For each $b \geq 1$, construct Y_b in the greedy manner: starting with an empty set $Y_b = \emptyset$, whenever there exists a vertex $c \in A_b \setminus Y_b$ with $\deg_{H[A_b \setminus Y_b]}(u) \geq 2^b h$, take an arbitrary set of $2^b h$ neighbors $u_1, u_2, \cdots, u_{2^b h} \in A_b \setminus Y_b$ and add this cluster $\{c, u_1, u_2, \cdots, u_{2^b h}\}$ to Y_b, with c being its center. In the end when the above procedure stops, define $Z_b = A_b \setminus Y_b$. Clearly this procedure takes $O(|F|) = O(n^2)$ time.

To verify property (2), by construction of the layering scheme, for each $z \in Z_b$, $\deg_{H[\cup_{i \geq b} A_i]}(z) \geq 2^b h$. So if z was not added to Y_b as a star center, then z must be adjacent to some vertices from $Y_b \cup \bigcup_{i>b} A_i$. ◀

Next, we try to maintain the layering $(A_0, A_1, \cdots)$ together with a clustering structure when edges are being deleted from G. During the execution of our decremental algorithm, we need to ensure a basic requirement.

▶ **Invariant 12.** *During the decremental algorithm, vertices could only move from layers A_{b+1} to A_b, $\forall b \geq 0$; in other words, vertices only move from upper layers to lower layers.*

The rest of the section would be devoted to the following statement.

▶ **Lemma 13.** *Suppose H undergoes a sequence of edge deletions. Then a clustering structure $\{(\mathcal{C}_b, Y_b, Z_b)\}_{b \geq 1}$ together with an induced edge weight ω in Definition 10 can be explicitly maintained using total update time $\tilde{O}(\sum_{b=1}^{\lceil \log(n/h) \rceil} 2^b D_b h + n^2)$, plus that the dynamic algorithm meets Invariant 12; here D_b is an upper bound on the number of edges which are incident on layer A_b right when they get deleted.*

3.1 Maintaining clusters under edge deletions

In this subsection we describe the algorithm behind Lemma 13. Let us first focus on a fixed layer A_b. Assume the updates to $H[A_b]$ are either edge deletions or vertex transfers from A_{b+1} to A_b; if $b = \lceil \log(n/h) \rceil$ then A_{b+1} is always empty. Initialize an arbitrary clustering $(\mathcal{C}_b, Y_b, Z_b)$ as in Lemma 11. We will be maintaining all adjacency lists in the induced subgraph $H[\bigcup_{i \geq b} A_i]$, and more importantly, the following auxiliary data structure based on edge orientation $\mathcal{O}$.

Auxiliary data structures based on edge orientation $\mathcal{O}$

- **(i)** For each $u \in A_b$, a list of all out-neighbors in H. By Invariant 12, these out-neighbors are all in $\bigcup_{i \geq b} A_i$.
- **(ii)** For each $u \in A_b$, a list of neighbors $Z^-(u) = \{z \mid (u,z) \in E, \mathcal{O}(u,z) = -1, z \in Z_b\}$.
- **(iii)** For each $u \in Z_b$, a list of neighbors $Z^+(u) = \{z \mid (u,z) \in E, \mathcal{O}(u,z) = 1, z \in Z_b\}$.
- **(iv)** A priority queue on all vertices in Z_b, which are ordered by degrees $\deg_{H[Z_b]}(u) = |Z^-(u)| + |Z^+(u)|, \forall u \in Z_b$ in the induced subgraph $H[Z_b]$.

In other words, for each vertex in Z_b, it knows both its out-neighbors and in-neighbors in $H[Z_b]$, while for each vertex in Y_b, it only knows its in-neighbors in Z_b. Plus, all vertices in Z_b are ordered by their degrees in the induced subgraph $H[Z_b]$.

Vertex insertions. Assume a new vertex u has moved to A_b from upper layer A_{b+1}. First, for each edge (u,v) where $v \in \bigcup_{i>b} A_i$, we need to adjust the orientation of $\mathcal{O}(u,v) = 1$, reassign edge weights for all edges $\omega(u,v) = 2b-1$.

By definition of layering, before u came downward from A_{b+1}, all its edges incident on A_b were directed inward. So when it joins A_b, $Z^+(u)$ should be empty. After insertion of u, we add u to Z_b, and initialize $Z^-(u)$ by scanning u's adjacency list in H, and initialize $Z^+(u) = \emptyset$. Then, for each $v \in A_b$ adjacent to u, add u to the list $Z^+(v)$. Next, update the keys of u and its neighbors in Z_b in the priority queue. Finally, if $v \in Y_b$, stay with $\omega(u,v) = 2b-1$, and otherwise reassign $\omega(u,v) = 2b-2$.

After that, repeat the following greedy procedure (1)(2)(3) to form star subgraphs out of $H[Z_b]$. For a more concise description of this procedure, check pseudo-code GreedyCluster.

- **(1)** If there exists $c \in Z_b$ such that $\deg_{H[Z_b]}(c) = |Z^-(c)| + |Z^+(c)| \geq 2^b h$, pick an arbitrary such vertex c; if there is none, then halt.
- **(2)** Find all of c's neighbors $u_1, u_2, \cdots, u_k \in Z_b$ by scanning $Z^-(c) \cup Z^+(c)$. Then move all vertices $c, u_1, u_2, \cdots, u_k$ from Z_b to Y_b, and add the star subgraph around all u_i's centered at c as a cluster to $\mathcal{C}_b$.
- **(3)** Lastly, we need to maintain all lists $Z^+(\cdot), Z^-(\cdot)$ and edge weights. To maintain lists and edge weights, for every vertex $x \in \{c, u_1, u_2, \cdots, u_k\}$, do the following steps.
  - **(a)** For each $z \in Z^-(x)$, remove x from $Z^+(z)$; for each $z \in Z^+(x)$, remove x from $Z^-(z)$. Then, for each such edge (x,z), reassign $\omega(x,z) = 2b-1$, and for all star edges $(c,u_i), 1 \leq i \leq k$, assign $\omega(c,u_i) = 2b$.
  - **(b)** Scan the out-neighborhood of x, then for each (x,y) such that $\mathcal{O}(x,y) = 1$ and $y \in Y_b$, remove x from $Z^-(y)$.

Non-star edge deletions. Now assume an edge deletion (u,v) occurs in subgraph $H[\bigcup_{i \geq b} A_i]$, and $u \in A_b$. Let us first study the simpler case where (u,v) is not a star edge in any cluster of $\mathcal{C}_b$.

A preliminary step is updating these four lists $Z^+(u), Z^-(u), Z^+(v), Z^-(v)$ due to (u,v)'s deletion. After this, some vertex $z \in Z_b$ might have a small degree in subgraph $H[\bigcup_{i \geq b} A_i]$, and in this case we would have to move it to A_{b-1} to ensure each $z \in Z_b$ is adjacent to some vertices in $Y_b \cup \bigcup_{i>b} A_i$. More specifically, while some vertices in $z \in Z_b$ have degree less than $2^b h$ in subgraph $H[\bigcup_{i \geq b} A_i]$, move z from Z_b to A_{b-1}, do the following steps to restore our data structures.

Algorithm 1 GreedyCluster.

while $\exists c \in Z_b$ *such that* $\deg_{H[Z_b]}(c) \geq 2^b h$ **do**
  list all neighbors of c in $H[Z_b]$ as $u_1, u_2, \cdots, u_k$;
  form a star subgraph around $u_1, u_2, \cdots, u_k$ centering at c, and add this as a cluster to $\mathcal{C}_b$;
  for each $x \in \{c, u_1, u_2, \cdots, u_k\}$, scan $Z^-(x), Z^+(x)$ to maintain all other affected lists $Z^-(z), Z^+(z), z \in Z_b$, and update the induced edge weights ω;
  for each $x \in \{c, u_1, u_2, \cdots, u_k\}$, scan its out-neighbors in $H[A_b]$ to update all $Z^-(y), \forall y \in Y_b$;

(1) Scan z's adjacency list, and for each of its neighbor $y \in A_b$, remove z from $Z^+(y), Z^-(y)$, and update the orientation $\mathcal{O}(z, y) = 1$ if necessary. Finally, remove z from the priority queue on Z_b.
(2) Besides, for each of z's neighbor x in $H[\bigcup_{i \geq b} A_i]$, reassign $\omega(z, x) = 2b - 3$.

The pseudo-code of the above procedure is presented below as GreedyPrune.

Algorithm 2 GreedyPrune.

while $\exists z \in Z_b$ *such that* $\deg_{H[\bigcup_{i \geq b} A_i]}(z) < 2^b h$ **do**
  move z from Z_b to A_{b-1}, and scan its adjacency list to update all affected lists $Z^+(\cdot), Z^-(\cdot)$, and update ω and $\mathcal{O}$ properly;

Star edge deletions. Now consider the harder case where the deleted edge is a star edge of a cluster in $\mathcal{C}_b$. In this case, let (c, u) be the deleted edge and c be the cluster center. After the edge deletion, one or more vertices in the cluster centered at c might have to leave Y_b. The two possibilities are the following.

- Excluding u, if the star subgraph now has at most $2^{b-1}h$ vertices, the whole cluster should be destroyed and removed from Y_b back to Z_b.
- Otherwise, only u needs to join Z_b.

Let vertex subset S be the set of all vertices in this cluster that might join Z_b shortly, so S is either a singleton $\{u\}$ or the whole star cluster. For each vertex $v \in S \setminus \{c\}$, a preliminary step is reweighing $\omega(c, v) = 2b - 1$ as (c, v) is no longer a star edge.

Go over all vertices $v \in S$ in an arbitrary order and do the following. Scan the list of all out-neighbors of v to find the set S_v of its out-neighbors in Z_b; remember that currently we do not maintain $Z^+(v)$ as $v \notin Z_b$, so instead of directly retrieving $S_v = Z^+(v)$, we have to scan all of v's out-neighbors. Now, as we have already maintained $Z^-(v)$, we can now decide whether v has at least $2^b h$ (undirected) neighbors in Z_b by checking if $|S_v| + |Z^-(v)| \geq 2^b h$. We need to study both possibilities.

- $|S_v| + |Z^-(v)| < 2^b h$. In this case, set $Z^+(v) = S_v$. Move v from Y_b to Z_b. To restore orientation-based data structures, first, for each $z \in Z^-(v)$ add v to $Z^+(z)$, and for every $z \in S_v$ add v to $Z^-(z)$; second, scan the out-neighbors of v, and for every (v, y) such that $y \in Y_b$, add v to $Z^-(y)$.
  Finally, to restore edge weights, for each of $z \in S_v \cup Z^-(v)$, change the edge weight $\omega(v, z)$ from $2b - 1$ to $2b - 2$.

- $|S_v| + |Z^-(v)| \geq 2^b h$. In this case, we would directly collect a star cluster around v using $S_v \cup Z^-(v)$. Namely, move all vertices in $S_v \cup Z^-(v)$ from Z_b to Y_b, and add the star subgraph centered at v as a cluster to $\mathcal{C}_b$. To restore the auxiliary data structures and edge weights, go over all vertices $x \in S_v \cup Z^-(v)$ and follow the same steps (3)(a) and (3)(b) when handling vertex insertions.

After we have iterated over all vertices from S, each $v \in S$ either has entered Z_b or formed a new cluster in $\mathcal{C}_b$. Subgraph $H[Z_b]$ could still contain vertices whose degree is at least $2^b h$, so we perform another round of GreedyCluster to exhaustively collect new clusters out of Z_b. Finally, to ensure property (2) in Definition 10, we invoke GreedyPrune on Z_b to remove vertices with small degrees.

Stacking all layers. In order to maintain the entire clustering structures $\{(\mathcal{C}_b, Y_b, Z_b)\}_{b\geq 1}$ across all layers $(A_0, A_1, \cdots)$ against edge deletions to H, we simply apply the above algorithm for each layer $A_b, b \geq 1$. When vertices move from upper layers to lower layers, it is equivalent to vertex insertions to lower layers, which can be handled by the algorithm.

3.2 Proof of correctness

The proof of correctness of our algorithm is divided into several lemmas.

▶ **Lemma 14.** *Invariant 12 is preserved by the algorithm, and $(A_0, A_1, \cdots)$ is always a layering satisfying the requirements in Definition 9.*

Proof. During the algorithm, vertices never move from lower levels A_b to higher levels A_{b+1}, so Invariant 12 is satisfied.

Now let us turn to verify properties of Definition 9. For property (1), when vertices move across different layers, our algorithm always adjusts $\mathcal{O}$ to ensure this requirement, so this property holds. As for property (2), by the algorithm description, whenever a vertex $z \in Z_b$ moves from A_b to A_{b-1}, it must be the case that $\deg_{H[\bigcup_{i\geq b} A_i]}(z) < 2^b h$. In the near future, as long as z stays in A_{b-1}, its out-neighbors under orientation $\mathcal{O}$ should always be a subset of its current neighbors in $\bigcup_{i\geq b} A_i$, and so the number of its out-neighbors is always bounded by $2^b h$, and thus property (2) holds. ◀

▶ **Lemma 15.** *During edge deletions, the algorithm correctly maintains the auxiliary data structures based on the edge orientation $\mathcal{O}$.*

Proof. Maintaining part (i)(iv) is straightforward, so we only focus on part (ii)(iii).

- **Vertex insertions.** Z_b could only change during GreedyCluster. Consider any new cluster $c, u_1, \cdots, u_k$ centered at c. To move them from Z_b to Y_b, for each $x \in \{c, u_1, \cdots, u_k\}$, on the one hand, we need to go over all of its neighbors in $H[Z_b]$ to fix lists $Z^-(z), Z^+(z)$ for all $z \in Z_b$; on the other hand, we only need to scan x's out-neighbors to fix lists $Z^-(y)$ for all $y \in Y_b$, since we do not require $Z^+(\cdot)$ for vertices in Y_b, which is a key point behind the design of the auxiliary data structures.
- **Non-star edge deletions.** In this case, some vertices might move from Z_b downward to A_{b-1}. As the maintenance of the auxiliary data structures is done in the straightforward manner, so correctness should follow easily.
- **Star edge deletions.** This case involves two rounds of vertex transfers between Y_b and Z_b. The first round is when it enumerates all vertices $v \in S$ and check if $|S_v| + |Z^-(v)| < 2^b h$. In this case, we have discussed two possibilities.

- If $|S_v| + |Z^-(v)| < 2^b h$, then the algorithm moves v from Y_b to Z_b. To restore part (ii)(iii), it suffices to scan $Z^-(v)$ and all out-neighbors of v, which is what the algorithm does.
- Otherwise if $|S_v| + |Z^-(v)| \geq 2^b h$, the algorithm would collect a star cluster around v with $S_v \cup Z^-(v)$. To move $S_v \cup Z^-(v)$ to Y_b, we would use a similar procedure as steps (3)(a)(b) of vertex insertions, and so the auxiliary data structures should be maintained correctly as well.

The second round is when it invokes GreedyCluster to further remove vertices $z \in Z_b$ such that $\deg_{H[Z_b]}(z) \geq 2^b h$. After that, the algorithm calls GreedyPrune to remove vertices $z \in Z_b$ such that $\deg_{H[\bigcup_{i \geq b} A_i]}(z) < 2^b h$. Using the same analysis as before, we know the algorithm always correctly maintains the auxiliary data structures. ◀

▶ **Lemma 16.** *A cluster structure $\{(\mathcal{C}_b, Y_b, Z_b)\}_{b \geq 1}$ from Definition 10 is correctly maintained by our algorithm.*

Proof. It is straightforward to verify that our algorithm correctly maintains the edge weight ω. So let us only focus on properties (1)(2).

Property (1) is automatically guaranteed by the algorithm, since the algorithm only collects star clusters of size $\geq 2^b h$, and whenever a star cluster becomes smaller than $2^{b-1} h$ after losing too many of its leaves due to edge deletions, the algorithm would try to move it back to Z_b. Now, let us turn to property (2).

▷ Claim 17. After every update, it holds that for each vertex $z \in Z_b$, $\deg_{H[\bigcup_{i \geq b} A_i]}(z) \geq 2^b h$, $\deg_{H[Z_b]}(z) < 2^b h$.

Proof of claim. When the cluster structure has just been initialized, this claim holds by construction of the layers $(A_0, A_1, \cdots)$. Next, consider vertex insertions and edge deletions separately.

- **Vertex insertions.** After a vertex insertion from upper layers, the degrees of vertices in Z_b in subgraph $H[\bigcup_{i \geq b} A_i]$ remains unchanged. By the GreedyCluster procedure, all vertices in Z_b should have degree less than $2^b h$ in $H[Z_b]$ afterwards.
- **Edge deletions.** After an edge deletion, by the end of the algorithm, it performs one round of GreedyCluster and then GreedyPrune, which first moves vertices in Z_b whose degree in $H[Z_b]$ is at least $2^b h$ to Y_b, and then moves vertices whose degree in $H[\bigcup_{i \geq b} A_i]$ is less than $2^b h$ downward to A_{b-1}. So the claim should hold when it finishes. ◀

By this claim, for any vertex $z \in Z_b$, on the one hand we know $\deg_{H[\bigcup_{i \geq b} A_i]}(z) \geq 2^b h$, and on the other hand $\deg_{H[Z_b]}(z) < 2^b h$, so there exists $y \in \bigcup_{i \geq b} A_i \setminus Z_b = Y_b \bigcup_{i > b} A_i$ adjacent to z, which is property (2). ◀

3.3 Running time analysis

▶ **Lemma 18.** *The total time of the algorithm for handling layer A_b is bounded by $O(2^b D_b h + n^2)$.*

Proof. Let us first analyze the total time of GreedyPrune throughout edge deletions. During this subroutine, the algorithm repeatedly picks $z \in Z_b$ such that $\deg_{H[\bigcup_{i \geq b} A_i]}(z) < 2^b h$ and move it to A_{b-1}. This requires scanning the adjacency list of z. Since each z can transfer from A_b to A_{b-1} for at most once, the total time is bounded by $O(|F|) = O(n^2)$.

For each cluster $C \in \mathcal{C}_b$, define $\mathsf{del}(C)$ to be the current number of star edges that have been deleted since C joined $\mathcal{C}_b$. Define a potential function, where K is a large constant:

$$\Phi = |F \cap (Z_b \times Z_b)| + 2^b h|Z_b| + K \cdot 2^b h \sum_{C \in \mathcal{C}_b} \mathsf{del}(C) = O(K \cdot 2^b D_b h + n^2)$$

▷ Claim 19. The total time of GreedyCluster across all updates is bounded by $O(2^b h D_b + n^2)$.

Proof of claim. In the while-loop, locating a $c \in Z_b$ such that $\deg_{H[Z_b]}(c) = |Z^-(c)| + |Z^+(c)| \geq 2^b h$ takes $O(\log n)$ time using the priority queue. By the algorithm, forming a cluster around c requires scanning all neighbors in $H[Z_b]$ for each $x \in \{c, u_1, u_2, \cdots, u_k\}$, plus x's out-neighbors in $H[\bigcup_{i \geq b} A_i]$. Suppose the total number of edges in $H[Z_b]$ incident on $\{c, u_1, u_2, \cdots, u_k\}$ is m_0. Then, on the one hand, the time cost would be $O(m_0 + 2^b h(k+1))$, as each x has out-neighbors at most $2^{b+1}h$ by Definition 9; on the other hand, the potential decrease of Φ is exactly $m_0 + 2^b h(k+1)$ as well, which cancels out the time cost. Hence, the total time of GreedyCluster across all updates is bounded by $O(2^b h D_b + n^2)$. ◁

A dominant part of the overall running time is handling star edge deletions, where we need to iterate over the set S. Let C be the star cluster centered at c. For any $v \in S$, computing the number of v's neighbors in Z_b takes time at most $2^{b+1}h$ as out-degree is bounded $\deg^+_{H,\mathcal{O}}(v) \leq 2^{b+1}h$ and $Z^-(v)$ is accessible as a list. Now consider two possibilities.

- $|S_v \cup Z^-(v)| < 2^b h$. In this case, the algorithm moves v back to Z_b by scanning $S_v \cup Z^-(v)$ plus all out-neighbors, and the running time is bounded by $O(2^b h)$. As for potential change, moving v to Z_b increases Φ by $|S_v \cup Z^-(v)| + 2^b h \leq 2^{b+1}h$, so the amortized cost is bounded by $O(2^b h)$.
- $|S_v \cup Z^-(v)| \geq 2^b h$. In this case, we would collect a new cluster around v. Similar to the analysis of GreedyCluster, the amortized cost of this part is zero.

If $S = \{v\}$, then this part of the amortized cost incurred by a star edge deletion is bounded by $O(2^b h)$. Plus that $\mathsf{del}(C)$ increases by 1, the amortized cost would be $K 2^b h$.

Otherwise, if S is the entire cluster centered at c, then on the one hand, summing up the above two cases, the amortized cost of iterating over S is $O(2^b h|S|)$; on the other hand, since the star C centered at c is canceled from $\mathcal{C}_b$, the potential decrease would be $K \cdot 2^b h \cdot \mathsf{del}(C)$. So the amortized cost of handling S would be $O(2^b h|S|) - K \cdot 2^b h \cdot \mathsf{del}(C)$.

A key point is that, by the algorithm, when C joined $\mathcal{C}_b$ it had at least $2^b h$ leaves by then, but now when C leaves it has $2^{b-1}h$ leaves. Therefore, $\mathsf{del}(C) \geq |S|$. Hence, if we choose K to be a sufficiently large constant, the amortized cost $O(2^b h|S|) - K \cdot 2^b h \cdot \mathsf{del}(C)$ would be negative.

To summarize, we have proved that each star edge deletion has amortized cost at most $K 2^b h$, plus that other costs are bounded by $O(2^b h D_b + n^2)$. So the total update time is $O(2^b h D_b + n^2)$. ◀

4 The main algorithm

Initialization. Let f be an empty flow, and we will keep augmenting f throughout $O(n^{2/3})$ iterations. Apply Lemma 13 on G to initialize a layering $(A_0, A_1, \cdots)$ as well as a clustering $\{(\mathcal{C}_b, Y_b, Z_b)\}_{b \geq 1}$ together with edge weights ω, where $h = \lceil \tau^{1/2} \rceil$; we can enforce $s, t \in A_0$ at the beginning since this would only increase $O(n)$ edges incident on A_0. We maintain two disjoint edge sets F_0 and F_1, where initially F_0 includes all edges in $G[V \setminus A_0]$, and let F_1 be the rest. So initially F_1 contains at most $O(nh)$ edges.

Throughout the iterations, $(V, F_0 \cup F_1)$ will be the residual graph G_f, and F_0 will be the set of all undirected edges used by the clustering structure, and F_1 could contain both undirected and directed edges in the residual graph G_f.

Iterations. In each iteration, we try to find a blocking flow with respect to f with increasing path (unweighted) length. Denote by G_f the residual graph with respect to f.

For each edge $(u, v) \in F_0 \cup F_1$, associate it with a binary weight μ such that $\mu(u, v) = 0$ if $(u, v) \in F_0$, and $\mu(u, v) = 1$ otherwise. To keep track of all connected components of (V, F_0), maintain a fully dynamic maximum weight spanning forest MSF on (V, F_0) according to weights ω using standard approach [8], and for each $u \in V$, let $\mathsf{mst}(u)$ denote the maximum weight spanning tree in MSF containing u (if $u \in A_0$, then $\mathsf{mst}(u)$ refers to u itself).

The following invariant regarding the s-t distance in G_f under edge weight μ will be guaranteed by our algorithm.

▶ **Invariant 20.** *At the beginning of the l-th iteration, the s-t distance in G_f is at least l.*

At the beginning of each iteration, perform Dijkstra's algorithm on $G_f = (V, F_0 \cup F_1, \mu)$ from s, and so each vertex $u \in V$ has a distance label $\mathsf{level}(u)$ which is set to the distance from s to u under edge weight μ. Since all edges in F_0 are undirected and have zero μ-weight, for shortest paths computations we could contract all edges in F_0, and so Dijkstra's algorithm takes time $O(|F_1| + n \log n)$; plus, vertices in any tree component in MSF can share the same label.

Later on when we search for augmenting paths, more edge weights $\mu(\cdot)$ would turn from 0 to 1 as edges are deleted from F_0. However, during this iteration, we will keep the labels $\mathsf{level}(\cdot)$ unchanged and ensure the following property.

▶ **Invariant 21.** *For any vertex $u \in V$, the distance from s to u in the residual graph G_f under edge weight μ is always at least $\mathsf{level}(u)$.*

Depth first search. Next, let focus on a single iteration. Each vertex in V has two phases: active or inactive. At the beginning of the search, activate all vertices whose label is $< l$ as well as terminal t. Starting with source vertex $u = s$. Assume inductively that we have found a sequence of vertices $s = u_0, u_1, \cdots, u_k$. As long as $k < l$ and s is active, repeat the following steps.

(1) Enumerate all vertices in $v \in \mathsf{mst}(u_k)$. For each v, try to find an edge (v, w) in G_f such that:
  (a) w is active, and $\mathsf{level}(w) = \mathsf{level}(v) + 1$.
  (b) Either the edge (v, w) is undirected or it is from v to w.
  If such an edge (v, w) is found, then assign $u_{k+1} \leftarrow w, k \leftarrow k + 1$.

(2) If no such edge (v, w) can be found, then deactivate all vertices in $\mathsf{mst}(u_k)$ and $k \leftarrow k - 1$.

Now, suppose at some point $k = l$. Since u_k is active, we know $u_k = t$. Then we try to send flows from s to t in G_f on an augmenting path of length l under edge weights μ. Recalling the way we found all of $u_0, u_1, \cdots, u_l$, each $\mathsf{mst}(u_i)$ and $\mathsf{mst}(u_{i+1})$ are connected by an edge (v_{i-1}, u_i) with positive capacity in G_f, and each pair of u_i, v_i are in the same tree component of (V, F_0), so we can route one unit of flow from s to t going through $u_0, u_1, \cdots, u_l$.

Let the augmenting path be $s = p_0, p_1, \cdots, p_e = t$, and augment f by sending one unit of flow along this path. After that, we need to update the residual graph G_f, the subgraph H on which the layered clustering is maintained, the maximum spanning forest MSF. Go over all $0 \le i < e$, and consider the following two cases.

- If (p_i, p_{i+1}) is an undirected edge in F_0, then delete it from F_0, and add a directed edge (p_{i+1}, p_i) to F_1 with residual capacity 2.
- If (p_i, p_{i+1}) is an undirected edge in F_1, then it is connecting two vertices with consecutive labels. In this case, add a directed edge (p_{i+1}, p_i) to F_1 with residual capacity 2, and delete the old edge from F_1.
- If (p_i, p_{i+1}) is a directed edge in F_1, then remove it from F_1 and add an undirected edge (p_i, p_{i+1}) with unit capacity to F_1.

After G_f is updated, we should reset $k \leftarrow 0$, and update the layered clustering structure, the maximum spanning forest accordingly.

The whole blocking flow computation is summarized as pseudo-code BlockingFlow.

Algorithm 3 BlockingFlow(f, l).

```
maintain layered clustering {(C_b, Y_b, Z_b)}_{b≥1}, maximum spanning forest MSF;
compute distance labels level(·);
activate all vertices whose label is < l as well as terminal t;
u_0 ← s, k ← 0;
while s is active do
    if k < l then
        if exists (v, w) such that w is active, v ∈ mst(u_k), level(w) = level(v) + 1 then
            u_{k+1} ← w and k ← k + 1;
        else
            deactivate the entire mst(u_k) and backtrack k ← k − 1;
    else
        send one unit of s-t flow;
        reset k ← 0, update residual graph, layered clustering, maximum spanning
         forest;
```

Now we can summarize our main algorithm as a piece of pseudo-code MaxFlow below.

Algorithm 4 MaxFlow(G, τ).

```
initialize empty flow f;
initialize a layered clustering structure on G parameterized by h = ⌈τ^{1/2}⌉;
for l = 1, 2, ··· , ⌈2n^{2/3}⌉ do
    call BlockingFlow(f, l);
decycle f using Lemma 4;
apply Lemma 5 on f exhaustively;
return f;
```

4.1 Proof of correctness

▶ **Lemma 22.** *Invariant 21 is preserved after G_f is updated.*

Proof. When a connected component in MSF is split due to edge deletions in F_0, the current distances in G_f cannot increase. If vertices move from $V \setminus A_0$ to A_0 due to degree losses, some inter-component edges would be inserted to G_f. However, such kind of edge insertions can never decrease distances, as they were connecting vertices with the same label value level$(\cdot)$. ◀

After the depth-first search is completed, we need to prove that the distance from s to t in the contracted graph G_f is at least $l+1$, namely preserving Invariant 20 for the next iteration.

▶ **Lemma 23.** *After the depth-first search which augments f, the distance from s to t in the residual graph G_f under edge weights μ is at least $l+1$.*

Proof. Suppose after all the augmentations by a blocking flow, there still exists an s-t flow with length at most l under edge weights μ. So there exists a path $s = p_0, p_1, \cdots, p_e = t$ in G_f such that $e \leq l$. Consider two possibilities.

- $\mathsf{level}(p_i) = i, \forall 0 \leq i \leq e$. Then it must be $e = l$. For any edge (p_i, p_{i+1}), the capacity from p_i to p_{i+1} should always be positive throughout the depth-first search, since the algorithm never pushes flows from level $i+1$ to i. Hence, the depth-first search could not have terminated, which is a contradiction.
- There exists $0 \leq i < e$ such that $\mathsf{level}(p_{i+1}) > \mathsf{level}(p_i) + 1$. By the algorithm, the depth-first search never adds flows directly across nonconsecutive levels, so the capacity from p_i to p_{i+1} should be positive at the beginning as well. Then, by the shortest paths computation, $\mathsf{level}(p_{i+1})$ should be at most $\mathsf{level}(p_i) + 1$, which makes a contradiction as well. ◀

By the above lemma together with Lemma 4.6 from [4], we have the following corollary.

▶ **Corollary 24** ([4]). *After $2n^{2/3}$ iterations, the residual flow of f (namely, maximum flow in G_f) becomes at most $n^{2/3}$.*

Proof. By the pigeon-hole principle, there exists a pair of consecutive levels in the residual graph G_f whose union contains at most $n^{1/3}$ vertices. Therefore, the capacity of the cut between these two levels is at most $2 \cdot (\frac{n^{1/3}}{2})^2 < n^{2/3}$. ◀

4.2 Running time analysis

Finally, let us analyze the running time of our max flow algorithm. To analyze the total time of maintaining the layered clusters, first we need some properties regarding the maximum spanning forest MSF.

▶ **Lemma 25.** *For each index b, all star edges in $\mathcal{C}_b$ are tree edges in the maximum spanning forest. Plus, for each vertex $z \in Z_b$, there exists $y \in Y_b \cup \bigcup_{i>b} A_i$ such that (y, z) is also a tree edge.*

Proof. Consider any star edge (c, u) with c being the center. If (c, u) is not a tree edge, then the tree path between u, c connects u to another vertex $v \neq c$. By definition of ω, $\omega(u, v) \leq 2b - 1$, so switching (c, u) for (u, v) in the spanning forest gives a strictly larger total weight, which makes a contradiction.

Now consider the second half of the statement. By property (2) of the clustering structure, there exists $y \in Y_b \cup \bigcup_{i>b} A_i$ adjacent to z, and so $\omega(y, z) = 2b-1$. If (y, z) is not a tree edge, then there exists a tree path that connects z to y. Consider the first edge (z, w) on this path. As the spanning forest has maximum total weight, it must be $\omega(z, w) \geq \omega(z, y) = 2b - 1$. As $z \notin Y_b$, it can only be the case that $\omega(z, w) = 2b - 1$, and so w cannot be in Z_b or $\bigcup_{i<b} A_i$. Hence, $w \in Y_b \bigcup_{i>b} A_i$, which completes the proof. ◀

▶ **Corollary 26.** *For each $u \in A_b$, there exists a tree path of at most $\log n$ edges connecting u to a star center in $\bigcup_{i \geq b} Y_i$.*

▶ **Lemma 27.** *For each index b, and for each spanning tree $T \in \mathsf{MSF}$, suppose it contains k centers in $\bigcup_{i \geq b} Y_i$, then any tree path of T contains at most $O(k \log n)$ vertices in A_b.*

Proof. Suppose otherwise a tree path contains $20k \log n$ vertices in A_b, then there exists a sub-sequence of this tree path $u_1, u_2, \cdots, u_{2k}$ vertices in A_b, such that for each $1 \leq j < 2k$, the tree path from u_j to u_{j+1} has at least $9 \log n$ tree edges. By Corollary 26, there exists a center $v_j \in \bigcup_{i \geq b} Y_i$ such that u_j, v_j are connected by a tree path of length at most $\log n$ in MSF. By the pigeon-hole principle, there exists distinct indices $j_1 < j_2$ such that $v_{j_1} = v_{j_2}$, which means u_{j_1}, u_{j_2} are connected by a tree path of at most $2 \log n$ edges, contradiction. ◀

▶ **Lemma 28.** *The overall running time throughout all iterations of maintaining the layered clusters*

$$\{(\mathcal{C}_b, Y_b, Z_b)\}_{b \geq 1}$$

together with its induced edge weight ω is bounded by $\tilde{O}(n^2 + n\tau)$. Also, the total update time of maintaining the maximum spanning forest MSF is also bounded by $\tilde{O}(n^2 + n\tau)$.

Proof. Consider any unit flow that we send from s to t during this round. Focus on the part within the same component of MSF. By the routing scheme, the flow always uses the tree edges of the maximum spanning tree. By Lemma 27, the length of the tree path is at most proportional to the number of centers. Therefore, for the b-th layer, the number of vertices in A_b on the unit flow is bounded by $O(\frac{n}{2^b h} \log n)$. Since the augmenting path is a simple path, the number of edge deletions incident on A_b is at most $O(\frac{n}{2^b h} \log n)$ as well. Summing over at most τ augmentations, the total edge deletions incident on A_b is at most $\tilde{O}(\frac{n\tau}{2^b h})$. By Lemma 13, the total update time of maintaining the layered clustering is $\tilde{O}(n^2 + n\tau)$.

As for the dynamic maximum spanning forest MSF, the number of changes to G_f and ω is always upper bounded by the time to maintain the layered clustering, so the total time of maintaining MSF is bounded similarly. ◀

Next we bound the total number of new inter-component edges added to F_1 throughout all $2n^{2/3}$ iterations for a single round.

▶ **Lemma 29.** *During the for-loop of MaxFlow, the total number of edges in F_1 is bounded by $O(n\tau^{1/2} \log n)$.*

Proof. Similar to the previous lemma, we can prove that each augmentation turns at most $O(n/h \log n)$ undirected edges from F_0 to be directed edges in F_1. Since F_1 contains at most $O(nh)$ initially before the first iteration, the overall edges to F_1 is bounded by $O(nh + \frac{n\tau \log n}{h}) = O(n\tau^{1/2} \log n)$. ◀

The above lemma helps us bounding the total time of computing blocking flows through all iterations.

▶ **Lemma 30.** *The running time of a single iteration takes time $\tilde{O}(n\tau^{1/2})$.*

Proof. First, invoking Dijkstra's algorithm takes time $O(|F_1| + n \log n) = \tilde{O}(n\tau^{1/2})$. Next, we need to argue that in each iteration, the blocking flow procedure takes time $\tilde{O}(|F_1| + n)$ as well. In fact, we claim that each edge $(v, w) \in F_1$ can be visited by at most twice during the depth-first search. If $\mathsf{level}(w) \neq \mathsf{level}(v) + 1$, then the algorithm does to need to consider it when searching for augmenting paths, as $\mathsf{level}(\cdot)$ does not change within this iteration. Now let us assume $\mathsf{level}(w) = \mathsf{level}(v) + 1$. If (v, w) is a directed edge, then after two augmentations it would be a directed edge from w to v, and so it would never be visited again; if (v, w) is an undirected edge, then after one augmentation it would be a directed one from w to v as well.

Finally, we efficiently enumerate edges incident on any tree in MSF under edge deletions, we could arrange all vertices in this tree according to the Euler-tour which supports fast link and cut operations on trees. ◀

By the above lemmas, the total time of for-loop is bounded by $\tilde{O}(n^2 + n\tau + n^{5/3}\tau^{1/2}) = \tilde{O}(n^{5/3}\tau^{1/2})$. After the while-loop, the total residual flow is bounded by $O(n^{2/3})$, so by Lemma 5 the rest takes time $\tilde{O}(n^{5/3}\tau^{1/2})$ as well.

References

1. Julia Chuzhoy, Yu Gao, Jason Li, Danupon Nanongkai, Richard Peng, and Thatchaphol Saranurak. A deterministic algorithm for balanced cut with applications to dynamic connectivity, flows, and beyond. In *2020 IEEE 61st Annual Symposium on Foundations of Computer Science (FOCS)*, pages 1158–1167. IEEE, 2020.
2. Julia Chuzhoy and Thatchaphol Saranurak. Deterministic algorithms for decremental shortest paths via layered core decomposition. In *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 2478–2496. SIAM, 2021.
3. Efim A Dinic. Algorithm for solution of a problem of maximum flow in networks with power estimation. In *Soviet Math. Doklady*, volume 11, pages 1277–1280, 1970.
4. Ran Duan. Breaking the $\mathcal{O}(n^{2.5})$ deterministic time barrier for undirected unit-capacity maximum flow. In *Proceedings of the twenty-fourth annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1171–1179. SIAM, 2013.
5. Lester Randolph Ford Jr and Delbert Ray Fulkerson. *Flows in networks*. Princeton university press, 2015.
6. Andrew V Goldberg and Satish Rao. Beyond the flow decomposition barrier. *Journal of the ACM (JACM)*, 45(5):783–797, 1998.
7. Andrew V Goldberg and Robert E Tarjan. A new approach to the maximum-flow problem. *Journal of the ACM (JACM)*, 35(4):921–940, 1988.
8. Jacob Holm, Kristian De Lichtenberg, and Mikkel Thorup. Poly-logarithmic deterministic fully-dynamic algorithms for connectivity, minimum spanning tree, 2-edge, and biconnectivity. *Journal of the ACM (JACM)*, 48(4):723–760, 2001.
9. Jacob Holm, Eva Rotenberg, and Christian Wulff-Nilsen. Faster fully-dynamic minimum spanning forest. In *Algorithms-ESA 2015*, pages 742–753. Springer, 2015.
10. David R Karger. Using random sampling to find maximum flows in uncapacitated undirected graphs. In *Proceedings of the twenty-ninth annual ACM symposium on Theory of computing*, pages 240–249, 1997.
11. David R Karger and Matthew S Levine. Finding maximum flows in undirected graphs seems easier than bipartite matching. In *Proceedings of the thirtieth annual ACM symposium on Theory of computing*, pages 69–78, 1998.
12. David R Karger and Matthew S Levine. Random sampling in residual graphs. In *Proceedings of the thiry-fourth annual ACM symposium on Theory of computing*, pages 63–66, 2002.
13. Tarun Kathuria. A potential reduction inspired algorithm for exact max flow in almost $\tilde{O}(m^{4/3})$ time. *arXiv preprint*, 2020. `arXiv:2009.03260`.
14. Yin Tat Lee and Aaron Sidford. Path finding methods for linear programming: Solving linear programs in $\tilde{O}(\sqrt{rank})$ iterations and faster algorithms for maximum flow. In *2014 IEEE 55th Annual Symposium on Foundations of Computer Science*, pages 424–433. IEEE, 2014.
15. Yang P Liu and Aaron Sidford. Faster divergence maximization for faster maximum flow. *arXiv preprint*, 2020. `arXiv:2003.08929`.
16. Yang P Liu and Aaron Sidford. Faster energy maximization for faster maximum flow. In *Proceedings of the 52nd Annual ACM SIGACT Symposium on Theory of Computing*, pages 803–814, 2020.

17 Aleksander Madry. Navigating central path with electrical flows: From flows to matchings, and back. In *2013 IEEE 54th Annual Symposium on Foundations of Computer Science*, pages 253–262. IEEE, 2013.

18 Aleksander Madry. Computing maximum flow with augmenting electrical flows. In *2016 IEEE 57th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 593–602. IEEE, 2016.

19 Jan van den Brand, Yin Tat Lee, Yang P Liu, Thatchaphol Saranurak, Aaron Sidford, Zhao Song, and Di Wang. Minimum cost flows, mdps, and *ell*_1-regression in nearly linear time for dense instances. *arXiv e-prints*, 2021. `arXiv:2101.05719`.

Arboreal Categories and Resources

Samson Abramsky
Department of Computer Science, University of Oxford, UK

Luca Reggio
Department of Computer Science, University of Oxford, UK

Abstract

We introduce *arboreal categories*, which have an intrinsic process structure, allowing dynamic notions such as bisimulation and back-and-forth games, and resource notions such as number of rounds of a game, to be defined. These are related to extensional or "static" structures via *arboreal covers*, which are resource-indexed comonadic adjunctions. These ideas are developed in a very general, axiomatic setting, and applied to relational structures, where the comonadic constructions for pebbling, Ehrenfeucht-Fraïssé and modal bisimulation games recently introduced in [1, 5, 6] are recovered, showing that many of the fundamental notions of finite model theory and descriptive complexity arise from instances of arboreal covers.

2012 ACM Subject Classification Theory of computation → Categorical semantics; Theory of computation → Finite Model Theory

Keywords and phrases factorisation system, embedding, comonad, coalgebra, open maps, bisimulation, game, resources, relational structures, finite model theory

Digital Object Identifier 10.4230/LIPIcs.ICALP.2021.115

Category Track B: Automata, Logic, Semantics, and Theory of Programming

Related Version *Extended Version*: `https://arxiv.org/abs/2102.08109`

Funding *Samson Abramsky*: Research supported by the EPSRC project EP/T00696X/1 "Resources and Co-Resources: a junction between categorical semantics and descriptive complexity".
Luca Reggio: Research supported by the European Union's Horizon 2020 research and innovation programme under the Marie Skłodowska-Curie grant agreement No 837724.

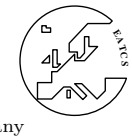
Acknowledgements Feedback from Tomáš Jakl and Dan Marsden is gratefully acknowledged.

1 Introduction

In previous work ([1, 5, 6]), it has been shown how a range of model comparison games which play a central role in finite model theory, including Ehrenfeucht-Fraïssé, pebbling, and bisimulation games, can be captured in terms of resource-indexed comonads on the category of relational structures and homomorphisms. This was done for k-pebble games in [1], and extended to Ehrenfeucht-Fraïssé games, and bisimulation games for the modal fragment, in [5]. In subsequent work, this has been further extended to games for generalized quantifiers [9], and for guarded fragments of first-order logic [3]. An important feature of this comonadic analysis is that it leads to novel characterisations of important combinatorial parameters such as tree-width and tree-depth. The coalgebras for each of these comonads correspond to certain forms of tree decompositions of structures, with the resource index matching the corresponding combinatorial parameter.

This leads to the question motivating the present paper:

> Can we capture the significant common elements of these constructions?



Our aim is to develop an elegant axiomatic account, based on clear conceptual principles, which will yield all these examples and more, and allow a deeper and more general understanding of resources.

Conceptually, a key ingredient is the assignment of a process structure – an intensional description – to an extensional object, such as a function, a set, or a relational structure. It is this process structure, unfolding in space and time, to which a resource parameter can be applied, which can then be transferred to the extensional object. At the basic level of computability, this happens when we assign a Turing machine description or a Gödel number to a recursive function. It is then meaningful to assign a complexity measure to the function. The same phenomenon arises in semantics: for example, the notion of *sequentiality* is applicable to a *process* computing a higher-order function. Reifying these processes in the form of *game semantics* led to a resolution of the famous full abstraction problem for PCF [2, 11], and to a wealth of subsequent results [20].

It is now becoming clear that this phenomenon is at play in the game comonads described in [1, 5, 6, 9, 3]. They build tree-structured covers of a given, purely extensional relational structure. Such a tree cover will in general not have the full properties of the original structure, but be a "best approximation" in some resource-restricted setting. More precisely, this means that we have an adjunction, yielding the corresponding comonad. The objects of the category where the approximations live have an intrinsic tree structure, which can be captured axiomatically. The tree encodes a process for generating (parts of) the relational structure, to which resource notions can be applied.

In this paper, we make this intuition precise. We introduce a notion of *arboreal category*, and show how all the examples of game comonads considered to date arise from *arboreal covers*, i.e. adjunctions between extensional categories of relational structures, and arboreal categories. Importantly, these adjunctions are comonadic, and the categories of coalgebras provide a setting for a general notion of bisimulation, which yields a wide range of logical equivalences in the examples. This notion refines the open maps formulation of bisimulation [13, 12] with the condition that the maps are *pathwise embeddings*, generalizing the ideas introduced in [6]. This allows a much wider range of logical equivalences to be captured.

After some preliminaries, we shall develop the axiomatization of paths, open pathwise embeddings and bisimulations, and arboreal categories. Then we establish the correspondence between bisimulations and back-and-forth equivalences in the setting of arboreal categories. Next, we show how many of the fundamental notions of finite model theory and descriptive complexity arise from instances of arboreal covers. We shall use the concrete constructions in finite model theory as running examples throughout.

We conclude this introduction by observing that the notion of *extendability*, a key ingredient for Rossman-type preservation theorems [19], can be defined in this general setting (more details are provided in the expanded version [4]).

2 Preliminaries

We shall assume familiarity with standard notions in category theory. All needed background can be found in [7, 16]. All categories under consideration are assumed to be locally small and *well-powered*, i.e. every object has a set of subobjects (as opposed to a proper class).

▶ **Example 1.** The extensional categories of primary interest in this paper are categories of relational structures. A relational vocabulary σ is a set of relation symbols R, each with a specified positive integer arity. A σ-structure $\mathcal{A}$ is given by a set A, the universe of the structure, and for each R in σ with arity n, a relation $R^{\mathcal{A}} \subseteq A^n$. A homomorphism

$h\colon \mathcal{A} \to \mathcal{B}$ is a function $h\colon A \to B$ such that, for each relation symbol R of arity n in σ, for all $a_1, \dots, a_n$ in A, $R^{\mathcal{A}}(a_1, \dots, a_n) \Rightarrow R^{\mathcal{B}}(h(a_1), \dots, h(a_n))$. We write $\mathsf{Struct}(\sigma)$ for the category of σ-structures and homomorphisms. The Gaifman graph of a structure $\mathcal{A}$ is a graph with vertices A, such that two distinct elements are adjacent if they both occur in some tuple $\vec{a} \in R^{\mathcal{A}}$ for some relation symbol R in σ.

2.1 Proper factorisation systems

We recall the notion of weak factorisation system in a category $\mathcal{C}$. Given arrows e and m in $\mathcal{C}$, we say that e has the *left lifting property* with respect to m, or that m has the *right lifting property* with respect to e, if for every commutative square as on the left-hand side below

there exists a (not necessarily unique) *diagonal filler*, i.e. an arrow d such that the right-hand diagram above commutes. If this is the case, we write $e \pitchfork m$. For any class $\mathcal{H}$ of morphisms in $\mathcal{C}$, let ${}^{\pitchfork}\mathcal{H}$ (respectively $\mathcal{H}^{\pitchfork}$) be the class of morphisms having the left (respectively right) lifting property with respect to every morphism in $\mathcal{H}$.

▶ **Definition 2.** *A pair of classes of morphisms $(\mathcal{Q}, \mathcal{M})$ in a category $\mathcal{C}$ is a* weak factorisation system *provided it satisfies the following conditions:*

(i) *every morphism f in $\mathcal{C}$ can be written as $f = m \circ e$ with $e \in \mathcal{Q}$ and $m \in \mathcal{M}$;*

(ii) $\mathcal{Q} = {}^{\pitchfork}\mathcal{M}$ *and* $\mathcal{M} = \mathcal{Q}^{\pitchfork}$.

A proper factorisation system *is a weak factorisation system $(\mathcal{Q}, \mathcal{M})$ such that $\mathcal{Q} \subseteq \{\text{epis}\}$ and $\mathcal{M} \subseteq \{\text{monos}\}$. A proper factorisation system is* stable *if, for any $e \in \mathcal{Q}$ and $m \in \mathcal{M}$ with common codomain, the pullback of e along m exists and belongs to $\mathcal{Q}$.*[1]

▶ Remark 3. Any proper factorisation system is an *orthogonal* factorisation system, i.e. the diagonal fillers are unique. In particular, factorisations are unique up to (unique) isomorphism.

▶ **Example 4.** If $\mathcal{A}$ is a relational structure, then for any $S \subseteq A$, there is an induced substructure with universe S. The inclusion map $S \hookrightarrow A$ is an *embedding*, i.e. an injective homomorphism which reflects as well as preserves relations. Any embedding $m\colon \mathcal{A} \to \mathcal{B}$ factors as $\mathcal{A} \cong \mathsf{Im}(m) \hookrightarrow \mathcal{B}$. Taking $\mathcal{Q}$ to be the surjective homomorphisms and $\mathcal{M}$ to be the embeddings gives a proper factorisation system on $\mathsf{Struct}(\sigma)$. This factorisation system is stable because pullbacks in $\mathsf{Struct}(\sigma)$ are computed in the category of sets and functions, where (surjections, injections) is a stable proper factorisation system.

Next, we state some well known properties of weak factorisation systems (cf. [10] or [18]):

▶ **Lemma 5.** *Let $(\mathcal{Q}, \mathcal{M})$ be a weak factorisation system in $\mathcal{C}$. The following hold:*

(a) *$\mathcal{Q}$ and $\mathcal{M}$ are closed under compositions.*

(b) $\mathcal{Q} \cap \mathcal{M} = \{\textit{isomorphisms}\}$.

(c) *The pullback in $\mathcal{C}$ of an $\mathcal{M}$-morphism along any morphism, if it exists, is again in $\mathcal{M}$.*

Moreover, if $(\mathcal{Q}, \mathcal{M})$ is proper, then the following hold:

(d) *$g \circ f \in \mathcal{Q}$ implies $g \in \mathcal{Q}$.*

(e) *$g \circ f \in \mathcal{M}$ implies $f \in \mathcal{M}$.*

[1] In the literature, the adjective *stable* is usually reserved for the stronger property stating that, for every $e \in \mathcal{Q}$, the pullback of e along any morphism exists and belongs to $\mathcal{Q}$.

Throughout this paper, we will refer to $\mathcal{M}$-morphisms as *embeddings* and denote them by $\rightarrowtail$. $\mathcal{Q}$-morphisms will be referred to as *quotients* and denoted by $\twoheadrightarrow$.

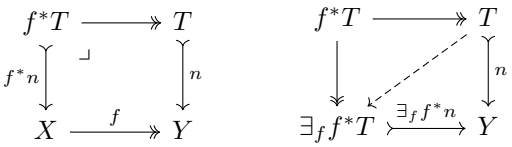
Assume $\mathcal{C}$ is a category admitting a proper factorisation system $(\mathcal{Q}, \mathcal{M})$. In the same way that one usually defines the poset of subobjects of a given object $X \in \mathcal{C}$, we can define the poset of $\mathcal{M}$-subobjects of X. Given embeddings $m\colon S \rightarrowtail X$ and $n\colon T \rightarrowtail X$, let us say that $m \trianglelefteq n$ provided there is a morphism $i\colon S \to T$ such that $m = n \circ i$ (note that i is necessarily an embedding). This yields a preorder on the class of all embeddings with codomain X. The symmetrization $\sim$ of $\trianglelefteq$ can be characterised as follows: $m \sim n$ if, and only if, there exists an isomorphism $i\colon S \to T$ such that $m = n \circ i$. Let $\mathbb{S}\,X$ be the class of $\sim$-equivalence classes of embeddings with codomain X, equipped with the natural partial order $\leq$ induced by $\trianglelefteq$. We shall systematically represent a $\sim$-equivalence class by any of its representatives. Because $\mathcal{C}$ is well-powered and $\mathcal{M} \subseteq \{\text{monos}\}$, we see that $\mathbb{S}\,X$ is a set.

For any morphism $f\colon X \to Y$ and embedding $m\colon S \rightarrowtail X$, we can consider the $(\mathcal{Q}, \mathcal{M})$-factorisation $S \twoheadrightarrow \exists_f S \rightarrowtail Y$ of $f \circ m$. This yields a monotone map $\exists_f\colon \mathbb{S}\,X \to \mathbb{S}\,Y$ sending m to the embedding $\exists_f S \rightarrowtail Y$. (Note that the map $\exists_f$ is well-defined because factorisations are unique up to isomorphism.) Further, if $(\mathcal{Q}, \mathcal{M})$ is stable and f is a quotient, we let $f^*\colon \mathbb{S}\,Y \to \mathbb{S}\,X$ be the monotone map sending $n\colon T \rightarrowtail Y$ to its pullback along f. It is not difficult to see that f^* is right adjoint to $\exists_f$.

▶ **Lemma 6.** *Let $\mathcal{C}$ be any category equipped with a stable proper factorisation system, and let $f\colon X \to Y$ be any morphism in $\mathcal{C}$. The following statements hold:*
(a) *If f is an embedding, then $\exists_f\colon \mathbb{S}\,X \to \mathbb{S}\,Y$ is an order-embedding.*
(b) *If f is a quotient, then $f^*\colon \mathbb{S}\,Y \to \mathbb{S}\,X$ is an order-embedding.*

Proof. For item (a) note that, as $f\colon X \to Y$ is an embedding, $\exists_f\colon \mathbb{S}\,X \to \mathbb{S}\,Y$ sends m to $f \circ m$. Let $m_1\colon S_1 \rightarrowtail X$ and $m_2\colon S_2 \rightarrowtail X$ be embeddings such that $f \circ m_1 \leq f \circ m_2$. Then there exists $k\colon S_1 \to S_2$ such that $f \circ m_1 = f \circ m_2 \circ k$. Because f is a monomorphism, it follows that $m_1 = m_2 \circ k$, i.e. $m_1 \leq m_2$. Hence, $\exists_f$ is an order-embedding.

For item (b), it is enough to prove that $\exists_f f^* n = n$ for any $n\colon T \rightarrowtail Y$, for then $f^* n_1 \leq f^* n_2$ implies $n_1 = \exists_f f^* n_1 \leq \exists_f f^* n_2 = n_2$. Consider the pullback of f along n, as displayed on the left-hand side below.



Since the square on the right-hand side above commutes, there exists a diagonal filler $T \to \exists_f f^* T$. Note that this diagonal filler must be both a quotient and an embedding, hence an isomorphism. Therefore, $\exists_f f^* n = n$ in $\mathbb{S}\,Y$. ◀

3 Path Categories

3.1 Paths

Throughout this section, we fix a category $\mathcal{C}$ equipped a stable proper factorisation system.

If $(P, \leq)$ is a poset, then $C \subseteq P$ is a *chain* if it is linearly ordered. $(P, \leq)$ is a *forest* if, for all $x \in P$, the set $\downarrow x := \{y \in P \mid y \leq x\}$ is a finite chain. The *height* of a forest is the supremum of the cardinalities of its chains. The *covering relation* $\prec$ associated with a partial order $\leq$ is defined by $u \prec v$ if and only if $u < v$ and there is no w such that $u < w < v$. It is

convenient to allow the empty forest. The *roots* of a forest are the minimal elements. A *tree* is a forest with at most one root. Morphisms of forests are maps which preserve roots and the covering relation. Equivalently, a monotone map $\varphi\colon U \to V$ between forests is a forest morphism if, for all $u \in U$, the restriction of φ yields a bijection between $\downarrow u$ and $\downarrow \varphi(u)$. The category of forests is denoted by $\mathcal{F}$, and the full subcategory of trees by $\mathcal{T}$. We equip these categories with the factorisation system (surjective morphisms, injective morphisms).

▶ **Definition 7.** *An object X of $\mathcal{C}$ is called a* path *provided the poset $\mathbb{S}\,X$ is a finite chain. Paths will be denoted by $P, Q, R, \ldots$.*

▶ **Example 8.** The paths in $\mathcal{F}$ and $\mathcal{T}$ are the finite chains, i.e. the trees consisting of a single branch.

▶ **Example 9.** We define a *forest-ordered σ-structure* $(\mathcal{A}, \leq)$ to be a σ-structure $\mathcal{A}$ with a forest order $\leq$ on A. A morphism of forest-ordered σ-structures $f\colon (\mathcal{A}, \leq) \to (\mathcal{B}, \leq')$ is a σ-homomorphism $f\colon \mathcal{A} \to \mathcal{B}$ that is also a forest morphism. This determines a category $\mathcal{R}(\sigma)$. We equip $\mathcal{R}(\sigma)$ with the factorisation system given by (surjective morphisms, embeddings), where an embedding is a morphism which is an embedding *qua* σ-homomorphism.

In [6], it is shown that the categories of coalgebras for the various comonads studied there are given, up to isomorphism, by subcategories of $\mathcal{R}(\sigma)$ (or minor variants thereof):

- For the Ehrenfeucht-Fraïssé comonad, this is the full subcategory $\mathcal{R}^E(\sigma)$ determined by those objects satisfying the condition (E): adjacent elements of the Gaifman graph of $\mathcal{A}$ are comparable in the forest order. For each $k > 0$, $\mathcal{R}^E_k(\sigma)$ is the full subcategory of $\mathcal{R}^E(\sigma)$ of those forest orders of height $\leq k$. The objects $(\mathcal{A}, \leq)$ of $\mathcal{R}^E_k(\sigma)$ are forest covers of $\mathcal{A}$ witnessing that its tree-depth is $\leq k$ [17].
- For the pebbling comonad, for each $k > 0$ this is the category $\mathcal{R}^P_k$ whose objects have the form $(\mathcal{A}, \leq, p)$, where $(\mathcal{A}, \leq)$ is a forest-ordered σ-structure, and $p\colon A \to [k]$ is a pebbling function. In addition to condition (E), these structures have to satisfy the condition (P): if a is adjacent to b in the Gaifman graph of $\mathcal{A}$, and $a < b$ in the forest order, then for all x such that $a < x \leq b$, $p(a) \neq p(x)$. It is shown in [6] that these structures are equivalent to the more familiar form of tree decomposition used to define tree-width [14]. Morphisms have to preserve the pebbling function.
- For the modal comonad, the category $\mathcal{R}^M_k$ has as objects the tree-ordered σ-structures of height $\leq k$ satisfying the condition (M): for $x, y \in A$, $x \prec y$ if and only if for some unique binary relation R_α in σ ("transition relation"), $R^{\mathcal{A}}_\alpha(x, y)$.

The paths in each of these categories are those structures in which the order is a finite chain. These are our key motivating examples for paths. Note that in the (multi-)modal case, ignoring propositional variables, these correspond to synchronization trees consisting of a single branch, i.e. traces.

The following fact is an immediate consequence of Lemma 6:

▶ **Lemma 10.** *Let $f\colon X \to Y$ be any morphism in $\mathcal{C}$. The following statements hold:*

(a) *If Y is a path and f is an embedding, then X is a path.*

(b) *If X is path and f is a quotient, then Y is a path.*

A *path embedding* is an embedding $P \rightarrowtail X$ whose domain is a path. Given any object X of $\mathcal{C}$, we let $\mathbb{P}\,X$ be the sub-poset of $\mathbb{S}\,X$ consisting of the path embeddings. By Lemma 10(b), for any arrow $f\colon X \to Y$, the monotone map $\exists_f\colon \mathbb{S}\,X \to \mathbb{S}\,Y$ restricts to a monotone map

$$\mathbb{P}\,f\colon \mathbb{P}\,X \to \mathbb{P}\,Y, \quad (m\colon P \rightarrowtail X) \mapsto (\exists_f m\colon \exists_f P \rightarrowtail Y).$$

By the uniqueness up to isomorphism of factorisations, this assignment is functorial.

3.2 Path categories

▶ **Definition 11.** *A* path category *is a category $\mathcal{C}$ satisfying the following conditions:*

(i) *$\mathcal{C}$ has a stable proper factorisation system;*

(ii) *$\mathcal{C}$ has all coproducts of small families of paths;*

(iii) *for any paths P, Q, R, if a composite $P \to Q \to R$ is a quotient, then so is $P \to Q$.*

▶ Remark 12. Item (iii) above is equivalent to the following condition: For any paths P, Q, R and morphisms $f\colon P \to Q$ and $g\colon Q \to R$, if any two of f, g, and $g \circ f$ are quotients, then so is the third. Thus, we shall refer to (iii) as the *2-out-of-3 condition*.

Any path category has an initial object, obtained as the coproduct of the empty family.

▶ **Example 13.** $\mathcal{F}$ and $\mathcal{T}$ are path categories. Coproducts of forests are given by disjoint union. For trees, coproducts are given by smash sum, with the bottom elements identified. Since forest morphisms preserve height, we see that $\mathcal{F}$ and $\mathcal{T}$ satisfy the 2-out-of-3 condition. Similarly, it is not difficult to see that $\mathcal{R}(\sigma)$ and its subcategories mentioned in Example 9 are all path categories ($\mathcal{R}(\sigma)$ has an initial object because we allow empty σ-structures).

▶ **Theorem 14.** *Let $\mathcal{C}$ be a path category. Then the assignment $X \mapsto \mathbb{P}\,X$ induces a functor $\mathbb{P}\colon \mathcal{C} \to \mathcal{T}$ into the category of trees.*

To prove this theorem, we start by showing that each poset $\mathbb{P}\,X$ is a tree.

▶ **Lemma 15.** *Let $\mathcal{C}$ be a path category. For any object X of $\mathcal{C}$, $\mathbb{P}\,X$ is a non-empty tree.*

Proof. Using Lemma 6(a), it is not difficult to see that the sub-poset of $\mathbb{P}\,X$ consisting of those elements that are below a given $P \in \mathbb{P}\,X$ is isomorphic to $\mathbb{P}\,P$. In turn, $\mathbb{P}\,P \cong \mathbb{S}\,P$ by Lemma 10(a). Since $\mathbb{S}\,P$ is a finite chain, we see that $\mathbb{P}\,X$ is a forest. Now, let $\mathbf{0} \twoheadrightarrow \widetilde{\mathbf{0}} \overset{m}{\rightarrowtail} X$ be the (quotient, embedding) factorisation of the unique morphism $\mathbf{0} \to X$ from the initial object. We claim that $m\colon \widetilde{\mathbf{0}} \rightarrowtail X$ is the unique root of $\mathbb{P}\,X$.

Note that $\mathbf{0}$ is a path: just observe that any embedding $S \rightarrowtail \mathbf{0}$ admits the unique morphism $\mathbf{0} \to S$ as a right inverse, and thus is a retraction. It follows that $S \cong \mathbf{0}$, i.e., $\mathbb{S}\,\mathbf{0}$ is the one-element poset. In particular, $\mathbf{0}$ is a path. Thus, $\widetilde{\mathbf{0}}$ is a path by Lemma 10(b). We show that $m\colon \widetilde{\mathbf{0}} \rightarrowtail X$ is the least element of $\mathbb{P}\,X$. If $m'\colon P \rightarrowtail X$ is any path embedding, we have a commutative square as follows.

$$\begin{array}{ccc} \mathbf{0} & \twoheadrightarrow & \widetilde{\mathbf{0}} \\ \downarrow & & \downarrow{\scriptstyle m} \\ P & \overset{m'}{\rightarrowtail} & X \end{array}$$

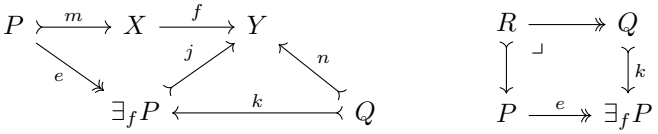
Hence there exists a diagonal filler $d\colon \widetilde{\mathbf{0}} \to P$, and so $m \leq m'$ in $\mathbb{P}\,X$. ◀

We next show that the functor $\mathbb{P}$ sends morphisms in a path category to tree morphisms, thus completing the proof of Theorem 14.

▶ **Proposition 16.** *Let $\mathcal{C}$ be a path category. For any arrow f in $\mathcal{C}$, $\mathbb{P}\,f$ is a tree morphism.*

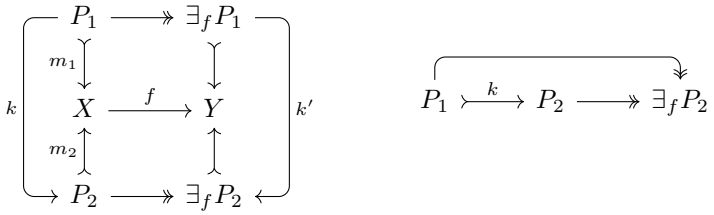
Proof. It is enough to show that, for any path embedding $m\colon P \rightarrowtail X$, the induced map $\mathbb{P}\,f\colon {\downarrow}m \to {\downarrow}\mathbb{P}\,f(m)$ is a bijection.

We start by establishing surjectivity, i.e. $\downarrow \mathbb{P}\, f(m) \subseteq \mathbb{P}\, f(\downarrow m)$. Let (e, j) be the (quotient, embedding) factorisation of $f \circ m$. If $n \colon Q \rightarrowtail Y$ is a path embedding such that $n \leq \mathbb{P}\, f(m)$ in $\mathbb{P}\, Y$, there exists an embedding $k \colon Q \rightarrowtail \exists_f P$ such that the left-hand diagram below commutes. Consider the pullback of k along e, as displayed in the right-hand diagram below.



Then R is a path by Lemmas 5(c) and 10(a), and the composite $i \colon R \rightarrowtail P \rightarrowtail X$ is a path embedding which is below m in the poset $\mathbb{P}\, P$. Further, the top horizontal arrow in the pullback square is a quotient and so, by the uniqueness up to isomorphism of factorisations, the (quotient, embedding) factorisation of $f \circ i$ is $R \twoheadrightarrow Q \overset{n}{\rightarrowtail} Y$, i.e., $\mathbb{P}\, f(i) = n$.

For injectivity, let $m_1 \colon P_1 \rightarrowtail X$ and $m_2 \colon P_2 \rightarrowtail X$ be path embeddings in $\downarrow m$. Since P is a path, m_1 and m_2 are comparable in the order of $\mathbb{P}\, X$. Assume without loss of generality that $m_1 \leq m_2$, i.e., there exists an embedding $k \colon P_1 \rightarrowtail P_2$ such that $m_1 = m_2 \circ k$. If $\mathbb{P}\, f(m_1) = \mathbb{P}\, f(m_2)$, there exists an isomorphism $k' \colon \exists_f P_1 \to \exists_f P_2$ making the left-hand diagram below commute.



In particular, the diagram on the right-hand side above commutes, where the top horizontal arrow is the composition of $P_1 \twoheadrightarrow \exists_f P_1$ with the isomorphism $k' \colon \exists_f P_1 \to \exists_f P_2$. By the 2-out-of-3 condition, k is an isomorphism, and so $m_1 = m_2$ in $\mathbb{P}\, X$. ◀

We conclude this section with the following useful observation:

▶ **Lemma 17.** *The following statements hold for any object X of a path category $\mathcal{C}$:*

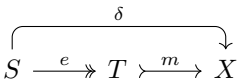
(a) *Any subset $\mathcal{U} \subseteq \mathbb{P}\, X$ admits a supremum $\bigvee \mathcal{U}$ in $\mathbb{S}\, X$.*

(b) *For any path embedding $m \in \mathbb{P}\, X$ and non-empty set $\mathcal{S} \subseteq \mathbb{S}\, X$, if $m = \bigvee \mathcal{S}$ then $m \in \mathcal{S}$.*

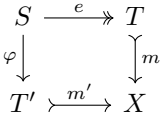
Proof. For item (a), consider a set of path embeddings

$$\mathcal{U} = \{m_i \colon P_i \rightarrowtail X \mid i \in I\} \subseteq \mathbb{P}\, X.$$

Let $S \coloneqq \coprod_{i \in I} P_i$ be the coproduct in $\mathcal{C}$ of the paths P_i and consider the (quotient, embedding) factorisation of the canonical morphism $\delta \colon S \to X$ whose component at P_i is m_i:



Each path embedding $m_i \in \mathcal{U}$ factors through m, thus m is an upper bound for $\mathcal{U}$. We claim that m is the least upper bound, i.e., $m = \bigvee \mathcal{U}$ in $\mathbb{S}\, X$. Suppose that all path embeddings in $\mathcal{U}$ factor through some embedding $m' \colon T' \rightarrowtail X$. By the universal property of S, we get a morphism $\varphi \colon S \to T'$. Further, using again the universal property of S, it is not difficult to see that $m' \circ \varphi$ coincides with δ, and so the following square commutes.



Therefore, there exists a diagonal filler $T \to T'$. In particular, the commutativity of the lower triangle entails that $m \leq m'$, as was to be proved.

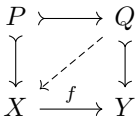
For item (b), let $m\colon P \rightarrowtail X$ be a path embedding and $\mathcal{S} \subseteq \mathbb{S}\,X$ a non-empty set such that $m = \bigvee \mathcal{S}$. Then $n \leq m$ for each $n \in \mathcal{S}$. Since P is a path, $\downarrow m$ is a finite chain in $\mathbb{S}\,X$ and so $\mathcal{S}$ must be a finite set whose largest element coincides with m. In particular, $m \in \mathcal{S}$. ◀

4 Pathwise Embeddings, Open Maps, and Bisimulations

Throughout this section, we fix a category $\mathcal{C}$ equipped with a stable proper factorisation system.

Pathwise embeddings and open maps

Following [6], let us say that a morphism $f\colon X \to Y$ in $\mathcal{C}$ is a *pathwise embedding* if, for all path embeddings $m\colon P \rightarrowtail X$, the composite $f \circ m$ is a path embedding. Hence, $\mathbb{P}\,f(m) = f \circ m$ for all $m \in \mathbb{P}\,X$. Following again [6], we introduce a notion of open map – inspired by [13] – that, combined with the concept of pathwise embedding, will allow us to define an appropriate notion of bisimulation. A morphism $f\colon X \to Y$ in $\mathcal{C}$ is said to be *open* if it satisfies the following path-lifting property: Given any commutative square



with P, Q paths, there exists a diagonal filler $Q \to X$ (i.e., an arrow $Q \to X$ making the two triangles commute). Note that, if it exists, such a diagonal filler must be an embedding.

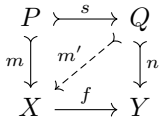
▶ Remark 18. The previous definition of open map differs from the one given in [13] because we require that, in the square above, the top horizontal morphism and the vertical ones be embeddings. However, a pathwise embedding is open in $\mathcal{C}$ (according to the definition above) if, and only if, it is open (in the sense of [13]) in the subcategory $\mathcal{C}_*$ of $\mathcal{C}$ having the same objects as $\mathcal{C}$ and morphisms the pathwise embeddings.

For pathwise embeddings $f\colon X \to Y$, openness can be characterised in terms of the corresponding monotone map $\mathbb{P}\,f\colon \mathbb{P}\,X \to \mathbb{P}\,Y$:

▶ **Proposition 19.** *The following are equivalent for any pathwise embedding $f\colon X \to Y$:*

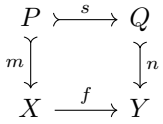
1. *f is open.*
2. *$\mathbb{P}\,f$ is a p-morphism, i.e. $\mathbb{P}\,f(\uparrow m) = \uparrow \mathbb{P}\,f(m)$ for all $m \in \mathbb{P}\,X$.*

Proof. (1) ⇒ (2). Suppose f is open, and let $m\colon P \rightarrowtail X$ be an arbitrary element of $\mathbb{P}\,X$. The inclusion $\mathbb{P}\,f(\uparrow m) \subseteq \uparrow \mathbb{P}\,f(m)$ follows at once from monotonicity of $\mathbb{P}\,f$. For the converse inclusion, assume that $n\colon Q \rightarrowtail Y$ is an element of $\mathbb{P}\,Y$ above $\mathbb{P}\,f(m) = f \circ m$. Then, the composite $f \circ m$ must factor through n, say $f \circ m = n \circ s$ for some embedding $s\colon P \rightarrowtail Q$. Hence, we have a commutative square as displayed below.

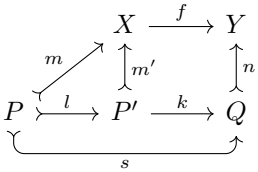


Since f is open, there exists a diagonal filler $m' \colon Q \rightarrowtail X$. The commutativity of the upper triangle entails that $m' \in \uparrow m$, while the commutativity of the lower triangle implies that $\mathbb{P}\, f(m') = n$. Therefore, $\uparrow \mathbb{P}\, f(m) \subseteq \mathbb{P}\, f(\uparrow m)$.

(2) $\Rightarrow$ (1). Assume $\mathbb{P}\, f$ is a p-morphism, and consider a commutative square as follows,



where P and Q are paths. We have $\mathbb{P}\, f(m) \leq n$ in $\mathbb{P}\, Y$, and thus there exists a path embedding $m' \colon P' \rightarrowtail X$ satisfying $m \leq m'$ and $\mathbb{P}\, f(m') = n$. The inequality $m \leq m'$ amounts to saying that $m = m' \circ l$ for some embedding $l \colon P \rightarrowtail P'$, while the equality $\mathbb{P}\, f(m') = n$ means that $f \circ m' = n \circ k$ for some isomorphism $k \colon P' \to Q$. We have a commutative diagram as displayed below.



We claim that $m' \circ k^{-1} \colon Q \to X$ satisfies $m' \circ k^{-1} \circ s = m$ and $f \circ m' \circ k^{-1} = n$, thus showing that f is open. To start with, note that $k \circ l = s$. Just observe that

$$n \circ k \circ l = f \circ m' \circ l = f \circ m = n \circ s,$$

and so $k \circ l = s$ because n is a monomorphism. Now, by diagram chasing we see that

$$m' \circ k^{-1} \circ s = m' \circ k^{-1} \circ k \circ l = m' \circ l = m$$

and $f \circ m' \circ k^{-1} = n \circ k \circ k^{-1} = n$. This concludes the proof. ◀

Bisimulations

A *bisimulation* between objects X, Y of $\mathcal{C}$ is a span of open pathwise embeddings $X \leftarrow Z \rightarrow Y$ in $\mathcal{C}$. If such a bisimulation exists, we say that X and Y are *bisimilar*.

▶ **Example 20.** This definition directly generalizes that in [6], and the notions of bisimulation given there for the Ehrenfeucht-Fraïssé, pebbling and modal comonads are the special cases arising in the categories $\mathcal{R}_k^E(\sigma)$, $\mathcal{R}_k^P(\sigma)$ and $\mathcal{R}_k^M(\sigma)$ respectively, as described in Example 9.

▶ Remark 21. Let $\mathcal{C}$ be a path category. If we regard trees as Kripke models where the accessibility relation is the tree order, then it follows from Theorem 14 and Proposition 19 that a span of pathwise embeddings $X \xleftarrow{f} Z \xrightarrow{g} Y$ in $\mathcal{C}$ is a bisimulation if, and only if, $\mathbb{P}\, X \xleftarrow{\mathbb{P}\, f} \mathbb{P}\, Z \xrightarrow{\mathbb{P}\, g} \mathbb{P}\, Y$ is a bisimulation of Kripke models in the usual sense.

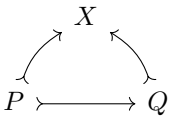
Given a bisimulation $X \leftarrow Z \rightarrow Y$, we would like to think of Z as providing a winning strategy for Duplicator in an appropriate game played "between X and Y". To substantiate this idea, in the next two sections we introduce *arboreal categories* – a refinement of the concept of path category – and show that, in these categories, bisimilarity is captured by back-and forth systems which model the dynamic nature of games.

5 Arboreal Categories

By Theorem 14, any path category $\mathcal{C}$ admits a functor $\mathbb{P}\colon \mathcal{C} \to \mathcal{T}$ into the category of trees. In general, the tree $\mathbb{P}\,X$ may retain little information about X. We are interested in the case where X is determined by $\mathbb{P}\,X$. This leads us to the notion of path-generated object.

Path-generated objects

Let $\mathcal{C}$ be a path category. For any object X of $\mathcal{C}$, we have a diagram with vertex X consisting of all path embeddings with codomain X:



The morphisms between paths are those which make the obvious triangles commute. Choosing representatives in an appropriate way, this can be seen as a cocone over the small diagram $\mathbb{P}\,X$. We say that X is *path-generated* provided this is a colimit cocone in $\mathcal{C}$. Intuitively, an object is path-generated if it is the colimit of its paths.

Let $\mathcal{C}_p$ be the full subcategory of $\mathcal{C}$ defined by the paths and recall that a functor $J\colon \mathcal{A} \to \mathcal{B}$ is *dense* if every $b \in \mathcal{B}$ is the colimit of the diagram $J{\downarrow}b \xrightarrow{\pi} \mathcal{A} \xrightarrow{J} \mathcal{B}$, where $J{\downarrow}b$ is the comma category and π is the natural forgetful functor.

▶ **Lemma 22.** *The following statements are equivalent for any path category $\mathcal{C}$:*
1. *Every object of $\mathcal{C}$ is path-generated.*
2. *The inclusion $\mathcal{C}_p \hookrightarrow \mathcal{C}$ is dense.*

Arboreal categories

We now state the axioms for an arboreal category. To this end, let us say that an object X of a path category $\mathcal{C}$ is *connected* if, for all small families of paths $\{P_i \mid i \in I\}$ in $\mathcal{C}$, any morphism $X \to \coprod_{i\in I} P_i$ factors through some coproduct injection $P_j \to \coprod_{i\in I} P_i$.

▶ **Definition 23.** *An* arboreal category *is a path category $\mathcal{C}$ satisfying the following conditions:*
- **(i)** *every object of $\mathcal{C}$ is path-generated;*
- **(ii)** *every path in $\mathcal{C}$ is connected.*

▶ **Example 24.** The category $\mathcal{T}$ of trees is arboreal; this is essentially the observation that (i) every tree is the colimit of the diagram given by its branches and the embeddings between them, and (ii) finite chains are connected in $\mathcal{T}$. Similarly, $\mathcal{F}$ is arboreal. Our key examples of the categories $\mathcal{R}_k^E(\sigma)$, $\mathcal{R}_k^P(\sigma)$ and $\mathcal{R}_k^M(\sigma)$ from Example 9 are also arboreal.

Note that, in view of Theorem 14, any arboreal category $\mathcal{C}$ admits a functor $\mathbb{P}\colon \mathcal{C} \to \mathcal{T}$ into the category of trees. This crucial fact is what will allow us, given an arboreal cover (cf. Section 7), to regard process structures as tree-like objects.

We collect some useful consequences of the axioms above.

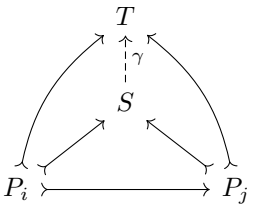
▶ **Lemma 25.** *Let $\mathcal{C}$ be an arboreal category. The following statements hold:*
(a) *Between any two paths there is at most one embedding.*
(b) *For any object X of $\mathcal{C}$ and any $m \in \mathbb{S}\,X$, $m = \bigvee \{p \in \mathbb{P}\,X \mid p \leq m\}$.*
(c) *If f is a quotient in $\mathcal{C}$, then $\mathbb{P}\,f$ is a surjection.*

Proof. For item (a), since there is at most on tree morphism between any two finite chains, it suffices to show that $\mathbb{P}\colon \mathcal{C} \to \mathcal{T}$ is faithful on embeddings between paths. That is, whenever $f, g\colon P \rightrightarrows Q$ are embeddings between paths, $\mathbb{P}\,f = \mathbb{P}\,g$ implies $f = g$. We show that, in fact, $\mathbb{P}$ is faithful on all pathwise embeddings in $\mathcal{C}$. Suppose $f, g\colon X \rightrightarrows Y$ are pathwise embeddings. If $\mathbb{P}\,f = \mathbb{P}\,g$ then, for all path embeddings $m\colon P \rightarrowtail X$,

$$f \circ m = \mathbb{P}\,f(m) = \mathbb{P}\,g(m) = g \circ m.$$

As X is path-generated, it follows that $f = g$.

For item (b), let $m\colon S \rightarrowtail X$ be an arbitrary embedding. Clearly, $\bigvee \{p \in \mathbb{P}\,X \mid p \leq m\} \leq m$. For the converse direction, assume that $n\colon T \rightarrowtail X$ is an upper bound for $\{p \in \mathbb{P}\,X \mid p \leq m\}$. This means that each path embedding $P \rightarrowtail X$ that factors through m must factor through n. We then have a commutative diagram as displayed below.



Because S is path-generated, the cocone with vertex S is a colimit cocone. Therefore, there exists a unique mediating arrow $\gamma\colon S \to T$ making the diagram commute. Using the universal property of S, it is not difficult to see that the composite $S \xrightarrow{\gamma} T \overset{n}{\rightarrowtail} X$ coincides with m, and so $m \leq n$.

For item (c), suppose that $f\colon X \twoheadrightarrow Y$ is a quotient in $\mathcal{C}$. We first assume that $Y = P$ is a path, and then settle the general case. To show that $\mathbb{P}\,f$ is a surjection, it suffices to prove that $\mathrm{id}_P \in \mathbb{P}\,f(\mathbb{P}\,X)$, where $\mathrm{id}_P\colon P \to P$ is the identity. We have

$$\begin{aligned} f^*\mathrm{id}_P &= \bigvee \{p \in \mathbb{P}\,X \mid p \leq f^*\mathrm{id}_P\} && \text{by item (b)} \\ &= \bigvee \{p \in \mathbb{P}\,X \mid \exists_f p \leq \mathrm{id}_P\} && \text{since } \exists_f \dashv f^* \\ &= \bigvee \mathbb{P}\,X, \end{aligned}$$

and so (using the fact that left adjoints preserve suprema)

$$\mathrm{id}_P = \exists_f f^*\mathrm{id}_P = \bigvee \mathbb{P}\,f(\mathbb{P}\,X),$$

where the first step follows from the fact that $\exists_f \circ f^*$ is the identity of $\mathbb{S}\,P$ (cf. the proof of Lemma 6(b)). Hence, $\mathrm{id}_P \in \mathbb{P}\,f(\mathbb{P}\,X)$ by Lemma 17(b).

For the general case, let $m\colon P \rightarrowtail Y$ be an arbitrary path embedding and consider the following pullback square in $\mathcal{C}$.

$$\begin{array}{ccc} f^*P & \xrightarrow{g} & P \\ {\scriptstyle f^*m}\downarrow & \lrcorner & \downarrow{\scriptstyle m} \\ X & \xrightarrow{f} & Y \end{array}$$

By the argument above, there is a path embedding $n\colon Q \rightarrowtail f^*P$ such that $\mathbb{P}\,g(n) = \mathrm{id}_P$. It follows easily that $\mathbb{P}\,f(f^*m \circ n) = m$. ◀

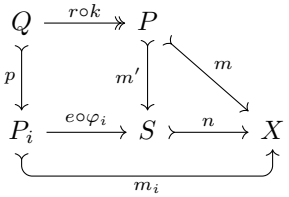
The next proposition will allow us to construct an object from a prescribed set of path embeddings, without adding any new paths in the process.

▶ **Proposition 26.** *Let $\mathcal{C}$ be an arboreal category, X an object of $\mathcal{C}$, and $\mathcal{U} \subseteq \mathbb{P}X$. A path embedding $m \in \mathbb{P}X$ is below $\bigvee \mathcal{U}$ if, and only if, it is below some element of $\mathcal{U}$.*

Proof. Fix an arbitrary object X of $\mathcal{C}$ and a set of path embeddings $\mathcal{U} = \{m_i \colon P_i \rightarrowtail X \mid i \in I\}$. Let $m \colon P \rightarrowtail X$ be an arbitrary path embedding. If m is below some element of $\mathcal{U}$, then clearly $m \leq \bigvee \mathcal{U}$. For the converse direction, suppose that $m \leq \bigvee \mathcal{U}$. Recall from the proof of Lemma 17(a) that the supremum of $\mathcal{U}$ is obtained by taking the (quotient, embedding) factorisation $\coprod_{i \in I} P_i \overset{e}{\twoheadrightarrow} S \overset{n}{\rightarrowtail} X$ of the canonical morphism $\coprod_{i \in I} P_i \to X$. With this notation, $\bigvee \mathcal{U} = n$. Since $m \leq \bigvee \mathcal{U}$, there exists an embedding $m' \colon P \rightarrowtail S$ such that $m = n \circ m'$. Consider the pullback of m' along e:

$$\begin{array}{ccc} T & \xrightarrow{r} & P \\ {\scriptstyle j}\downarrow & \lrcorner & \downarrow{\scriptstyle m'} \\ \coprod_{i \in I} P_i & \xrightarrow{e} & S \end{array}$$

Applying Lemma 25(c) to the quotient r, we see that there exists a path embedding $k \colon Q \rightarrowtail T$ such that $r \circ k$ is a quotient. Because Q is connected, $j \circ k \colon Q \rightarrowtail \coprod_{i \in I} P_i$ must factor through some coproduct injection $\varphi_i \colon P_i \to \coprod_{i \in I} P_i$, i.e., $j \circ k = \varphi_i \circ p$ for some path embedding $p \colon Q \rightarrowtail P$. We then have a commutative diagram as follows.



As $m \circ r \circ k = m_i \circ p$ and the right-hand side of the equation is an embedding, $r \circ k$ is an isomorphism. So $m \leq m_i \in \mathcal{U}$, thus concluding the proof. ◀

6 Back-and-Forth Systems and Games

Throughout this section, we work in a fixed arboreal category $\mathcal{C}$. First, we introduce back-and-forth systems in $\mathcal{C}$ and show that they capture precisely the bisimilarity relation defined in Section 4 in terms of spans of open pathwise embeddings. Then, we show that back-and-forth systems can be equivalently seen as appropriate back-and-forth games.

Back-and-forth systems

Given objects X and Y of $\mathcal{C}$, we consider spans of (equivalence classes of) path embeddings of the form $X \overset{m}{\hookleftarrow} P \overset{n}{\rightarrowtail} Y$. Such a span can be thought of as a partial isomorphism "of shape P" between X and Y. A back-and-forth system between X and Y is a collection of such spans containing an "initial element" and satisfying an appropriate extension property.

Let X, Y be any two objects of $\mathcal{C}$. Given $m \in \mathbb{P}X$ and $n \in \mathbb{P}Y$, we write $[\![m, n]\!]$ to denote that $\mathrm{dom}(m) \cong \mathrm{dom}(n)$. Observe that (i) any two embeddings in the same $\sim$-equivalence class have isomorphic domains, and (ii) given $[\![m, n]\!]$, there exist path embeddings $m' \sim m$ and $n' \sim n$ such that $\mathrm{dom}(m') = \mathrm{dom}(n')$. Hence, the pairs of the form $[\![m, n]\!]$ capture the partial isomorphisms $X \overset{m}{\hookleftarrow} P \overset{n}{\rightarrowtail} Y$ "of shape P".

▶ **Definition 27.** *A* back-and-forth system *between objects X, Y of $\mathcal{C}$ is a set $\mathcal{B} = \{[\![m_i, n_i]\!] \mid m_i \in \mathbb{P}\,X,\ n_i \in \mathbb{P}\,Y,\ i \in I\}$ satisfying the following conditions:*

- **(i)** *$[\![\bot_X, \bot_Y]\!] \in \mathcal{B}$, where $\bot_X$ and $\bot_Y$ are the least elements of $\mathbb{P}\,X$ and $\mathbb{P}\,Y$, respectively;*
- **(ii)** *if $[\![m, n]\!] \in \mathcal{B}$ and $m' \in \mathbb{P}\,X$ are such that $m \prec m'$, there exists $n' \in \mathbb{P}\,Y$ satisfying $n \prec n'$ and $[\![m', n']\!] \in \mathcal{B}$;*
- **(iii)** *if $[\![m, n]\!] \in \mathcal{B}$ and $n' \in \mathbb{P}\,Y$ are such that $n \prec n'$, there exists $m' \in \mathbb{P}\,X$ satisfying $m \prec m'$ and $[\![m', n']\!] \in \mathcal{B}$.*

A back-and-forth system $\mathcal{B}$ is strong *if, for all $[\![m, n]\!] \in \mathcal{B}$ and all $m' \in \mathbb{P}\,X$, $n' \in \mathbb{P}\,Y$, if $m' \prec m$ and $n' \prec n$ then $[\![m', n']\!] \in \mathcal{B}$.*

Two objects X and Y of $\mathcal{C}$ are said to be *(strong) back-and-forth equivalent* if there exists a (strong) back-and-forth system between them.

▶ Remark 28. The definition of (strong) back-and-forth system given above is a variant of the notion of *(strong) path bisimulation* from [12]. The nomenclature adopted here is motivated by the analogy with back-and-forth systems of partial isomorphisms from model theory [15].

The aim of this section is to prove the following result:

▶ **Theorem 29.** *In an arboreal category with binary products, any two objects are bisimilar if, and only if, they are strong back-and-forth equivalent.*

In all our key examples of arboreal categories, binary products exist:

▶ **Example 30.** The category $\mathcal{T}$ has binary products. These are computed as "synchronous products" consisting of the pairs (x, y) of elements having the same height, with the componentwise order. Similarly, $\mathcal{F}$ has binary products, and so does the category $\mathcal{R}(\sigma)$ if the relations in the synchronous product are interpreted componentwise. As synchronous products do not increase the height of forests, we see that the category $\mathcal{R}_k^E(\sigma)$ from Example 9 has binary products. Finally, binary products exist in $\mathcal{R}_k^P(\sigma)$ and $\mathcal{R}_k^M(\sigma)$ and can be described again as variants of synchronous products.

▶ Remark 31. Direct inspection of the relevant proofs shows that Theorem 29 can be slightly generalized to the effect that, for any two objects X and Y of an arboreal category, if the product $X \times Y$ exists, then X and Y are bisimilar precisely when they are strong back-and-forth equivalent.

We start by establishing the easy direction of Theorem 29.

▶ **Proposition 32.** *Any two bisimilar objects of an arboreal category are strong back-and-forth equivalent.*

Proof. Suppose that $X \xleftarrow{f} Z \xrightarrow{g} Y$ is a span of open pathwise embeddings in an arboreal category $\mathcal{C}$. We claim that

$$\mathcal{B} \coloneqq \{[\![\mathbb{P}\,f(m), \mathbb{P}\,g(m)]\!] \mid m \in \mathbb{P}\,Z\}$$

is a strong back-and-forth system between X and Y. We show that items (i), (ii), and (iii) in Definition 27 are satisfied.

For item (i), let $\bot_Z$ be the least element of $\mathbb{P}\,Z$. Then $\mathbb{P}\,f(\bot_Z) = \bot_X$ and $\mathbb{P}\,g(\bot_Z) = \bot_Y$ because tree morphisms preserve roots, and so $[\![\bot_X, \bot_Y]\!] \in \mathcal{B}$. For item (ii), let $[\![\mathbb{P}\,f(m), \mathbb{P}\,g(m)]\!] \in \mathcal{B}$ and $m' \in \mathbb{P}\,X$ be such that $\mathbb{P}\,f(m) \prec m'$. Let us denote $P \coloneqq \mathrm{dom}(m)$ and $P' \coloneqq \mathrm{dom}(m')$. As $f \circ m \leq m'$ in $\mathbb{P}\,X$, there exists $k \colon P \rightarrowtail P'$ such that $f \circ m = m' \circ k$ in $\mathcal{C}$. Therefore, we have a commutative square as follows.

$$\begin{array}{ccc} P & \xrightarrow{k} & P' \\ {\scriptstyle m}\downarrow & & \downarrow{\scriptstyle m'} \\ Z & \xrightarrow{f} & X \end{array}$$

Since f is open, there exists a diagonal filler $n\colon P' \rightarrowtail Z$. Thus, $m' = \mathbb{P}\, f(n)$ and $[\![m', \mathbb{P}\, g(n)]\!] \in \mathcal{B}$. It remains to show that $\mathbb{P}\, g(m) \prec \mathbb{P}\, g(n)$. Note that $m \leq n$ and $\mathbb{P}\, f(m) \prec \mathbb{P}\, f(n)$ entail $m \prec n$, and so $\mathbb{P}\, g(m) \prec \mathbb{P}\, g(n)$ because $\mathbb{P}\, g$ preserves the covering relation.

The proof of item (iii) is the same, mutatis mutandis, as for (ii). Finally, observe that the back-and-forth system $\mathcal{B}$ is strong. To see this, suppose that $[\![\mathbb{P}\, f(m), \mathbb{P}\, g(m)]\!] \in \mathcal{B}$, and let $p_1 \in \mathbb{P}\, X$ and $p_2 \in \mathbb{P}\, Y$ satisfy $p_1 \prec \mathbb{P}\, f(m)$ and $p_2 \prec \mathbb{P}\, g(m)$. As $\mathbb{P}\, f$ is a tree morphism, there exists $m' \in \mathbb{P}\, Z$ such that $m' \prec m$ and $\mathbb{P}\, f(m') = p_1$. But then $\mathbb{P}\, g(m') \prec \mathbb{P}\, g(m)$ implies $\mathbb{P}\, g(m') = p_2$, and therefore $[\![p_1, p_2]\!] \in \mathcal{B}$. ◀

To establish the other direction of Theorem 29, we start by considering a strong back-and-forth system $\mathcal{B} = \{[\![m_i, n_i]\!] \mid i \in I\}$ between X and Y, and attempt to construct an object Z and a span of open pathwise embeddings $X \leftarrow Z \rightarrow Y$. Intuitively, Z is obtained by gluing together the paths $P_i \coloneqq \mathrm{dom}(m_i)$, for $i \in I$, by taking a colimit in $\mathcal{C}$. This colimit can be equivalently described as the supremum of a set of path embeddings as we now explain.

Consider an arbitrary $[\![m_i, n_i]\!] \in \mathcal{B}$ and assume without loss of generality that $\mathrm{dom}(m_i) = P_i = \mathrm{dom}(n_i)$ for some path P_i. Then the product arrow $\langle m_i, n_i \rangle\colon P_i \to X \times Y$ is an embedding. In fact, it suffices that m_i be an embedding (or, symmetrically, that n_i be an embedding), for then $m_i = \pi_X \circ \langle m_i, n_i \rangle$ entails that $\langle m_i, n_i \rangle$ is an embedding, where $\pi_X\colon X \times Y \to X$ is the projection. Therefore, we can identify each $[\![m_i, n_i]\!] \in \mathcal{B}$ with a path embedding $\langle m_i, n_i \rangle \in \mathbb{P}\,(X \times Y)$ and compute the supremum $m\colon Z \rightarrowtail X \times Y$ in $\mathbb{S}\,(X \times Y)$ of all these path embeddings. (It is not difficult to see that the assignment $[\![m_i, n_i]\!] \mapsto \langle m_i, n_i \rangle \in \mathbb{P}\,(X \times Y)$ does not depend on the choice of the representatives in the equivalence classes of m_i and n_i.) We note in passing the following immediate fact:

▶ **Lemma 33.** *Let $\mathcal{B} = \{[\![m_i, n_i]\!] \mid i \in I\}$ be a back-and-forth system between X and Y. If $\mathcal{B}$ is strong, then $\{\langle m_i, n_i \rangle \in \mathbb{P}\,(X \times Y) \mid i \in I\}$ is downwards closed in $\mathbb{S}\,(X \times Y)$.*

To show that the span $X \xleftarrow{\pi_X \circ m} Z \xrightarrow{\pi_Y \circ m} Y$ is a bisimulation, we exploit the fact that Z does not admit more path embeddings than those prescribed (cf. Proposition 26). The following proposition then completes the proof of Theorem 29:

▶ **Proposition 34.** *In an arboreal category with binary products, any two strong back-and-forth equivalent objects are bisimilar.*

Proof. Let $\mathcal{C}$ be an arboreal category with binary products, and let X, Y be any two objects of $\mathcal{C}$. Assume that there is a strong back-and-forth system $\mathcal{B} = \{[\![m_i, n_i]\!] \mid i \in I\}$ between X and Y, and consider the set

$$\mathcal{U} \coloneqq \{\langle m_i, n_i \rangle \in \mathbb{P}\,(X \times Y) \mid i \in I\}.$$

Let $m\colon Z \rightarrowtail X \times Y$ be the supremum of $\mathcal{U}$ in $\mathbb{S}\,(X \times Y)$. We claim that

$$X \xleftarrow{\pi_X \circ m} Z \xrightarrow{\pi_Y \circ m} Y$$

is a bisimulation between X and Y.

To see that this is a span of pathwise embeddings, consider an arbitrary path embedding $n\colon P \rightarrowtail Z$. In view of Proposition 26 and Lemma 33, $m \circ n \in \mathcal{U}$. That is, $m \circ n = \langle m_i, n_i \rangle$ in $\mathbb{P}(X \times Y)$ for some $[\![m_i, n_i]\!] \in \mathcal{B}$. It follows that $m \circ n = \langle m_i, n_i \rangle \circ \varphi$ in $\mathcal{C}$ for some isomorphism φ, and so $\pi_X \circ m \circ n$ and $\pi_Y \circ m \circ n$ are embeddings because $\pi_X \circ m \circ n = m_i \circ \varphi$ and $\pi_Y \circ m \circ n = n_i \circ \varphi$.

It remains to show that $\pi_X \circ m$ and $\pi_Y \circ m$ are open. We prove that $\pi_X \circ m$ is open; the proof for $\pi_Y \circ m$ follows by symmetry. Consider a commutative square in $\mathcal{C}$ as displayed below, where P and Q are paths.

$$\begin{array}{ccc} P & \overset{k}{\rightarrowtail} & Q \\ {\scriptstyle n}\downarrow & & \downarrow{\scriptstyle m_j} \\ Z & \xrightarrow{\pi_X \circ m} & X \end{array}$$

Reasoning as above, we see that $m \circ n = \langle m_i, n_i \rangle$ in $\mathbb{P}(X \times Y)$ for some $[\![m_i, n_i]\!] \in \mathcal{B}$. Therefore, in $\mathbb{P}X$, we have $m_i = \pi_X \circ m \circ n \leq m_j$. Applying item (ii) in Definition 27 (possibly finitely many times), it follows that there exists $n_j \in \mathbb{P}Y$ such that $n_i \leq n_j$ and $[\![m_j, n_j]\!] \in \mathcal{B}$. Suppose without loss of generality that $\mathrm{dom}(m_j) = \mathrm{dom}(n_j)$. Then, $\langle m_j, n_j \rangle \in \mathcal{U}$ and so $\langle m_j, n_j \rangle\colon Q \rightarrowtail X \times Y$ factors through the supremum $m\colon Z \rightarrowtail X \times Y$ of $\mathcal{U}$. That is, $\langle m_j, n_j \rangle = m \circ h$ in $\mathcal{C}$ for some morphism $h\colon Q \to Z$. We claim that the following diagram commutes, thus establishing that $\pi_X \circ m$ is open.

$$\begin{array}{ccc} P & \overset{k}{\rightarrowtail} & Q \\ {\scriptstyle n}\downarrow & \overset{h}{\swarrow} & \downarrow{\scriptstyle m_j} \\ Z & \xrightarrow{\pi_X \circ m} & X \end{array}$$

For the commutativity of the lower triangle, just observe that

$$\pi_X \circ m \circ h = \pi_X \circ \langle m_j, n_j \rangle = m_j.$$

Now, assume without loss of generality that $\mathrm{dom}(m_i) = R = \mathrm{dom}(n_i)$ for some path R. As already observed above, $m \circ n = \langle m_i, n_i \rangle$ in $\mathbb{P}(X \times Y)$ implies that $m \circ n = \langle m_i, n_i \rangle \circ \varphi$ in $\mathcal{C}$ for some isomorphism $\varphi\colon P \to R$. Thus, for the upper triangle, we have

$$\begin{aligned} n = h \circ k &\Leftrightarrow m \circ n = m \circ h \circ k \\ &\Leftrightarrow \langle m_i, n_i \rangle \circ \varphi = \langle m_j, n_j \rangle \circ k \\ &\Leftrightarrow \begin{cases} m_i \circ \varphi = m_j \circ k \\ n_i \circ \varphi = n_j \circ k \end{cases} \end{aligned}$$

where in the first step we used the fact that m is a monomorphism. In turn, the inequalities $m_i \leq m_j$ and $n_i \leq n_j$ entail the existence of embeddings $k_1, k_2\colon R \rightrightarrows Q$ such that $m_i = m_j \circ k_1$ and $n_i = n_j \circ k_2$. By Lemma 25(a) we have $k_1 \circ \varphi = k = k_2 \circ \varphi$. It follows that $m_i \circ \varphi = m_j \circ k$ and $n_i \circ \varphi = n_j \circ k$, and so $n = h \circ k$. ◀

Back-and-forth games

Let $\mathcal{C}$ be an arboreal category and let X, Y be any two objects of $\mathcal{C}$. We define a back-and-forth game $\mathcal{G}(X, Y)$ played by Spoiler and Duplicator on X and Y as follows. Positions in the game are pairs of (equivalence classes of) path embeddings $(m, n) \in \mathbb{P}X \times \mathbb{P}Y$. The winning relation $\mathcal{W}(X, Y) \subseteq \mathbb{P}X \times \mathbb{P}Y$ consists of the pairs (m, n) such that $\mathrm{dom}(m) \cong \mathrm{dom}(n)$.

Let $\bot_X : P \rightarrowtail X$ and $\bot_Y : Q \rightarrowtail Y$ be the roots of $\mathbb{P}\,X$ and $\mathbb{P}\,Y$, respectively. If $P \ncong Q$, then Duplicator loses the game. Otherwise, the initial position is $(\bot_X, \bot_Y)$. At the start of each round, the position is specified by a pair $(m, n) \in \mathbb{P}\,X \times \mathbb{P}\,Y$, and the round proceeds as follows: Either Spoiler chooses some $m' \succ m$ and Duplicator must respond with some $n' \succ n$, or Spoiler chooses some $n'' \succ n$ and Duplicator must respond with $m'' \succ m$. Duplicator wins the round if they are able to respond and the new position is in $\mathcal{W}(X, Y)$. Duplicator wins the game if they have a strategy which is winning after t rounds, for all $t \geq 0$.

▶ Remark 35. It is shown in [6] that the abstract game $\mathcal{G}(X, Y)$ restricts, in the case of the arboreal categories $\mathcal{R}_k^E(\sigma)$, $\mathcal{R}_k^P(\sigma)$ and $\mathcal{R}_k^M(\sigma)$, to the usual k-round Ehrenfeucht-Fraïssé, k-pebble and k-round bisimulation games, respectively.

The following straightforward observation makes precise the translation between strong back-and-forth systems and back-and-forth games.

▶ **Lemma 36.** *Two objects X, Y of an arboreal category $\mathcal{C}$ are strong back-and-forth equivalent if, and only if, Duplicator has a winning strategy in the game $\mathcal{G}(X, Y)$.*

Proof. Clearly, if $\mathcal{B} = \{[\![m_i, n_i]\!] \mid i \in I\}$ is a strong back-and-forth system between X and Y, then the plays in the set

$$\{(m_i, n_i) \mid i \in I\} \subseteq \mathbb{P}\,X \times \mathbb{P}\,Y$$

yield a winning strategy for Duplicator in the game $\mathcal{G}(X, Y)$.

Conversely, a winning strategy for Duplicator in the game $\mathcal{G}(X, Y)$ determines a set $W \subseteq \mathcal{W}(X, Y)$ of the plays following this strategy, and

$$\mathcal{B} \coloneqq \{[\![m, n]\!] \mid (m, n) \in W\}$$

is a back-and-forth system. It is not difficult to see that $\mathcal{B}$ is strong. ◀

The previous lemma, combined with Theorem 29, yields at once the following result:

▶ **Theorem 37.** *Let $\mathcal{C}$ be an arboreal category with binary products. Any two objects X, Y of $\mathcal{C}$ are bisimilar if, and only if, Duplicator has a winning strategy in the game $\mathcal{G}(X, Y)$.*

7 Arboreal Covers

We return to the underlying motivation for the axiomatic development in this paper. Arboreal categories have a rich intrinsic process structure, which allows "dynamic" notions such as bisimulation and back-and-forth games, and resource notions such as the height of a tree, to be defined. A key idea is to relate these process notions to extensional, or "static" structures. In particular, much of finite model theory and descriptive complexity can be seen in this way.

In the general setting, we have an arboreal category $\mathcal{C}$, and another category $\mathcal{E}$, which we think of as the extensional category.

▶ **Definition 38.** *An* arboreal cover *of $\mathcal{E}$ by $\mathcal{C}$ is given by a comonadic adjunction*

$$\mathcal{C} \underset{R}{\overset{L}{\rightleftarrows}} \mathcal{E}\,.$$

As for any adjunction, this induces a comonad on $\mathcal{E}$. The comonad is (G, ε, δ), where $G \coloneqq LR$, ε is the counit of the adjunction, and $\delta_a : LRa \to LRLRa$ is given by $\delta_a \coloneqq L(\eta_{Ra})$, with η the unit of the adjunction. The comonadicity condition states that the Eilenberg-Moore

category of coalgebras for this comonad is isomorphic to $\mathcal{C}$. The idea is then that we can use the arboreal category $\mathcal{C}$, with its rich process structure and all the associated notions, to study the extensional category $\mathcal{E}$ via the adjunction. Both the co-Kleisli category of the comonad, and the full Eilenberg-Moore category, are useful in this regard.

We now bring resources into the picture.

▶ **Definition 39.** *Let $\mathcal{C}$ be an arboreal category, with full subcategory of paths $\mathcal{C}_p$. We say that $\mathcal{C}$ is* resource-indexed *by a resource parameter k if for all $k \geq 0$, there is a full subcategory $\mathcal{C}_p^k$ of $\mathcal{C}_p$ closed under embeddings*[2] *with*

$$\mathcal{C}_p^0 \hookrightarrow \mathcal{C}_p^1 \hookrightarrow \mathcal{C}_p^2 \hookrightarrow \cdots$$

This induces a corresponding tower of full subcategories $\mathcal{C}_k$ of $\mathcal{C}$, with the objects of $\mathcal{C}_k$ those whose cocone of path embeddings with domain in $\mathcal{C}_p^k$ is a colimit cocone in $\mathcal{C}$.

▶ **Example 40.** One resource parameter which is always available is to take $\mathcal{C}_p^k$ to be given by those paths in $\mathcal{C}$ whose chain of subobjects is of length $\leq k$. In the case of $\mathcal{F}$ and $\mathcal{T}$, the corresponding categories $\mathcal{F}_k$ and $\mathcal{T}_k$ are the forests and trees of height $\leq k$. We can think of this as a temporal parameter, restricting the number of sequential steps, or the number of rounds in a game. For the Ehrenfeucht-Fraïssé and modal comonads, we recover $\mathcal{R}_k^E$ and $\mathcal{R}_k^M$ as described in Example 9, corresponding to k-round versions of the Ehrenfeucht-Fraïssé and modal bisimulation games respectively [6]. However, note that for the pebbling comonad, the relevant resource index is the number of pebbles, which is a memory restriction along a computation or play of a game. This leads to $\mathcal{R}_k^P$ as described in Example 9.

In Proposition 42 below we shall see that, given a resource-indexed arboreal category $\{\mathcal{C}_k\}$, each category $\mathcal{C}_k$ is arboreal. This allows us to exploit the ideas developed in this paper for any choice of the resource parameter k. We start by proving the following fact:

▶ **Lemma 41.** *Let $\{\mathcal{C}_k\}$ be a resource-indexed arboreal category and suppose that $X \rightarrowtail Y$ is an embedding in $\mathcal{C}$. For any k, if $Y \in \mathcal{C}_k$ then also $X \in \mathcal{C}_k$.*

Proof. We start by showing that, for any path embedding $P \rightarrowtail Y$ in $\mathcal{C}$, if $Y \in \mathcal{C}_k$ then $P \in \mathcal{C}_p^k$. Consider the set

$$\mathcal{U} \coloneqq \{p \in \mathbb{P}Y \mid \mathrm{dom}(p) \in \mathcal{C}_p^k\}.$$

(Note that $\mathcal{U}$ is well-defined because any two representatives in the equivalence class of p have isomorphic domains, and $\mathcal{C}_p^k$ is closed under isomorphisms.) As Y is the colimit in $\mathcal{C}$ of the subdiagram of $\mathbb{P}Y$ consisting of those path embeddings whose domain is in $\mathcal{C}_p^k$, it follows that $\bigvee \mathcal{U} = \mathrm{id}_Y$ in $\mathbb{S}Y$. If there exists a path embedding $m \colon P \rightarrowtail Y$, then $m \leq \bigvee \mathcal{U}$ in $\mathbb{P}Y$ and so, by Proposition 26, m factors through some $p \in \mathcal{U}$. In particular, there exists an embedding $P \rightarrowtail \mathrm{dom}(p)$. Because $\mathrm{dom}(p) \in \mathcal{C}_p^k$, we see that $P \in \mathcal{C}_p^k$.

Now, suppose that $j \colon X \rightarrowtail Y$ is an embedding in $\mathcal{C}$ and $Y \in \mathcal{C}_k$. Since X is path-generated, it is the colimit in $\mathcal{C}$ of the small diagram $\mathbb{P}X$. We show that, for any path embedding $P \rightarrowtail X$, the path P must belong to $\mathcal{C}_p^k$. It then follows immediately that $X \in \mathcal{C}_k$. Let $m \colon P \rightarrowtail X$ be an arbitrary path embedding. The composite $j \circ m \colon P \rightarrowtail Y$ is also a path embedding, and so $P \in \mathcal{C}_p^k$ by the argument above. ◀

[2] That is, for any embedding $P \rightarrowtail Q$ in $\mathcal{C}$ with P, Q paths, if $Q \in \mathcal{C}_p^k$ then also $P \in \mathcal{C}_p^k$.

▶ **Proposition 42.** *Let $\{\mathcal{C}_k\}$ be a resource-indexed arboreal category. Then $\mathcal{C}_k$ is an arboreal category for each k.*

Proof. If $\mathcal{C}$ is equipped with the stable proper factorisation system $(\mathcal{Q}, \mathcal{M})$, consider the classes of morphisms $\mathcal{Q}' \coloneqq \mathcal{Q} \cap \mathcal{C}_k$ and $\mathcal{M}' \coloneqq \mathcal{M} \cap \mathcal{C}_k$. It is not difficult to see that $(\mathcal{Q}', \mathcal{M}')$ is a proper factorisation system in $\mathcal{C}_k$. Just observe that, whenever $W \to Z$ is a morphism in $\mathcal{C}_k$ and

$$W \twoheadrightarrow X \rightarrowtail Y$$

is its (quotient, embedding) factorisation in $\mathcal{C}$, then $X \in \mathcal{C}_k$ by Lemma 41. Using again Lemma 41, along with the fact that embeddings are stable under pullbacks, it follows at once that $(\mathcal{Q}', \mathcal{M}')$ is stable (and, in fact, pullbacks of $\mathcal{Q}'$-morphisms along $\mathcal{M}'$-morphisms are computed in $\mathcal{C}$). With respect to this factorisation system, it is not difficult to see that the paths in $\mathcal{C}_k$ are precisely the objects of $\mathcal{C}_p^k$.

Moreover, $\mathcal{C}_k$ has all coproducts of small families of paths, and they are computed in $\mathcal{C}$. To see this, consider a set of paths $\{P_i \in \mathcal{C}_p^k \mid i \in I\}$ and let $\coprod_{i\in I} P_i$ be the coproduct in $\mathcal{C}$. If $m\colon P \rightarrowtail \coprod_{i\in I} P_i$ is any path embedding in $\mathcal{C}$ then, because P is connected, m must factor through some coproduct injection. In particular, there exist $i \in I$ and an embedding $P \rightarrowtail P_i$. Since $P_i \in \mathcal{C}_p^k$, we get $P \in \mathcal{C}_p^k$. As $\coprod_{i\in I} P_i$ is path-generated in $\mathcal{C}$, it follows at once that $\coprod_{i\in I} P_i \in \mathcal{C}_k$. Hence, $\coprod_{i\in I} P_i$ coincides with the coproduct of the family $\{P_i \in \mathcal{C}_p^k \mid i \in I\}$ in $\mathcal{C}_k$.

We conclude that $\mathcal{C}_k$ is a path category. Further, every object of $\mathcal{C}_k$ is path-generated by definition, and paths in $\mathcal{C}_k$ are connected because any path in $\mathcal{C}_k$ is also a path in $\mathcal{C}$ and, as observed above, coproducts of paths in $\mathcal{C}_k$ are computed in $\mathcal{C}$. Therefore, $\mathcal{C}_k$ is arboreal. ◀

▶ **Definition 43.** *Let $\{\mathcal{C}_k\}$ be a resource-indexed arboreal category. A* resource-indexed arboreal cover *of $\mathcal{E}$ by $\mathcal{C}$ is an indexed family of comonadic adjunctions*

$$\mathcal{C}_k \underset{R_k}{\overset{L_k}{\rightleftarrows}} \mathcal{E}$$

with corresponding comonads G_k on $\mathcal{E}$.

▶ **Example 44.** Our key examples arise by taking the extensional category $\mathcal{E}$ to be $\mathsf{Struct}(\sigma)$. For each $k \geq 0$, there are evident forgetful functors

$$L_k^E\colon \mathcal{R}_k^E \to \mathsf{Struct}(\sigma), \quad L_k^P\colon \mathcal{R}_k^P \to \mathsf{Struct}(\sigma), \quad L_k^M\colon \mathcal{R}_k^M \to \mathsf{Struct}(\sigma)$$

which forget the forest order, and in the case of $\mathcal{R}_k^P$, also the pebbling function. These functors are all comonadic over $\mathsf{Struct}(\sigma)$. (To be precise, in the modal logic case the category $\mathcal{E}$ consists of *pointed* structures, cf. [6].) The right adjoints build a forest over a structure $\mathcal{A}$ by forming sequences of elements over the universe A, suitably labelled and with the σ-relations interpreted so as to satisfy the conditions (E), (P) and (M) respectively. This yields the comonads described concretely in [1, 6]. The sequences correspond to plays in the Ehrenfeucht-Fraïssé, pebbling and modal bisimulation games respectively.

We now show how resource-indexed arboreal covers can be used to define important notions on the extensional category. For a resource-indexed arboreal cover of $\mathcal{E}$ by $\mathcal{C}$, with adjunctions L_k, R_k and comonads G_k, we define three resource-indexed equivalence relations on objects of $\mathcal{E}$. The first two use the co-Kleisli category of G_k, while the third uses the Eilenberg-Moore category.

▶ **Definition 45.** *Consider a resource-indexed arboreal cover of $\mathcal{E}$ by $\mathcal{C}$, and objects $a, b \in \mathcal{E}$.*
1. *We define $a \rightleftarrows_k^{\mathcal{C}} b$ iff there are co-Kleisli morphisms $G_k a \to b$ and $G_k b \to a$.*
2. *We define $a \cong_k^{\mathcal{C}} b$ iff a and b are isomorphic in the co-Kleisli category of G_k.*
3. *We define $a \leftrightarrow_k^{\mathcal{C}} b$ iff there is a bisimulation between $R_k a$ and $R_k b$ in $\mathcal{C}_k$.*

▶ **Proposition 46.** *Assume $\mathcal{E}$ has binary products. For objects a and b of $\mathcal{E}$, $a \leftrightarrow_k^{\mathcal{C}} b$ iff $R_k a$ and $R_k b$ are strong back-and-forth equivalent, iff Duplicator has a winning strategy in the game $\mathcal{G}(R_k a, R_k b)$.*

Proof. This follows directly from Theorems 29 and 37, and Proposition 42 (cf. also Remark 31). Just observe that, since right adjoints preserve limits, the product of $R_k a$ and $R_k b$ in $\mathcal{C}_k$ exists and can be identified with $R_k(a \times b)$. ◀

What do these notions mean in Example 44? For each of our three types of model comparison game, there are corresponding fragments $\mathcal{L}_k$ of first-order logic [15, 8]:
- For Ehrenfeucht-Fraïssé games, $\mathcal{L}_k$ is the fragment of quantifier-rank $\leq k$.
- For pebble games, $\mathcal{L}_k$ is the k-variable fragment.
- For bisimulation games, $\mathcal{L}_k$ is the modal fragment with modal depth $\leq k$.

In each case, we write $\exists\mathcal{L}_k$ for the existential positive fragment of $\mathcal{L}_k$, and $\mathcal{L}_k(\#)$ for the extension of $\mathcal{L}_k$ with counting quantifiers [15]. For each logic $\mathcal{L}$, there is the usual equivalence on σ-structures: $\mathcal{A} \equiv^{\mathcal{L}} \mathcal{B}$ iff for all φ in $\mathcal{L}$, $\mathcal{A} \models \varphi \iff \mathcal{B} \models \varphi$.

We now have the following result from [6]:

▶ **Theorem 47.** *For σ-structures $\mathcal{A}$ and $\mathcal{B}$:*
1. $\mathcal{A} \equiv^{\exists\mathcal{L}_k} \mathcal{B} \iff \mathcal{A} \rightleftarrows_k^{\mathcal{C}} \mathcal{B}$.
2. $\mathcal{A} \equiv^{\mathcal{L}_k} \mathcal{B} \iff \mathcal{A} \leftrightarrow_k^{\mathcal{C}} \mathcal{B}$.
3. $\mathcal{A} \equiv^{\mathcal{L}_k(\#)} \mathcal{B} \iff \mathcal{A} \cong_k^{\mathcal{C}} \mathcal{B}$.

Note that this is really a family of three theorems, one for each type of game arising from a resource-indexed arboreal cover $\mathcal{C}$ as in Example 44. Thus in each case, we capture the salient logical equivalences in syntax-free, categorical form.

We return to the general setting. Given a resource-indexed arboreal cover of $\mathcal{E}$ by $\mathcal{C}$, we know by comonadicity that for each k, $\mathcal{C}_k$ is isomorphic to the Eilenberg-Moore category for the comonad G_k. For each $a \in \mathcal{E}$, we can ask if it carries a G_k-coalgebra structure; that is, whether there is a morphism $\alpha\colon a \to G_k a$ satisfying the G_k coalgebra conditions. Moreover, we can ask for the *least* k such that this is the case. We call this the *coalgebra number* of a.

The intuition behind this, as explained in [1, 6], is that the resource parameter is bounding access to the structure, making it more difficult to have a morphism in $\mathcal{E}$ with codomain $G_k a$. So the least k for which this *is* possible is a significant invariant of the structure. This intuition is born out by the following result from [1, 6].

▶ **Theorem 48.**
1. *For the Ehrenfeucht-Fraïssé comonad, the coalgebra number of $\mathcal{A}$ corresponds precisely to the* tree-depth *of $\mathcal{A}$.*
2. *For the pebbling comonad, the coalgebra number of $\mathcal{A}$ corresponds precisely to the* tree-width *of $\mathcal{A}$.*
3. *For the modal comonad, the coalgebra number of $\mathcal{A}$ corresponds precisely to the* modal unfolding depth *of $\mathcal{A}$.*

What underlies these results is the comonadicity of the arboreal covers, which means that the coalgebras are witnesses for the various forms of tree decompositions of structures in $\mathcal{E}$ corresponding to these combinatorial invariants.

As a further illustration of the use of our axiomatic setting, we note that it is possible to give an account of the key notion of *extendability*, used by Rossman in his seminal results on homomorphism preservation [19], at this level of generality. For more details, see [4].

References

1 S. Abramsky, A. Dawar, and P. Wang. The pebbling comonad in finite model theory. In *32nd Annual ACM/IEEE Symposium on Logic in Computer Science (LICS)*, pages 1–12, 2017.

2 S. Abramsky, R. Jagadeesan, and P. Malacaria. Full abstraction for PCF. *Information and computation*, 163(2):409–470, 2000.

3 S. Abramsky and D. Marsden. Comonadic semantics for guarded fragments. Preprint available at `arXiv:2008.11094`, 2020.

4 S. Abramsky and L. Reggio. Arboreal categories: An axiomatic theory of resources. Extended version. Preprint available at `arXiv:2102.08109`, 2021.

5 S. Abramsky and N. Shah. Relating structure and power: Comonadic semantics for computational resources. In *27th EACSL Annual Conference on Computer Science Logic, CSL, September 4-7, 2018, Birmingham, UK*, pages 2:1–2:17, 2018.

6 S. Abramsky and N. Shah. Relating structure and power: Comonadic semantics for computational resources. Extended version to appear in Journal of Logic and Computation. Preprint available at `arXiv:2010.06496`, 2021.

7 J. Adámek, H. Herrlich, and G.E. Strecker. *Abstract and concrete categories. The joy of cats.* Online edition, 2004.

8 P. Blackburn, M. De Rijke, and Y. Venema. *Modal Logic*, volume 53 of *Cambridge Tracts in Theoretical Computer Science*. Cambridge University Press, 2002.

9 A. Ó Conghaile and A. Dawar. Game comonads & generalised quantifiers. In *29th EACSL Annual Conference on Computer Science Logic, CSL*, volume 183 of *LIPIcs*, pages 16:1–16:17. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021.

10 P.J. Freyd and G.M. Kelly. Categories of continuous functors, I. *Journal of Pure and Applied Algebra*, 2(3):169–191, 1972.

11 J.M.E. Hyland and C.-H.L Ong. On full abstraction for PCF: I, II, and III. *Information and computation*, 163(2):285–408, 2000.

12 A. Joyal, M. Nielsen, and G. Winskel. Bisimulation from open maps. *Information and Computation*, 127(2):164–185, 1996.

13 A. Joyal, M. Nielson, and G. Winskel. Bisimulation and open maps. In *Proceedings of 8th Annual IEEE Symposium on Logic in Computer Science*, pages 418–427, 1993.

14 T. Kloks. *Treewidth: computations and approximations*, volume 842 of *Springer Lecture Notes in Computer Science*. Springer Science & Business Media, 1994.

15 L. Libkin. *Elements of finite model theory.* Texts in Theoretical Computer Science. An EATCS Series. Springer-Verlag, Berlin, 2004.

16 S. Mac Lane. *Categories for the working mathematician*, volume 5 of *Graduate Texts in Mathematics*. Springer-Verlag, New York, second edition, 1998.

17 J. Nešetřil and P. Ossona de Mendez. Tree-depth, subgraph coloring and homomorphism bounds. *European Journal of Combinatorics*, 27(6):1022–1041, 2006.

18 E. Riehl. Factorization systems. Notes available at `http://www.math.jhu.edu/~eriehl/factorization.pdf`.

19 B. Rossman. Homomorphism preservation theorems. *J. ACM*, 55(3):15:1–15:53, 2008.

20 ACM SIGLOG. Alonzo Church Award, 2017. URL: `https://siglog.org/winners-of-the-2017-alonzo-church-award/`.

Dynamic Membership for Regular Languages

Antoine Amarilli
LTCI, Télécom Paris, Institut Polytechnique de Paris, France

Louis Jachiet
LTCI, Télécom Paris, Institut Polytechnique de Paris, France

Charles Paperman
Univ. Lille, CNRS, INRIA, Centrale Lille, UMR 9189 CRIStAL, F-59000 Lille, France

Abstract

We study the *dynamic membership problem* for regular languages: fix a language L, read a word w, build in time $O(|w|)$ a data structure indicating if w is in L, and maintain this structure efficiently under letter substitutions on w. We consider this problem on the unit cost RAM model with logarithmic word length, where the problem always has a solution in $O(\log|w| / \log\log|w|)$ per operation.

We show that the problem is in $O(\log\log|w|)$ for languages in an algebraically-defined, decidable class **QSG**, and that it is in $O(1)$ for another such class **QLZG**. We show that languages not in **QSG** admit a reduction from the prefix problem for a cyclic group, so that they require $\Omega(\log|w| / \log\log|w|)$ operations in the worst case; and that **QSG** languages not in **QLZG** admit a reduction from the prefix problem for the multiplicative monoid $U_1 = \{0, 1\}$, which we conjecture cannot be maintained in $O(1)$. This yields a conditional trichotomy. We also investigate intermediate cases between $O(1)$ and $O(\log\log|w|)$.

Our results are shown via the dynamic word problem for monoids and semigroups, for which we also give a classification. We thus solve open problems of the paper of Skovbjerg Frandsen, Miltersen, and Skyum [29] on the dynamic word problem, and additionally cover regular languages.

2012 ACM Subject Classification Theory of computation → Formal languages and automata theory

Keywords and phrases regular language, membership, RAM model, updates, dynamic

Digital Object Identifier 10.4230/LIPIcs.ICALP.2021.116

Category Track B: Automata, Logic, Semantics, and Theory of Programming

Related Version *Full Version*: `https://arxiv.org/abs/2102.07728`

Funding The authors have been partially supported by the ANR project EQUUS ANR-19-CE48-0019. Funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) – 431183758.

Acknowledgements We thank Jean-Éric Pin and Jorge Almeida for their advice on **ZG** and **SG**, and thank the ICALP referees for their helpful feedback.

1 Introduction

This paper studies how to handle letter substitution updates on a word while maintaining the information of whether the word belongs to a regular language. Specifically, we fix a regular language L – for instance $L = a^*b^*$. We are then given an input word w, e.g., $w = aaaa$. We first preprocess w in linear time to build a data structure, which we can use in particular to test if $w \in L$. Now, w is edited by letter substitutions, and we want to update w and keep track at each step of whether $w \in L$. For instance, an update can replace the third letter of w by a b, so that $w = aaba$, which is no longer in L. Then another update can replace, e.g., the fourth letter of w by a b, so that $w = aabb$, and now we have $w \in L$ again. Our problem, called *dynamic membership*, is to devise a data structure to handle such update

operations and determine whether $w \in L$, as efficiently as possible. We study this task from a theoretical angle, but it can also be useful in practice to maintain a Boolean condition (expressed as a regular language) on a user-edited word.

Dynamic membership was studied for various update operations, e.g., append operations for streaming algorithms or the sliding window model [14, 13, 15], letter substitutions for the dynamic word problem for monoids [29], or concatenations and splits [22]. It was also studied in the case of *pattern matching*, where we check if the word contains some target pattern [9], which is also assumed to be editable. It is also connected to the incremental validation problem, which has been studied for strings and for XML documents [5]. The problem was also studied from the angle of *dynamic complexity*, which does not restrict the running time but the logical language used to handle updates [16]; and very recently refined to a study of the amount of *parallel work* required [28].

Our focus in this work is to identify classes of fixed regular languages for which dynamic membership can be solved extremely efficiently, e.g., in constant time or sublogarithmic time. Our update language only allows letter substitutions to the input word, in particular the length of the input word can never be changed by updates. We make this choice because insertions and deletions already make it challenging to efficiently maintain the word itself (see Section 7). We work within the computational model of the unit-cost RAM, with logarithmic cell size.

Dynamic word problem for monoids [29]. Our problem closely relates to the work by Skovbjerg Frandsen, Miltersen, and Skyum on the *dynamic word problem for monoids* [29]: fix a finite monoid, read a *word* which is a sequence of monoid elements, and maintain under substitution updates the composition of these elements according to the monoid's internal law. Indeed, the dynamic membership problem for a language L reduces to the dynamic word problem for any monoid that recognizes L; but the converse is not true. Hence, studying the dynamic word problem for monoids is coarser than studying the dynamic membership problem for languages, although it is a natural first step and is already very challenging.

In the context of monoids, Skovbjerg Frandsen et al. [29] propose a general algorithm for the dynamic word problem that can handle each operation in time $O(\log n / \log\log n)$, for n the length of the word. This is a refinement of the elementary $O(\log n)$ algorithm that decomposes the word as a balanced binary tree whose nodes are annotated with the monoid image of the corresponding infix. They show that the $O(\log n / \log\log n)$ bound is tight for some monoids, namely noncommutative groups, and a generalization of them defined via an equation. This is obtained by a reduction from the so-called *prefix-$\mathbb{Z}_d$ problem*, for which an $\Omega(\log n / \log\log n)$ lower bound [12] is known in the cell probe model [17]. We will reuse this lower bound in our work.

They also show that the problem is easier for some monoids. For instance, commutative monoids can be maintained in $O(1)$, simply by maintaining the number of element occurrences. They also show a trickier $O(\log\log n)$ upper bound for *group-free monoids*: this is based on a so-called Krohn-Rhodes decomposition [27] and uses a predecessor data structure implemented as a van Emde Boas tree [34]. However, there are non-commutative monoids for which the problem is in $O(1)$ (as we will show), and there is still a gap between group-free monoids (with an upper bound in $O(\log\log n)$) and the monoids for which the $\Omega(\log n / \log\log n)$ lower bound applies. This was claimed as open in [29] and not addressed afterwards. While there is a more recent study by Pătraşcu and Tarniţă [23], it focuses on single-bit memory cells.

Our contributions. In this paper, we attack these problems using algebraic monoid theory. This unlocks new results: first on the dynamic word problem for monoids, where we extend the results of [29], and then on the dynamic membership problem for regular languages.

We start with our results on the dynamic word problem for monoids, which are summarized in Figure 1 along with a table of the main classes in Table 1. First, in Section 3, we show how a more elaborate $O(\log \log n)$ algorithm can cover all monoids to which the $\Omega(\log n / \log \log n)$ lower bound of [29] does not apply: we dub this class **SG** and characterize it by the equation $x^{\omega+1}yx^{\omega} = x^{\omega}yx^{\omega+1}$, for any elements x and y, where ω denotes the idempotent power. Our algorithm shares some ideas with the $O(\log \log n)$ algorithm of [29], in particular it uses van Emde Boas trees, but it faces significant new challenges. For instance, we can no longer use a Krohn-Rhodes decomposition, and proceed instead by a rather technical induction on the $\mathcal{J}$-classes of the monoid. Thus, we have an unconditional dichotomy between monoids in **SG**, which are in $O(\log \log n)$, and monoids outside of **SG**, which are in $\Theta(\log n / \log \log n)$.

Second, in Section 4, we generalize the $O(1)$ result on commutative monoids to the monoid class **ZG** [4]. This class is defined via the equation $x^{\omega+1}y = yx^{\omega+1}$, i.e., only the elements that are part of a group are required to commute with all other elements. We show that the dynamic word problem for these monoids is in $O(1)$: we use an algebraic characterization to reduce them to commutative monoids and to monoids obtained from nilpotent semigroups, for which we design a simple but somewhat surprising algorithm. We also show a conditional lower bound: for any monoid M not in **ZG**, we can reduce the *prefix-U_1* problem to the dynamic word problem for M. This is the problem of maintaining a binary word under letter substitution updates while answering queries asking if a prefix contains a 0. It can be seen as a priority queue (slightly weakened), so we conjecture that no $O(1)$ data structure for this problem exists in the RAM model. If this conjecture holds, **ZG** is exactly the class of monoids having a dynamic word problem in $O(1)$.

We then extend our results in Section 5 from monoids to the dynamic word problem for semigroups. Our results for **SG** extend directly: the upper bound on **SG** also applies to semigroups in **SG**, and semigroups not in **SG** must contain a submonoid not in **SG** so covered by the lower bound. For **ZG**, there are major complications, and we must study the class **LZG** of semigroups where all submonoids are in **ZG**. Semigroups not in **LZG** are covered by our conditional lower bound on prefix-U_1, but it is very tricky to show the converse, i.e., that imposing the condition on **LZG** suffices to ensure tractability. We do so by showing tractability for $\mathbf{ZG} * \mathbf{D}$, the semigroups generated by *semidirect products* of **ZG** semigroups and so-called *definite semigroups*, and by showing in [3] that $\mathbf{ZG} * \mathbf{D} = \mathbf{LZG}$, a *locality result* of possible independent interest.

Next, we extend our results in Section 6 from semigroups to languages. This is done using the notion of *stable semigroup* [6, 7], denoted as the **Q** operator; and specifically the class **QSG** of regular languages where the stable semigroup of the syntactic morphism in is **SG**, and the class **QLZG** where all local monoids of the stable semigroup of the syntactic morphism are in **ZG**. We obtain:

▶ **Theorem 1.1.** *Let L be a regular language, and consider the dynamic membership problem for L on the unit-cost RAM with logarithmic word length under letter substitution updates:*

- *If L is in the class* **QLZG***, then the problem is in $O(1)$.*
- *If L is not in the class* **QLZG** *but is in the class* **QSG***, then the dynamic membership problem is in $O(\log \log n)$ with n the length of the word. Further, solving it in $O(1)$ time gives an $O(1)$ implementation of a structure for the prefix-U_1 problem.*
- *If L is not in the class* **QSG***, then the dynamic membership problem is in $\Theta(\log n / \log \log n)$.*

We last present in Section 7 some extensions and questions for future work: preliminary observations on the precise complexity of languages in **QSG** $\setminus$ **QLZG** (as the $O(\log \log n)$ bound is not shown to be tight), the complexity of deciding which case of the theorem applies, the support for insertion and deletion updates, and the support for infix queries. Because of space constraints, the complete proofs of results are deferred to the full version [2].

2 Preliminaries and Problem Statement

Computation model. We work in the RAM model with unit cost, i.e., each cell can store integers of value at most polynomial in $O(|w|)$ where $|w|$ is the length of the input, and arithmetic operations (addition, successor, modulo, etc.) on two cells take unit time. As the integers have at most polynomial value, the memory usage is also polynomially bounded.

We consider *dynamic problems* where we are given an input word, preprocess it in linear time to build a data structure, and must then handle *update operations* on the word (by reflecting them in the data structure), and *query operations* on the current state of the word (using the data structure). The *complexity* of the problem is the worst-case complexity of handling an update or answering a query.

Like in [29], the lower bounds that we show actually hold in the coarser *cell probe* model, which only considers the number of memory cells accessed during a computation. Furthermore, they hold even without the assumption that the preprocessing is linear.

Given two dynamic problems A and B, we say that A has a *(constant-time) reduction* to B if we can implement a data structure for problem A having constant-time complexity when using as oracle constantly many data structures for problem B (built during the preprocessing). In other words, queries and updates on the structure for A can perform constant-time computations using its own memory, but they can also use the data structures for B as an oracle, i.e., perform a constant number of queries and updates on them, which are considered to run in $O(1)$. We similarly talk of a dynamic problem having a *(constant-time) reduction* to multiple problems, meaning we can use all of them as oracle. If problem A reduces to problems $B_1, \ldots, B_n$, then any complexity upper bound that holds on all problems $B_1, \ldots, B_n$ also holds for A, and any complexity lower bound on A extends to at least one of the B_i.

Problem statement. Our problems require some algebraic prerequisites. We refer the reader to the book of Pin [25] and his lecture notes [26] for more details. A *semigroup* is a set S equipped with an associative composition law (written multiplicatively), and a *monoid* is a semigroup M with a *neutral element*, i.e., an element 1 such that $1x = x1 = x$ for all $x \in M$; the neutral element is then unique. One example of a monoid is the *free monoid* Σ^* defined for a finite alphabet Σ and consisting of all words with letters in Σ (including the empty word), with concatenation as its law. Except for the free monoid, all semigroups and monoids considered are finite.

A semigroup element $x \in S$ is *idempotent* if $xx = x$. For $x \in S$, we denote by ω the idempotent power of x, i.e., the least positive integer such that x^ω is idempotent. A *zero* for S is an element $0 \in S$ such that $0x = x0 = 0$ for all $x \in S$: if it exists, it is unique. Given a semigroup S, we write S^1 for the monoid obtained by adding a fresh neutral element to S if it does not already have one.

A *morphism* from a semigroup S to a semigroup S' is a map $\mu \colon S \to S'$ such that for any $x, y \in S$, we have $\mu(xy) = \mu(x)\mu(y)$. A *morphism* from a monoid M to a monoid M' must additionally verify that $\mu(1) = 1'$, for 1 and $1'$ the neutral elements of M and M' respectively.

The *direct product* of two monoids M_1 and M_2 is $M_1 \times M_2$ with componentwise composition; it is also a monoid. A *quotient* of a monoid M is a monoid M' such that there is a surjective morphism from M to M'. A *submonoid* is a subset of a monoid which is also a monoid. The analogous notions for semigroups are defined in the expected way. A *pseudovariety* of monoids (resp., semigroups) is a class of monoids (resp., semigroups) closed under direct product, quotient and submonoid (resp., subsemigroup). The pseudovariety of monoids (resp., semigroups) *generated* by a class $\mathbf{V}$ of monoids (resp., of semigroups) is the least pseudovariety closed under these operations and containing $\mathbf{V}$. As we are working with finite semigroups and monoids, we refer to pseudovarieties simply as varieties.

We consider dynamic problems where we maintain a word w on a finite alphabet Σ, every letter being stored in a cell. We allow *letter substitution* updates of the form (i, a) for $1 \leq i \leq |w|$ and $a \in \Sigma$. The letter substitution update (i, a) replaces the i-th letter of w by a; the size $|w|$ of the word never changes. Given the input word w, we first preprocess it in time $O(|w|)$ to build a data structure. The data structure must then support update operations for letter substitution updates, and some query operations (to be defined below). The complexity that we measure is the worst-case complexity of an update operation or query operation, as a function of $|w|$. Our definition does not limit the memory used. However, all our upper complexity bounds actually have memory usage in $O(|w|)$. Further, all our lower bounds hold even when no assumption is made on the memory usage.

We focus on three related dynamic problems, allowing different query operations. The first is the *dynamic word problem for monoids*: fix a monoid M, the alphabet Σ is M, and the query returns the *evaluation* of the current word w, i.e., the product of the elements of w (it is an element of M). This is the problem studied in [29]. The second is the *dynamic word problem for semigroups*, which is the same but with a semigroup, and assuming that $|w| > 0$. The third is the *dynamic membership problem for regular languages*: we fix a regular language L on the alphabet Σ, and the query checks whether the current word belongs to L.

We study the data complexity of these problems in the rest of this paper, i.e., the complexity expressed as a function of w, with the monoid, semigroup, or language being fixed. Let us first observe that, for monoids and more generally for semigroups, the usual algebraic operators of quotient, subsemigroup, and direct product, do not increase the complexity of the problem:

▶ **Proposition 2.1.** *Let S and T be finite semigroups. The dynamic word problem of subsemigroups or quotients of S reduces to the same problem for S, and the dynamic word problem of $S \times T$ reduces to the same problem for S and T.*

Hard problems. All our lower bounds are obtained by reducing from the problem *prefix-M*, for M a fixed monoid. In this problem, we maintain a word of M^* under letter substitution updates, and handle *prefix queries*: given a prefix length, return the evaluation of the prefix of that length.

In particular, for $d \geq 2$, we consider the problem *prefix-$\mathbb{Z}_d$* for $\mathbb{Z}_d$ the cyclic group of order d, i.e., $\mathbb{Z}_d = \{0, \ldots, d-1\}$ with addition modulo d, where the evaluation of prefix is the sum of its elements modulo d. The following lower bound is known already in the cell probe model:

▶ **Theorem 2.2** ([12, 29])**.** *For any fixed $d \geq 2$, any structure for prefix-$\mathbb{Z}_d$ on a word of length n has complexity $\Omega(\log n / \log \log n)$.*

We also consider the problem *prefix-U_1*, where $U_1 = \{0, 1\}$ is the multiplicative monoid whose composition is the logical AND, i.e., prefix queries check if the prefix contains an occurrence of 0. Equivalently, we must maintain a subset S of a universe $\{1, \ldots, n\}$ (intuitively

n is the length of the word) under insertions and deletions, and support *threshold queries* that ask, given $0 \leq i \leq n$, whether S contains some element which is $\leq i$ (i.e., if some position before i has a 0). The prefix-U_1 problem can be solved in $O(\log \log n)$ [33] with a priority queue data structure, or even in expected $O(\sqrt{\log \log n})$ if we allow randomization [18]. Note that prefix-U_1 is slightly weaker than a priority queue as we can only *compare* the minimal value to a value given as input. We do not know of lower bounds on prefix-U_1, but conjecture [20] that it cannot be solved in $O(1)$:

▶ **Conjecture 2.3.** *There is no structure for prefix-U_1 with complexity $O(1)$.*

Note that the best algorithm for prefix-U_1 works by sorting small sets of large integers. This takes linear time in the cell probe model, so does not rule out the existence of an $O(1)$ priority queue [33]. Hence, a lower bound for prefix-U_1 would need to apply to the RAM model specifically, which would require new techniques.

Our last hard problem is prefix-U_2 where U_2 is the monoid $\{1, a, b\}$ with composition law $xy = y$ for $x, y \in \{a, b\}$, i.e., queries check if the last non-neutral element is a or b (or nothing). By adapting known results on the *colored predecessor problem* [24], we have:

▶ **Theorem 2.4** (Adapted from [24])**.** *Any structure for prefix-U_2 on a word of length n must be in $\Omega(\log \log n)$.*

General algorithms. Of course, the "hard" prefix problems, and the three problems that we study, can all be solved in $O(|w|)$ by re-reading the whole word at each update. We can improve this to $O(\log |w|)$ by maintaining a balanced binary tree on the letters of the word, with each node of the tree carrying the evaluation in the monoid of the letters reachable from that node. Any letter substitution update on the word can be propagated up to the root in logarithmic time, and the annotation of the root is the evaluation of the word. This algorithm has been implemented in practice [22]. A finer bound is given in [29] using a folklore technique of working with $(\log n)$-ary trees rather than binary trees, and using the power of the RAM model. We recall it here for monoids (but it applies to all three problems):

▶ **Theorem 2.5** ([29])**.** *For any fixed monoid M, the dynamic word problem and prefix problem for M are in $O(\log n / \log \log n)$.*

Our goal in this paper is to solve the dynamic word problem and dynamic membership problem more efficiently for specific classes of monoids, semigroups, and languages. We start our study with monoids in the next two sections, by studying the varieties **SG** and **ZG**.

3 Dynamic Word Problem for Monoids in SG

We define the class **SG** of monoids by the equation $x^{\omega+1}yx^{\omega} = x^{\omega}yx^{\omega+1}$ for all x, y. It incidentally occurs in [10, Theorem 3.1], but to our knowledge was not otherwise studied. The name **SG** means *swappable groups*. Intuitively, a monoid M is in **SG** iff, for any two elements $r, t \in M$ belonging to the same subgroup of M, we can *swap* them, i.e., $rst = tsr$ for all $s \in M$. We first recall the lower bound from [29] on the dynamic word problem for monoids *not in* **SG**, and then show an upper bound for monoids in **SG**.

Lower bound. The monoids not in **SG** are in fact those covered by the lower bound of [29]. Namely, we have the following, implying the $\Omega(\log n / \log \log n)$ lower bound by Theorem 2.2:

▶ **Theorem 3.1** ([29], Theorem 2.5.1)**.** *For any monoid M not in* **SG***, there exists $d \geq 2$ such that the prefix-$\mathbb{Z}_d$ problem reduces to the dynamic word problem for M.*

Upper bound. The rest of this section presents our upper bound on monoids in **SG**. In fact, we show a more general claim on the dynamic word problem for *semigroups* in **SG**, i.e., those satisfying the equation of **SG**. This covers in particular the monoids of **SG**:

▶ **Theorem 3.2.** *The dynamic word problem for any semigroup in* **SG** *is in* $O(\log \log n)$.

This result extends the result of [29] on group-free monoids, because **SG** contains all aperiodic monoids and all commutative monoids. Indeed, an aperiodic monoid satisfies the equation $x^{\omega+1} = x^\omega$, and then $x^{\omega+1}yx^\omega = x^\omega yx^\omega = x^\omega yx^{\omega+1}$. Besides, commutative monoids clearly satisfy the equation. Of course, **SG** captures monoids not covered by [29], e.g., products of a commutative monoid and an aperiodic monoid.

The result of [29] uses van Emde Boas trees [34], which we extend to store values in an alphabet Σ. Fixing an alphabet Σ, a *vEB tree* (or *vEB*) is a data structure parametrized by an integer n called its *span*, which stores a set $X \subseteq \{1, \ldots, n\}$ and a mapping $\mu\colon X \to \Sigma$, and supports the following operations: *inserting* an integer $x \in \{1, \ldots, n\} \setminus X$ with a label $\mu(x) := a$; *retrieving* the label of $x \in \{1, \ldots, n\}$ if $x \in X$ (or a special value if $x \notin X$); *removing* an integer $x \in X$ and its label; *finding the next integer* of X that follows an input $x \in \{1, \ldots, n\}$ (or a special value if none exists); and *finding the previous integer*.

We can implement vEBs so that these five operations run in $O(\log \log n)$ time in the worst case, and so that a vEB can be constructed in linear time from an ordered list.

We use vEBs in our inductive proof to represent words with "gaps": a vEB represents the word obtained by concatenating the labels of the elements of X in order. For a semigroup S and span $n \in \mathbb{N}$, the *dynamic word problem on vEBs* for S is to maintain a vEB T of span n on alphabet S under insertions and deletions, and to answer queries asking the evaluation in S of the word currently represented by T. As before, the complexity is the worst-case complexity of an insertion, deletion, or query, measured as a function of the span n (which never changes). The data structure for this problem must be initialized during a preprocessing phase on the initial vEB T, which must run in $O(n)$. Note that when T is empty then its evaluation is not expressible in the semigroup S: we then return a special value.

It is then clear that Theorem 3.2 follows from its generalization to vEBs, as a word in the usual sense can be converted in linear time to a vEB where $X = \{1, \ldots, n\}$:

▶ **Theorem 3.3.** *Let S be a semigroup in* **SG**. *The dynamic word problem for S on a vEB of span n is in* $O(\log \log n)$.

We show this result in the rest of the section. We assume without loss of generality that S has a zero, as otherwise we can simply add one. We first introduce some algebraic preliminaries, and then present the proof, which is an induction on $\mathcal{J}$-classes of the semigroup.

Preliminaries and proof structure. The *$\mathcal{J}$-order* of S is the preorder $\leq_\mathcal{J}$ on S defined by $s \leq_\mathcal{J} s'$ if $S^1 s S^1 \subseteq S^1 s' S^1$, recalling that S^1 is the monoid where we add a neutral element to S if it does not already have one. The equivalence classes of the symmetric closure of this preorder are called *$\mathcal{J}$-classes*. We lift the $\mathcal{J}$-order to $\mathcal{J}$-classes C, C' by writing $C \leq_\mathcal{J} C'$ if $u \leq_\mathcal{J} v$ for all $u \in C$ and $u' \in C'$. A $\mathcal{J}$-class is *maximal* if it is maximal for this order.

We show Theorem 3.3 by induction on the number of $\mathcal{J}$-classes of the semigroup. More precisely, at every step, we consider a maximal $\mathcal{J}$-class C, and remove it by reducing to the semigroup $S \setminus C$. Remember that the number of classes only depends on the fixed semigroup S, so it is constant. Thus, as the constant number of operations on vEBs at each class each take time $O(\log \log n)$, the overall bound is indeed in $O(\log \log n)$.

The base case of the induction is that of a semigroup with a single $\mathcal{J}$-class; from our assumption that the semigroup has a zero, that $\mathcal{J}$-class must then consist of the zero, i.e., we have the trivial monoid $\{0\}$, and the image is always 0 (or undefined if the word is empty).

We now show the induction step of Theorem 3.3. Take a semigroup S with more than one $\mathcal{J}$-class, and fix a maximal $\mathcal{J}$-class C of S: we know that $S \setminus C$ is not empty. What is more:

▷ Claim 3.4. For any x, y of S with $xy \in C$, then $x \in C$ and $y \in C$.

Thus, whenever a combination of elements "falls" outside of the maximal class C, then it remains in $S \setminus C$; and we can see $S \setminus C$ as a semigroup, which still has a zero, and has strictly less $\mathcal{J}$-classes. So we will study how to reduce to $S \setminus C$. We now make a case disjunction depending on whether C is *regular*, i.e., whether it contains an idempotent element.

Non-regular maximal classes. This case is easy, because products of two or more elements of C are never in C. To formalize this property, for a maximal $\mathcal{J}$-class C of S, we call a word w on S^* *pair-collapsing* for C if the product of any two adjacent letters of w is in $S \setminus C$. We show:

▶ **Lemma 3.5.** *Let C be a maximal $\mathcal{J}$-class. If C is non-regular, then any word is pair-collapsing: for any $x, y \in C$, we have $xy \in S \setminus C$.*

We can then show the following, which we will reuse for regular maximal classes:

▶ **Lemma 3.6.** *Let S be a semigroup and let C be a maximal $\mathcal{J}$-class of S. Consider the dynamic word problem for S on vEBs of some span n where we assume that, at every step, the represented word is pair-collapsing for C. Then that problem reduces to the dynamic word problem for $S \setminus C$ on vEBs of span n.*

Thanks to Lemma 3.5, this allows us to settle the case of a non-regular maximal $\mathcal{J}$-class, using the induction hypothesis to maintain the problem on $S \setminus C$.

Case of a regular maximal class. We now consider a maximal $\mathcal{J}$-class C that is regular. Consider the semigroup $C^0 := C \cup \{0\}$ for a fresh zero 0, i.e., the multiplication is that of C except that $x0 = 0x = 0$ for all $x \in C^0$, and that $xy = 0$ in C^0 for all $x, y \in C$ for which the product xy in S is not an element of C. Note that 0 is unrelated to the zero which S was assumed to have; intuitively, the 0 of C^0 stands for combinations of elements that are not in C. Another way to see C^0 is as the quotient of S by the ideal $S \setminus C$, i.e., we identify all elements of $S \setminus C$ to 0. By Prop. 4.35 of Chapter V of [26], we know that C^0 is a so-called *0-simple semigroup*. By the Rees-Sushkevich theorem (Theorem 3.33 of [26]), S is isomorphic to some *Rees matrix semigroup with 0*. This is a semigroup $M^0(G, I, J, P)$ with I and J two non-empty sets, G a group called the *structuring group*, and P a matrix indexed by $J \times I$ having values in G^0. The elements of the semigroup are the elements of $I \times G \times J$ and the element 0, with $x0 = 0x = 0$ for any element $x \in I \times G \times J$, and for (i, g, j) and (i', g', j') two elements of $I \times G \times J$, their product is 0 if $p_{j,i'} = 0$, and $(i, gp_{j,i'}g', j')$ otherwise.

With this representation, the idea is to collapse together the maximal runs of consecutive elements of C^0 whose product is not 0, i.e., does not "fall" outside of C. Once this is done, the product of two elements always falls in $S \setminus C$, so we can conclude using Lemma 3.6.

However, we cannot do this in a naive fashion. For instance, if we insert a letter in the vEB in the middle of such a maximal run, we cannot hope to split the run and know the exact group annotation of the two new runs – this could amount to solving a prefix-$\mathbb{Z}_d$ problem. Instead, we must now use the fact that S is in **SG**, and derive the consequences

of the equation in terms of the Rees-Sushkevich representation. Intuitively, the equation ensures that the structuring group G is commutative, and that annotations in G can "move around" relative to other elements in S without changing the evaluation. Formally:

▷ Claim 3.7. The structuring group G is commutative.

▷ Claim 3.8. Let $r, s, t \in S^*$ and $(i, g, j), (i', g', j') \in I \times G \times J$. Write $w = r(i, g, j)s(i', g', j')t$ and $w' = r(i, gg', j)s(i', 1, j')t$ where 1 is the neutral element of G. Then $\mathrm{eval}(w) = \mathrm{eval}(w')$.

This allows us to reduce the dynamic word problem on S to the same problem where we assume that the word is always pair-collapsing:

▷ Claim 3.9. The dynamic word problem for S on vEBs (of some span n) reduces to the same problem on vEBs of span n where we additionally require that, at every step, the represented word is pair-collapsing for the maximal $\mathcal{J}$-class C.

Proof sketch. We maintain a mapping where all maximal runs of word elements evaluating to C are collapsed to a single element, which we can evaluate following the Rees-Sushkevich representation. The tricky case is whenever an update breaks a maximal run into two parts: we cannot recover the G-component of the annotation of each part, but we use Claim 3.8 to argue that we can simply put it on the left part without altering the evaluation in S. ◁

This claim together with Lemma 3.6 implies that the dynamic word problem for S reduces to the same problem for $S \setminus C$, for which we use the induction hypothesis. This establishes the induction step and concludes the proof of Theorem 3.2.

4 Dynamic Word Problem for Monoids in ZG

We pursue our study of the dynamic word problem for monoids with the class **ZG**, introduced in [4] and defined by the equation $x^{\omega+1}y = yx^{\omega+1}$ for all x, y. This asserts that elements of the form $x^{\omega+1}$, which are the ones belonging to a *subgroup* of the monoid, are *central*, i.e., commute with all other elements. By the equations, and recalling that $x^{\omega+1} = x^\omega x^{\omega+1}$, clearly **ZG** $\subseteq$ **SG**. In this section, we show an $O(1)$ upper bound on the dynamic word problem for monoids in **ZG**, and a conditional lower bound for any monoid not in **ZG**.

Upper bound. Recall the result on commutative monoids from [29]:

▶ **Theorem 4.1** ([29]). *The dynamic word problem for any commutative monoid is in* $O(1)$.

Our goal is to generalize it to the following result:

▶ **Theorem 4.2.** *The dynamic word problem for any monoid in* **ZG** *is in* $O(1)$.

This generalizes Theorem 4.1 (as commutative monoids are clearly in **ZG**) and covers other monoids, e.g., the monoid $M = \{1, a, b, ab, 0\}$ with $a^2 = b^2 = ba = 0$, where it intuitively suffices to track the position of a's and b's and compare them if there is only one of each.

We now prove Theorem 4.2. A semigroup S is *nilpotent* if it has a zero and there exists $k > 0$ such that $S^k = \{0\}$, i.e., all products of k elements are equal to 0. Alternatively [26, Chapter X, Section 4], S is nilpotent iff it satisfies the equation $x^\omega y = yx^\omega = x^\omega$. We then consider the monoids of the form S^1 where S is nilpotent – an example of this is the monoid M described above. The variety generated by such monoids is called **MNil** and was studied by Straubing [30]. We can show:

▶ **Proposition 4.3.** *For any nilpotent S, the dynamic word problem for S^1 is in $O(1)$.*

Proof sketch. We maintain a (non-sorted) doubly-linked list L of the positions of the word w that contain a non-neutral element. As S is nilpotent, the evaluation of w is 0 unless constantly many non-neutral letters remain, which we can then find in $O(1)$ with L. ◀

In [3] we show that **ZG** is generated by such monoids S^1 and by commutative monoids:

▶ **Proposition 4.4** (Corollary 3.6 of [3]). *The variety* **ZG** *is generated by commutative monoids and monoids of the form S^1 for S a nilpotent semigroup.*

In view of Theorem 4.1 and Proposition 4.3, the dynamic word problem is in $O(1)$ for the semigroups that generate the variety **ZG** (Proposition 4.4). Theorem 4.2 then follows from Proposition 2.1.

Lower bound. We now show a conditional lower bound on the dynamic word problem for monoids outside of **ZG**. To do this, we will reduce from the prefix-U_1 problem:

▶ **Theorem 4.5.** *For any monoid M in $\mathbf{SG} \setminus \mathbf{ZG}$, the prefix-$U_1$ problem reduces to the dynamic word problem for M.*

Proof sketch. We consider the variety **ZE** [1] of monoids whose idempotents are central, i.e., the variety defined by the equation $x^\omega y = yx^\omega$. We show that $\mathbf{ZG} = \mathbf{SG} \cap \mathbf{ZE}$. We then show that, for any monoid not in **ZE**, we can reduce from the prefix-U_1 problem by encoding the elements 0 and 1 of U_1 using carefully chosen elements of the monoid. ◀

Using Conjecture 2.3, and together with Theorem 3.1 for the monoids not in **SG**, this implies a conditional super-constant lower bound for monoids outside **ZG**.

5 Dynamic Word Problem for Semigroups

We have classified the complexity of the dynamic word problem for monoids: it is in $O(\log \log n)$ for monoids in **SG**, in $O(1)$ for monoids in **ZG**, in $\Omega(\log n / \log \log n)$ for monoids not in **SG**, and non-constant for monoids not in **ZG** conditionally to Conjecture 2.3. In this section, we extend our results from monoids to *semigroups*.

Submonoids and local monoids. A *submonoid* of a semigroup S is a subset of the semigroup which is stable under its composition law and is a monoid. We first notice via Proposition 2.1 that a semigroup that contains a hard submonoid is also hard:

▷ Claim 5.1. The dynamic word problem for any submonoid of a semigroup S reduces to the same problem for S.

We will investigate if studying the submonoids of a semigroup suffices to understand the complexity of its dynamic word problem. To do so, we focus on a certain kind of submonoids: the *local monoids*. A submonoid N of S is a *local monoid* if there exists an idempotent element e of S such that $N = eSe$, i.e., N is the set of elements that can be written as ese for some $s \in S$. The point of local monoids is that they are maximal in the sense that every submonoid T of S is a submonoid of a local monoid: indeed, taking 1 the neutral element of T, all elements of T can be written as $1T1 \subseteq 1S1$ and $1S1$ is a local monoid. For **V** a variety of monoids, we denote by **LV** the variety of semigroups such that all local monoids are in **V**. As we explained, this is equivalent to imposing that all submonoids are in **V** (since varieties are closed under the submonoid operation).

Case of SG. We now revisit our results on monoids to extend them to semigroups, starting with **SG**. We denote by **LSG** the variety of semigroups whose local monoids are in **SG**. We show that a semigroup where all local monoids are in **SG** must itself be in **SG**:

▷ Claim 5.2. We have **LSG** = **SG** as varieties of semigroups.

Semigroups in **SG** are already covered by our upper bound (Theorem 3.2), and semigroups not in **LSG** have a submonoid not in **SG**, so we can use Claim 5.1 and Theorem 3.1. Hence:

▶ **Corollary 5.3.** *Let S be a semigroup. If S is in* **SG***, then the dynamic word problem for S is in $O(\log\log n)$. Otherwise, the dynamic word problem for S is in $\Omega(\log n/\log\log n)$.*

Case of ZG. The variety **ZG** is not equal to **LZG**. For instance, let S be the syntactic semigroup of a^*b^*, that is the semigroup $\{a, b, ab, 0\}$ defined with $a^2 = a$, $b^2 = b$ and $ba = 0$. It is not in **ZG**, since a and b are idempotents that do not commute. However, its local monoids are either trivial or U_1, so they are all in **ZG**, showing that this semigroup is in **LZG**. Still, we can extend our characterization from monoids to semigroups up to studying **LZG**:

▶ **Theorem 5.4.** *Let S be a semigroup. If S is in* **LZG***, then the dynamic word problem for S is in $O(1)$. Otherwise, unless prefix-U_1 is in $O(1)$, the dynamic word problem for S is not in $O(1)$.*

The second part of the claim is by Claim 5.1 and Theorem 3.1, but the first part is much trickier. We use a characterization of **LZG** as a *semidirect product* **ZG** $*$ **D**, which follows from a very technical *locality result* on **ZG** [3], and then design an algorithm for the dynamic word problem for semigroups in **ZG** $*$ **D**. We prove Theorem 5.4 in the rest of this section.

Given two semigroups S and T, a *semigroup action* of S on T is defined by a map $\mathrm{act} : S \times T \to T$ such that $\mathrm{act}(s_1, \mathrm{act}(s_2, t)) = \mathrm{act}(s_1 s_2, t)$ and $\mathrm{act}(s, t_1 t_2) = \mathrm{act}(s, t_1)\mathrm{act}(s, t_2)$. We then define the product $\circ_{\mathrm{act}}$ on the set $T \times S$ as follows: for all s_1, s_2 in S and t_1, t_2 in T, we have: $(t_1, s_1) \circ_{\mathrm{act}} (t_2, s_2) := (t_1 \mathrm{act}(s_1, t_2), s_1 s_2)$. The set $T \times S$ equipped with the product $\circ_{\mathrm{act}}$ is a semigroup called the *semidirect product* of S by T, denoted $T \circ_{\mathrm{act}} S$.

We say that a semigroup D is *definite* if there exists an integer $k \in \mathbb{N}$ such that for all $y, x_1, \ldots, x_k$ in D, we have $yx_1 \cdots x_k = x_1 \cdots x_k$. Alternatively, a semigroup is definite iff it satisfies the equation $yx^\omega = x^\omega$ [31, Proposition 2.2] for all x, y in D. In particular, every nilpotent semigroup is definite. We write **D** for the variety of definite semigroups.

Our alternative definition of **LZG** will be the variety of semigroups **ZG** $*$ **D** generated by semigroups that are the semidirect product of a **ZG** monoid by a definite semigroup.

The variety **ZG** $*$ **D** of semigroups is not immediately related to the variety **LZG** defined above. One can easily show that **ZG** $*$ **D** $\subseteq$ **LZG**, but the other direction is much more challenging to establish. We show this as a so-called *locality theorem*, which we defer to a separate paper [3] because it uses different tools and is of possible independent interest:

▶ **Theorem 5.5** ([3], Theorem 1.1)**.** *We have:* **ZG** $*$ **D** = **LZG**.

To conclude the proof of Theorem 5.4, by the locality theorem above, it suffices to solve the dynamic word problem for semigroups in **ZG** $*$ **D**. By Proposition 2.1, it suffices to consider the semigroups that generate the variety. We do this below, establishing Theorem 5.4:

▶ **Proposition 5.6.** *Let S be a definite semigroup, let T be a semigroup of* **ZG***, and let* act *be a semigroup action of S on T. The dynamic word problem for the semigroup $T \circ_{\mathrm{act}} S$ reduces to the same problem for T.*

Proof sketch. We express the direct product of the letters of the input word as a product involving elements of T and prefix sums of elements of S, which we can maintain in $O(1)$. ◀

6 Dynamic Word Problem for Languages

We now turn to the dynamic membership problem for regular languages, and show Theorem 1.1 using the three previous sections and some extra algebraic results.

Connection to the dynamic word problem. A regular language L is *recognized* by a finite monoid if there exists a morphism $\eta\colon \Sigma^* \to M$ such that $L = \eta^{-1}(\eta(L))$. The *syntactic congruence* of L is the equivalence relation on Σ^* where $u, v \in \Sigma^*$ are equivalent iff, for each $r, t \in \Sigma^*$, either $rut \in L$ and $rvt \in L$, or $rut \notin L$ and $rvt \notin L$. The *syntactic monoid* M of L is the quotient of Σ^* by the syntactic congruence of L, and the *syntactic morphism* is the morphism mapping Σ^* to M; the syntactic morphism witnesses that the syntactic monoid recognizes L.

The dynamic membership problem for a language clearly reduces to the dynamic word problem for its syntactic monoid. However, the converse is not true: the language $L := (aa)^*ba^*$ has a syntactic monoid M that can be shown to be outside of **SG**, but we can solve dynamic membership for L in $O(1)$ by counting the b's at even and odd positions. Intuitively, M has a neutral element 1 so that the dynamic word problem for M has a reduction from prefix-$\mathbb{Z}_2$, but 1 is not achieved by a letter of the alphabet so dynamic membership for L is easier.

We extend our results to languages using the notion of *stable semigroup* [6, 7]. This allows us to remove the neutral element (as it is a semigroup not a monoid) and ensures that all semigroup elements can be achieved by subwords of some constant length (the *stability index*).

Formally, let L be a regular language and $\eta\colon \Sigma^* \to M$ its syntactic morphism. The *powerset* of M is the monoid whose elements are subsets of M and for $E, F \subseteq M$, the product EF is $\{xy \mid x \in E, y \in F\}$. The *stability index* of L is the idempotent power s of $\eta(\Sigma)$ in the powerset monoid. Intuitively, this choice of s ensures that, for any two words $w_1, w_2 \in \Sigma^s$, the value $\eta(w_1w_2)$ of their concatenation in the monoid can be achieved by another word of Σ^s, i.e., $\eta(w_1w_2) = \eta(w)$ for some $w \in \Sigma^s$. Then $\eta(\Sigma^s)$ is a subsemigroup of M, because $(\eta(\Sigma^s))^2 = \eta(\Sigma^s)$: we call it the *stable semigroup* of L. For any class of semigroups $\mathbf{V}$, we denote by $\mathbf{QV}$ the class of languages whose stable semigroup is in $\mathbf{V}$.

Upper bounds. We first show that the dynamic membership problem for a regular language reduces more specifically to the dynamic word problem for its *stable semigroup*:

▶ **Proposition 6.1.** *Let L be a regular language. The dynamic membership problem for L reduces to the dynamic word problem for the stable semigroup of L.*

Proof sketch. We partition the word of L into chunks of size s (plus one of size $\leq s$) for s the stability index, and feed them to the data structure for the stable semigroup of L. ◀

By Corollary 5.3 and Theorem 5.4, this implies that languages in **QSG** (resp., in **QLZG**) have a dynamic membership problem in $O(\log \log n)$ (resp., in $O(1)$).

Lower bounds. We now show that languages whose stable semigroup is not in **LV** admit a reduction from the dynamic word problem of a monoid of **V**.

▶ **Proposition 6.2.** *Let $\mathbf{V}$ be a variety of monoids and let L be a regular language not in $\mathbf{QLV}$. There is a monoid not in $\mathbf{V}$ whose dynamic word problem reduces to the dynamic membership problem for L.*

Proof sketch. If L is not in **QLV**, then its stable semigroup contains a submonoid M not in **V**, and all elements can be achieved by a block of s letters for s the stability index. ◀

Again by Corollary 5.3 and Theorem 5.4, we deduce that languages outside of **QSG** have complexity at least $\Omega(\log n/\log\log n)$. Further, assuming Conjecture 2.3, and languages outside of **QLZG** do not have complexity $O(1)$.

7 Extensions, Problem Variants, and Future Work

We have presented our results on the dynamic word problem for monoids and semigroups, and on the dynamic membership problem for regular languages. We conclude the paper by a discussion of problems for further study. We first discuss the question of *intermediate complexities* between $O(1)$ and $O(\log\log n)$. We then study the complexity of *deciding which case applies* as a function of the target language, semigroup, or monoid. We last explore the issue of *handling insertions and deletions* on the input word, and of supporting *infix queries*.

Intermediate complexities. Our $O(\log\log n)$ upper bound in Theorem 3.2 and its variants may not be tight. Still, we can identify a language L_{U_2} in $\mathbf{QSG} \setminus \mathbf{QLZG}$ for which we show that the dynamic membership problem is in $\Theta(\log\log n)$ (even allowing randomization and allowing a probably correct answer), because the prefix-U_2 problem reduces to it.

We can also identify a language of $\mathbf{QSG} \setminus \mathbf{QLZG}$ that reduces to prefix-U_1 and so can be solved in expected $O(\sqrt{\log\log n})$. This shows that languages in $\mathbf{QSG} \setminus \mathbf{QLZG}$ have different complexity regimes, at least when allowing randomization.

▶ **Proposition 7.1.** *There is a language L_{U_2} in $\mathbf{QSG} \setminus \mathbf{QLZG}$ which is equivalent to prefix-U_2 under constant-time reductions, and a language L_{U_1} in $\mathbf{QSG} \setminus \mathbf{QLZG}$ which is equivalent to prefix-U_1 under constant-time reductions.*

Deciding which case applies. A natural question about our results is the question of efficiently identifying, given a regular language, which of the cases of Theorem 1.1 applies, or in particular of determining, given an input monoid or semigroup, if it is in **SG**, or in **ZG**. This depends on how the input is represented. If we are given a monoid explicitly (as a table of its operations), then the equations of **ZG** and of **SG** can be checked in polynomial time. If the monoid is represented more concisely as the transition monoid of some automaton, then the verification can be performed in PSPACE. We do not know if the problems are PSPACE-hard, though this seems likely at least for **SG** because of its proximity to aperiodic monoids [8]. We leave open the precise complexities of this task, in particular for the **L** and **Q** operators.

Handling insertions and deletions. Another natural question is to handle insertion and deletion updates, i.e., *inserting* letter a at position k transforms the word $w_1 \cdots w_{k-1} w_k \cdots w_n$ into $w_1 \cdots w_{k-1} a w_k \cdots w_n$, and *deleting* at position k does the opposite. Any regular language can be maintained under such updates in $O(\log n)$ using a Fenwick tree, but it makes the problem much harder for most languages. For example, if the alphabet has two letters a and b, just testing if the word that we maintain contains an a requires $\Omega(\log n/\log\log n)$ by [21]. This is why we do not study such updates in this work. Interestingly, notice that our algorithm in Theorem 3.3 supports insertions and deletions on words represented as vEBs, but the semantics are different (they use explicit positions in a fixed range).

Infix queries. A natural extension of dynamic membership for a regular language L is the *dynamic infix membership problem*, where we can query any *infix* of the word (identified by its endpoints) to ask whether it is in L. The $O(\log n/\log\log n)$ algorithm of Theorem 2.5 supports this, and so can the $O(\log\log n)$ algorithm of [29] for group-free monoids. However, the infix problem can be harder. Consider for instance the language L_2 on $\{a, b\}$ of words with an even number of a's. Dynamic membership has complexity $O(1)$ because L_2 is commutative, but infix queries (or even prefix queries) require $\Omega(\log n/\log\log n)$ as prefix-$\mathbb{Z}_2$ reduces to it.

We leave open the study of the complexity of the infix problem. We note, however, that this problem for a language L can be studied as the dynamic membership problem for a regular language defined from L. So our results cover the infix problem via this transformation; we leave to future work a characterization of the resulting classes.

▷ Claim 7.2. For any fixed regular language L, the dynamic infix membership problem is equivalent up to constant-time reductions to the dynamic membership problem for the language $\Sigma^* x L x \Sigma^*$ where x is a fresh letter.

Other open questions. A natural question for future work would be to study the complexity of our problems in weaker models, e.g., pointer machines [32], or machines with counters. One could also extend our study to languages that are not regular, e.g., generalizing bounds on maintaining the language of well-parenthesized strings ([19, Proposition 1] and [11]).

References

1 Jorge Almeida and Assis Azevedo. Implicit operations on certain classes of semigroups. In *Semigroups and their Applications*. Springer, 1987.

2 Antoine Amarilli, Louis Jachiet, and Charles Paperman. Dynamic membership for regular languages, 2021. `arXiv:2003.02576`.

3 Antoine Amarilli and Charles Paperman. Locality and centrality: The variety **ZG**, 2021. `arXiv:2102.07724`.

4 Karl Auinger. Semigroups with central idempotents. In *Algorithmic problems in groups and semigroups*. Springer, 2000.

5 Andrey Balmin, Yannis Papakonstantinou, and Victor Vianu. Incremental validation of XML documents. *TODS*, 29(4), 2004. URL: `http://db.ucsd.edu/wp-content/uploads/pdfs/212.pdf`.

6 David A. Mix Barrington, Kevin J. Compton, Howard Straubing, and Denis Thérien. Regular languages in NC^1. *J. Comput. Syst. Sci.*, 44(3), 1992. URL: `https://www.sciencedirect.com/science/article/pii/002200009290014A`.

7 Laura Chaubard, Jean-Eric Pin, and Howard Straubing. First order formulas with modular predicates. In *LICS*, 2006. URL: `https://hal.archives-ouvertes.fr/hal-00112846/document`.

8 Sang Cho and Dung T. Huynh. Finite-automaton aperiodicity is PSPACE-complete. *Theoretical Computer Science*, 88(1), 1991. `doi:10.1016/0304-3975(91)90075-d`.

9 Raphael Clifford, Allan Grønlund, Kasper Green Larsen, and Tatiana Starikovskaya. Upper and lower bounds for dynamic data structures on strings. In *STACS*, 2018. `doi:10.4230/LIPIcs.STACS.2018.22`.

10 Assis de Azevedo. The join of the pseudovariety **J** with permutative pseudovarieties. In *Lattices, Semigroups, and Universal Algebra*. Springer, 1990.

11 Gudmund Skovbjerg Frandsen, Thore Husfeldt, Peter Bro Miltersen, Theis Rauhe, and Søren Skyum. Dynamic algorithms for the Dyck languages. In *WADS*, 1995. URL: `https://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.57.4615&rep=rep1&type=pdf`.

12 Michael Fredman and Michael Saks. The cell probe complexity of dynamic data structures. In *STOC*, 1989. URL: https://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.453.9085&rep=rep1&type=pdf.

13 Moses Ganardi, Danny Hucke, Daniel König, Markus Lohrey, and Konstantinos Mamouras. Automata theory on sliding windows. In *STACS*, 2018. doi:10.4230/LIPIcs.STACS.2018.31.

14 Moses Ganardi, Danny Hucke, and Markus Lohrey. Querying regular languages over sliding windows. In *FSTTCS*, 2016. doi:10.4230/LIPIcs.FSTTCS.2016.18.

15 Moses Ganardi, Danny Hucke, Markus Lohrey, and Tatiana Starikovskaya. Sliding window property testing for regular languages. In *ISAAC*, 2019. doi:10.4230/LIPIcs.ISAAC.2019.6.

16 Wouter Gelade, Marcel Marquardt, and Thomas Schwentick. The dynamic complexity of formal languages. *TOCL*, 13(3), 2012. arXiv:0812.1915.

17 Kasper Green Larsen, Jonathan Lindegaard Starup, and Jesper Steensgaard. Further unifying the landscape of cell probe lower bounds. In *SOSA*, 2021. URL: https://epubs.siam.org/doi/pdf/10.1137/1.9781611976472.25.

18 Yijie Han and Mikkel Thorup. Integer sorting in $O(n\sqrt{\log\log n})$ expected time and linear space. In *FOCS*, 2002.

19 Thore Husfeldt and Theis Rauhe. Hardness results for dynamic problems by extensions of Fredman and Saks' chronogram method. In *ICALP*, 1998. URL: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.29.6315&rep=rep1&type=pdf.

20 Louis Jachiet. Computing and maintaining the minimum of a set S of integers while allowing updates on S. Theoretical Computer Science Stack Exchange. URL: https://cstheory.stackexchange.com/q/47831 (version: 2020-11-08).

21 Louis Jachiet. On the complexity of a "list" datastructure in the RAM model. Theoretical Computer Science Stack Exchange. URL: https://cstheory.stackexchange.com/q/46746 (version: 2020-05-01).

22 Eugene Kirpichov. Incremental regular expressions. Code available at: https://github.com/jkff/ire, 2012. URL: http://jkff.info/articles/ire/.

23 Mihai Pătraşcu and Corina E Tarniţă. On dynamic bit-probe complexity. *Theoretical Computer Science*, 380(1-2), 2007. URL: https://www.sciencedirect.com/science/article/pii/S0304397507001624.

24 Mihai Pătraşcu and Mikkel Thorup. Randomization does not help searching predecessors. In *SODA*, 2007. URL: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.208.6464&rep=rep1&type=pdf.

25 Jean-Éric Pin. *Varieties of formal languages*. Foundations of Computer Science. Plenum Publishing Corp., New York, 1986. With a preface by M.-P. Schützenberger, Translated from the French by A. Howie. doi:10.1007/978-1-4613-2215-3.

26 Jean-Éric Pin. Mathematical foundations of automata theory, 2019. URL: https://www.irif.fr/~jep/PDF/MPRI/MPRI.pdf.

27 John Rhodes. The fundamental lemma of complexity for arbitrary finite semigroups. *Bulletin of the American Mathematical Society*, 74(6), 1968. doi:10.1090/s0002-9904-1968-12064-6.

28 Jonas Schmidt, Thomas Schwentick, Till Tantau, Nils Vortmeier, and Thomas Zeume. Work-sensitive dynamic complexity of formal languages, 2021. arXiv:2101.08735.

29 Gudmund Skovbjerg Frandsen, Peter Bro Miltersen, and Sven Skyum. Dynamic word problems. *JACM*, 44(2), 1997. URL: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.868.4450&rep=rep1&type=pdf.

30 Howard Straubing. The variety generated by finite nilpotent monoids. *Semigroup Forum*, 24(1), 1982. doi:10.1007/bf02572753.

31 Howard Straubing. Finite semigroup varieties of the form V*D. *Journal of Pure and Applied Algebra*, 36, 1985. URL: https://www.sciencedirect.com/science/article/pii/0022404985900623.

32 Robert Endre Tarjan. A class of algorithms which require nonlinear time to maintain disjoint sets. *Journal of Computer and System Sciences*, 18(2), 1979. doi:10.1016/0022-0000(79)90042-4.

33 Mikkel Thorup. Equivalence between priority queues and sorting. *JACM*, 54(6), 2007.

34 Peter van Emde Boas, Robert Kaas, and Erik Zijlstra. Design and implementation of an efficient priority queue. *Mathematical systems theory*, 10(1), 1976.

Appendix

Table 1 Summary of the main classes of monoids and semigroups used in the paper.

Class	Description	Equation	References
ZE	Monoids/semigroups with central idempotents	$x^\omega y = yx^\omega$	[1]
ZG	Monoids/semigroups with central groups	$x^{\omega+1}y = yx^{\omega+1}$	[4]
SG	Monoids/semigroups with swappable groups	$x^{\omega+1}yx^\omega = x^\omega yx^{\omega+1}$	[10, 29]
A	Aperiodic semigroups/monoids	$x^{\omega+1} = x^\omega$	[26]
Com	Commutative semigroups/monoids	$xy = yx$	[26]
Nil	Nilpotent semigroups	$x^\omega y = yx^\omega$	[26]
MNil	Monoids dividing a nilpotent semigroup		[30]
D	Definite semigroups	$yx^\omega = x^\omega$	[31]

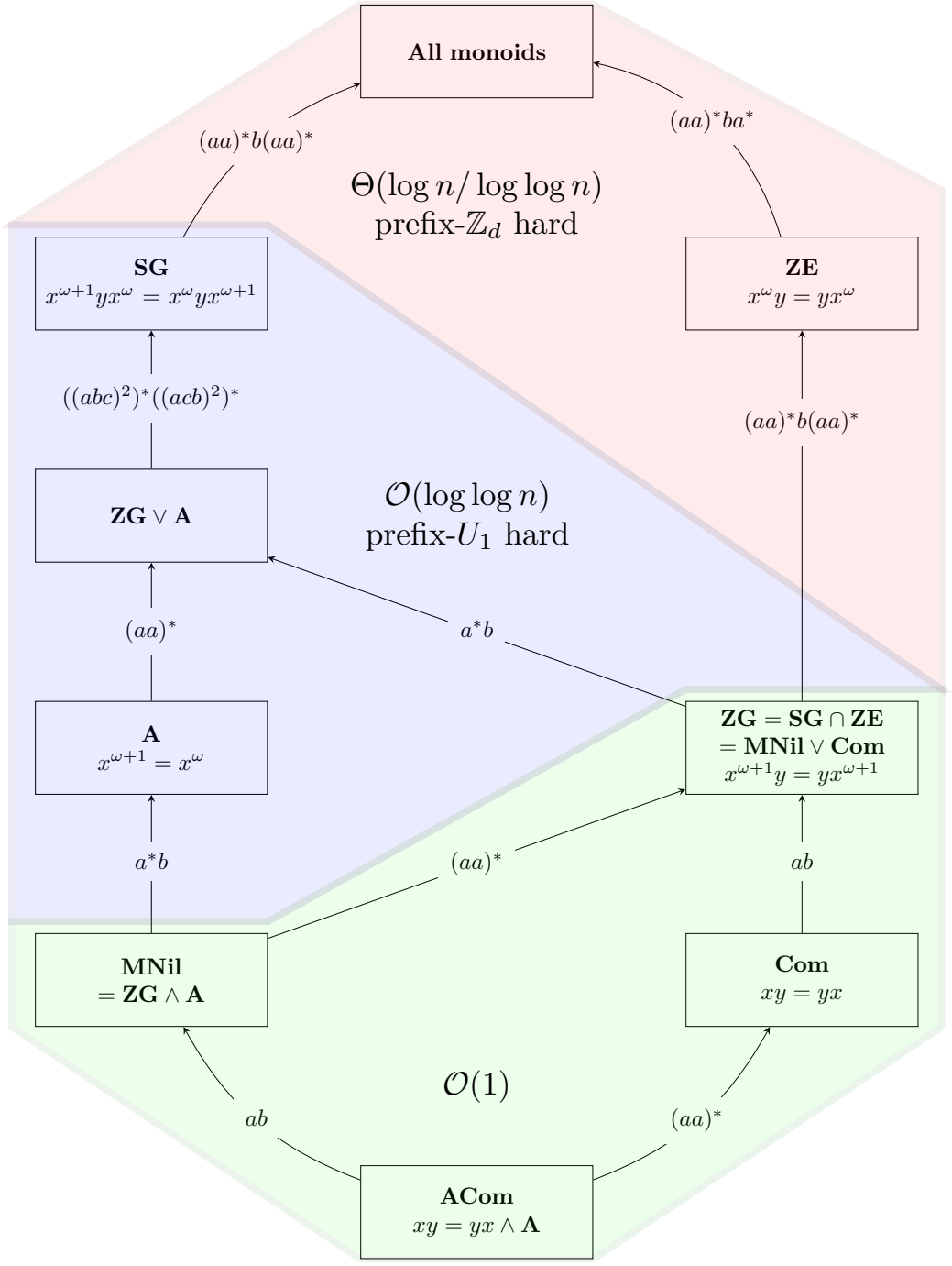


Figure 1 Complexity of the dynamic word problem for common classes of monoids. Arrows denote inclusion and are labeled with languages (with an implicit neutral letter e) whose syntactic monoids separate the classes. The classes **ZG** and **SG** are maximal for the $O(1)$ region and $O(\log\log n)$ region respectively.

A Rice's Theorem for Abstract Semantics

Paolo Baldan
Dipartimento di Matematica, University of Padova, Italy

Francesco Ranzato
Dipartimento di Matematica, University of Padova, Italy

Linpeng Zhang
Department of Computer Science, University College London, UK

Abstract

Classical results in computability theory, notably Rice's theorem, focus on the extensional content of programs, namely, on the partial recursive functions that programs compute. Later and more recent work investigated intensional generalisations of such results that take into account the way in which functions are computed, thus affected by the specific programs computing them. In this paper, we single out a novel class of program semantics based on abstract domains of program properties that are able to capture nonextensional aspects of program computations, such as their asymptotic complexity or logical invariants, and allow us to generalise some foundational computability results such as Rice's Theorem and Kleene's Second Recursion Theorem to these semantics. In particular, it turns out that for this class of abstract program semantics, any nontrivial abstract property is undecidable and every decidable overapproximation necessarily includes an infinite set of false positives which covers all values of the semantic abstract domain.

2012 ACM Subject Classification Theory of computation → Computability; Theory of computation → Recursive functions; Theory of computation → Abstraction

Keywords and phrases Computability Theory, Recursive Function, Rice's Theorem, Kleene's Second Recursion Theorem, Program Analysis, Affine Program Invariants

Digital Object Identifier 10.4230/LIPIcs.ICALP.2021.117

Category Track B: Automata, Logic, Semantics, and Theory of Programming

Related Version *Full Version*: `https://arxiv.org/abs/2105.14579`

Funding *Paolo Baldan*: Partially funded by *University of Padova*, under the SID2018 project "Analysis of STatic Analyses (ASTA)"; *Italian Ministry of University and Research*, under the PRIN2017 project no. 201784YSZ5 "AnalysiS of PRogram Analyses (ASPRA)".
Francesco Ranzato: Partially funded by *University of Padova*, under the SID2018 project "Analysis of STatic Analyses (ASTA)"; *Italian Ministry of University and Research*, under the PRIN2017 project no. 201784YSZ5 "AnalysiS of PRogram Analyses (ASPRA)"; *Facebook Research*, under a "Probability and Programming Research Award".

Acknowledgements We are grateful to Roberto Giacobazzi for thorough discussions and comments.

1 Introduction

Most classical results in computability theory focus on the so-called *extensional* properties of programs, i.e., on the properties of the partial functions they compute. Notably, the renowned Rice's Theorem [25] states that any nontrivial extensional property of programs is undecidable. Despite being very general, Rice's Theorem and similar results in computability theory, due to the requirement of extensionality, leave out several *intensional* properties which are of utmost importance in the practice of programming. Essential intensional properties of programs include their asymptotic complexity of computation, their logical invariants (e.g., relations between variables at program points), or any event that might happen during program computation while not affecting the program output.

State-of-the-Art

A generalisation of well-established results of computability theory to the realm of program complexity has been put forward by Asperti [1]. A first observation is that Blum's complexity classes [2], i.e., sets of recursive functions (rather than sets of programs) with some given (lower or upper) bound on their (space and/or time) complexity, are not adequate for investigating the decidability aspects of program complexity: in fact, viewed as program properties they are trivially extensional. Thus, a key idea in [1] is to focus on the so-called *complexity cliques*, namely, sets of programs (i.e., program indices) closed with respect to their extensional input/output behaviour and their asymptotic complexity. Asperti [1] showed how this approach enables intensional versions of Rice's theorem, Rice-Shapiro theorem, and Kleene's second recursion theorem ([8, 28] are standard references for these foundational results) for complexity cliques.

More recently, a different approach has been considered by Moyen and Simonsen in [19], where the classical definition of extensionality has been weakened to a notion of *partial extensionality*. Roughly, a given set of programs is partially extensional if it includes the set of all programs computing a given partial recursive function. It is shown in [19] that if a set of programs and its complement are partially extensional, then they cannot be both recursive. Interestingly, this result can be further generalised by replacing the extensionality with an equivalence relation on programs satisfying some suitable structural conditions, notably, the existence of a so-called intricated switching family. Moyen and Simonsen [19] show how to derive within their framework intensional versions of Rice's Theorem – generalising Asperti's result [1] – and Rice-Shapiro Theorem.

Main Contributions

Along the lines traced by Asperti [1], we investigate whether and how some fundamental extensional results of computability theory can be systematically generalised to intensional aspects of computation, but rather than focussing on specific intensional properties we deal with generic *abstract program semantics*. More in detail, we distill two fundamental properties of abstract program semantics in our approach: the *strong smn property* and the existence of a *universal fair program*, roughly, an interpreter that preserves the abstract semantics. We show that for abstract semantics satisfying the strong smn property and admitting a universal fair program, a generalisation of Kleene's second recursion theorem can be proved. This, in turn, leads to a generalisation of Rice's theorem. Besides relying on a general abstract program semantics, inspired by Moyen and Simonsen's approach [19], we also relax the extensionality condition to partial extensionality. This weakening provides stronger impossibility results as it allows us to show that it is undecidable whether a given program *can* have a particular semantics, i.e., even nontrivial overapproximations of such properties are undecidable. On a different route, we establish a precise connection with Moyen and Simonsen's work [19] by showing that for any abstract program semantics satisfying the strong smn property and a structural *branching* condition (roughly, expressing some form of conditional choice), we can prove the existence of an intricated switching family, which turns out to be the crucial hypothesis in [19] for deriving an intensional version of Rice's theorem.

Therefore, on the one hand, we generalise the results in [1], going beyond complexity cliques, and, on the other hand, we provide an explicit characterisation of a class of program semantics that admit intricated switching families so that the results in [19] can be applied.

Finally, we show some applications of our intensional Rice's theorem that generalise some undecidability results for intensional properties used in static program analysis. In particular, we focus on program analysis in Karr's abstract domain of affine relations between program

variables [13]. By exploiting an acute reduction to the undecidable Post correspondence problem, Müller-Olm and Seidl [20] prove that for affine programs with positive affine guards it is undecidable whether a given nontrivial affine relation holds at a given program point or not. Here, we first show that this class of affine programs with positive affine guards, modeled as control flow graphs, turns out to be Turing complete since, by selecting a suitable program semantics, these programs can simulate a URM. Then, this allows us to derive the undecidability result in [20] as a consequence of our results.

The rest of the paper is structured as follows. In Section 2, we provide some background and our basic notions. In Section 3, we introduce the strong smn property, fair universal programs, and the branching condition that will play a fundamental role in our results. In Section 4, we provide our generalisation of Kleene's second recursion theorem and use it to derive our intensional Rice's theorem. We also establish an explicit connection with the notion of intricated switching family given in [19]. Section 5 provides some applications of our results to the analysis of affine programs. Section 6 discusses in detail the relation with some of Asperti's results [1] and with Rogers' systems of indices [27, 28]. Finally, Section 7 concludes and outlines some directions of future work.

2 Basic Notions

Given an n-ary partial function $f : \mathbb{N}^n \to \mathbb{N}$, we denote by $\mathrm{dom}(f)$ the domain of f and by $\mathrm{rng}(f) \triangleq \{f(\vec{x}) : \vec{x} \in \mathrm{dom}(f)\}$ its range. We write $f(\vec{x})\downarrow$ if $\vec{x} \in \mathrm{dom}(f)$ and $f(\vec{x})\uparrow$ if $\vec{x} \notin \mathrm{dom}(f)$. Moreover, $\lambda\vec{x}.\uparrow$ denotes the always undefined function. We denote by $\mathcal{F}_n \triangleq \mathbb{N}^n \to \mathbb{N}$ the class of all n-ary (possibly partial) functions and by $\mathcal{F} \triangleq \bigcup_n \mathcal{F}_n$ the class of all such functions. Additionally, $\mathcal{C}_n \subseteq \mathcal{F}_n$ denotes the subset of n-ary partial recursive functions ($\mathcal{C}$ stands for computable) and $\mathcal{C} \triangleq \bigcup_n \mathcal{C}_n$ the set of all partial recursive functions.

▶ **Assumption 2.1** (Turing completeness)**.** Throughout the paper, we assume a fixed Turing complete model and we denote by $\mathcal{P}$ the corresponding set of programs. Moreover, we consider a fixed Gödel numbering for the programs in $\mathcal{P}$ and, given an index $a \in \mathbb{N}$, we write P_a for the a-th program in $\mathcal{P}$. A program can take a varying number n of inputs and we denote by $\phi_a^{(n)} \in \mathcal{C}_n$ the n-ary partial function computed by P_a. Therefore, by Turing completeness, $\{\phi_a^{(n)} \mid a, n \in \mathbb{N}\} = \mathcal{C}$ must hold. ⌟

The binary relation between programs that compute the same n-ary function is called *Rice's equivalence* and denoted by $\sim_R^n$, i.e.,

$$a \sim_R^n b \overset{\triangle}{\iff} \phi_a^{(n)} = \phi_b^{(n)}.$$

Classical Rice's theorem [25] compares the extension of programs, i.e., the functions they compute, and shows that unions of equivalence classes of programs computing the same function are undecidable. In Asperti's work [1], by relying on the notion of complexity clique, the asymptotic program complexity can be taken into account. The idea here is to further generalise the approach in [1] by considering generic program semantics rather than asymptotic program complexity. Additionally, an equivalence relation on program semantics allows us to further abstract and identify programs with different extensional semantics. More precisely, such an equivalence relation allows us to reason on semantic program properties that may not hold with functional equivalence.

▶ **Definition 2.2** (Abstract semantics)**.** An *abstract semantics* is a pair $\langle \pi, \equiv_\pi \rangle$ where:

(1) $\pi : \mathbb{N}^2 \to \mathcal{F}$ associates a program index a and arity n with an n-ary function $\pi_a^{(n)} \in \mathcal{F}_n$, called *semantics* of a;

(2) $\equiv_\pi \subseteq \mathcal{F} \times \mathcal{F}$ is an equivalence relation between functions.

Given $n \in \mathbb{N}$, the *n-ary program equivalence* induced by an abstract semantics $\langle \pi, \equiv_\pi \rangle$ is the equivalence $\sim_\pi^n \subseteq \mathbb{N} \times \mathbb{N}$ defined as follows: for all $a, b \in \mathbb{N}$,

$$a \sim_\pi^n b \overset{\triangle}{\iff} \pi_a^{(n)} \equiv_\pi \pi_b^{(n)}.$$ ⌟

The notation for the case of arity $n = 1$ will be simplified by omitting the arity, e.g., ϕ_a instead of $\phi_a^{(1)}$ and $\sim_\pi$ in place of $\sim_\pi^1$. Abstract semantics can be viewed as a generalisation of the notion of system of indices (or numbering), as found in standard reference textbooks [22, 28] and discussed in detail later in Section 6.2. Let us now show how the standard extensional interpretation of programs, complexity and complexity cliques can be cast into our setting.

▶ **Example 2.3** (Concrete semantics)**.** The concrete input/output semantics can be trivially seen as an abstract semantics $\langle \phi, = \rangle$ where $\phi_a^{(n)}$ is the n-ary function computed by P_a and $=$ is the equality between functions. Observe that this concrete semantics induces an n-ary program equivalence which is Rice's equivalence $\sim_R^n$. ⌟

▶ **Example 2.4** (Domain semantics)**.** For a given set of inputs $S \subseteq \mathbb{N}$, consider $\langle \phi, \equiv_S \rangle$ where $\phi_a^{(n)}$ is the n-ary function computed by P_a and for $f, g : \mathbb{N}^n \to \mathbb{N}$, their equivalence is defined by $f \equiv_S g \overset{\triangle}{\iff} \mathrm{dom}(f) \cap S = \mathrm{dom}(g) \cap S$. ⌟

▶ **Example 2.5** (Blum complexity)**.** Let $\Phi : \mathbb{N}^2 \to \mathcal{C}$ be a Blum complexity [2], i.e., for all $a \in \mathbb{N}$ and $\vec{x} \in \mathbb{N}^n$, (1) $\Phi_a^{(n)}(\vec{x}) \downarrow \Leftrightarrow \phi_a^{(n)}(\vec{x}) \downarrow$ holds, and (2) for all $m \in \mathbb{N}$, the predicate $\Phi_a^{(n)}(\vec{x}) = m$ is decidable. Letting $\Theta(f)$ to denote the standard Big Theta complexity class of a function f, the pair $\langle \Phi, \equiv_\Phi \rangle$ defined by

$$\Phi_a^{(n)} \equiv_\Phi \Phi_b^{(n)} \overset{\triangle}{\iff} \Phi_a^{(n)} \in \Theta(\Phi_b^{(n)})$$

is an abstract semantics. ⌟

▶ **Example 2.6** (Complexity clique)**.** *Complexity cliques* as defined by Asperti in [1] can be viewed as an abstract semantics $\langle \pi, \equiv_\pi \rangle$, that we will refer to as the complexity clique semantics. For each arity n and program index a let us define:

$$\pi_a^{(n)} \triangleq \lambda \vec{y}. \langle\!\langle \phi_a^{(n)}(\vec{y}), \Phi_a^{(n)}(\vec{y}) \rangle\!\rangle$$

where $\langle\!\langle \_, \_ \rangle\!\rangle : \mathbb{N}^2 \to \mathbb{N}$ is an effective bijective encoding for pairs and $\Phi : \mathbb{N}^2 \to \mathcal{C}$ is a Blum complexity. The equivalence $\equiv_\pi$ is defined as follows: for all $a, b, n \in \mathbb{N}$,

$$\pi_a^{(n)} \equiv_\pi \pi_b^{(n)} \overset{\triangle}{\iff} \phi_a^{(n)} = \phi_b^{(n)} \wedge \Phi_a^{(n)} \equiv_\Phi \Phi_b^{(n)}.$$ ⌟

Classical Rice's theorem states the undecidabilty of extensional program properties. Following [19], we parameterise extensional sets by means of a generic equivalence relation.

▶ **Definition 2.7** (∼-extensional set)**.** Let $\sim \subseteq \mathbb{N} \times \mathbb{N}$ be an equivalence relation between programs whose equivalence classes are denoted by $[a]_\sim$. A set of indices $A \subseteq \mathbb{N}$ is called:

- *∼-extensional* when for all $a, b \in \mathbb{N}$, if $a \in A$ and $a \sim b$ then $b \in A$;
- *partially ∼-extensional* when there exists $a \in \mathbb{N}$ such that $[a]_\sim \subseteq A$;
- *universally ∼-extensional* when for all $a \in \mathbb{N}$, $[a]_\sim \cap A \neq \varnothing$. ⌟

In words, a set A is $\sim$-extensional if A is a union of $\sim$-equivalence classes, partially $\sim$-extensional if A contains at least a whole $\sim$-equivalence class, and universally $\sim$-extensional if A contains at least an element from each $\sim$-equivalence class, i.e., its complement $\mathbb{N} \setminus A$ is not partially $\sim$-extensional. Notice that if A is not trivial (i.e., $A \neq \varnothing$ and $A \neq \mathbb{N}$) and $\sim$-extensional then A is partially $\sim$-extensional and not universally $\sim$-extensional. Let us observe that $\sim_R$-extensionality is the standard notion of extensionality so that classical Rice's theorem [25] states that if A is $\sim_R$-extensional and not trivial then A is not recursive.[1]

3 Fair and Strong smn Semantics

In this section, we identify some fundamental properties of abstract semantics that will be later used in our intensional computability results. A first basic property stems from the fundamental smn theorem and intuitively amounts to requiring that the operation of fixing some parameters of a program is effective and preserves its abstract semantics.

▶ **Definition 3.1** (Strong smn semantics). An abstract semantics $\langle \pi, \equiv_\pi \rangle$ has the *strong smn* (*ssmn*) *property* if, given $m, n \geq 1$, there exists a total computable function $s : \mathbb{N}^{m+2} \to \mathbb{N}$ such that for all $a, b \in \mathbb{N}$, $\vec{x} \in \mathbb{N}^m$:

$$\lambda \vec{y}.\pi_a^{(n+1)}(\phi_b^{(m)}(\vec{x}), \vec{y}) \equiv_\pi \pi^{(n)}_{s(a,b,\vec{x})}. \tag{1}$$

In such a case, the abstract semantics $\langle \pi, \equiv_\pi \rangle$ is called *strong smn*. ⌟

The above definition requires the property (1) which is slightly stronger than one would expect. The natural generalisation of the standard smn property, in the style, e.g., of [1], would amount to asking that, given $m, n \geq 1$, there exists a total computable function $s : \mathbb{N}^{m+1} \to \mathbb{N}$ such that for any program index $a \in \mathbb{N}$ and input $\vec{x} \in \mathbb{N}^m$, it holds $\lambda \vec{y}.\pi_a^{(m+n)}(\vec{x}, \vec{y}) \equiv_\pi \pi^{(n)}_{s(a,\vec{x})}$. The concrete semantics $\langle \phi, = \rangle$ of Example 2.3 clearly satisfies this ssmn property. In fact, the function $\lambda a, b, \vec{y}.\pi_a^{(n+1)}(\phi_b^{(m)}(\vec{x}), \vec{y})$ is computable (by composition, relying on the existence of universal functions), hence the existence of a total computable $s : \mathbb{N}^{m+2} \to \mathbb{N}$ such that $\lambda \vec{y}.\pi_a^{(n+1)}(\phi_b^{(m)}(\vec{x}), \vec{y}) \equiv_\pi \pi^{(n)}_{s(a,b,\vec{x})}$ holds, as prescribed by Definition 3.1, follows by the standard smn theorem. It is easily seen that the same applies to the domain semantics of Example 2.4.

The reason for the stronger requirement (1) in Definition 3.1 is that, to deal with generic abstract semantics, thus going beyond asymptotic complexity, a suitable smn definition needs to embody a condition on program composition (of a and b in Definition 3.1). Indeed, if we consider the semantics based on program complexity (i.e., Examples 2.5 and 2.6), it turns out that whenever they enjoy the smn property in [1, Definition 11] and, additionally, they satisfy the linear time composition hypothesis in [1, Section 4] relating the asymptotic complexities of a program composition to those of its components, then they are ssmn semantics according to Definition 3.1. More details on the relationship with Asperti's approach [1] will be given later in Section 6.1.

Note that for an ssmn abstract semantics $\langle \pi, \equiv_\pi \rangle$, there always exists a program whose denotation is equivalent to the always undefined function, namely,

$$\text{for any arity } n \in \mathbb{N} \text{ there exists a program index } e_0 \in \mathbb{N} \text{ such that } \pi^{(n)}_{e_0} \equiv_\pi \lambda \vec{y}.\uparrow. \tag{2}$$

[1] In [19], the term "extensional" is replaced by "compatible" when one refers to generic equivalence relations $\sim$.

In fact, if b is a program index for the always undefined function $\lambda\vec{y}.\uparrow$ then, by (1), we have that $\lambda\vec{y}.\pi_0^{(n+1)}(\phi_b(0),\vec{y}) = \lambda\vec{y}.\uparrow \equiv_\pi \pi^{(n)}_{s(0,b,0)}$, so that we can pick $e_0 \triangleq s(0,b,0)$.

It is also worth exhibiting an example of abstract semantics which is not ssmn. Let $\pi_a(\vec{x})$ be defined as the number of different variables accessed in a computation of the program a on the input $\vec{x}$. Then, let us observe that the mere fact that π_a is always a total function trivially makes the abstract semantics $\langle \pi, = \rangle$ non-ssmn.

To generalise Kleene's second recursion theorem, besides the ssmn property, we need to postulate the existence of so-called *fair universal programs*, namely, programs that can simulate every other program w.r.t. a given abstract semantics. This generalises the analogous notion in [1, Definition 26], where this simulation is specific to complexity cliques and must preserve both the computed function and its asymptotic complexity.

▶ **Definition 3.2** (Fair semantics). An index $u \in \mathbb{N}$ is a *fair universal program* for an abstract semantics $\langle \pi, \equiv_\pi \rangle$ and an arity $n \in \mathbb{N}$ if for all $a \in \mathbb{N}$:

$$\pi_a^{(n)} \equiv_\pi \lambda\vec{y}.\pi_u^{(n+1)}(a,\vec{y}).$$

An abstract semantics is *fair* if it admits a fair universal program for every arity. ⌟

Clearly, the concrete (Example 2.3) and domain (Example 2.4) semantics are fair. In general, as noted in [1], the existence of a fair universal program may not only depend on the reference abstract semantics, but also on the underlying computational model. For instance, when considering program complexity, as argued by Asperti [1, Section 6] by relying on some remarks by Blum [3], multi-tape Turing machines seem not to admit fair universal programs. By contrast, single tape Turing machines do have fair universal programs, despite the fact that this is commonly considered a folklore fact and cannot be properly quoted. Hereafter, when referring to the complexity-based semantics of Examples 2.5 and 2.6, we will implicitly use that they are ssmn and fair semantics.

4 Kleene's Second Recursion Theorem and Rice's Theorem

In this section, we show how some foundational results of computability theory can be extended to a general abstract semantics. The first approach relies on a generalisation of Kleene's second recursion theorem, which is then used to derive a corresponding Rice's theorem. A second approach consists in identifying conditions that ensure the existence of an intricated switching family in the sense of [19], from which Rice's theorem also follows.

4.1 Kleene's Second Recursion Theorem

We show that Kleene's second recursion theorem holds for any fair ssmn abstract semantics. This generalises the analogous result proved by Asperti [1, Section 5] for complexity cliques.

▶ **Theorem 4.1** (Intensional Second Recursion Theorem). *Let $\langle \pi, \equiv_\pi \rangle$ be a fair ssmn abstract semantics. For any total computable function $h : \mathbb{N} \to \mathbb{N}$ and arity $n \in \mathbb{N}$, there exists an index $a \in \mathbb{N}$ such that $a \sim^n_\pi h(a)$.*

As an example, this result, instantiated to the complexity semantics of Example 2.5, entails the impossibility of designing a program transform that modifies the asymptotic complexity of every program, even without preserving its input-output behavior.

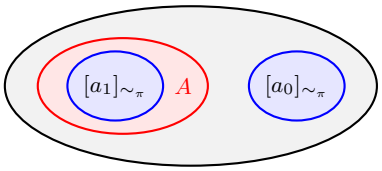


Figure 1 A graphical representation of Theorem 4.3.

▶ **Example 4.2** (Fixpoints of Blum complexity semantics)**.** Let $\langle \Phi, \equiv_\Phi \rangle$ be the Blum complexity semantics of Example 2.5. A program transform $h : \mathbb{N} \to \mathbb{N}$ is a total computable function which maps indices of programs into indices of transformed programs. By applying Theorem 4.1, for any arity $n \in \mathbb{N}$, we know that there exists an index of a program a such that $a \sim_\pi^n h(a)$ holds, so that the program transform h necessarily does not alter the asymptotic complexity of, at least, the program a. ⌟

This second recursion theorem allows us to obtain an intensional version of Rice's theorem for fair and ssmn abstract semantics. Inspired by [19], we generalise the statement to cover partially extensional properties.

▶ **Theorem 4.3** (Rice by fair and ssmn semantics)**.** *Let $\langle \pi, \equiv_\pi \rangle$ be a fair and ssmn semantics. If $A \subseteq \mathbb{N}$ is partially $\sim_\pi^n$-extensional and not universally $\sim_\pi^n$-extensional, for some arity $n \in \mathbb{N}$, then A is not recursive.*

Fig. 1 provides a graphical representation of this result: if we can find two program indices $a_0, a_1 \in \mathbb{N}$ such that A overapproximates the $\equiv_\pi$-equivalence class $[a_1]_{\sim_\pi}$ and A does not intersect $[a_0]_{\sim_\pi}$, then A cannot be recursive. For example, as observed in Section 3, the asymptotic complexity on a suitable computational model such as single tape Turing machines is a fair ssmn semantics, so that Theorem 4.3 applies. Let us illustrate some further applications of Theorem 4.3.

▶ **Example 4.4** (Halting set)**.** Let $\langle \phi, \equiv_\mathbb{N} \rangle$ be the domain semantics of Example 2.4 with $S = \mathbb{N}$, hence $f \equiv_\mathbb{N} g$ when $\mathrm{dom}(f) = \mathrm{dom}(g)$. The halting set $K \triangleq \{a \in \mathbb{N} \mid \phi_a(a)\!\downarrow\}$ can be proved to be non-recursive by resorting to Theorem 4.3 for $\langle \phi, \equiv_\mathbb{N} \rangle$. Let $e_0, e_1 \in \mathbb{N}$ be such that $\phi_{e_0} = \lambda x.\uparrow$ and $\phi_{e_1} = \lambda x.1$. Since $[e_1]_{\equiv_\mathbb{N}}$ is the set of programs that compute total functions, we have that $[e_1]_{\equiv_\mathbb{N}} \subseteq K$. Moreover, $[e_0]_{\equiv_\mathbb{N}}$ is the set of nonterminating programs for any input, so that $[e_0]_{\equiv_\mathbb{N}} \cap K = \varnothing$. This means that $\langle \phi, \equiv_\mathbb{N} \rangle$ satisfies the hypotheses of Theorem 4.3, thus entailing that K is not recursive. ⌟

▶ **Example 4.5** (Complexity sets)**.** Let $\langle \phi, = \rangle$, $\langle \Phi, \equiv_\Phi \rangle$ be, resp., the semantics of Examples 2.3 and 2.5. Let $sort : \mathbb{N} \to \mathbb{N}$ be a total function that takes as input an encoded sequence of numbers and outputs the encoding of the corresponding sorted sequence. It turns out that by applying Theorem 4.3, the following sets can be proved to be non-recursive:
(1) $A \triangleq \{a \mid \Phi_a \in \Theta(n \log n) \wedge \phi_a = sort\}$,
(2) $B \triangleq \{a \mid \Phi_a \in \mathcal{O}(n \log n)\}$,
(3) $C \triangleq \{a \mid \Phi_a \in \Omega(n \log n)\}$.
Let *is*, *ms* be different implementations of *sort*, i.e., $\phi_{is} = \phi_{ms} = sort$, such that $\Phi_{is} \in \Theta(n^2)$ and $\Phi_{ms} \in \Theta(n \log n)$ – *is* and *ms* could be, resp., insertion and merge sort. Recall that $\sim_R$ denotes the Rice equivalence induced by $\langle \phi, = \rangle$ (i.e., $a \sim_R b \Leftrightarrow \phi_a = \phi_b$), and, in turn, let $\sim_{\Phi R} = \sim_\Phi \cap \sim_R$ be the equivalence induced by the complexity clique semantics of Example 2.6, which is a fair ssmn semantics. Then, we have that:

(1) since $[is]_{\sim_{\Phi R}} \cap A = \varnothing$ and $[ms]_{\sim_{\Phi R}} \subseteq A$, by Theorem 4.3 A is non-recursive;
(2) since $[is]_{\sim_{\Phi}} \cap B = \varnothing$ and $[ms]_{\sim_{\Phi}} \subseteq B$, by Theorem 4.3 B is non-recursive;
(3) let e be any program index such that $\Phi_e \in \Theta(1)$. Since $[e]_{\sim_{\Phi}} \cap C = \varnothing$ and $[is]_{\sim_{\Phi}} \subseteq C$, by Theorem 4.3, the set C is non-recursive. ⌟

It is worth remarking that in Example 4.5, $n \log n$ could be replaced by any function, thus showing the undecidability of the asymptotic complexities "big O" (case (2)) and "big Omega" (case (3)). Let us also point out that Example 4.4 shows how easily the halting set K can be proved to be non-recursive by applying Theorem 4.3.

4.2 Branching Semantics

Let us investigate the connection between our results and the key notion of intricated switching family used by Moyen and Simonsen [19] for proving their intensional version of Rice's theorem. Firstly, we argue that every ssmn abstract semantics admits an intricated switching family whenever it is able to express a suitable form of *conditional branching*. This allows us to derive an intensional Rice's theorem. Moreover, we show that for fair and ssmn semantics, the identity can always play the role of intricated switching family.

▶ **Definition 4.6** (Branching and discharging semantics)**.** An abstract semantics $\langle \pi, \equiv_\pi \rangle$ is *branching* if, given $n \geq 1$, there exists a total computable function $r : \mathbb{N}^4 \to \mathbb{N}$ such that $\forall a, b, c_1, c_2, x \in \mathbb{N}$ such that $c_1 \neq c_2$:

$$\lambda \vec{y}.\pi^{(n)}_{r(a,b,c_1,c_2)}(x, \vec{y}) \equiv_\pi \begin{cases} \lambda \vec{y}.\pi^{(n)}_a(x, \vec{y}) & \text{if } x = c_1 \\ \lambda \vec{y}.\pi^{(n)}_b(x, \vec{y}) & \text{if } x = c_2 \\ \lambda \vec{y}.\uparrow & \text{otherwise} \end{cases}$$

Moreover, $\langle \pi, \equiv_\pi \rangle$ is (variable) *discharging* if, for all $n \geq 1$, there exists a total computable function $t : \mathbb{N} \to \mathbb{N}$ such that for all $a, x \in \mathbb{N}$:

$$\pi^{(n)}_a \equiv_\pi \lambda \vec{y}.\pi^{(n+1)}_{t(a)}(x, \vec{y}).$$ ⌟

Hence, intuitively, an abstract semantics is branching when it is able to model the branching structure of conditional statements with multiple positive guards, while the property of being variable discharging holds when one can freely add fresh and unused variables without altering the abstract semantics. Let us recall the notion of intricated switching family from [19, Definition 5].[2]

▶ **Definition 4.7** (Intricated switching family [19, Definition 5])**.** Let $\sim \subseteq \mathbb{N} \times \mathbb{N}$ be an equivalence relation on program indices. An *intricated switching family* (ISF) w.r.t. $\sim$ is an indexed set of total computable functions $\{\sigma_{a,b}\}_{a,b \in \mathbb{N}}$, with $\sigma_{a,b} : \mathbb{N} \to \mathbb{N}$, such that for all $a, b \in \mathbb{N}$, the sets $A_{a,b} = \{x \in \mathbb{N} \mid \sigma_{a,b}(x) \sim a\}$ and $B_{a,b} = \{x \in \mathbb{N} \mid \sigma_{a,b}(x) \sim b\}$ are recursively inseparable (i.e., no recursive C exists such that $A_{a,b} \subseteq C$ and $C \cap B_{a,b} = \varnothing$). ⌟

Moyen and Simonsen [19, Theorem 3] show that if an equivalence $\sim$ admits an ISF, then every partially $\sim$-extensional and not universally $\sim$-extensional set is not recursive. A simplified version of their intensional result, tailored for our setting, can be stated as follows.

[2] For the sake of simplicity, [19, Definition 5] is here instantiated to the case of recursive sets.

▶ **Theorem 4.8** ([19, Theorem 3]). *Let $\sim \subseteq \mathbb{N} \times \mathbb{N}$ be an equivalence relation. If $A \subseteq \mathbb{N}$ is partially $\sim$-extensional, not universally $\sim$-extensional and there exists an ISF w.r.t. $\sim$ then A is not recursive.*

Branching semantics allow us to derive the following intensional version of Rice's Theorem.

▶ **Theorem 4.9** (Rice by branching, discharging and ssmn semantics). *Let $\langle \pi, \equiv_\pi \rangle$ be a branching, discharging and ssmn semantics. If $A \subseteq \mathbb{N}$ is partially $\sim_\pi^n$-extensional and not universally $\sim_\pi^n$-extensional for some arity $n \in \mathbb{N}$, then A is not recursive.*

Let us discuss more in detail the relationship with the approach in [19]. Firstly, it turns out that a fair ssmn semantics always admits a canonical ISF, namely, the identity $\textsc{Id} \triangleq \{(\lambda x.x)_{a,b}\}_{a,b\in\mathbb{N}}$.

▶ **Proposition 4.10.** *Let $\langle \pi, \equiv_\pi \rangle$ be a fair and ssmn semantics. Then, the identity* Id *is an ISF w.r.t. $\sim_\pi^n$, for all $n \geq 1$.*

Let us point out that the identity function has not been exploited in [19], that instead focuses on the standard switching family. It turns out that the identity function plays a key role as ISF.

▶ **Proposition 4.11.** *Let $\sim \subseteq \mathbb{N} \times \mathbb{N}$ be an equivalence relation. The following statements are equivalent:*

(1) *Every set $A \subseteq \mathbb{N}$ partially $\sim$-extensional and not universally $\sim$-extensional is non-recursive.*

(2) *The identity* Id *is an ISF w.r.t. $\sim$.*

(3) *There exists an ISF w.r.t. $\sim$.*

Therefore, the above result roughly states that the identity function is the "canonical" ISF, meaning that if an ISF exists, then Id is an ISF as well. Moreover, the intensional Rice's Theorem 4.8 of [19] provides a sufficient condition (i.e., the existence of an ISF) for a partially and not universally extensional set to be undecidable. Proposition 4.11 enhances Theorem 4.8 by showing that such a sufficient condition is necessary as well, or, equivalently, that a partially and not universally extensional set is undecidable iff there exists an ISF.

We conclude this section by discussing an alternative notion of branching, which requires the preservation of a full conditional statement with positive and negative guards. This is an adaptation to our framework of a property that would be needed to exploit a so-called standard switching family as defined in [19, Example 1].

▶ **Definition 4.12** (Strongly branching semantics). An abstract semantics $\langle \pi, \equiv_\pi \rangle$ is *strongly branching* if, given $n \geq 1$, there exists a total computable function $r : \mathbb{N}^3 \to \mathbb{N}$ such that for all $a, b, c, x \in \mathbb{N}$:

$$\lambda\vec{y}.\pi^{(n)}_{r(a,b,c)}(x,\vec{y}) \equiv_\pi \begin{cases} \lambda\vec{y}.\pi^{(n)}_a(x,\vec{y}) & \text{if } x = c \\ \lambda\vec{y}.\pi^{(n)}_b(x,\vec{y}) & \text{otherwise} \end{cases}$$ ⌟

Despite appearing to be more natural, the preservation of conditionals with positive and negative conditions is a stronger requirement than the one we considered in Definition 4.6. Indeed, it turns out that every ssmn and strongly branching semantics is a branching semantics.

▶ **Proposition 4.13** (Strongly branching implies branching). *If $\langle \pi, \equiv_\pi \rangle$ is an ssmn and strongly branching semantics, then $\langle \pi, \equiv_\pi \rangle$ is branching.*

4.3 An Application to Static Program Verifiers

We adapt the general definition of static program verifier of Cousot et al. [7, Definition 4.3] to our framework. Given a program property $P \subseteq \mathbb{N}$ to check, a static program verifier is a total recursive function $\mathcal{V} : \mathbb{N} \to \{0, 1\}$, which is *sound* when for all $p \in \mathbb{N}$, $\mathcal{V}(p) = 1 \Rightarrow p \in P$, while $\mathcal{V}$ is *precise* if the reverse implication also holds, i.e., when $\mathcal{V}(p) = 1 \Leftrightarrow p \in P$ holds. Informally, soundness guarantees that only false negatives are allowed, i.e., $\mathbb{N} \setminus P$ is merely a subset of $\{p \in \mathbb{N} : \mathcal{V}(p) = 0\}$, while precise verifiers output true positives and true negatives only (i.e., they decide P).

Classical Rice's theorem clearly entails the impossibility of designing a precise verifier for a nontrivial extensional property. However, one may wonder whether there exist sound verifiers with "few" false negatives. By applying our intensional Theorem 4.3, we are able to show that sound but imprecise verifiers necessarily have at least one false negative for each equivalence class of programs, even for intensional properties.

▶ **Example 4.14** (Constant value verifier)**.** Assume we are interested in checking if a program can output a given constant value, for instance, zero with the aim of statically detecting division-by-zero bugs. Let $\mathcal{V}$ be a sound static verifier for the set $P_{=0} \triangleq \{p \in \mathbb{N} \mid 0 \in \mathrm{rng}(\phi_p)\}$ of programs that output zero for some input. The set $N \triangleq \{p \in \mathbb{N} \mid \mathcal{V}(p) = 0\}$ is recursive since $\mathcal{V}$ is assumed to be a total computable function. By soundness of $\mathcal{V}$, we have that $\mathbb{N} \setminus P_{=0} \subseteq N$, so that N includes, for example, the programs computing the constant function $\lambda x.1$. Therefore, N is partially extensional, and, by Theorem 4.3, N has to be universally extensional. This means that for any computable function $f \in \mathcal{C}$ there exists a program $p \in \mathbb{N}$ that computes f such that $\mathcal{V}(p) = 0$. Thus, when $0 \in \mathrm{rng}(f)$ holds (e.g., for $f = \lambda x.0$), $\mathcal{V}$ necessarily outputs a false negative for p. Hence, $\mathcal{V}$ outputs infinitely many false negatives. ⌟

▶ **Example 4.15** (Complexity verifier)**.** Consider a speculative sound static verifier $\mathcal{V}$ for recognizing programs that meet some lower bound, for instance, programs having a cubic lower bound $P_{\Omega(n^3)} \triangleq \{p \in \mathbb{N} \mid \Phi_p = \Omega(n^3)\}$. Thus, $N \triangleq \{p \in \mathbb{N} \mid \mathcal{V}(p) = 0\}$ has to be recursive and if $\sim_\Phi$ is the program equivalence induced by the Blum complexity semantics $\langle \Phi, \equiv_\Phi \rangle$ of Example 2.5 then, by soundness of $\mathcal{V}$, we have, for example, $\{p \in \mathbb{N} \mid \Phi_p = \Theta(1)\} \subseteq N$. This means that N is partially $\sim_\Phi$-extensional and, by Theorem 4.3, N is universally extensional, namely, $\mathcal{V}$ will output 0 for at least a program in each Blum complexity class. For instance, even some programs with an exponential lower bound will be wrongly classified by $\mathcal{V}$ as programs that do not meet a cubic lower bound. ⌟

As shown by Cousot et al. [7, Theorem 5.4], precise static verifiers cannot be designed (unless for trivial program properties). The examples above prove that, additionally, we cannot have any certain information on an input program p whenever the output of a sound (and imprecise) verifier for p is 0. In fact, when this happens, p could compute any partial function (cf. Example 4.14) or have any complexity (cf. Example 4.15).

5 On the Decidability of Affine Program Invariants

Karr's abstract domain [13] consisting of affine equalities between program variables, such as $2x - 3y = 1$, is well known and widely used in static program analysis [18, 26]. Karr [13] put forward an algorithm that infers for each program point q of a control flow graph modelling an affine program P (i.e., an unguarded program with non-deterministic branching and affine assignments) a set of affine equalities that hold among the variables of P when the control reaches q, namely, an *affine invariant* for P. Müller-Olm and Seidl [20] show that

Karr's algorithm actually computes the strongest affine invariant for affine programs (this result has been extended to a slightly larger class of affine programs in [23, Theorem 5.1]). Moreover, they design a more efficient algorithm implementing this static analysis and they extend in [21] the algorithm for computing bounded polynomial invariants, i.e., the strongest polynomial equalities of degree at most a given $d \in \mathbb{N}$. Later, Hrushovski et al. [11] put forward a sophisticated algorithm for computing the strongest unbounded polynomial invariants of affine programs, by relying on the Zariski closure of semigroups.

On the impossibility side, Müller-Olm and Seidl [20, Section 7] prove that for affine programs allowing positive affine guards it is undecidable whether a given nontrivial affine equality holds at a given program point or not. In practical applications, static analyses on Karr's domain of guarded affine programs ignore non-affine Boolean guards, while for an affine guard b, the current affine invariant i is propagated through the positive branch of b by the intersection $i \cap b$, that remains an affine subspace. By the aforementioned undecidability result [20, Section 7], this latter analysis algorithm for guarded affine programs turns out to be sound but necessarily imprecise, thus inferring affine invariants which are not the strongest ones.

Müller-Olm and Seidl [20, Section 7] prove their undecidability result by exploiting an acute reduction to the undecidable Post correspondence problem, inspired by early reductions explored in data flow analysis [9, 12]. In this section, we show that our Theorem 4.9 allows us to derive and extend this undecidability result by exploiting an orthogonal intensional approach. More precisely, we prove that any nontrivial (and not necessarily affine) relation on the states of control flow graphs of programs allowing: (1) zero, variable and successor assignments, resp., $x := 0$, $x := y$ and $x := y + 1$, and (2) positive equality guards $x = y?$ and $x = v?$, turns out to be undecidable. Since these control flow graphs form a subclass of affine programs with positive affine guards, the undecidability result of Müller-Olm and Seidl [20, Section 7] is retrieved as a consequence.

We consider control flow graphs that consist of program points connected by edges labeled by assignments and guards. Variables are denoted by x_i, with $i \in \mathbb{N}$, and store values ranging in $\mathbb{N}$, while Karr's abstract domain is designed for variables assuming values in $\mathbb{Q}$. Clearly, from a computability perspective, this is not a restriction simply by considering a computable bijection between $\mathbb{N}$ and $\mathbb{Q}$.

▶ **Definition 5.1** (Basic affine control flow graph)**.** A *basic affine control flow graph* (BACFG) is a tuple $G = (N, E, s, e)$, where N is a finite set of nodes, $s, e \in N$ are the start and end nodes, and $E \subseteq N \times \mathrm{Com} \times N$ is a set of labelled edges, where the set Com of commands consists of assignments of type $x_n := 0$, $x_n := x_m$, $x_n := x_m + 1$, and equality guards of type $x_n = x_m?$, $x_n = v?$, with $v \in \mathbb{N}$. ⌟

Let us remark that BACFGs only include basic affine assignments and positive affine guards, in particular inequality checks such as $x_n \neq x_m?$ and $x_n \neq v?$ are not allowed. Thus, BACFGs are a subclass of affine programs with positive affine guards.

As in dataflow analysis and abstract interpretation [6, 9, 26], BACFGs have a *collecting semantics* where, given a set of input states In, each program point is associated with the set of states that occur in some program execution from some state in In. A finite number of variables may occur in a BACFG, so that a state of a BACFG G is a tuple $(x_1, \ldots, x_k) \in \mathbb{N}^k$, where k is the maximum variable index occuring in G and $k = 0$ is a degenerate case for trivial BACFGs with $\mathbb{N}^0 = \{\bullet\}$. The *collecting transfer function* $f_{(\cdot)}(\cdot) : \mathrm{Com} \to \wp(\mathbb{N}^k) \to \wp(\mathbb{N}^k)$ for $k \in \mathbb{N}$ variables and with $n, m \in [1, k]$ is defined as follows:

$$
\begin{aligned}
f_{x_n:=0}(S) &\triangleq \{(x_1,\ldots,x_{n-1},0,x_{n+1},\ldots,x_k) \mid \vec{x} \in S\},\\
f_{x_n:=x_m}(S) &\triangleq \{(x_1,\ldots,x_{n-1},x_m,x_{n+1},\ldots,x_k) \mid \vec{x} \in S\},\\
f_{x_n:=x_m+1}(S) &\triangleq \{(x_1,\ldots,x_{n-1},x_m+1,x_{n+1},\ldots,x_k) \mid \vec{x} \in S\},\\
f_{x_n=v?}(S) &\triangleq \{\vec{x} \in S \mid x_n = v\},\\
f_{x_n=x_m?}(S) &\triangleq \{\vec{x} \in S \mid x_n = x_m\}.
\end{aligned}
$$

A no-op ϵ command is a syntactic sugar for $x_1 := x_1$, i.e., $f_\epsilon \triangleq f_{x_1:=x_1} = \lambda S.S$. Given $k, k' \in \mathbb{N}$ and $S \in \wp(\mathbb{N}^{k'})$, the projection $S\restriction_k \in \wp(\mathbb{N}^k)$ is defined as follows:

$$
S\restriction_k \triangleq \begin{cases} S \times \mathbb{N}^{k-k'} & \text{if } 0 \leq k' < k \\ S & \text{if } k' = k \\ \{(x_1,\ldots,x_k) \mid \vec{x} \in S\} & \text{if } k < k' \end{cases}
$$

▶ **Definition 5.2** (Collecting semantics of BACFGs)**.** Given a BACFG $G = (N, E, s, e)$ with $k \in \mathbb{N}$ variables and a set of input states $S \subseteq \mathbb{N}^{k'}$, with $k' \leq k$, the *collecting semantics* $[\![G]\!]_S : N \to \wp(\mathbb{N}^k)$ is the least, w.r.t. pointwise set inclusion, solution in $\wp(\mathbb{N}^k)^{|N|}$ of the following system of constraints:

$$
\begin{cases} [\![G]\!]_S[s] \supseteq S\restriction_k & \text{for the start node } s \\ [\![G]\!]_S[v] \supseteq f_c([\![G]\!]_S[u]) & \text{for each edge } (u,c,v) \in E \end{cases}
$$

⌟

Let us observe that, since the collecting transfer functions f_c are additive on the complete lattice $\langle \wp(\mathbb{N}^k), \subseteq \rangle$, by Knaster-Tarski fixpoint theorem, $[\![G]\!]_S$ is well defined. For $\vec{x} \in \mathbb{N}^{k'}$, we write $[\![G]\!]_{\vec{x}}$ instead of $[\![G]\!]_{\{\vec{x}\}}$. Notice that $[\![G]\!]_{(\cdot)}$ is an additive function, so that, for any program point $u \in N$, $[\![G]\!]_S[u] = \bigcup_{\vec{x} \in S} [\![G]\!]_{\vec{x}}[u]$ holds.

5.1 Turing Completeness of BACFGs

Let us recall that an ssmn abstract semantics needs an underlying Turing complete concrete semantics of programs (cf. Assumption 2.1). A crucial observation is that any URM (Unlimited Register Machine[3]) program, provided with suitable operational semantics, can be simulated by a BACFG, that is, BACFGs turn out to be Turing complete despite not including full (both positive and negative) Boolean tests.

▶ **Theorem 5.3** (Turing completeness of BACFGs)**.** BACFGs *are a Turing complete computational model.*

It is worth providing an intuition of the proof of Theorem 5.3. First, we point out that all four types of instructions of URMs, namely, using the definition and notation of [8],

- $z(n)$: sets register r_n to 0 ($r_n \leftarrow 0$) and transfers the control to the next instruction;
- $s(n)$: increments register r_n by 1 ($r_n \leftarrow r_n + 1$) and transfers the control to the next instruction;
- $t(m,n)$: sets register r_n to r_m ($r_n \leftarrow r_m$) and transfers the control to the next instruction;
- $j(m,n,p)$: if $r_m = r_n$ and I_p is a proper instruction, then it jumps to the instruction I_p; otherwise, it skips to the next instruction;

[3] Recall that URMs are a Turing complete computational model [8].

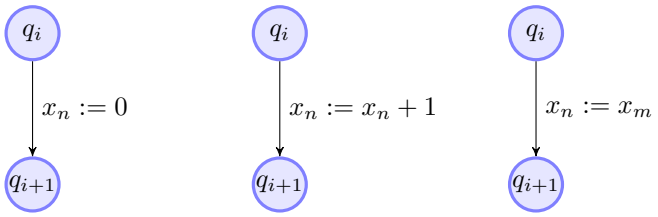


Figure 2 BACFGs simulating: $z(n)$ (left), $s(n)$ (center), $t(m, n)$ (right).

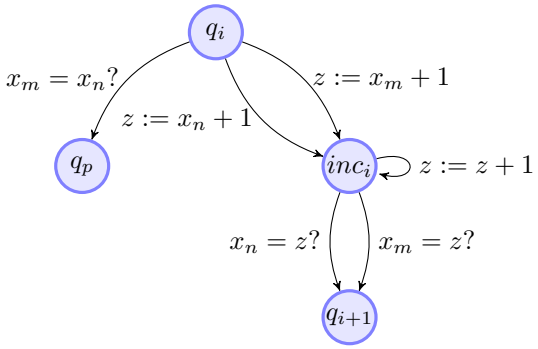


Figure 3 BACFG simulating a jump instruction $j(m, n, p)$.

can be simulated by the BACFGs depicted in Figures 2 and 3. While the BACFGs in Figure 2 are trivial, let us describe more in detail how to simulate a jump instruction by the BACFG in Figure 3. Intuitively, a difficulty arises for simulating the negative branch $x_n \neq x_m?$. Here, the BACFG at node q_i initialises a fresh unused variable z with both $x_n + 1$ and $x_m + 1$ and transfers the control to a node inc_i where z is incremented infinitely many times. Thus, in the least fixpoint solution, at node inc_i the variable z stores any value $v > \min(x_m, x_n)$, including $z = \max(x_m, x_n)$. Suppose now that $x_n > x_m$ holds: in this case, the guard $x_n = z?$ between nodes inc_i and q_{i+1} eventually will be made true and at the node q_{i+1} the store will retain the original values of all variables (x_m and x_n included), except for the new variable z which will be ignored by the remaining nodes. The case $x_m > x_n$ is analogous. Therefore, it turns out that the node q_{i+1} will be reached if and only if $x_m \neq x_n$ holds, while q_p will be reached if and only if $x_m = x_n$ holds, thus providing a simulation for the jump instruction $j(m, n, p)$.

5.2 Concrete and Abstract Semantics

One key insight is that the concrete semantics is defined on the URM programs that satisfy the Assumption 2.1 of Turing completeness, while the abstract semantics is defined on BACFGs. Let us consider two Gödel numberings for BACFGs and URMs, so that for an index $a \in \mathbb{N}$, G_a and RM_a denote, resp., the a-th BACFG and URM programs. The concrete semantics $\langle \phi, = \rangle$ of URMs, for an index $a \in \mathbb{N}$ and an arity $n \in \mathbb{N}$, is defined as follows: for all $\vec{x} \in \mathbb{N}^n$,

$$\phi_a^{(n)}(\vec{x}) \triangleq \begin{cases} y & \text{if } RM_a \text{ on input } \vec{x} \text{ halts with value } y \text{ on its first register,} \\ \uparrow & \text{otherwise.} \end{cases}$$

On the other hand, the abstract semantics of BACFGs is as follows.

▶ **Definition 5.4** (State semantics of BACFGs)**.** Let $Q \subseteq \wp(\mathbb{N}^t)$ be a predicate on sets of states with $t \in \mathbb{N}$ variables. The *state semantics* $\langle Q, = \rangle$ of BACFGs, for any index $a \in \mathbb{N}$ and arity $n \in \mathbb{N}$, is given by the function $Q_a^{(n)} : \mathbb{N}^n \to \{0, 1\}$ defined as follows: for all $\vec{x} \in \mathbb{N}^n$,

$$Q_a^{(n)}(\vec{x}) \triangleq \begin{cases} 1 & \text{if } [\![G_a]\!]_{\vec{x}}[e_a] \neq \varnothing \wedge [\![G_a]\!]_{\vec{x}}[e_a] \restriction_t \in Q \\ 0 & \text{if } [\![G_a]\!]_{\vec{x}}[e_a] \neq \varnothing \wedge [\![G_a]\!]_{\vec{x}}[e_a] \restriction_t \notin Q \\ \uparrow & \text{if } [\![G_a]\!]_{\vec{x}}[e_a] = \varnothing \end{cases}$$

where e_a is the end node of the a-th BACFG G_a. ⌟

Predicates of type $Q \subseteq \wp(\mathbb{N}^t)$ are also known as hyperproperties [5] and the state semantics of Definition 5.4 models the validity of a given predicate Q at the end node of a BACFG. Note that it is not restrictive to consider the end node, since this can be arbitrarily chosen in a BACFG.

▶ **Theorem 5.5.** *The state semantics of Definition 5.4 is ssmn, branching and discharging.*

Let us now consider a state semantics $\langle Q, = \rangle$ for some predicate $Q \subseteq \wp(\mathbb{N}^t)$. For all $n \geq 1$, let us define two sets $A^{\forall Q}$ and $A^{\exists Q}$, by distinguishing two cases depending on whether Q includes the empty set, that models nontermination, or not:

(1) if $\varnothing \notin Q$ then $A^{\forall Q} \triangleq \{a \in \mathbb{N} \mid \forall \vec{y}.\, Q_a^{(n)}(\vec{y}) = 1\}$ and $A^{\exists Q} \triangleq \{a \in \mathbb{N} \mid \exists \vec{y}.\, Q_a^{(n)}(\vec{y}) = 1\}$;
(2) if $\varnothing \in Q$ then $A^{\forall Q} \triangleq \{a \in \mathbb{N} \mid \forall \vec{y}.\, Q_a^{(n)}(\vec{y}) \in \{1, \uparrow\}\}$ and $A^{\exists Q} \triangleq \{a \in \mathbb{N} \mid \exists \vec{y}.\, Q_a^{(n)}(\vec{y}) \in \{1, \uparrow\}\}$.

Hence, $A^{\forall Q}$ ($A^{\exists Q}$) is the set of BACFGs such that Q holds at e_a for any (some) input state. It turns out that if $A^{\forall Q}$ is nontrivial then it is not recursive. Indeed, observe that $A^{\forall Q}$ is $\sim_Q$-extensional, so that Theorem 5.5 enables applying Theorem 4.9 to $\langle Q, = \rangle$. The same argument applies to the existential version $A^{\exists Q}$. We have therefore the following consequence.

▶ **Corollary 5.6.** *If Q is not trivial then $A^{\forall Q}$ and $A^{\exists Q}$ are not recursive.*

Corollary 5.6 means that we cannot decide if a nontrivial predicate Q holds at a given program point of a BACFG for all input states, neither whether there exists an input state that will make Q true. It is worth remarking that the predicates Q are arbitrary and include, but are not limited to, relational predicates between program variables such as affine equalities of Karr's abstract domain. Let us define some noteworthy examples of predicates:

(1) Given a set of affine equalities $\mathit{aff} = \{\vec{a_j} \cdot \vec{x} = b_j\}_{j=1}^m$, with $\vec{a_j} \in \mathbb{Z}^t$ and $b_j \in \mathbb{Z}$, $Q_{\mathit{aff}} \triangleq \{S \in \wp(\mathbb{N}^t) \mid \forall \vec{v} \in S. \forall j \in [1, m].\, \vec{a_j} \cdot \vec{v} = b_j\}$;
(2) Given $i \in [1, t]$ and $c \in \mathbb{N}$, $Q_{=c} \triangleq \{S \in \wp(\mathbb{N}^t) \mid \exists \vec{v} \in S.\ v_i = c\}$;
(3) Given a size $k \in \mathbb{N}$, $Q_{\mathrm{fin}_k} \triangleq \{S \in \wp(\mathbb{N}^t) \mid |S| = k\}$ and $Q_{\mathrm{fin}} \triangleq \cup_{k \in \mathbb{N}} Q_{\mathrm{fin}_k}$.

Therefore, Corollary 5.6 for $A^{\forall Q_{\mathit{aff}}}$ entails the undecidability result of Müller-Olm and Seidl [20, Section 7] discussed above. The predicate $Q_{=c}$ can be used to derive the undecidability of checking if some variable x_i may store a given constant c for affine programs with positive affine guards, e.g., for $c = 0$ this amounts to the undecidability of detecting division-by-zero bugs. Finally, with Q_{fin_0} we obtain the undecidability of dead code elimination, Q_{fin_1} entails the well-known undecidability of constant detection [9, 24], while the existential predicate Q_{fin} encodes whether some program point may only have finitely many different states.

6 Discussion of Related Work

In this section we discuss in detail the relation with some of Asperti's results [1] and with Rogers' systems of indices [27, 28].

6.1 Relation with Asperti's Approach

We show that our ssmn property in Definition 3.1 is a generalisation of the smn property in Asperti's approach [1], in a way that the Kleene's second recursion theorem and Rice's theorem for complexity cliques in [1] arise as instances of the corresponding results in our approach. Let us first recall and elaborate on the axioms for the complexity of function composition studied by Lischke [15, 16, 17] and assumed in [1, Section 4].

▶ **Definition 6.1** (Linear time and space complexity composition)**.** Consider a given concrete semantics ϕ and a Blum complexity Φ. The pair $\langle \phi, \Phi \rangle$ has the *linear time composition* property if there exists a total computable function $h : \mathbb{N}^2 \to \mathbb{N}$ such that for all $i, j \in \mathbb{N}$:
(1) $\phi_{h(i,j)} = \phi_i \circ \phi_j$,
(2) $\Phi_{h(i,j)} \in \Theta(\Phi_i \circ \phi_j + \Phi_j)$.
If (2) is replaced by
(2′) $\Phi_{h(i,j)} \in \Theta(\max\{\Phi_i \circ \phi_j, \Phi_j\})$
then $\langle \phi, \Phi \rangle$ has the *linear space composition* property. ⌟

Roughly speaking, the linear time composition property states that there exists a program $h(i, j)$ which computes the composition $\phi_i(\phi_j(x))$ in an amount of time which is asymptotically equivalent to the sum of the time needed for computing P_i on input $\phi_j(x)$ and the time to compute P_j on input x. On the other hand, the linear space composition property aims at modeling the needed space, so that rather than adding the complexities of P_i and P_j, their maximum is considered, since this intuitively is the maximum amount of space needed for computing a composition of programs.

By observing that $\Theta(\max\{\Phi_i \circ \phi_j, \Phi_j\}) = \Theta(\Phi_i \circ \phi_j + \Phi_j)$ we can merge the linear time and space properties of Definition 6.1 and extend them for n-ary compositions as follows.

▶ **Definition 6.2** (Linear complexity composition)**.** Given a concrete semantics ϕ and a Blum complexity Φ, the pair $\langle \phi, \Phi \rangle$ has the *linear complexity composition* property if, given $n, m \geq 1$, there exists a total computable function $h : \mathbb{N}^2 \to \mathbb{N}$ such that for all $i, j \in \mathbb{N}$:

$$\phi^{(m+n)}_{h(i,j)} = \lambda\vec{x}\lambda\vec{y}.\ \phi^{(n+1)}_i(\phi^{(m)}_j(\vec{x}), \vec{y}),$$
$$\Phi^{(m+n)}_{h(i,j)} \in \Theta(\lambda\vec{x}\lambda\vec{y}.\ (\Phi^{(n+1)}_i(\phi^{(m)}_j(\vec{x}), \vec{y})) + \Phi^{(m)}_j(\vec{x}))).$$ ⌟

We can now recall the smn property as defined in [1, Definition 11].

▶ **Definition 6.3** (Asperti's smn property)**.** Given a concrete semantics ϕ, a Blum complexity Φ and $m, n \geq 1$, the pair $\langle \phi, \Phi \rangle$ has the *Asperti's smn* property if there exists a total computable function $s : \mathbb{N}^{m+1} \to \mathbb{N}$ such that $\forall e \in \mathbb{N}, \vec{x} \in \mathbb{N}^m$:

$$\lambda\vec{y}.\phi^{(m+n)}_e(\vec{x}, \vec{y}) = \phi^{(n)}_{s(e,\vec{x})},$$
$$\lambda\vec{y}.\Phi^{(m+n)}_e(\vec{x}, \vec{y}) \in \Theta(\lambda\vec{y}.\Phi^{(n)}_{s(e,\vec{x})}(\vec{y})).$$ ⌟

Informally, the smn property of Definition 6.3 states that the operation of fixing parameters preserves both the concrete semantics and the asymptotic complexity. Under these assumptions, we can show that Asperti's complexity clique semantics satisfies our ssmn property.

▶ **Lemma 6.4.** *Let $\langle \pi, \equiv_\pi \rangle$ be the complexity clique semantics of Example 2.6. If $\langle \pi, \equiv_\pi \rangle$ satisfies Asperti's smn and linear complexity composition properties then $\langle \pi, \equiv_\pi \rangle$ is ssmn.*

This result, together with the observation that the notion of fairness (Definition 3.2) instantiated to the complexity clique semantics is exactly that of [1, Definition 26], allows us to retrieve Kleene's second recursion theorem and Rice's theorem for complexity cliques in [1] as instances of our corresponding results given in Section 4.1.

6.2 Relation with Systems of Indices

As mentioned in Section 2, our definition of abstract semantics resembles the acceptable systems of indices [22, Definition II.5.1] or numberings [28, Exercise 2-10], firstly studied by Rogers [27]. In this section we discuss how such notions compare.

▶ **Definition 6.5** (System of indices [22, Definition II.5.1])**.** A *system of indices* is a family of functions $\{\psi^n\}_{n\in\mathbb{N}}$ such that each $\psi^n : \mathbb{N} \to \mathcal{C}_n$ is a surjective map that associates program indices to n-ary partial recursive functions.

- $\{\psi^n\}_{n\in\mathbb{N}}$ has the *parametrization* (or smn) property if for every $m, n \in \mathbb{N}$ there is a total computable function $s : \mathbb{N}^{m+1} \to \mathbb{N}$ such that $\forall e \in \mathbb{N}, \vec{x} \in \mathbb{N}^m$:

$$\lambda \vec{y}.\psi_e^{m+n}(\vec{x}, \vec{y}) = \psi^n_{s(e,\vec{x})}.$$

- $\{\psi^n\}_{n\in\mathbb{N}}$ has the *enumeration* property if for every $n \in \mathbb{N}$ there exists $u \in \mathbb{N}$ such that for all and $e \in \mathbb{N}$ and $\vec{y} \in \mathbb{N}^n$:

$$\psi_e^n = \lambda \vec{y}.\psi_u^{n+1}(e, \vec{y}).$$ ⌟

Any standard Gödel numbering associating a program with the function it computes is a system of indices with the parametrization and enumeration properties. Moreover, exactly as we did in Example 2.3, any system of indices $\{\psi^n\}_{n\in\mathbb{N}}$ can be viewed as an abstract semantics $\langle \pi, = \rangle$ with $\pi_n^a \triangleq \psi_a^n$. In this context, the enumeration and parametrization properties correspond to our fairness and ssmn conditions: fairness is exactly enumeration while ssmn follows from parametrization and enumeration, as discussed in Section 3 for the concrete semantics (cf. Example 2.3).

A system of indices is defined to be *acceptable* if it allows to get back and forth with a given system of indices satisfying the parametrization and enumeration properties through a pair of total computable functions.

▶ **Definition 6.6** (Acceptable system of indices [27, Definition 4])**.** Let $\{\varphi^n\}_{n\in\mathbb{N}}$ be a given system of indices with the parametrization and enumeration properties. A system of indices $\{\psi^n\}_{n\in\mathbb{N}}$ is *acceptable* if there exist two total computable functions $f, g : \mathbb{N} \to \mathbb{N}$ such that for all $a, n \in \mathbb{N}$:

$$\psi_a^n = \varphi^n_{f(a)} \quad \text{and} \quad \varphi_a^n = \psi^n_{g(a)}.$$ ⌟

As shown in [22, Proposition II.5.3], it turns out that a system of indices is acceptable if and only if it satisfies both enumeration and parametrization (a proof of this characterization was first given by Rogers [27, Section 2]). Consequently, an acceptable system of indices $\{\psi^n\}_{n\in\mathbb{N}}$ can be viewed as an abstract semantic $\langle \pi, = \rangle$, where $\pi_a^n = \psi_a^n$, which, by this characterization of acceptability, is ssmn and fair, and therefore, by Theorem 4.1 it enjoys Kleene's second recursion theorem, as already known from [22, Corollary II.5.4]. Under this

perspective, a generic abstract semantics according to Definition 2.2 can be viewed as a proper generalisation of the notion of acceptable system of indices, which merely encodes a change of program numbering and does not allow to take into account an actual abstraction of the concrete input/output behaviour of programs.

7 Conclusion and Future Work

This work generalises some traditional extensional results of computability theory, notably Kleene's second recursion theorem and Rice's theorem, to intensional abstract program semantics that include the complexity cliques investigated by Asperti [1]. Our approach was also inspired by Moyen and Simonsen [19] and relies on weakening the classical definition of extensional program property to a notion of partial extensionality w.r.t. abstract program semantics that satisfy some structural conditions. As an application, we strengthened and generalised a result by Müller-Olm and Seidl [20] proving that for affine programs with positive affine guards it is undecidable whether an affine relation holds at a given program point. Our results also shed further light on the claim that these undecidability results hinge on the Turing completeness of the underlying computational model, as argued in [19].

As future work, a natural question would be to investigate intensional extensions of Rice-Shapiro's theorem that fit our framework based on abstract semantics. This appears to be a nontrivial challenge. Generalisations of Rice-Shapiro's theorem have been given in [1, Section 5] and [19, Section 5.1]. A generalisation in the vein of the approach in [1] seems to be viable, but would require structural assumptions on abstract program semantics that, while natural in [1] whose focus is on complexity properties, would be artificial for abstract program semantics and would limit a general applicability. A further stimulating research topic is to apply our approach to abstract semantics as defined by abstract interpretation of programs, in particular for investigating the relationship with the notion of abstract extensionality studied by Bruni et al. [4]. Finally, while our framework relies on the assumption of an underlying Turing complete computational model, in a different direction, one could try to consider intensional properties for classes of programs indexing subrecursive functions (e.g., primitive recursive functions), whose extensional properties have been already studied (see, e.g., [10, 14]). Despite the fact that we suppose that our approach will fall short on these program classes, as one cannot expect to have a universal program inside the class itself or the validity of Kleene's second recursion theorem, we think that this represents an intriguing research challenge.

References

1 Andrea Asperti. The intensional content of Rice's theorem. In *Proceedings of the 35th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, POPL 2008, page 113–119, New York, NY, USA, 2008. ACM. doi:10.1145/1328438.1328455.

2 Manuel Blum. A machine-independent theory of the complexity of recursive functions. *J. ACM*, 14(2):322–336, April 1967. doi:10.1145/321386.321395.

3 Manuel Blum. On effective procedures for speeding up algorithms. *J. ACM*, 18(2):290–305, April 1971. doi:10.1145/321637.321648.

4 Roberto Bruni, Roberto Giacobazzi, Roberta Gori, Isabel Garcia-Contreras, and Dusko Pavlovic. Abstract extensionality: on the properties of incomplete abstract interpretations. *Proceedings of the ACM on Programming Languages (POPL 2020)*, 4:1–28, December 2020. doi:10.1145/3371096.

5 Michael R. Clarkson and Fred B. Schneider. Hyperproperties. *J. Comput. Secur.*, 18(6):1157–1210, 2010. doi:10.3233/JCS-2009-0393.

6 Patrick Cousot and Radhia Cousot. Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *Proc. 4th ACM Symp. on Principles of Programming Languages (POPL 1977)*. ACM, 1977. doi:10.1145/512950.512973.

7 Patrick Cousot, Roberto Giacobazzi, and Francesco Ranzato. Program analysis is harder than verification: A computability perspective. In *Proc. Int. Conf. on Computer Aided Verification (CAV 2018)*, pages 75–95. Springer, 2018.

8 Nigel Cutland. *Computability: An Introduction to Recursive Function Theory*. Cambridge University Press, 1980. doi:10.1017/CBO9781139171496.

9 Matthew S. Hecht. *Flow Analysis of Computer Programs*. Elsevier, 1977.

10 Mathieu Hoyrup. The decidable properties of subrecursive functions. In *Proc. Int. Coll. on Automata, Languages and Programming (ICALP 2016)*, volume 55 of *LIPIcs*, pages 108:1–108:13, 2016. doi:10.4230/LIPIcs.ICALP.2016.108.

11 Ehud Hrushovski, Joël Ouaknine, Amaury Pouly, and James Worrell. Polynomial invariants for affine programs. In *Proceedings of the 33rd Annual ACM/IEEE Symposium on Logic in Computer Science*, (LICS 2018), page 530–539, New York, NY, USA, 2018. ACM. doi:10.1145/3209108.3209142.

12 John B. Kam and Jeffrey D. Ullman. Monotone data flow analysis frameworks. *Acta Informatica*, 7:305–317, 1977. doi:10.1007/BF00290339.

13 Michael Karr. Affine relationships among variables of a program. *Acta Inf.*, 6:133–151, 1976. doi:10.1007/BF00268497.

14 Dexter Kozen. Indexings of subrecursive classes. *Theor. Comput. Sci.*, 11:277–301, 1980. doi:10.1016/0304-3975(80)90017-1.

15 Gerhard Lischke. Über die erfüllung gewisser erhaltungssätze durch kompliziertheitsmasse. *Mathematical Logic Quarterly*, 21(1):159–166, 1975. URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/malq.19750210121.

16 Gerhard Lischke. Natürliche kompliziertheitsmasze und erhaltungssätze I. *Mathematical Logic Quarterly*, 22(1):413–418, 1976. doi:10.1002/malq.19760220150.

17 Gerhard Lischke. Natürliche kompliziertheitsmasze und erhaltungssätze II. *Mathematical Logic Quarterly*, 23(13-15):193–200, 1977. doi:10.1002/malq.19770231302.

18 Antoine Miné. Tutorial on static inference of numeric invariants by abstract interpretation. *Foundations and Trends in Programming Languages*, 4(3-4):120–372, 2017. doi:10.1561/2500000034.

19 Jean-Yves Moyen and Jakob Grue Simonsen. More intensional versions of Rice's theorem. In *Proc. Computability in Europe (CIE 2019), Computing with Foresight and Industry*, pages 217–229. Springer, 2019. doi:10.1007/978-3-030-22996-2_19.

20 Markus Müller-Olm and Helmut Seidl. A note on Karr's algorithm. In *Proc. Int. Coll. on Automata, Languages and Programming (ICALP 2004)*, pages 1016–1028. Springer, 2004. doi:10.1007/978-3-540-27836-8_85.

21 Markus Müller-Olm and Helmut Seidl. Precise interprocedural analysis through linear algebra. In *Proceedings of the 31st ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, (POPL 2004), page 330–341, New York, NY, USA, 2004. ACM. doi:10.1145/964001.964029.

22 Piergiorgio Odifreddi. *Classical Recursion Theory: The Theory of Functions and Sets of Natural Numbers*. Sole Distributors for the Usa and Canada, Elsevier Science Pub. Co., 1989.

23 Francesco Ranzato. Decidability and synthesis of abstract inductive invariants. In *Proc. 31st International Conference on Concurrency Theory (CONCUR 2020)*, volume 171 of *LIPIcs*, pages 30:1–30:21, 2020. doi:10.4230/LIPIcs.CONCUR.2020.30.

24 John H. Reif and Harry R. Lewis. Symbolic evaluation and the global value graph. In *Proc. 4th ACM Symp. on Principles of Programming Languages (POPL 1977)*, pages 104–118. ACM, 1977. doi:10.1145/512950.512961.

25 H.G. Rice. Classes of recursively enumerable sets and their decision problems. *Transactions of the American Mathematical Society*, 74:358–366, 1953. doi:10.2307/1990888.

26 Xavier Rival and Kwangkeun Yi. *Introduction to Static Analysis – An Abstract Interpretation Perspective*. MIT Press, 2020.

27 Hartley Rogers. Gödel numberings of partial recursive functions. *Journal of Symbolic Logic*, 23(3):331–341, 1958. doi:10.2307/2964292.

28 Hartley Rogers. *Theory of Recursive Functions and Effective Computability*. Higher Mathematics Series. McGraw-Hill, 1967.

Optimal Spectral-Norm Approximate Minimization of Weighted Finite Automata

Borja Balle
DeepMind, London, UK

Clara Lacroce[1]
School of Computer Science, McGill University, Montréal, Canada
Mila, Montréal, Canada

Prakash Panangaden
School of Computer Science, McGill University, Montréal, Canada
Mila, Montréal, Canada

Doina Precup
School of Computer Science, McGill University, Montréal, Canada
Mila, Montréal, Canada

Guillaume Rabusseau[2]
DIRO, Université de Montréal, Montréal, Canada
CIFAR AI Chair, Mila, Montréal, Canada

Abstract

We address the approximate minimization problem for weighted finite automata (WFAs) with weights in $\mathbb{R}$, over a one-letter alphabet: to compute the best possible approximation of a WFA given a bound on the number of states. This work is grounded in Adamyan-Arov-Krein approximation theory, a remarkable collection of results on the approximation of Hankel operators. In addition to its intrinsic mathematical relevance, this theory has proven to be very effective for model reduction. We adapt these results to the framework of weighted automata over a one-letter alphabet. We provide theoretical guarantees and bounds on the quality of the approximation in the spectral and ℓ^2 norm. We develop an algorithm that, based on the properties of Hankel operators, returns the optimal approximation in the spectral norm.

2012 ACM Subject Classification Theory of computation → Quantitative automata; Theory of computation → Probabilistic computation; Theory of computation → Markov decision processes

Keywords and phrases Weighted finite automata, approximate minimization, Hankel matrices, AAK Theory

Digital Object Identifier 10.4230/LIPIcs.ICALP.2021.118

Category Track B: Automata, Logic, Semantics, and Theory of Programming

Related Version *Full Version*: `https://arxiv.org/abs/2102.06860` [10]

Funding *Clara Lacroce*: Research supported by NSERC (Canada).
Prakash Panangaden: Research supported by NSERC (Canada).
Doina Precup: Research supported by NSERC (Canada).
Guillaume Rabusseau: Research supported by CIFAR (Canadian AI chair program).

Acknowledgements The authors would like to thank Tianyu Li, Harsh Satija and Alessandro Sordoni for feedback on earlier drafts of this work, Gheorghe Comanici for a detailed review, and Maxime Wabartha for fruitful discussions and comments on proofs.

[1] Corresponding author.
[2] The names of the authors appear in alphabetical order.

1 Introduction

Weighted finite automata (WFAs) are an expressive class of models representing functions defined over sequences. The *approximate minimization problem* is concerned with finding an automaton that approximates the behaviour of a given minimal WFA, while being smaller in size. This second automaton recognizes a different language, and the objective is to minimize the approximation error [11, 12]. Approximate minimization becomes particularly useful in the context of spectral learning algorithms [5, 7, 9, 23]. When applied to a learning task, such algorithms can be viewed as working in two steps. First, they compute a minimal WFA that explains the training data exactly. Then, they obtain a model that generalizes to the unseen data by producing a smaller approximation to the minimal WFA. It is not just a question of saving space by having a smaller state space; the exact machine will *overfit* the data and generalize poorly. To obtain accurate results it is crucial to guess correctly the size of the minimal WFA, in particular when the data is generated by this type of machine.

The minimization task is greatly shaped by the way we decide to measure the approximation error. It is thus natural to wonder if there are norms that are preferable to others. We believe that the chosen norm should be computationally reasonable to minimize. For instance, the distance between WFAs can be computed using a metric based on bisimulation [8]. While this approach could still be interesting, the fact that this metric is hard to compute makes it unsuitable for our purposes. Moreover, this metric is specifically designed for WFAs, so it is not directly applicable to other models dealing with sequential data. We think that being transferable is a second important feature for a norm. In fact, being able to compare different classes of models is desirable for future applications of this method. For example, one can think of the burgeoning literature on approximating Recurrent Neural Networks (RNNs) using WFAs, where the objective is to extract from a trained RNN an automaton that accurately mimics its behaviour [37, 41, 32, 4, 19]. With this in mind, we think that it is preferable to consider a norm defined on the input-output function – or the Hankel matrix – rather than the parameters of the specific model considered. Finally, it is important to choose a norm that can be computed accurately. The spectral norm seems to be a good candidate for the task. In particular, it allows us to exploit the work of Adamyan, Arov and Krein which has come to be known as AAK theory [1]: a series of results connecting the theory of complex functions on the unit circle to Hankel matrices, a mathematical object representing functions defined over sequences. The core of this theory provides us with theoretical guarantees for the exact computation of the spectral norm of the error, and a method to construct the optimal approximation. We summarize our main contributions:

- We use AAK theory to study the approximate minimization problem of WFAs. To connect those areas, we establish a correspondence between the parameters of a WFA and the coefficients of a complex function on the unit circle. To the best of our knowledge, this paper represents the first attempt to apply AAK theory to WFAs.
- We present a theoretical analysis of the optimal spectral-norm approximate minimization problem for WFAs, based on their connection with finite-rank infinite Hankel matrices. We provide a closed form solution for real weighted automata $A = \langle \boldsymbol{\alpha}, \mathbf{A}, \boldsymbol{\beta} \rangle$ over a one-letter alphabet, under the assumption $\rho(\mathbf{A}) < 1$ on the spectral radius. We bound the approximation error, both in terms of spectral and ℓ^2 norm.
- We propose a self-contained algorithm that returns the unique optimal spectral-norm approximation of a given size.
- We tighten the connection, made in [12], between WFAs and discrete dynamical systems, by adapting some of the control theory concepts, *e.g.* the *allpass system*.

2 Background

2.1 Preliminaries

We denote with $\mathbb{N}$, $\mathbb{Z}$ and $\mathbb{R}$ the set of natural, integers and real numbers, respectively. We use bold letters for vectors and matrices; all vectors considered are column vectors. We denote with $\mathbf{1}$ the identity matrix, specifying its dimension only when not clear from the context. We refer to the i-th row and the j-th column of $\mathbf{M}$ by $\mathbf{M}(i,:)$ and $\mathbf{M}(:,j)$. Given a matrix $\mathbf{M} \in \mathbb{R}^{p\times q}$ of rank n, a *rank factorization* is a factorization $\mathbf{M} = \mathbf{PQ}$, where $\mathbf{P} \in \mathbb{R}^{p\times n}$, $\mathbf{Q} \in \mathbb{R}^{n\times q}$ and $\text{rank}(\mathbf{P}) = \text{rank}(\mathbf{Q}) = n$. Let $\mathbf{M} \in \mathbb{R}^{p\times q}$ of rank n, the compact *singular value decomposition* SVD of $\mathbf{M}$ is the factorization $\mathbf{M} = \mathbf{UDV}^\top$, where $\mathbf{U} \in \mathbb{R}^{p\times n}$, $\mathbf{D} \in \mathbb{R}^{n\times n}$, $\mathbf{V} \in \mathbb{R}^{q\times n}$ are such that $\mathbf{U}^\top\mathbf{U} = \mathbf{V}^\top\mathbf{V} = \mathbf{1}$, and $\mathbf{D}$ is a diagonal matrix. The columns of $\mathbf{U}$ and $\mathbf{V}$ are called left and right *singular vectors*, while the entries of $\mathbf{D}$ are the *singular values*. The *Moore-Penrose pseudo-inverse* $\mathbf{M}^+$ of $\mathbf{M}$ is the unique matrix such that $\mathbf{MM}^+\mathbf{M} = \mathbf{M}$, $\mathbf{M}^+\mathbf{MM}^+ = \mathbf{M}^+$, with $\mathbf{M}^+\mathbf{M}$ and $\mathbf{MM}^+$ Hermitian [43]. The *spectral radius* $\rho(\mathbf{M})$ of a matrix $\mathbf{M}$ is the largest modulus among its eigenvalues.

A *Hilbert space* is a complete normed vector space where the norm arises from an inner product. A linear operator $T : X \to Y$ between Hilbert spaces is *bounded* if it has finite operator norm, *i.e.* $\|T\|_{op} = \sup_{\|g\|_X \leq 1} \|Tg\|_Y < \infty$. We denote by $\mathbf{T}$ the (infinite) matrix associated with T by some (canonical) orthonormal basis on H. An operator is *compact* if the image of the unit ball in X is relatively compact. Let $T : X \to Y$ be a compact operator between Hilbert spaces, the *adjoint operator* T^* is the linear operator $T^* : Y \to X$ such that $\langle Tx, y\rangle_Y = \langle x, T^*y\rangle_X$, with $x \in X$, $y \in Y$ and $\langle\cdot,\cdot\rangle$ denotes the inner product of the corresponding Hilbert space. The *singular numbers* $\{\sigma_n\}_{n\geq 0}$ of T are the square roots of the non-negative eigenvalues of the self-adjoint operator T^*T, arranged in decreasing order. A *σ-Schmidt pair* $\{\boldsymbol{\xi}, \boldsymbol{\eta}\}$ for T is a couple of norm 1 vectors such that: $\mathbf{T}\boldsymbol{\xi} = \sigma\boldsymbol{\eta}$ and $\mathbf{T}^*\boldsymbol{\eta} = \sigma\boldsymbol{\xi}$. The Hilbert-Schmidt decomposition provides a generalization of the compact SVD for the infinite matrix of a compact operator T using singular numbers and orthonormal Schmidt pairs: $\mathbf{Tx} = \sum_{n\geq 0} \sigma_n \langle \mathbf{x}, \boldsymbol{\xi}_n\rangle \boldsymbol{\eta}_k$ [43]. The *spectral norm* $\|\mathbf{T}\|$ of the matrix representing the operator T is the largest singular number of T. Note that the spectral norm of $\mathbf{T}$ corresponds to the operator norm of T.

Let ℓ^2 be the Hilbert space of square-summable sequences over Σ^*, with norm $\|f\|_2^2 = \sum_{x\in\Sigma^*} |f(x)|^2$ and inner product $\langle f, g\rangle = \sum_{x\in\Sigma^*} f(x)g(x)$ for $f, g \in \mathbb{R}^{\Sigma^*}$. Let $\mathbb{T} = \{z \in \mathbb{C} : |z| = 1\}$ be the complex unit circle, $\mathbb{D} = \{z \in \mathbb{C} : |z| < 1\}$ the (open) complex unit disc. Let $1 < p < \infty$, $\mathcal{L}^p(\mathbb{T})$ be the space of measurable functions on $\mathbb{T}$ for which the p-th power of the absolute value is Lebesgue integrable. For $p = \infty$, we denote with $\mathcal{L}^\infty(\mathbb{T})$ the space of measurable functions that are bounded, with norm $\|f\|_\infty = \sup\{|f(x)| : x \in \mathbb{T}\}$.

2.2 Weighted Finite Automata

Let Σ be a fixed finite alphabet, Σ^* the set of all finite strings with symbols in Σ. We use ε to denote the empty string. Given $p, s \in \Sigma$, we denote with ps their concatenation.

A *weighted finite automaton* (WFA) of n states over Σ is a tuple $A = \langle \boldsymbol{\alpha}, \{\mathbf{A}_a\}, \boldsymbol{\beta}\rangle$, where $\boldsymbol{\alpha}$, $\boldsymbol{\beta} \in \mathbb{R}^n$ are the vector of initial and final weights, respectively, and $\mathbf{A}_a \in \mathbb{R}^{n\times n}$ is the matrix containing the transition weights associated with each symbol a. Every WFA A with real weights realizes a function $f_A : \Sigma^* \to \mathbb{R}$, *i.e.* given a string $x = x_1 \cdots x_t \in \Sigma^*$, it returns $f_A(x) = \boldsymbol{\alpha}^\top \mathbf{A}_{x_1} \cdots \mathbf{A}_{x_t} \boldsymbol{\beta} = \boldsymbol{\alpha}^\top \mathbf{A}_x \boldsymbol{\beta}$. A function $f : \Sigma^* \to \mathbb{R}$ is called *rational* if there exists a WFA A that realizes it. The *rank* of the function is the size of the smallest WFA realizing f. Given $f : \Sigma^* \to \mathbb{R}$, we can consider a matrix $\mathbf{H}_f \in \mathbb{R}^{\Sigma^*\times\Sigma^*}$ having rows and columns indexed by strings and defined by $\mathbf{H}_f(p, s) = f(ps)$ for $p, s \in \Sigma^*$.

▶ **Definition 1.** *A (bi-infinite) matrix* $\mathbf{H} \in \mathbb{R}^{\Sigma^* \times \Sigma^*}$ *is* ***Hankel*** *if for all* $p, p', s, s' \in \Sigma^*$ *such that* $ps = p's'$*, we have* $\mathbf{H}(p,s) = \mathbf{H}(p',s')$*. Given a Hankel matrix* $\mathbf{H} \in \mathbb{R}^{\Sigma^* \times \Sigma^*}$*, there exists a unique function* $f : \Sigma^* \to \mathbb{R}$ *such that* $\mathbf{H}_f = \mathbf{H}$*.*

▶ **Theorem 2** ([16, 20])**.** *A function* $f : \Sigma^* \to \mathbb{R}$ *is realized by a WFA* A *if and only if* $\mathbf{H}_f$ *has finite rank* n*. In that case,* n *is the minimal number of states of any* A *such that* $f = f_A$*.*

Given a WFA $A = \langle \boldsymbol{\alpha}, \{\mathbf{A}_a\}, \boldsymbol{\beta} \rangle$, the *forward matrix* of A is the infinite matrix $\mathbf{F}_A \in \mathbb{R}^{\Sigma^* \times n}$ given by $\mathbf{F}_A(p,:) = \boldsymbol{\alpha}^\top \mathbf{A}_p$ for any $p \in \Sigma^*$, while the *backward matrix* of A is $\mathbf{B}_A \in \mathbb{R}^{\Sigma^* \times n}$, given by $\mathbf{B}_A(s,:) = (\mathbf{A}_s \boldsymbol{\beta})^\top$ for any $s \in \Sigma^*$. Let $\mathbf{H}_f$ be the Hankel matrix of f, its forward-backward (FB) factorization is: $\mathbf{H}_f = \mathbf{F}\mathbf{B}^\top$. A WFA with n states is *reachable* if $\operatorname{rank}(\mathbf{F}_A) = n$, while it is *observable* if $\operatorname{rank}(\mathbf{B}_A) = n$. A WFA is *minimal* if it is reachable and observable. If A is minimal, the FB factorization is a rank factorization [7].

We recall the definition of the singular value automaton, a canonical form for WFAs [11].

▶ **Definition 3.** *Let* $f : \Sigma^* \to \mathbb{R}$ *be a rational function and suppose* $\mathbf{H}_f$ *admits an SVD,* $\mathbf{H}_f = \mathbf{U}\mathbf{D}\mathbf{V}^\top$*. A* ***singular value automaton*** *(SVA) for* f *is the minimal WFA* A *realizing* f *such that* $\mathbf{F}_A = \mathbf{U}\mathbf{D}^{1/2}$ *and* $\mathbf{B}_A = \mathbf{V}\mathbf{D}^{1/2}$*.*

The SVA can be computed with an efficient algorithm relying on the following matrices [12].

▶ **Definition 4.** *Let* $f : \Sigma^* \to \mathbb{R}$ *be a rational function,* $\mathbf{H}_f = \mathbf{F}\mathbf{B}^\top$ *a FB factorization. If the matrices* $\mathbf{P} = \mathbf{F}^\top\mathbf{F}$ *and* $\mathbf{Q} = \mathbf{B}^\top\mathbf{B}$ *are well defined, we call* $\mathbf{P}$ *the* ***reachability Gramian*** *and* $\mathbf{Q}$ *the* ***observability Gramian****.*

Note that if A is an SVA, then the Gramians associated with its FB factorization satisfy $\mathbf{P}_A = \mathbf{Q}_A = \mathbf{D}$, where $\mathbf{D}$ is the matrix of singular values of its Hankel matrix. The Gramians can alternatively be characterized (and computed [12]) using fixed point equations, corresponding to Lyapunov equations when $|\Sigma| = 1$ [28].

▶ **Theorem 5.** *Let* $|\Sigma| = 1$*,* $A = \langle \boldsymbol{\alpha}, \mathbf{A}, \boldsymbol{\beta} \rangle$ *a WFA with* n *states and well-defined Gramians* $\mathbf{P}$*,* $\mathbf{Q}$*. Then* $X = \mathbf{P}$ *and* $Y = \mathbf{Q}$ *solve the equations* $X - \mathbf{A}X\mathbf{A}^\top = \boldsymbol{\beta}\boldsymbol{\beta}^\top$ *and* $Y - \mathbf{A}^\top Y \mathbf{A} = \boldsymbol{\alpha}\boldsymbol{\alpha}^\top$*.*

Finally, we recall the definition of *generative probabilistic automata* (GPA). A WFA $A = \langle \boldsymbol{\alpha}, \{\mathbf{A}_a\}, \boldsymbol{\beta} \rangle$ is a GPA if $f_A(x) \geq 0$ for every x and $\sum_{x \in \Sigma^*} f_A(x) = 1$, *i.e.* if f_A computes a probability distribution over Σ^*.

▶ **Example 6.** Let $|\Sigma| = 1$, $\Sigma = \{x\}$. The WFA A: $\mathbf{A} = \begin{pmatrix} 0 & \frac{1}{2} \\ \frac{1}{2} & 0 \end{pmatrix}$, $\boldsymbol{\alpha} = \boldsymbol{\beta} = \begin{pmatrix} \frac{\sqrt{3}}{2} \\ 0 \end{pmatrix}$, is a GPA. Ideed, $f_A(x) \geq 0$ and $\sum_{x \in \Sigma^*} f_A(x) = 1$, since the rational function is:

$$f_A(x \cdots x) = f_A(k) = \boldsymbol{\alpha}^\top \mathbf{A}^k \boldsymbol{\beta} = \begin{cases} 0 & \text{if } k \text{ is odd} \\ \frac{3}{4} 2^{-k} & \text{if } k \text{ is even} \end{cases}$$

where k corresponds to the string where x is repeated k-times. We remark that A is minimal and in its SVA form, with Gramians $\mathbf{P} = \mathbf{Q} = \begin{pmatrix} \frac{4}{5} & 0 \\ 0 & \frac{1}{5} \end{pmatrix}$, and f_A has rank 2. Finally, the corresponding Hankel matrix, also of rank 2, is:

$$\mathbf{H} = \begin{pmatrix} f_A(0) & f_A(1) & f_A(2) & \cdots \\ f_A(1) & f_A(2) & f_A(3) & \cdots \\ f_A(2) & f_A(3) & f_A(4) & \cdots \\ \vdots & \vdots & \vdots & \ddots \end{pmatrix} = \begin{pmatrix} \frac{3}{4} & 0 & \frac{3}{16} & \cdots \\ 0 & \frac{3}{16} & 0 & \cdots \\ \frac{3}{16} & 0 & \frac{3}{64} & \cdots \\ \vdots & \vdots & \vdots & \ddots \end{pmatrix}. \tag{1}$$

2.3 AAK Theory

Theorem 2 provides us with a way to associate a minimal WFA A with n states to a Hankel matrix $\mathbf{H}$ of rank n. The approach we propose to approximate A is to find the WFA corresponding to the matrix that minimizes $\mathbf{H}$ in the spectral norm. We recall the fundamental result of Schmidt, Eckart, Young and Mirsky [18].

▶ **Theorem 7** ([18]). *Let $\mathbf{H}$ be a Hankel matrix corresponding to a compact Hankel operator of rank n, and σ_m, with $0 \leq m < n$ and $\sigma_0 \geq \cdots \geq \sigma_{n-1} > 0$, its singular numbers. Then, if $\mathbf{R}$ is a matrix of rank k, we have: $\|\mathbf{H} - \mathbf{R}\| \geq \sigma_k$. The equality is attained when $\mathbf{R}$ corresponds to the truncated SVD of $\mathbf{H}$.*

Note that a low-rank approximation obtained by truncating the SVD is not in general a Hankel matrix. This is problematic, since $\mathbf{G}$ needs to be Hankel in order to be the matrix of a WFA. Surprisingly, we can obtain a result comparable to the one of Theorem 7 while preserving the Hankel property. This is the possible thanks to AAK theory [1], a theory of optimal approximation. To apply this theory, we will need to rewrite the approximation problem in functional analysis terms. First, we will associate a linear operator to the Hankel matrix. Then, we will use Fourier analysis to reformulate the problem in a function space. A comprehensive presentation of the concepts recalled in this section can be found in [31, 34, 29].

Let $f : \Sigma^* \to \mathbb{R}$ be a rational function, we interpret the corresponding Hankel matrix $\mathbf{H}_f$ as the expression of a linear (Hankel) operator $H_f : \ell^2 \to \ell^2$ in terms of the canonical basis. We recall that a Hankel operator H_f is bounded if and only if $f \in \ell^2$ [12]. This property, together with the fact that we only consider finite rank operators (corresponding to the Hankel matrices of WFAs), is sufficient to guarantee compactness.

To introduce AAK theory, we need to consider a second realization of Hankel operators on complex spaces. Since in this paper we work with two classes of functions – functions over sequences and complex functions – to avoid any confusion we will make explicit the dependence on the complex variable $z = e^{it}$. We start by recalling a few fundamental definitions from the theory of complex functions. Note that a function $\phi(z) \in \mathcal{L}^2(\mathbb{T})$ can be represented, using the orthonormal basis $\{z^n\}_{n\in\mathbb{Z}}$, by means of its Fourier series: $\phi(z) = \sum_{n\in\mathbb{Z}} \widehat{\phi}(n) z^n$, with Fourier coefficients $\widehat{\phi}(n) = \int_{\mathbb{T}} \phi(z)\bar{z}^n dz$, $n \in \mathbb{Z}$. This establishes an isomorphism between the function $\phi(z)$ and the sequence of the corresponding Fourier coefficients $\widehat{\phi}$. Thus, we can partition the function space $\mathcal{L}^2(\mathbb{T})$ into two subspaces.

▶ **Definition 8.** *For $0 < p \leq \infty$, the **Hardy space** $\mathcal{H}^p$ and the **negative Hardy space** $\mathcal{H}^p_-$ on $\mathbb{T}$ are the subspaces of $\mathcal{L}^p(\mathbb{T})$ defined as:*

$$\mathcal{H}^p = \{\phi(z) \in \mathcal{L}^p(\mathbb{T}) : \widehat{\phi}(n) = 0, n < 0\}, \qquad \mathcal{H}^p_- = \{\phi(z) \in \mathcal{L}^p(\mathbb{T}) : \widehat{\phi}(n) = 0, n \geq 0\}.$$

Interestingly, the elements of the Hardy space can be canonically identified with the set of functions analytic in $\mathbb{D}$, with the property that the p-th power of their absolute value is integrable on $\mathbb{T}$ (a proof can be found in [31]). Thus, we will make no difference between these functions in the unit disc and their boundary value on the circle.

We can now embed the sequence space ℓ^2 into $\ell^2(\mathbb{Z})$ by "duplicating" each vector, *i.e.* by associating $\boldsymbol{\mu} = (\mu_0, \mu_1, \dots) \in \ell^2$ to $\boldsymbol{\mu}^{(2)} = (\dots, \mu_1, \mu_0, \mu_1, \dots) \in \ell^2(\mathbb{Z})$. Then, we can use the Fourier isomorphism to map the vector $\boldsymbol{\mu}^{(2)} \in \ell^2(\mathbb{Z})$ to the function space $\mathcal{L}^2(\mathbb{T})$. In this way each vector $\boldsymbol{\mu} \in \ell^2$ corresponds to two functions in the Hardy spaces:

$$\mu^-(z) = \sum_{j=0}^{\infty} \boldsymbol{\mu}_j z^{-j-1} \in \mathcal{H}^2_-, \qquad \mu^+(z) = \sum_{j=0}^{\infty} \boldsymbol{\mu}_j z^j \in \mathcal{H}^2. \tag{2}$$

This leads to an alternative characterization of Hankel operators in Hardy spaces.

▶ **Definition 9.** *Let $\phi(z)$ be a function in the space $\mathcal{L}^2(\mathbb{T})$. A **Hankel operator** is an operator $H_\phi : \mathcal{H}^2 \to \mathcal{H}^2_-$ defined by $H_\phi f(z) = \mathbb{P}_- \phi f(z)$, where $\mathbb{P}_-$ is the orthogonal projection from $\mathcal{L}^2(\mathbb{T})$ onto $\mathcal{H}^2_-$. The function $\phi(z)$ is called a **symbol** of the Hankel operator H_ϕ.*

We briefly recall the following theorem, due to Nehari [30], characterizing bounded Hankel operators.

▶ **Theorem 10** ([30])**.** *Let $\phi \in \mathcal{L}^2(\mathbb{T})$ be a symbol of the Hankel operator on Hardy spaces $H_\phi : \mathcal{H}^2 \to \mathcal{H}^2_-$. Then, H_ϕ is bounded on $\mathcal{H}^2$ if and only if there exists $\psi \in \mathcal{L}^\infty(\mathbb{T})$ such that $\widehat{\psi}(m) = \widehat{\phi}(m)$ for all $m < 0$. If the conditions above are satisfied, then:*

$$\|H_\phi\| = \inf\{\|\psi\|_\infty : \widehat{\psi}(m) = \widehat{\phi}(m),\, m < 0\}. \tag{3}$$

As a consequence of Theorem 10, if H_ϕ is a bounded operator, we can consider without loss of generality $\phi(z) \in \mathcal{L}^\infty(\mathbb{T})$. We remark that a Hankel operator has infinitely many different symbols, since $H_\phi = H_{\phi+\psi}$ for $\psi(z) \in \mathcal{H}^\infty$.

▶ Remark 11. Note that, in the standard orthonormal bases $\{z^k\}_{k\geq 0}$ of $\mathcal{H}^2$ and $\{z^{-(j+1)}\}_{j\geq 0}$ of $\mathcal{H}^2_-$, H_ϕ has Hankel matrix $\mathbf{H}(j,k) = \widehat{\phi}(-j-k-1)$ for $j,k \geq 0$.

▶ **Example 12.** In the case of the Hankel matrix in Example 6, we have that $\mathbf{H}(j,k) = 0$ if $j+k$ is odd, and $\mathbf{H}(j,k) = \frac{3}{4}2^{-(j+k)}$ if $j+k$ is even. Since $\mathbf{H}(j,k) = \widehat{\phi}(-j-k-1)$, we can recover the corresponding symbol: $\mathbb{P}_-\phi = \sum_{n\geq 0} \widehat{\phi}(-n-1)z^{-n-1} = \sum_{n\geq 0} \frac{3}{4}2^{-2n}z^{-2n-1} = \frac{3z}{4z^2-1}$.

▶ **Definition 13.** *The complex function $\phi(z)$ is **rational** if $\phi(z) = p(z)/q(z)$, with $p(z)$ and $q(z)$ polynomials. The rank of $\phi(z)$ is the maximum between the degrees of $p(z)$ and $q(z)$. A rational function is **strictly proper** if the degree of $p(z)$ is strictly smaller than that of $q(z)$.*

We remark that the poles of a complex function $\phi(z)$ correspond to the zeros of $1/\phi(z)$. The following result of Kronecker relates finite-rank infinite Hankel matrices to rational functions.

▶ **Theorem 14** ([25])**.** *Let H_ϕ be a bounded Hankel operator with matrix $\mathbf{H}$. Then $\mathbf{H}$ has finite rank if and only if $\mathbb{P}_-\phi$ is a strictly proper rational function. Moreover the rank of $\mathbf{H}$ is equal to the number of poles (with multiplicities) of $\mathbb{P}_-\phi$ inside the unit disc.*

▶ **Example 15.** The function in Example 12 is rational with degree 2 and has two poles inside the unit disc at $z = \pm\frac{1}{2}$. Thus, the Hankel matrix associated has rank 2.

We state as remark an important takeaway from this section.

▶ Remark 16. Given a rank n Hankel matrix $\mathbf{H}$, we can look at it in two alternative ways. On the one hand we can consider the Hankel operator over sequences $H_f : \ell^2 \to \ell^2$, associated to a function $f : \Sigma^* \to \mathbb{R}$. In this case $\mathbf{H}(i,j) = f(i+j)$ for $i,j \geq 0$, and f is rational in the sense that it is realized by a WFA of size n. On the other hand, we can consider the Hankel operator over complex (Hardy) spaces $H_\phi : \mathcal{H}^2 \to \mathcal{H}^2_-$, associated to a function $\phi(z) \in \mathcal{L}^2(\mathbb{T})$, the symbol. In this case $\mathbf{H}(j,k) = \widehat{\phi}(-j-k-1)$ for $j,k \geq 0$, and $\mathbb{P}_-\phi = \widehat{\phi}(-j-k-1)$ is rational of rank n in the sense of Definition 13.

We can introduce now the main result of Adamyan, Arov and Krein [1]. The theorem, stated for Hankel operators over Hardy spaces, shows that for infinite dimensional Hankel matrices the constraint of preserving the Hankel property does not affect the achievable approximation error.

▶ **Theorem 17** (AAK-1[1]). *Let H_ϕ be a compact Hankel operator of rank n and singular numbers σ_m, with $0 \leq m < n$ and $\sigma_0 \geq \cdots \geq \sigma_{n-1} > 0$. Then there exists a unique Hankel operator G of rank $k < n$ such that:*

$$\|\mathbf{H} - \mathbf{G}\| = \sigma_k. \tag{4}$$

We can reformulate the theorem in terms of symbols. Let $\mathcal{R}_k \subset \mathcal{H}^\infty_-$ be the set of strictly proper rational functions of rank k, and $\mathcal{H}^\infty_k = \{\psi \in \mathcal{L}^\infty(\mathbb{T}) : \ \exists g \in \mathcal{R}_k, \exists l \in \mathcal{H}^\infty, \ \psi = g + l\}$.

▶ **Theorem 18** (AAK-2 [1]). *Let $\phi(z) \in \mathcal{L}^\infty(\mathbb{T})$. Then, there exists $\psi(z) \in \mathcal{H}^\infty_k$ such that:*

$$\|\phi(z) - \psi(z)\|_\infty = \sigma_k(H_\phi). \tag{5}$$

The solutions of Theorem 17 and 18 are strictly related.

▶ **Corollary 19.** *Let $\psi(z) = g(z) + l(z) \in \mathcal{H}^\infty_k$, with $g(z) \in \mathcal{R}_k$, $l(z) \in \mathcal{H}^\infty$. If $\psi(z)$ solves Equation 5, then $G = H_g$ is the unique Hankel operator from Theorem 17.*

Proof of Corollary 19. Let H_ϕ be a Hankel operator with symbol $\phi(z) \in \mathcal{L}^\infty(\mathbb{T})$ and matrix $\mathbf{H}$. Let $\psi(z) = g(z) + l(z) \in \mathcal{H}^\infty_k$ be the solution of Equation 5. We have:

$$\begin{aligned} \|H_\phi - H_\psi\| &= \|H_{\phi-\psi}\| && (6)\\ &= \|H_{\sigma_k \eta_k^-(z)/\xi_k^+(z)}\| && (7)\\ &\leq \sigma_k \|\eta_k^-(z)/\xi_k^+(z)\|_\infty = \sigma_k && (8) \end{aligned}$$

where first we used Corollary 20 and then we applied Theorem 10. Now, using the definition of Hankel operator, we have:

$$\|H_\phi - H_\psi\| = \|H_\phi - H_g\| = \|\mathbf{H} - \mathbf{G}\| \leq \sigma_k. \tag{9}$$

Since $\|\mathbf{H} - \mathbf{G}\| \geq \sigma_k$ (from Eckart-Young theorem [18]), it follows that $\|\mathbf{H} - \mathbf{G}\| = \sigma_k$. Note that $\mathbf{G}$ has rank k, as required, because $g \in \mathcal{R}_k$(Theorem 14). ◀

We state as corollary the key point of the proof of AAK Theorem, that provides us with a practical way to find the best approximating symbol.

▶ **Corollary 20.** *Let $\phi(z)$ and $\{\boldsymbol{\xi}_k, \boldsymbol{\eta}_k\}$ be a symbol and a σ_k-Schmidt pair for H_ϕ. A function $\psi(z) \in \mathcal{L}^\infty$ is the best AAK approximation according to Theorem 18, if and only if:*

$$(\phi(z) - \psi(z))\xi_k^+(z) = \sigma_k \eta_k^-(z). \tag{10}$$

Moreover, the function $\psi(z)$ does not depend on the particular choice of the pair $\{\boldsymbol{\xi}_k, \boldsymbol{\eta}_k\}$.

3 Approximate Minimization

3.1 Assumptions

To apply AAK theory to the approximate minimization problem, we consider only automata with real weights, defined over a one-letter alphabet. In this case, the free monoid generated by the single letter can be identified with $\mathbb{N}$, and canonically embedded into $\mathbb{Z}$. This passage is fundamental to use Fourier analysis and the isomorphism that leads to Theorem 17 and 18. Unfortunately, this idea cannot be directly generalized to bigger alphabets, since in this case we would obtain a free non-abelian monoid (not identifiable with $\mathbb{Z}$).

Theorem 17 requires the Hankel operator H to be bounded. To ensure that a minimal WFA $A = \langle \boldsymbol{\alpha}, \mathbf{A}, \boldsymbol{\beta} \rangle$ satisfies this condition, we assume $\rho(\mathbf{A}) < 1$, where ρ is the spectral radius of $\mathbf{A}$ [12]. As a matter of fact, to guarantee the boundness of the Hankel operator it is enough that the considered WFA computes a function $f \in \ell^2$ [12]. However, the stricter assumption on the spectral radius is needed when computing the symbol associated to a WFA. This condition directly implies the existence of the SVA, and of the Gramian matrices $\mathbf{P}$ and $\mathbf{Q}$, with $\mathbf{P} = \mathbf{Q}$ and diagonal [12]. We assume $A = \langle \boldsymbol{\alpha}, \mathbf{A}, \boldsymbol{\beta} \rangle$ is in SVA form. In this case, given the size of the alphabet, the Hankel matrix $\mathbf{H}$ is symmetric, so $\mathbf{A} = \mathbf{A}^\top$. Moreover, if we denote with λ_i the i-th non-zero eigenvalue of $\mathbf{H}$, and we consider the coordinates of $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$, we have that $\boldsymbol{\alpha}_i = \operatorname{sgn}(\lambda_i)\boldsymbol{\beta}_i$, where $\operatorname{sgn}(\lambda_i) = \lambda_i/|\lambda_i|$.

For example, we note that a minimal GPA computes a function $f \in \ell^1$, so the condition on $\rho(\mathbf{A})$ is automatically satisfied by this class of WFAs [12]. Possible relaxations of the spectral radius assumption are discussed in Section 5, together with an alternative method to find the optimal spectral-norm approximation of a Hankel matrix without extracting a WFA.

Finally, in this paper we consider only automata with weights in $\mathbb{R}$, though results remain true for complex numbers. The method we present can be easily extended to vector-valued automata [36], but the solution to the optimal approximation problem will not be unique [34].

3.2 Problem Formulation

Let $A = \langle \boldsymbol{\alpha}, \mathbf{A}, \boldsymbol{\beta} \rangle$ be a minimal WFA with n states in SVA form, defined over a one-letter alphabet. Let $\mathbf{H}$ be the Hankel matrix of A, we denote with σ_i, for $0 \leq i < n$, the singular numbers. Given a target number of states $k < n$, we say that a WFA $\widehat{A}_k$ with k states solves the *optimal spectral-norm approximate minimization* problem if the Hankel matrix $\mathbf{G}$ of $\widehat{A}_k$ satisfies $\|\mathbf{H} - \mathbf{G}\| = \sigma_k(\mathbf{H})$. Note that the content of the "optimal spectral-norm approximate minimization" is equivalent to the problem solved by Theorem 17, with the exception that here we insist on representing the inputs and outputs of the problem effectively by means of WFAs. Based on the AAK theory sketched in Section 2.3, we draw the following steps:

1. *Compute a symbol $\phi(z)$ for H using Remark 16.* We obtain the negative Fourier coefficients of $\phi(z)$ and derive its Fourier series.
2. *Compute the optimal symbol $\psi(z)$ using Corollary 20.* The main challenge here is to find a suitable representation for the functions $\psi(z)$ and $e(z) = \phi(z) - \psi(z)$. We define them in terms of two auxiliary WFAs. The key point is to select constraints on their parameters to leverage the properties of weighted automata, while still keeping the formulation general.
3. *Extracting the rational component by solving for $g(z)$ in Corollary 19.* This step is arguably the most conceptually challenging, as it requires to identify the position of the function's poles. In fact, we know from Theorem 14 that $g(z)$ has k poles, all inside the unit disc.
4. *Find a WFA representation for $g(z)$.* Since in Step 2 we parametrized the functions using WFAs, the expression of $g(z)$ directly reveals the WFA $\widehat{A}_k$.

3.3 Spectral-Norm Approximate Minimization

In the following sections we will consider a minimal WFA $A = \langle \boldsymbol{\alpha}, \mathbf{A}, \boldsymbol{\beta} \rangle$ with n states in SVA form, defined over a one-letter alphabet $\Sigma = \{a\}$, its Hankel matrix $\mathbf{H}$, corresponding to the bounded operator H, and the singular numbers σ_i, for $0 \leq i < n$. Let $f : \Sigma^* \to \mathbb{R}$ be the function realized by A. We denote by x the string where a is repeated x times, so we have $f(x) = \boldsymbol{\alpha}^\top \mathbf{A}^x \boldsymbol{\beta}$.

3.3.1 Computation of a Symbol for A

To determine the symbol $\phi(z)$ of H, we recall that each entry of the Hankel matrix corresponds simultaneously to the values of f and to the negative Fourier coefficients of $\phi(z)$. In fact, as seen in Remark 16, we have:

$$\mathbf{H} = \begin{pmatrix} f_A(0) & f_A(1) & \dots \\ f_A(1) & f_A(2) & \dots \\ \vdots & \vdots & \ddots \end{pmatrix} = \begin{pmatrix} \widehat{\phi}(-1) & \widehat{\phi}(-2) & \dots \\ \widehat{\phi}(-2) & \widehat{\phi}(-3) & \dots \\ \vdots & \vdots & \ddots \end{pmatrix}. \tag{11}$$

We obtain:

$$\mathbb{P}_-\phi(z) = \sum_{k\geq 0} f(k) z^{-k-1} = \sum_{k\geq 0} \boldsymbol{\alpha}^\top \mathbf{A}^k \boldsymbol{\beta} z^{-k-1} = \boldsymbol{\alpha}^\top (z\mathbf{1} - \mathbf{A})^{-1} \boldsymbol{\beta} \tag{12}$$

where we use the fact that $\rho(A) < 1$ for the last equality. Since the function obtained is already bounded, we can directly consider $\phi(z) = \boldsymbol{\alpha}^\top (z\mathbf{1} - \mathbf{A})^{-1}\boldsymbol{\beta}$ as a symbol for H.

▶ **Example 21.** If we apply the formula in Equation 12 to the GPA in Example 6, we recover the rational function $\phi(z) = \frac{3z}{4z^2-1}$ found in Example 12.

3.3.2 Computation of the Optimal Symbol

We consider two auxiliary WFAs. Let $\widehat{A} = \langle \widehat{\boldsymbol{\alpha}}, \widehat{\mathbf{A}}, \widehat{\boldsymbol{\beta}} \rangle$ be a WFA with $j \geq k$ states, satisfying the following properties:

1. 1 is not an eigenvalue of $\widehat{\mathbf{A}}$
2. the automaton $E = \langle \boldsymbol{\alpha}_e, \mathbf{A}_e, \boldsymbol{\beta}_e \rangle$ is minimal, with

$$\mathbf{A}_e = \begin{pmatrix} \mathbf{A} & \mathbf{0} \\ \mathbf{0} & \widehat{\mathbf{A}} \end{pmatrix}, \quad \boldsymbol{\alpha}_e = \begin{pmatrix} \boldsymbol{\alpha} \\ -\widehat{\boldsymbol{\alpha}} \end{pmatrix}, \quad \boldsymbol{\beta}_e = \begin{pmatrix} \boldsymbol{\beta} \\ \widehat{\boldsymbol{\beta}} \end{pmatrix}. \tag{13}$$

Using the parameters of the automaton $\widehat{A}$ and a constant C, we define a function $\psi(z) = \widehat{\boldsymbol{\alpha}}^\top (z\mathbf{1} - \widehat{\mathbf{A}})^{-1}\widehat{\boldsymbol{\beta}} + C$. We remark that the poles of $\psi(z)$ correspond to the eigenvalues of $\widehat{\mathbf{A}}$, counted with their multiplicities. By assumption, 1 is not an eigenvalue of $\widehat{A}$, so $\psi(z)$ does not have any poles on the unit circle, and therefore $\psi(z) \in \mathcal{L}^\infty(\mathbb{T})$. Analogously, the function $e(z) = \phi(z) - \psi(z) = \boldsymbol{\alpha}_e^\top (z\mathbf{1} - \mathbf{A}_e)^{-1}\boldsymbol{\beta}_e - C$ is also bounded on the circle.

Our objective is to compute the parameters of $\widehat{A} = \langle \widehat{\boldsymbol{\alpha}}, \widehat{\mathbf{A}}, \widehat{\boldsymbol{\beta}} \rangle$ that make $\psi(z)$ the best approximation of $\phi(z)$ according to Theorem 18. In particular, we will use Corollary 20 to find the triple $\widehat{\boldsymbol{\alpha}}, \widehat{\mathbf{A}}, \widehat{\boldsymbol{\beta}}$ such that $\psi(z)$ satisfies Equation 10. Note that, with this purpose, the constant term $C \in H^\infty$ becomes necessary to characterize $\psi(z)$. In fact, while the H^∞-component of the symbol does not affect the Hankel norm, it plays a role in the computation of the $\mathcal{L}^\infty$-norm (in Equation 5) according to Theorem 10, so it cannot be dismissed.

The following theorem provides us with an explicit expression for the functions in the Hardy space corresponding to a σ_k-Schmidt pair.

▶ **Theorem 22.** *Let σ_k be a singular number of the Hankel operator H. The singular functions associated with the σ_k-Schmidt pair $\{\boldsymbol{\xi}_k, \boldsymbol{\eta}_k\}$ of H are:*

$$\xi_k^+(z) = \sigma_k^{-1/2} \boldsymbol{\alpha}^\top (\mathbf{1} - z\mathbf{A})^{-1} \mathbf{e}_k \tag{14}$$

$$\eta_k^-(z) = \sigma_k^{-1/2} \boldsymbol{\beta}^\top (z\mathbf{1} - \mathbf{A})^{-1} \mathbf{e}_k. \tag{15}$$

*If $\psi(z)$ is the best approximation to the symbol, then $\sigma_k^{-1} e(z)$ has modulus 1 almost everywhere on the unit circle (*i.e. *it is unimodular).*

Proof. Let $\mathbf{F}$ and $\mathbf{B}$ be the forward and backward matrices, respectively, with $\mathbf{H} = \mathbf{F}\mathbf{B}^\top$, $\mathbf{P} = \mathbf{F}^\top\mathbf{F}, \mathbf{Q} = \mathbf{B}^\top\mathbf{B}$. We consider the σ_k-Schmidt pair $\{\boldsymbol{\xi}_k, \boldsymbol{\eta}_k\}$. By definition, we have $\mathbf{H}^\top\mathbf{H}\boldsymbol{\xi}_k = \sigma_k^2\boldsymbol{\xi}_k$. Rewriting in terms of the FB factorization, we obtain:

$$\mathbf{H}^\top\mathbf{H}\boldsymbol{\xi}_k = \sigma_k^2\boldsymbol{\xi}_k \tag{16}$$
$$\mathbf{B}\mathbf{F}^\top\mathbf{F}\mathbf{B}^\top\boldsymbol{\xi}_k = \sigma_k^2\boldsymbol{\xi}_k \tag{17}$$
$$\mathbf{B}\mathbf{P}\mathbf{B}^\top\boldsymbol{\xi}_k = \sigma_k^2\boldsymbol{\xi}_k \tag{18}$$
$$\mathbf{B}\mathbf{P}\mathbf{e}_k = \sigma_k^2\boldsymbol{\xi}_k \tag{19}$$

where in the last step we set $\mathbf{e}_k = \mathbf{B}^\top\boldsymbol{\xi}_k$, to reduce the SVD problem of $\mathbf{H}$ to the one of $\mathbf{QP}$. Note that, since $\mathbf{P}$ and $\mathbf{Q}$ are diagonal, $\mathbf{e}_k$ is the k-th coordinate vector $(0, \dots, 0, 1, 0, \dots, 0)^\top$. Since $\mathbf{e}_k$ is an eigenvector of $\mathbf{QP}$ for σ_k^2, we get:

$$\mathbf{B}\mathbf{Q}^{-1}\mathbf{Q}\mathbf{P}\mathbf{e}_k = \sigma_k^2\boldsymbol{\xi}_k \tag{20}$$
$$\mathbf{B}\mathbf{Q}^{-1}\mathbf{e}_k = \boldsymbol{\xi}_k. \tag{21}$$

Moreover, $\mathbf{H}$ is symmetric, so we have that the singular vectors $\boldsymbol{\eta}_k$ and $\boldsymbol{\xi}_k$ have the same coordinates up to the sign of the corresponding eigenvalues. We obtain:

$$\xi_k^+(z) = \sum_{j=0}^{\infty} \sigma_k^{-1/2}\boldsymbol{\alpha}^\top\mathbf{A}^j\mathbf{e}_k z^j = \sigma_k^{-1/2}\boldsymbol{\alpha}^\top(\mathbf{1} - z\mathbf{A})^{-1}\boldsymbol{e}_k \tag{22}$$

$$\eta_k^-(z) = \sum_{j=0}^{\infty} \sigma_k^{-1/2}\boldsymbol{\beta}^\top\mathbf{A}^j\mathbf{e}_k z^{-j-1} = \sigma_k^{-1/2}\boldsymbol{\beta}^\top(z\mathbf{1} - \mathbf{A})^{-1}\boldsymbol{e}_k \tag{23}$$

where the singular functions have been computed following Equation 2. If r is the multiplicity of σ_k, from Corollary 20 we get the following fundamental equation:

$$(\phi(z) - \psi(z))\boldsymbol{\alpha}^\top(\mathbf{1} - z\mathbf{A})^{-1}\mathbf{V} = \sigma_k\boldsymbol{\beta}^\top(z\mathbf{1} - \mathbf{A})^{-1}\mathbf{V}$$

where $\mathbf{V} = \begin{pmatrix}\mathbf{0} & \mathbf{1}_r\end{pmatrix}^\top$ is a $n \times r$ matrix. Consequently, we obtain the unimodular function:

$$\sigma_k^{-1}e(z) = \frac{\boldsymbol{\beta}^\top(z\mathbf{1} - \mathbf{A})^{-1}\mathbf{V}}{\boldsymbol{\alpha}^\top(\mathbf{1} - z\mathbf{A})^{-1}\mathbf{V}}$$

which is unimodular, since $\boldsymbol{\alpha}_i = \operatorname{sgn}(\lambda_i)\boldsymbol{\beta}_i$. ◀

It is reasonable to wonder how the fact that $\sigma_k^{-1}e(z)$ is unimodular reflects on the structure of the WFA $E = \langle\boldsymbol{\alpha}_e, \mathbf{A}_e, \boldsymbol{\beta}_e\rangle$ associated with it. We remark that, *a priori*, the controllability and observability Gramians of E might not be well defined. The following theorem provides us with two matrices $\mathbf{P}_e$ and $\mathbf{Q}_e$ satisfying properties similar to those of the Gramians. This theorem is the analogous of a control theory result [17], rephrased in terms of WFAs. The proof, that relies on the minimality of the WFA E [39], can be found in [17].

▶ **Theorem 23** ([17]). *Consider the function $e(z) = \boldsymbol{\alpha}_e^\top(z\mathbf{1} - \mathbf{A}_e)^{-1}\boldsymbol{\beta}_e - C$ and the corresponding minimal WFA $E = \langle\boldsymbol{\alpha}_e, \mathbf{A}_e, \boldsymbol{\beta}_e\rangle$ associated with it. If $\sigma_k^{-1}e(z)$ is unimodular, then there exists a unique pair of symmetric invertible matrices $\mathbf{P}_e$ and $\mathbf{Q}_e$ satisfying:*

(a) $\mathbf{P}_e - \mathbf{A}_e\mathbf{P}_e\mathbf{A}_e^\top = \boldsymbol{\beta}_e\boldsymbol{\beta}_e^\top$
(b) $\mathbf{Q}_e - \mathbf{A}_e^\top\mathbf{Q}_e\mathbf{A}_e = \boldsymbol{\alpha}_e\boldsymbol{\alpha}_e^\top$
(c) $\mathbf{P}_e\mathbf{Q}_e = \sigma_k^2\mathbf{1}$

We can now derive the parameters of the WFA $\widehat{A} = \langle\widehat{\boldsymbol{\alpha}}, \widehat{\mathbf{A}}, \widehat{\boldsymbol{\beta}}\rangle$.

▶ **Theorem 24.** *Let $A = \langle \boldsymbol{\alpha}, \mathbf{A}, \boldsymbol{\beta} \rangle$ be a minimal WFA with n states in its SVA form, and let $\phi(z) = \boldsymbol{\alpha}^\top (z\mathbf{1} - \mathbf{A})^{-1}\boldsymbol{\beta}$ be a symbol for its Hankel operator H. Let σ_k be a singular number of multiplicity r for H, with $\sigma_0 \geq \cdots > \sigma_k = \cdots = \sigma_{k+r-1} > \sigma_{k+r} \geq \cdots \geq \sigma_{n-1} > 0$. We can partition the Gramian matrices $\mathbf{P}$, $\mathbf{Q}$ and, conformally to them, $\mathbf{A}$, $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$, as follows:*

$$\mathbf{P} = \mathbf{Q} = \begin{pmatrix} \boldsymbol{\Sigma} & \mathbf{0} \\ \mathbf{0} & \sigma_k \mathbf{1}_r \end{pmatrix}, \quad \mathbf{A} = \begin{pmatrix} \mathbf{A}_{11} & \mathbf{A}_{12} \\ \mathbf{A}_{12}^\top & \mathbf{A}_{22} \end{pmatrix}, \quad \boldsymbol{\alpha} = \begin{pmatrix} \boldsymbol{\alpha}_1 \\ \boldsymbol{\alpha}_2 \end{pmatrix}, \quad \boldsymbol{\beta} = \begin{pmatrix} \boldsymbol{\beta}_1 \\ \boldsymbol{\beta}_2 \end{pmatrix}. \tag{24}$$

where $\boldsymbol{\Sigma} \in \mathbb{R}^{(n-r)\times(n-r)}$ is the diagonal matrix containing the remaining singular numbers. Let $\mathbf{R} = \sigma_k^2 \mathbf{1}_{n-r} - \boldsymbol{\Sigma}^2$, we denote by $(\cdot)^+$ the Moore-Penrose pseudo-inverse. If the function $\psi(z) = \widehat{\boldsymbol{\alpha}}^\top (z\mathbf{1} - \widehat{\mathbf{A}})^{-1}\widehat{\boldsymbol{\beta}} + C$ is the best approximation of $\phi(z)$, then:

- *If $\boldsymbol{\alpha}_2 \neq \mathbf{0}$:*

$$\begin{cases} \widehat{\boldsymbol{\beta}} = -\widehat{\mathbf{A}}\mathbf{A}_{12}(\boldsymbol{\beta}_2^\top)^+ \\ \widehat{\boldsymbol{\alpha}} = \widehat{\mathbf{A}}^\top \mathbf{R}\mathbf{A}_{12}(\boldsymbol{\alpha}_2^\top)^+ \\ \widehat{\mathbf{A}}(\mathbf{A}_{11} - \mathbf{A}_{12}(\boldsymbol{\beta}_2^\top)^+\boldsymbol{\beta}_1^\top) = \mathbf{1} \end{cases} \tag{25}$$

- *If $\boldsymbol{\alpha}_2 = \mathbf{0}$:*

$$\begin{cases} \widehat{\boldsymbol{\beta}} = (\mathbf{1} - \widehat{\mathbf{A}}\mathbf{A}_{11})(\boldsymbol{\beta}_1^\top)^+ \\ \widehat{\boldsymbol{\alpha}} = -(\mathbf{R} - \widehat{\mathbf{A}}^\top \mathbf{R}\mathbf{A}_{11})(\boldsymbol{\alpha}_1^\top)^+ \\ \widehat{\mathbf{A}}\mathbf{A}_{12} = \mathbf{0} \end{cases} \tag{26}$$

Proof. Since $\sigma^{-1}e(z) = \phi(z) - \psi(z)$ is unimodular, from Theorem 23 there exist two symmetric nonsingular matrices $\mathbf{P}_e$, $\mathbf{Q}_e$ satisfying the fixed point equations:

$$\mathbf{P}_e - \mathbf{A}_e \mathbf{P}_e \mathbf{A}_e^\top = \boldsymbol{\beta}_e \boldsymbol{\beta}_e^\top \tag{27}$$

$$\mathbf{Q}_e - \mathbf{A}_e^\top \mathbf{Q}_e \mathbf{A}_e = \boldsymbol{\alpha}_e \boldsymbol{\alpha}_e^\top \tag{28}$$

and such that $\mathbf{P}_e\mathbf{Q}_e = \sigma_k^2\mathbf{1}$. We can partition $\mathbf{P}_e$ and $\mathbf{Q}_e$ according to the definition of $\mathbf{A}_e$ (see Eq. 13):

$$\mathbf{P}_e = \begin{pmatrix} \mathbf{P}_{11} & \mathbf{P}_{12} \\ \mathbf{P}_{12}^\top & \mathbf{P}_{22} \end{pmatrix}, \quad \mathbf{Q}_e = \begin{pmatrix} \mathbf{Q}_{11} & \mathbf{Q}_{12} \\ \mathbf{Q}_{12}^\top & \mathbf{Q}_{22} \end{pmatrix}.$$

From Equation 27 and 28, we note that $\mathbf{P}_{11}$ and $\mathbf{Q}_{11}$ corresponds to the controllability and observability Gramians of A:

$$\mathbf{P}_{11} = \mathbf{Q}_{11} = \mathbf{P} = \begin{pmatrix} \boldsymbol{\Sigma} & \mathbf{0} \\ \mathbf{0} & \sigma_k \mathbf{1} \end{pmatrix}.$$

Moreover, since $\mathbf{P}_e\mathbf{Q}_e = \sigma_k^2\mathbf{1}$, we get $\mathbf{P}_{12}\mathbf{Q}_{12}^\top = \sigma_k^2\mathbf{1} - \mathbf{P}^2$. It follows that $\mathbf{P}_{12}\mathbf{Q}_{12}^\top$ has rank $n - r$. Without loss of generality we can set $\dim \widehat{\mathbf{A}} = j = n - r$, and choose an appropriate basis for the state space such that $\mathbf{P}_{12} = \begin{pmatrix} \mathbf{1} & \mathbf{0} \end{pmatrix}^\top$ and $\mathbf{Q}_{12} = \begin{pmatrix} \mathbf{R} & \mathbf{0} \end{pmatrix}^\top$, with $\mathbf{R} = \sigma_k^2\mathbf{1} - \boldsymbol{\Sigma}^2$. Once $\mathbf{P}_{12}$ and $\mathbf{Q}_{12}$ are fixed, the values of $\mathbf{P}_{22}$ and $\mathbf{Q}_{22}$ are automatically determined. We obtain:

$$\mathbf{P}_e = \begin{pmatrix} \boldsymbol{\Sigma} & \mathbf{0} & \mathbf{1} \\ \mathbf{0} & \sigma_k\mathbf{1} & \mathbf{0} \\ \mathbf{1} & \mathbf{0} & -\boldsymbol{\Sigma}\mathbf{R}^{-1} \end{pmatrix}, \quad \mathbf{Q}_e = \begin{pmatrix} \boldsymbol{\Sigma} & \mathbf{0} & \mathbf{R} \\ \mathbf{0} & \sigma_k\mathbf{1} & \mathbf{0} \\ \mathbf{R} & \mathbf{0} & -\boldsymbol{\Sigma}\mathbf{R} \end{pmatrix}. \tag{29}$$

Now that we have an expression for the matrices $\mathbf{P}_e$ and $\mathbf{Q}_e$ of Theorem 23, we can rewrite the fixed point equations to derive the parameters $\widehat{\boldsymbol{\alpha}}$, $\widehat{\mathbf{A}}$ and $\widehat{\boldsymbol{\beta}}$. We obtain the following systems:

$$\begin{cases} \mathbf{P} - \mathbf{A}\mathbf{P}\mathbf{A} = \boldsymbol{\beta}\boldsymbol{\beta}^\top \\ \mathbf{N} - \mathbf{A}\mathbf{N}\widehat{\mathbf{A}}^\top = \boldsymbol{\beta}\widehat{\boldsymbol{\beta}}^\top \\ -\boldsymbol{\Sigma}\mathbf{R}^{-1} + \widehat{\mathbf{A}}\boldsymbol{\Sigma}\mathbf{R}^{-1}\widehat{\mathbf{A}}^\top = \widehat{\boldsymbol{\beta}}\widehat{\boldsymbol{\beta}}^\top \end{cases} \qquad \begin{cases} \mathbf{P} - \mathbf{A}\mathbf{P}\mathbf{A} = \boldsymbol{\alpha}\boldsymbol{\alpha}^\top \\ \mathbf{M} - \mathbf{A}^\top\mathbf{M}\widehat{\mathbf{A}} = -\boldsymbol{\alpha}\widehat{\boldsymbol{\alpha}}^\top \\ -\boldsymbol{\Sigma}\mathbf{R} + \widehat{\mathbf{A}}^\top\boldsymbol{\Sigma}\mathbf{R}\widehat{\mathbf{A}} = \widehat{\boldsymbol{\alpha}}\widehat{\boldsymbol{\alpha}}^\top \end{cases} \tag{30}$$

where $\mathbf{N} = \begin{pmatrix} \mathbf{1} \\ \mathbf{0} \end{pmatrix}$ and $\mathbf{M} = \begin{pmatrix} \mathbf{R} \\ \mathbf{0} \end{pmatrix}$. We can rewrite the second equation of each system as follows:

$$\begin{cases} \mathbf{1} - \mathbf{A}_{11}\widehat{\mathbf{A}}^\top = \boldsymbol{\beta}_1\widehat{\boldsymbol{\beta}}^\top \\ -\mathbf{A}_{12}^\top\widehat{\mathbf{A}}^\top = \boldsymbol{\beta}_2\widehat{\boldsymbol{\beta}}^\top \end{cases} \qquad \begin{cases} \mathbf{R} - \mathbf{A}_{11}\mathbf{R}\widehat{\mathbf{A}} = -\boldsymbol{\alpha}_1\widehat{\boldsymbol{\alpha}}^\top \\ \widehat{\mathbf{A}}^\top\mathbf{R}\mathbf{A}_{12} = \widehat{\boldsymbol{\alpha}}\boldsymbol{\alpha}_2^\top \end{cases} \tag{31}$$

If $\boldsymbol{\alpha}_2 \neq \mathbf{0}$, then also $\boldsymbol{\beta}_2 \neq \mathbf{0}$ (recall that $\boldsymbol{\alpha}_i = \operatorname{sgn}(\lambda_i)\boldsymbol{\beta}_i$), and we have:

$$\begin{cases} \widehat{\boldsymbol{\beta}} = -\widehat{\mathbf{A}}\mathbf{A}_{12}(\boldsymbol{\beta}_2^\top)^+ \\ \widehat{\boldsymbol{\alpha}} = \widehat{\mathbf{A}}^\top\mathbf{R}\mathbf{A}_{12}(\boldsymbol{\alpha}_2^\top)^+ \\ \widehat{\mathbf{A}}(\mathbf{A}_{11} - \mathbf{A}_{12}(\boldsymbol{\beta}_2^\top)^+\boldsymbol{\beta}_1^\top) = \mathbf{1} \end{cases} \tag{32}$$

with $(\boldsymbol{\alpha}_2^\top)^+ = \frac{\boldsymbol{\alpha}_2}{\boldsymbol{\alpha}_2^\top\boldsymbol{\alpha}_2}$ and $(\boldsymbol{\beta}_2^\top)^+ = \frac{\boldsymbol{\beta}_2}{\boldsymbol{\beta}_2^\top\boldsymbol{\beta}_2}$.

If $\boldsymbol{\alpha}_2 = \mathbf{0}$, we have $\widehat{\mathbf{A}}\mathbf{A}_{12} = \mathbf{0}$. We remark that $\widehat{\mathbf{A}}$ has size $(n-r)\times(n-r)$, while $\mathbf{A}_{12}$ is $(n-r)\times r$, so the system of equations corresponding to $\widehat{\mathbf{A}}\mathbf{A}_{12} = \mathbf{0}$ is underdetermined if $r < \frac{n}{2}$, in which case we can find an alternative set of solutions:

$$\begin{cases} \widehat{\boldsymbol{\beta}} = (\mathbf{1} - \widehat{\mathbf{A}}\mathbf{A}_{11})(\boldsymbol{\beta}_1^\top)^+ \\ \widehat{\boldsymbol{\alpha}} = -(\mathbf{R} - \widehat{\mathbf{A}}^\top\mathbf{R}\mathbf{A}_{11})(\boldsymbol{\alpha}_1^\top)^+ \\ \widehat{\mathbf{A}}\mathbf{A}_{12} = \mathbf{0} \end{cases} \tag{33}$$

with $\widehat{\mathbf{A}} \neq \mathbf{0}$. On the other hand, if $r \geq \frac{n}{2}$, *i.e.* if the multiplicity of the singular number σ_k is more than half the size of the original WFA, the system might not have any solution unless $\widehat{\mathbf{A}} = \mathbf{0}$ (or unless $\mathbf{A}_{12}$ was zero to begin with). In this setting, the method proposed returns $\widehat{\mathbf{A}} = \mathbf{0}$. An alternative in this case is to search for an approximation of size $k-1$ or $k+1$, so that $r < \frac{n}{2}$, and the system in Equation 33 is underdetermined. ◀

3.3.3 Extraction of the Rational Component

The function $\psi(z) = \widehat{\boldsymbol{\alpha}}^\top(z\mathbf{1} - \widehat{\mathbf{A}})^{-1}\widehat{\boldsymbol{\beta}} + C$ found in Theorem 24 corresponds to the solution of Theorem 18. To find the solution to the approximation problem we only need to "isolate" the function $g(z) \in \mathcal{R}_k$, *i.e.* the *rational component*. To do this, we study the position of the poles of $\psi(z)$, since the poles of a strictly proper rational function lie in the unit disc (Theorem 14). As noted before, we parametrized $\psi(z)$ so that its poles correspond to the eigenvalues of $\widehat{A}$. After a change of basis (detailed in Paragraph 3.3.3), we can rewrite $\widehat{\mathbf{A}}$ in block-diagonal form:

$$\widehat{\mathbf{A}} = \begin{pmatrix} \widehat{\mathbf{A}}_+ & \mathbf{0} \\ \mathbf{0} & \widehat{\mathbf{A}}_- \end{pmatrix} \tag{34}$$

where the modulus of the eigenvalues of $\widehat{\mathbf{A}}_+$ (resp. $\widehat{\mathbf{A}}_-$) is smaller (resp. greater) than one. We apply the same change of coordinates on $\widehat{\boldsymbol{\alpha}}$ and $\widehat{\boldsymbol{\beta}}$.

To conclude the study of the eigenvalues of $\widehat{\mathbf{A}}$, we need the following auxiliary result from Ostrowski [33]. A proof of this theorem can be found in [42].

▶ **Theorem 25** ([33]). *Let $|\Sigma| = 1$, and let $\mathbf{P}$ be a solution to the fixed point equation $X - \mathbf{A}X\mathbf{A}^\top = \boldsymbol{\beta}\boldsymbol{\beta}^\top$ for the WFA $A = \langle \boldsymbol{\alpha}, \mathbf{A}, \boldsymbol{\beta} \rangle$. If A is reachable, then:*

- *The number of eigenvalues λ of $\mathbf{A}$ such that $|\lambda| < 1$ is equal to the number of positive eigenvalues of $\mathbf{P}$.*
- *The number of eigenvalues λ of $\mathbf{A}$ such that $|\lambda| > 1$ is equal to the number of negative eigenvalues of $\mathbf{P}$.*

We can finally find the rational component of the function $\psi(z)$, *i.e.* the function $g(z)$ from Corollary 19 necessary to solve that approximate minimization problem.

▶ **Theorem 26.** *Let $\widehat{\mathbf{A}}_+, \widehat{\boldsymbol{\alpha}}_+, \widehat{\boldsymbol{\beta}}_+$ be as in Theorem 24. The rational component of $\psi(z)$ is the function $g(z) = \widehat{\boldsymbol{\alpha}}_+^\top (z\mathbf{1} - \widehat{\mathbf{A}}_+)^{-1} \widehat{\boldsymbol{\beta}}_+$.*

Proof. Clearly $\psi(z) = g(z) + l(z)$, with $l(z) = \widehat{\boldsymbol{\alpha}}_-^\top (z\mathbf{1} - \widehat{\mathbf{A}}_-)^{-1} \widehat{\boldsymbol{\beta}}_-$, $l \in \mathcal{H}^\infty$. To conclude the proof we need to show that $g(z)$ has k poles inside the unit disc, and therefore has rank k. We do this by studying the position of the eigenvalues of $\widehat{\mathbf{A}}_+$.

Since $\widehat{A}$ is reachable (follows from the minimality of E), we can use Theorem 25 and solve the problem by directly examining the eigenvalues of $-\boldsymbol{\Sigma}\mathbf{R}$. From the proof of Theorem 24 we have $-\boldsymbol{\Sigma}\mathbf{R} = \boldsymbol{\Sigma}(\boldsymbol{\Sigma}^2 - \sigma_k^2 \mathbf{1})$, where $\boldsymbol{\Sigma}$ is the diagonal matrix having as elements the singular numbers of H different from σ_k. It follows that $-\boldsymbol{\Sigma}\mathbf{R}$ has only k strictly positive eigenvalues, and $\widehat{\mathbf{A}}$ has k eigenvalues with modulus smaller than 1. Thus, $\widehat{\mathbf{A}}_+$ has k eigenvalues, corresponding to the poles of $g(z)$. ◀

Block Diagonalization

In this paragraph we detail the technical steps necessary to rewrite $\widehat{\mathbf{A}}$ in block-diagonal form. The problem of computing the Jordan form of a matrix is ill-conditioned, hence it is not suitable for our algorithmic purposes [40]. Instead, we compute the Schur decomposition, *i.e.* we find an orthogonal matrix $\mathbf{U}$ such that $\mathbf{U}^\top \widehat{\mathbf{A}} \mathbf{U}$ is upper triangular, with the eigenvalues of $\widehat{\mathbf{A}}$ on the diagonal. We obtain:

$$\mathbf{T} = \mathbf{U}^\top \widehat{\mathbf{A}} \mathbf{U} = \begin{pmatrix} \widehat{\mathbf{A}}_+ & \widehat{\mathbf{A}}_{12} \\ \mathbf{0} & \widehat{\mathbf{A}}_- \end{pmatrix} \tag{35}$$

where the eigenvalues are arranged in increasing order of modulus, and the modulus of those in $\widehat{\mathbf{A}}_+$ (resp. $\widehat{\mathbf{A}}_-$) is smaller (resp. greater) than one. To transform this upper triangular matrix into a block-diagonal one, we use the following result.

▶ **Theorem 27** ([38]). *The matrix $\mathbf{X}$ is a solution of the equation $\widehat{\mathbf{A}}_+\mathbf{X} - \mathbf{X}\widehat{\mathbf{A}}_- + \widehat{\mathbf{A}}_{12} = \mathbf{0}$ if and only if $\mathbf{M} = \begin{pmatrix} \mathbf{1} & \mathbf{X} \\ \mathbf{0} & \mathbf{1} \end{pmatrix}$ and $\mathbf{M}^{-1} = \begin{pmatrix} \mathbf{1} & -\mathbf{X} \\ \mathbf{0} & \mathbf{1} \end{pmatrix}$ satisfy $\mathbf{M}^{-1}\mathbf{T}\mathbf{M} = \begin{pmatrix} \widehat{\mathbf{A}}_+ & \mathbf{0} \\ \mathbf{0} & \widehat{\mathbf{A}}_- \end{pmatrix}$, where $\mathbf{T}$ is the matrix defined in Equation 35.*

Setting $\boldsymbol{\Gamma} = \begin{pmatrix} \mathbf{1}_k & \mathbf{0} \end{pmatrix}$ we can now derive the rational component of the WFA:

$$\widehat{\mathbf{A}}_+ = \boldsymbol{\Gamma}\mathbf{M}^{-1}\mathbf{U}^\top \widehat{\mathbf{A}} \mathbf{U}\mathbf{M}\boldsymbol{\Gamma}^\top, \quad \widehat{\boldsymbol{\alpha}}_+ = \boldsymbol{\Gamma}\mathbf{M}^\top \mathbf{U}^\top \widehat{\boldsymbol{\alpha}}, \quad \widehat{\boldsymbol{\beta}}_+ = \boldsymbol{\Gamma}\mathbf{M}^{-1}\mathbf{U}^\top \widehat{\boldsymbol{\beta}}. \tag{36}$$

3.3.4 Solution to the Approximation Problem

In the previous sections, we have derived the rational function $g(z)$ corresponding to the symbol of G, the operator that solves Theorem 17. To find the solution to the approximation problem we only need to find the parameters of $\widehat{A}_k$, the optimal approximating WFA. These are directly revealed by the expression of $g(z)$, thanks to the way we parametrized the functions.

▶ **Theorem 28.** *Let $A = \langle \boldsymbol{\alpha}, \mathbf{A}, \boldsymbol{\beta} \rangle$ be a minimal WFA with n states over a one-letter alphabet. Let A be in its SVA form. The optimal spectral-norm approximation of rank k is given by the WFA $\widehat{A}_k = \langle \widehat{\boldsymbol{\alpha}}_+, \widehat{\mathbf{A}}_+, \widehat{\boldsymbol{\beta}}_+ \rangle$.*

Proof. From Corollary 19 we know that $g(z)$ is the rational function associated with the Hankel matrix of the best approximation. Given the correspondence between the Fourier coefficients of $g(z)$ and the entries of the matrix (Remark 16), we have:

$$g(z) = \widehat{\boldsymbol{\alpha}}_+^\top (z\mathbf{1} - \widehat{\mathbf{A}}_+)^{-1} \widehat{\boldsymbol{\beta}}_+ = \sum_{k \geq 0} \widehat{\boldsymbol{\alpha}}_+^\top \widehat{\mathbf{A}}_+^k \widehat{\boldsymbol{\beta}}_+ z^{-k-1} = \sum_{k \geq 0} \bar{f}(k) z^{-k-1} \tag{37}$$

where $\bar{f} : \Sigma^* \to \mathbb{R}$ is the function computed by $\widehat{A}_k$, and $\widehat{\boldsymbol{\alpha}}_+, \widehat{\mathbf{A}}_+, \widehat{\boldsymbol{\beta}}_+$ are the parameters. ◀

3.4 Error Analysis

The theoretical foundations of AAK theory guarantee that the construction detailed in Section 3.3 produces the rank k optimal spectral-norm approximation of a WFA satisfying our assumptions, and the singular number σ_k provides the exact error.

Similarly to the case of SVA truncation [12], due to the ordering of the singular numbers, the error decreases when k increases, meaning that allowing $\widehat{A}_k$ to have more states guarantees a better approximation of A. We remark that, while the solution we propose is optimal in the spectral norm, the same is not necessarily true in other norms. Nonetheless, we have the following bound between ℓ^2-norm and spectral-norm.

▶ **Theorem 29.** *Let A be a minimal WFA computing $f : \Sigma^* \to \mathbb{R}$, with matrix $\mathbf{H}$. Let $\widehat{A}_k$ be its optimal spectral-norm approximation, computing $g : \Sigma^* \to \mathbb{R}$, with matrix $\mathbf{G}$. Then:*

$$\|f - g\|_{\ell^2} \leq \|\mathbf{H} - \mathbf{G}\| = \sigma_k. \tag{38}$$

Proof. Let $\mathbf{e}_0 = \begin{pmatrix} 1 & 0 & \cdots \end{pmatrix}^\top$, $f : \Sigma^* \to \mathbb{R}$, $g : \Sigma^* \to \mathbb{R}$ with Hankel matrices $\mathbf{H}$ and $\mathbf{G}$, respectively. We have:

$$\|f-g\|_{\ell^2} = \left(\sum_{n=0}^{\infty} |f_n - g_n|^2 \right)^{1/2} = \|(\mathbf{H}-\mathbf{G})\mathbf{e}_0\|_{\ell^2} \leq \sup_{\|\mathbf{x}\|_{\ell^2}=1} \|(\mathbf{H}-\mathbf{G})\mathbf{x}\|_{\ell^2} = \|\mathbf{H}-\mathbf{G}\| = \sigma_k$$

where the second equation follows by definition and by observing that matrix difference is computed entry-wise. ◀

4 Algorithm

In this section we describe the algorithm for spectral-norm approximate minimization. The algorithm takes as input a target number of states $k < n$, a minimal WFA A with $\rho(\mathbf{A}) < 1$, $\boldsymbol{\alpha}_2 \neq 0$, n states and in SVA form, and its Gramian $\mathbf{P}$. Note that, in the case of $\boldsymbol{\alpha}_2 = 0$, it is

Algorithm 1 `AAKapproximation`.

input : A minimal WFA A, with $\boldsymbol{\alpha}_2 \neq 0$, n states and in SVA form, its Gramian $\mathbf{P}$, a target number of states $k < n$
output : A WFA $\widehat{A}_k$ with k states

Let $\boldsymbol{\alpha}_1, \boldsymbol{\alpha}_2, \boldsymbol{\beta}_1, \boldsymbol{\beta}_2, \mathbf{A}_{11}, \mathbf{A}_{12}, \mathbf{A}_{22}, \boldsymbol{\Sigma}$ be the blocks defined in Eq. 24
Let $(\boldsymbol{\alpha}_2^\top)^+ = \frac{\boldsymbol{\alpha}_2}{\boldsymbol{\alpha}_2^\top \boldsymbol{\alpha}_2}$, $(\boldsymbol{\beta}_2^\top)^+ = \frac{\boldsymbol{\beta}_2}{\boldsymbol{\beta}_2^\top \boldsymbol{\beta}_2}$
Let $\mathbf{R} = \sigma_k^2 \mathbf{1} - \boldsymbol{\Sigma}^2$
Let $\widehat{\mathbf{A}} = (\mathbf{A}_{11} - \mathbf{A}_{12}(\boldsymbol{\beta}_2^\top)^+ \boldsymbol{\beta}_1^\top)^{-1}$
Let $\widehat{\boldsymbol{\alpha}} = \widehat{\mathbf{A}}^\top \mathbf{R} \mathbf{A}_{12} (\boldsymbol{\alpha}_2^\top)^+$
Let $\widehat{\boldsymbol{\beta}} = -\widehat{\mathbf{A}} \mathbf{A}_{12} (\boldsymbol{\beta}_2^\top)^+$
Let $\widehat{A} = \langle \widehat{\boldsymbol{\alpha}}, \widehat{\mathbf{A}}, \widehat{\boldsymbol{\beta}} \rangle$
Let $\widehat{A}_k \leftarrow$ `BlockDiagonalize`$(\widehat{A})$
return $\widehat{A}_k$

Algorithm 2 `BlockDiagonalize`.

input : A WFA $\widehat{A}$
output : A WFA $\widehat{A}_k$ wit $\rho < 1$

Compute the Schur decomposition of $\widehat{\mathbf{A}} = \mathbf{U}\mathbf{T}\mathbf{U}^\top$, where $|T_{11}| \leq |T_{22}| \leq \dots$
Solve $\widehat{\mathbf{A}}_{11}\mathbf{X} - \mathbf{X}\widehat{\mathbf{A}}_{22} + \widehat{\mathbf{A}}_{12} = \mathbf{0}$ for $\mathbf{X}$
Let $\mathbf{M} = \begin{pmatrix} \mathbf{1} & \mathbf{X} \\ \mathbf{0} & \mathbf{1} \end{pmatrix}$ and $\mathbf{M}^{-1} = \begin{pmatrix} \mathbf{1} & -\mathbf{X} \\ \mathbf{0} & \mathbf{1} \end{pmatrix}$
Let $\boldsymbol{\Gamma} = \begin{pmatrix} \mathbf{1}_k & \mathbf{0} \end{pmatrix}$
Let $\widehat{\mathbf{A}}_+ = \boldsymbol{\Gamma}\mathbf{M}^{-1}\mathbf{U}^\top \widehat{\mathbf{A}} \mathbf{U} \mathbf{M} \boldsymbol{\Gamma}^\top$
Let $\widehat{\boldsymbol{\alpha}}_+ = \boldsymbol{\Gamma}\mathbf{M}^\top \mathbf{U}^\top \widehat{\boldsymbol{\alpha}}$
Let $\widehat{\boldsymbol{\beta}}_+ = \boldsymbol{\Gamma}\mathbf{M}^{-1}\mathbf{U}^\top \widehat{\boldsymbol{\beta}}$
Let $\widehat{A}_k = \langle \widehat{\boldsymbol{\alpha}}_+, \widehat{\mathbf{A}}_+, \widehat{\boldsymbol{\beta}}_+ \rangle$
return $\widehat{A}_k$

enough to substitute the Steps $4, 5, 6$ with the analogous from Equation 26. The constraints on the WFA A to be minimal and in SVA form are non essential. In fact a WFA with n states can be minimized in time $O(n^3)$ [15], and the SVA computed in $O(n^3)$ [12].

Using the results of Theorem 24, we outline in Algorithm 1, `AAKapproximation`, the steps necessary to extract the best spectral-norm approximation of a WFA.

The algorithm involves a call to Algorithm 2, `BlockDiagonalize`. This corresponds to the steps, outlined in Paragraph 3.3.3, necessary to derive the WFA $\widehat{A}_k$ corresponding to the rational function $g(z)$. We remark that Step 2 in `BlockDiagonalize` can be performed using the Bartels-Stewart algorithm [14].

To compute the computational cost we recall the following facts [40]:

- The product of two $n \times n$ matrices can be computed in time $O(n^3)$ using a standard iterative algorithm, but can be reduced to $O(n^\omega)$ with $\omega < 2.4$.
- The inversion of a $n \times n$ matrix can be computed in time $O(n^3)$ using Gauss-Jordan elimination, but can be reduced to $O(n^\omega)$ with $\omega < 2.4$.
- The computation of the Schur decomposition of a $n \times n$ matrix can be done with a two-step algorithm, where each step takes $O(n^3)$, using the Hessenberg form of the matrix.
- The Bartels-Stewart algorithm applied to upper triangular matrices to find a matrix $m \times n$ takes $O(mn^2 + nm^2)$.

The running time of `BlockDiagonalize` with input a WFA $\widehat{A}$ with $(n-r)$ states is thus in $O((n-r)^3)$, where r is the multiplicity of the singular value considered. The running time of `AAKapproximation` for an input WFA $\widehat{A}$ with n states is in $O((n-r)^3)$.

5 Possible Extensions

5.1 Relaxing the Spectral Radius Assumption

It is possible to extend part of our method to WFAs over a one-letter alphabet with $\rho(\mathbf{A}) \neq 1$, but the approximation recovered is not optimal in the spectral norm.

Let $A = \langle \boldsymbol{\alpha}, \mathbf{A}, \boldsymbol{\beta} \rangle$, with $\rho(\mathbf{A}) \neq 1$, be a WFA with n states that we want to minimize. The idea is to block-diagonalize $\mathbf{A}$ like we did in Section 3.3.3, and tackle each component separately. The case of $A_+ = \langle \boldsymbol{\alpha}_+, \mathbf{A}_+, \boldsymbol{\beta}_+ \rangle$, the component having $\rho(\mathbf{A}) < 1$, can be dealt with in the way presented in the previous sections. This means that we can find an optimal spectral-norm approximation of the desired size for A_+. Now we can consider the second component, $A_- = \langle \boldsymbol{\alpha}_-, \mathbf{A}_-, \boldsymbol{\beta}_- \rangle$. The key idea is to apply the transformation $z^{j-1} \mapsto z^{-j}$ for $j \geq 1$ to the function $\phi''(z)$ associated to A_-. Then, the function

$$\phi''(z^{-1}) = \sum_{k \geq 0} \boldsymbol{\alpha}_-^\top \mathbf{A}_-^k z^k \boldsymbol{\beta}_- = \boldsymbol{\alpha}_-^\top (\mathbf{1} - z\mathbf{A}_-)^{-1} \boldsymbol{\beta}_- \tag{39}$$

is well defined, as the series converges for z with small enough modulus. Using this transformation we obtain a function with poles inside the unit disc and we can apply the method presented in the paper. An important choice to make is the size of the approximation of A_-, as it can influence the quality of the approximation. Analyzing the effects of this parameter on the approximation error constitutes an interesting direction for future work, both in the theoretical and experimental side. We refer the reader to the control theory literature [21], where some theoretical work has been done to study an analogous approach for continuous time systems and their approximation error.

5.2 Polynomial method

We remark that Equation 10 from Corollary 20 can be rewritten as

$$\psi(z) = \phi(z) - \frac{H\xi_k^+(z)}{\xi_k^+(z)}, \tag{40}$$

where $\xi_k^+(z)$ is the function in $\mathcal{H}^2$ associated to the vector $\boldsymbol{\xi}_k \in \mathrm{Ker}(\mathbf{H}^*\mathbf{H} - \sigma_k^2 \mathbf{1})$ (and $\psi(z)$ does not depend on the choice of the specific $\boldsymbol{\xi}_k$). There is an alternative way to find the best approximation, particularly useful when the objective is to approximate a finite-rank infinite Hankel matrix with another Hankel matrix, without necessarily extract a WFA. We can consider the adjoint operator H^* and its matrix $\mathbf{H}^*$. The singular numbers and singular vectors of H correspond to the eigenvalues and eigenvectors of $\mathbf{R} = (\mathbf{H}^*\mathbf{H})^{1/2}$. Hence, it is possible to compute σ_k and a corresponding singular vector $\boldsymbol{\xi}_k$. The function $\xi_k^+(z)$ is then obtained following Equation 2. Then, the Hankel matrix $\mathbf{G}$ that best approximates $\mathbf{H}$ is given by $\mathbf{G} = \mathbf{H} - \mathbf{M}$, where $\mathbf{M}$ is the Hankel matrix having $\frac{H\xi_k^+(z)}{\xi_k^+(z)}$ as symbol.

6 Related Work

The study of approximate minimization for WFAs is very recent, and only a few works have been published on the subject. In [11, 12] the authors present an approximate minimization technique using a canonical expression for WFAs, and provide bounds on the error in the

ℓ^2 norm. The result is supported by strong theoretical guarantees, but it is not optimal in any norm. An extension of this method to the case of Weighted Tree Automata can be found in [13]. A similar problem is addressed in [26], with less general results. In [27], the authors connect spectral learning to the approximate minimization problem of a small class of Hidden Markov models, bounding the error in terms of the total variation distance.

The control theory community has largely studied approximate minimization in the context of linear time-invariant systems [3]. A parallel with those results can be drown by noting that the impulse response of a discrete Single-Input-Single-Output SISO system can be parametrized as a WFA over a one-letter alphabet. In [21] Glover presents a state-space solution for the case of continuous Multi-Input-Multi-Output MIMO systems. His method led to a widespread application of these results, thanks to its computational and theoretical simplicity. This stems from the structure of the continuous Lyapunov equations. For discrete systems, though, the quadratic nature of the Lyapunov equations does not allow for a simple closed form formula for the state space solution [17]. Thus, most of the results for the discrete case work with a suboptimal version of the problem [6, 2, 24]. A solution for the SISO case can be found using a polynomial approach, but it does not provide an explicit representation of the state space nor it generalizes to the MIMO setting. The first to actually extend Glover results is Gu, who provides an elegant solution for the MIMO discrete problem [22]. Glover and Gu's solutions rely on building an *all-pass system*, equivalent to the WFA E in our case. Part of our contribution is the adaptation of some of the control theory tools to WFAs.

7 Conclusion

In this paper we applied the AAK theory for Hankel operators and complex functions to the framework of WFAs in order to construct the best possible approximation to an automaton given a bound on the size. We provide an algorithm to find the parameters of the best WFA approximation in the spectral norm, and bounds on the error. Our method applies to real WFAs $A = \langle \boldsymbol{\alpha}, \mathbf{A}, \boldsymbol{\beta} \rangle$, defined over a one-letter alphabet, with $\rho(\mathbf{A}) < 1$. While this setting is certainly restricted, we believe that this work constitutes a first fundamental step towards optimal approximation. Furthermore, the use of AAK techniques has proven to be very fruitful in related areas like control theory; we think that automata theory can also benefit from it. The use of such methods can help deepen the understanding of the behaviour of rational functions. This paper highlights and strengthens the interesting connections between functional analysis, automata theory and control theory, unifying tools from different domains in one formalism.

A compelling direction for future work is to extend our results to the multi-letter case. The work of Adamyan, Arov and Krein provides us with a powerful theory connecting sequences to the study of complex functions. We note that, unfortunately, this approach cannot be directly generalized to the multi-letter case because of the non-commutative nature of the monoid considered. Extending this work would require adapting standard harmonic analysis results to the non-abelian case. A recent line of work in functional analysis is centered around extending this theory to the case of non-commutative operators, and in [35] a non-commutative version of the AAK theorem is presented. However, those results are non-constructive, making this direction, already challenging, even harder to pursue.

References

1. Vadim M. Adamyan, Damir Zyamovich Arov, and Mark Grigorievich Krein. Analytic Properties of Schmidt Pairs for a Hankel Operator and the Generalized Schur–Takagi problem. *Mathematics of The Ussr-sbornik*, 15:31–73, 1971.
2. M.M Al-Hussari, I.M. Jaimoukha, and D.J.N. Limebeer. A Descriptor Approach for the Solution of the One-Block Distance Problem. In *In Proceedings of the IFAC World Congress*, 1993.
3. Athanasios C. Antoulas. *Approximation of Large-Scale Dynamical Systems*. SIAM, 2005.
4. Stéphane Ayache, Rémi Eyraud, and Noé Goudian. Explaining Black Boxes on Sequential Data Using Weighted Automata. In *Proceedings of the 14th International Conference on Grammatical Inference, ICGI 2018, Wrocław, Poland, September 5-7, 2018*, volume 93 of *Proceedings of Machine Learning Research*, pages 81–103. PMLR, 2018. URL: `http://proceedings.mlr.press/v93/ayache19a.html`.
5. Raphaël Bailly, François Denis, and Liva Ralaivola. Grammatical Inference as a Principal Component Analysis Problem. In *Proceedings of the 26th Annual International Conference on Machine Learning*, ICML '09, pages 33—40, New York, NY, USA, 2009. Association for Computing Machinery. `doi:10.1145/1553374.1553379`.
6. Joseph A. Ball and Andre CM. Ran. Optimal Hankel Norm Model Reductions and Wiener–Hopf Factorization I: The Canonical Case. *SIAM Journal on Control and Optimization*, 25(2):362–382, 1987.
7. Borja Balle, Xavier Carreras, Franco M. Luque, and Ariadna Quattoni. Spectral Learning of Weighted Automata - A Forward–Backward Perspective. *Mach. Learn.*, 96(1-2):33–63, 2014. `doi:10.1007/s10994-013-5416-x`.
8. Borja Balle, Pascale Gourdeau, and Prakash Panangaden. Bisimulation Metrics and Norms for Weighted Finite Automata. In Ioannis Chatzigiannakis, Piotr Indyk, Fabian Kuhn, and Anca Muscholl, editors, *44th International Colloquium on Automata, Languages, and Programming, ICALP 2017, July 10-14, 2017, Warsaw, Poland*, volume 80 of *LIPIcs*, pages 103:1–103:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2017. `doi:10.4230/LIPIcs.ICALP.2017.103`.
9. Borja Balle, William L. Hamilton, and Joelle Pineau. Methods of moments for learning stochastic languages: Unified presentation and empirical comparison. In *Proceedings of the 31th International Conference on Machine Learning, ICML 2014, Beijing, China, 21-26 June 2014*, volume 32 of *JMLR Workshop and Conference Proceedings*, pages 1386–1394. JMLR.org, 2014. URL: `http://proceedings.mlr.press/v32/balle14.html`.
10. Borja Balle, Clara Lacroce, Prakash Panangaden, Doina Precup, and Guillaume Rabusseau. Optimal Spectral–Norm Approximate Minimization of Weighted Finite Automata. *arXiv preprint*, 2021. `arXiv:2102.06860`.
11. Borja Balle, Prakash Panangaden, and Doina Precup. A Canonical Form for Weighted Automata and Applications to Approximate Minimization. In *30th Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2015, Kyoto, Japan, July 6-10, 2015*, pages 701–712. IEEE Computer Society, 2015. `doi:10.1109/LICS.2015.70`.
12. Borja Balle, Prakash Panangaden, and Doina Precup. Singular Value Automata and Approximate Minimization. *Math. Struct. Comput. Sci.*, 29(9):1444–1478, 2019. `doi:10.1017/S0960129519000094`.
13. Borja Balle and Guillaume Rabusseau. Approximate minimization of weighted tree automata. *Information and Computation*, page 104654, 2020.
14. R. H. Bartels and G. W. Stewart. Solution of the matrix equation ax + xb = c [f4]. *Commun. ACM*, 15(9):820—-826, 1972.
15. Jean Berstel and Christophe Reutenauer. *Noncommutative rational series with applications*, volume 137. Cambridge University Press, 2011.
16. J.W. Carlyle and A. Paz. Realizations by stochastic finite automata. *Journal of Computer and System Sciences*, 5(1):26–40, 1971.

17 Charles K. Chui and Guanrong Chen. *Discrete H^∞ Optimization With Applications in Signal Processing and Control Systems*. Springer-Verlag, 1997.

18 C. Eckart and G. Young. The approximation of one matrix by another of lower rank. *Psychometrika*, 1:211–218, 1936. doi:10.1007/BF02288367.

19 Rémi Eyraud and Stéphane Ayache. Distillation of Weighted Automata from Recurrent Neural Networks Using a Spectral Approach. *CoRR*, abs/2009.13101, 2020. arXiv:2009.13101.

20 M. Fliess. Matrice de Hankel. *Journal de Mathématique Pures et Appliquées*, 5:197–222, 1974.

21 Keith Glover. All optimal Hankel-norm approximations of linear multivariable systems and their $\mathcal{L}_\infty$-error bounds. *International Journal of Control*, 39(6):1115–1193, 1984. doi:10.1080/00207178408933239.

22 Guoxiang Gu. All optimal Hankel-norm approximations and their error bounds in discrete-time. *International Journal of Control*, 78(6):408–423, 2005. doi:10.1080/00207170500110988.

23 Daniel Hsu, Sham M. Kakade, and Tong Zhang. A spectral algorithm for learning hidden markov models. *J. Comput. Syst. Sci.*, 78(5):1460–1480, 2012. doi:10.1016/j.jcss.2011.12.025.

24 Vlad Ionescu and Cristian Oara. The four-block Adamjan-Arov-Kein problem for discrete-time systems. In *Linear Algebra and its Application*, pages 95–119. Elsevier, 2001.

25 L. Kronecker. Zur Theorie der Elimination einer Variablen aus zwei algebraischen Gleichungen. *Montasber. Königl. Preussischen Acad Wies*, pages 535–600, 1881.

26 Alex Kulesza, Nan Jiang, and Satinder Singh. Low-Rank Spectral Learning with Weighted Loss Functions. In *Artificial Intelligence and Statistics*, pages 517–525. PMLR, 2015.

27 Alex Kulesza, N. Raj Rao, and Satinder Singh. Low-Rank Spectral Learning. In Samuel Kaski and Jukka Corander, editors, *Proceedings of the Seventeenth International Conference on Artificial Intelligence and Statistics*, volume 33 of *Proceedings of Machine Learning Research*, pages 522–530, Reykjavik, Iceland, April 22–25 2014. PMLR. URL: http://proceedings.mlr.press/v33/kulesza14.html.

28 Aersity M Lyapunov. The General Problem of the Stability of Motion [in Russian]. *Gostekhizdat, Moscow*, 1950.

29 Jean Meinguet. A Simplified Presentation of the Adamjan-Arov-Krein Approximation Theory. In H. Werner, L. Wuytack, E. Ng, and H. J. Bünger, editors, *Computational Aspects of Complex Analysis*, pages 217–248, Dordrecht, 1983. Springer Netherlands. doi:10.1007/978-94-009-7121-9_9.

30 Zeev Nehari. On Bounded Bilinear Forms. *Annals of Mathematics*, 65(1):153–162, 1957.

31 Nikolai K. Nikol'Skii. *Operators, Functions and Systems: An Easy Reading*, volume 92 of *Mathematical Surveys and Monographs*. American Mathematical Society, 2002.

32 Takamasa Okudono, Masaki Waga, Taro Sekiyama, and Ichiro Hasuo. Weighted Automata Extraction from Recurrent Neural Networks via Regression on State Spaces. In *The Thirty-Fourth AAAI Conference on Artificial Intelligence, 2020, The Thirty-Second Innovative Applications of Artificial Intelligence Conference, IAAI 2020, The Tenth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2020, New York, NY, USA, February 7-12, 2020*, pages 5306–5314. AAAI Press, 2020. URL: https://aaai.org/ojs/index.php/AAAI/article/view/5977.

33 Alexander Ostrowski and Hans Schneider. Some Theorems on the Inertia of General Matrices. *J. Math. Anal. Appl*, 4(1):72–84, 1962.

34 Vladimir Peller. *Hankel Operators and their Applications*. Springer Science & Business Media, 2012.

35 Gelu Popescu. Multivariable Nehari Problem and Interpolation. *Journal of Functional Analysis*, 200:536–581, 2003. doi:10.1016/S0022-1236(03)00078-8.

36 Guillaume Rabusseau, Borja Balle, and Joelle Pineau. Multitask Spectral Learning of Weighted Automata. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, pages 2585–2594, 2017.

37 Guillaume Rabusseau, Tianyu Li, and Doina Precup. Connecting Weighted Automata and Recurrent Neural Networks through Spectral Learning. In Kamalika Chaudhuri and Masashi Sugiyama, editors, *The 22nd International Conference on Artificial Intelligence and Statistics, AISTATS 2019, 16-18 April 2019, Naha, Okinawa, Japan*, volume 89 of *Proceedings of Machine Learning Research*, pages 1630–1639. PMLR, 2019. URL: http://proceedings.mlr.press/v89/rabusseau19a.html.

38 William E. Roth. The equations ax - yb = c and ax - xb = c in matrices. *Proceedings of the American Mathematical Society*, 3(3):392–396, 1952.

39 B.De Schutter. Minimal State-Space Realization in Linear System Theory: an Overview. *Journal of Computational and Applied Mathematics*, 121(1):331–354, 2000. doi:10.1016/S0377-0427(00)00341-1.

40 Lloyd N Trefethen and David Bau III. *Numerical linear algebra*, volume 50. Siam, 1997.

41 Gail Weiss, Yoav Goldberg, and Eran Yahav. Learning Deterministic Weighted Automata with Queries and Counterexamples. In Hanna M. Wallach, Hugo Larochelle, Alina Beygelzimer, Florence d'Alché-Buc, Emily B. Fox, and Roman Garnett, editors, *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*, pages 8558–8569, 2019. URL: https://proceedings.neurips.cc/paper/2019/hash/d3f93e7766e8e1b7ef66dfdd9a8be93b-Abstract.html.

42 H.K Wimmer. On the Ostrowski-Schneider Inertia Theorem. *Journal of Mathematical Analysis and Applications*, 41(1):164–169, 1973. doi:10.1016/0022-247X(73)90190-X.

43 Kehe Zhu. *Operator Theory in Function Spaces*, volume 138. American Mathematical Society, 1990.

Property Testing of Regular Languages with Applications to Streaming Property Testing of Visibly Pushdown Languages

Gabriel Bathie
École normale supérieure de Lyon, France

Tatiana Starikovskaya
DIENS, École normale supérieure de Paris, PSL Research University, France

Abstract

In this work, we revisit the problem of testing membership in regular languages, first studied by Alon et al. [1]. We develop a one-sided error property tester for regular languages under weighted edit distance that makes $\mathcal{O}(\varepsilon^{-1}\log(1/\varepsilon))$ non-adaptive queries, assuming that the language is described by an automaton of constant size. Moreover, we show a matching lower bound, essentially closing the problem for the edit distance. As an application, we improve the space bound of the current best streaming property testing algorithm for visibly pushdown languages from $\mathcal{O}(\varepsilon^{-4}\log^6 n)$ to $\mathcal{O}(\varepsilon^{-3}\log^5 n \log\log n)$, where n is the size of the input. Finally, we provide a $\bar{\Omega}(\max(\varepsilon^{-1}, \log n))$ lower bound on the memory necessary to test visibly pushdown languages in the streaming model, significantly narrowing the gap between the known bounds.

2012 ACM Subject Classification Theory of computation → Regular languages

Keywords and phrases property testing, regular languages, streaming algorithms, visibly pushdown languages

Digital Object Identifier 10.4230/LIPIcs.ICALP.2021.119

Category Track B: Automata, Logic, Semantics, and Theory of Programming

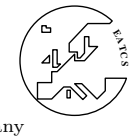
Funding This work was partially funded by the grants ANR-19-CE48-0016 and ANR-20-CE48-0001 from the French National Research Agency (ANR).

1 Introduction

A one-sided error ε-property tester for a language L is a randomised algorithm that accepts an input u of length n if $u \in L$ with probability one, and rejects if the distance from u to L is at least εn with probability $\geq 2/3$. In the property testing setting, we do not have access to the whole input u, but instead must take the decision by querying as few symbols of the input as possible. The number of queried symbols is called the query complexity of the tester.

The study of property testing of formal languages was initiated in the seminal paper of Alon et al. [1], who showed a property tester for regular languages as well as lower bounds for context-free languages. Property testing of regular languages has been also studied in [6, 17, 19, 20]. Apart from regular languages, there is a series of work that studied the question of testing membership in $\textsc{Dyck}(s)$, the language of well-parenthesized expressions with s types of parentheses [1, 4, 21]. When the distance allows sufficient modifications of the input, such as moves of arbitrarily large factors, it was shown that any context-free language is testable with a constant number of queries [3].

In this work, we revisit the problem of property testing of regular languages. Recall that the Hamming distance is the number of mismatches between two equal-length strings, and the edit distance between two strings is the smallest number of insertions, deletions, and substitutions required to convert one string into another. The weighted edit distance is a



generalisation of the edit distance for weighted words (see Section 2 for a definition). Fix a regular language specified by an automaton with m states and k connected components. The first property tester for regular languages was given by Alon et al. [1]. The tester of Alon et al. is for the Hamming distance, it queries $\mathcal{O}(k^2 m \cdot \varepsilon^{-1} \log^3(m/\varepsilon))$ symbols of the input, and its runtime is exponential in the size of the automaton that defines the regular language. Alon et al. [1] also showed a $\bar{\Omega}(1/\varepsilon)$ lower bound for such testers[1]. Magniez and De Rougemont [17] built upon [1] to show a property tester for regular languages under the edit distance with moves. The query complexity of their algorithm is $\mathcal{O}(m^3 \cdot \varepsilon^{-1} \log^2(m/\varepsilon))$ and its running time exponential. Later, Fischer et al. [3] improved the query complexity to remove the dependency on m. Ndione et al. [19, 20] continued this line of work with a goal of devising property testers that run in polynomial time, both for the Hamming distance and the edit distance. Their tester for the Hamming distance has query complexity $\mathcal{O}(k^2 m^4 \cdot \varepsilon^{-1} \log^3(km^4/\varepsilon))$ and runtime $\mathcal{O}(k^2 m^{10} \cdot \varepsilon^{-1} \log^3(km^4/\varepsilon))$. Unfortunately, their tester for the edit distance, which is of more interest to us, contains a fatal error in one of the key lemmas, which is why we do not give its complexities here[2]. Finally, François et al. [5, 6] gave a tester for regular languages under the weighted edit distance with query complexity $\mathcal{O}(m^3 \cdot \varepsilon^{-2})$ and $\mathcal{O}(km^5 \cdot \varepsilon^{-2})$ running time.

Table 1 Summary of property testing algorithms for regular languages, assuming that a regular language is described by an automaton on a constant-size alphabet Σ with m states and k strongly connected components.

	Queries	**Time**	**Distance**
Alon et al. [1]	$\mathcal{O}(k^2 m \cdot \varepsilon^{-1} \log^3(m/\varepsilon))$	$\mathcal{O}(2^{m^2} + \varepsilon^{-k})$	Hamming
Magniez et De Rougemont [17]	$\mathcal{O}(m^3 \cdot \varepsilon^{-1} \log^2(m/\varepsilon))$	$\mathcal{O}(2^m m^5 \cdot \varepsilon^{-1} \log^2(m/\varepsilon))$	edit w. moves
Fischer et al. [3]	$\mathcal{O}(\frac{\Sigma^{2/\varepsilon} \log \lvert\Sigma\rvert}{\varepsilon^4})$	$m^{\lvert\Sigma\rvert^{\mathcal{O}(1/\varepsilon)}}$	edit w. moves
Ndione et al. [19, 20]	$\mathcal{O}(k^2 m^4 \cdot \varepsilon^{-1} \log^3(km^4/\varepsilon))$	$\mathcal{O}(k^2 m^{10} \cdot \varepsilon^{-1} \log^3(km^4/\varepsilon))$	Hamming
François et al. [6]	$\mathcal{O}(m^3 \cdot \varepsilon^{-2})$	$\mathcal{O}(km^5 \cdot \varepsilon^{-2})$	weight. edit
This work	$\mathcal{O}(km \cdot \varepsilon^{-1} \log(m/\varepsilon))$	$\mathcal{O}(km^3 \cdot \varepsilon^{-1} \log(m/\varepsilon))$	weight. edit

Our main contribution is a tight bound on query complexity of property testing of regular languages under the weighted edit distance. First, we show a new property tester with query complexity $\mathcal{O}(km \cdot \varepsilon^{-1} \log(m/\varepsilon))$ and time complexity $\mathcal{O}(km^3 \cdot \varepsilon^{-1} \log(m/\varepsilon))$(Theorem 5). Essentially, our tester is very simple: it samples a set of short factors of the input string u, and checks whether the factors can be complemented to a word that belongs to the given regular language L. The analysis is much more involved. Our inspiration originates from the ideas from [5, 19, 20], which we extend in a non-trivial way to show a better (and correct) bound. In Theorem 15, we complement the upper bound with a matching lower bound. As the Hamming distance is always larger than the edit distance, the bound also holds for the Hamming distance, improving the lower bound by Alon et al. [1]. See Table 1 for a summary of previously known results and comparison with our work.

[1] We use the following asymptotic notation: For functions $f, g : \mathbb{N} \to \mathbb{R}_{\geq 0}$, $f(n) = \bar{\Omega}(g(n))$ holds if $f(n) \geq c \cdot g(n)$ for some $c > 0$ and infinitely many $n \in \mathbb{N}$.

[2] The error is in [19, Lemma 12], namely, the value l' chosen in the last two sentences of the proof does not necessarily exist (confirmed via personal communication with the authors).

As an almost straightforward application of our result, in Section 4 we plug our property tester into the algorithm of François et al. [5] to show an improved space bound for streaming property testing for the class of visibly pushdown languages (VPL) (that, in particular, contains regular languages and $\textsc{Dyck}(s)$). Informally, a streaming property tester for a language L is an algorithm that receives an input word u of length n as a stream, one symbol at a time, and must accept if $u \in L$ with probability one and reject if the distance from u to L is at least εn with probability at least $2/3$. Assuming that the automaton that specifies a VPL L is of constant size, the streaming property tester of François et al. [6] requires $\mathcal{O}(\varepsilon^{-4}\log^6 n)$ bits of space. Our result improves this bound to $\mathcal{O}(\varepsilon^{-3}\log^5 n \log\log n)$ (Corollary 22). François et al. [5] showed that for $\varepsilon = 0$, a streaming property tester for visibly pushdown languages must use $\Omega(n)$ bits, even when randomisation is allowed. As our final contribution, we show a space lower bound of $\tilde{\Omega}(\max(\varepsilon^{-1}, \log n))$ bits (Theorem 23), thus narrowing the gap between the best existing bounds by $\log^2 n/\log\log n$.

Apart from VPL, the problem of testing membership in formal languages in the streaming setting has been studied for $\textsc{Dyck}(s)$ [14, 16, 18], and for DLIN and $\mathsf{LL}(k)$ [2]. A variant of the streaming setting, called the sliding window model, where one must decide the membership for the n-length suffix of the stream after each symbol arrival, has been considered for regular languages [9, 10, 11, 12], VPL [8], and context-free languages [13].

2 Preliminaries

An alphabet Σ is a finite set the elements of which are called symbols. The length of a word u, denoted $|u|$, is the number of symbols in u. For $1 \le i \le j \le |u|$, we let $u[i]$ denote the i-th symbol in u, and $v = u[i,j]$ the word $u[i]\dots u[j]$, which we call a *factor* of u. A factor of length l is called an *l-factor*. If $i = 1$, then v is called a *prefix* of u, and if $j = n$, a *suffix*. We let Σ^n denote the set of all n-length words over Σ and $\Sigma^* = \bigcup_{n\in\mathbb{N}} \Sigma^n \cup \epsilon$, where ϵ is the empty word. Any subset of Σ^* is called a *language*.

The *edit distance* between two words u and v, $\mathsf{ed}(u,v)$, is defined as the minimum number of deletions, insertions, and substitutions of symbols required to transform u into v. The *indel distance* between two words u, v, $\delta(u,v)$, is defined as the smallest number of insertions and deletions needed to convert u into v.

We say that a word $u \in \Sigma^*$ is *weighted* if each position i of u has a non-zero integer weight that we denote by $\mathsf{weight}(u[i])$. We define the weight of u, $\mathsf{weight}(u)$, as the sum of the weights of its positions. The *weighted edit distance* between a weighted word u and a (non-weighted) word v, $\mathsf{wed}(u,v)$, is defined as the minimum cost of deletions and insertions of symbols that we must apply to u to obtain v. The cost of deletion or insertion of a symbol is equal to its weight. (Substitutions are not allowed.) For example, if $u = abb$ and the weights of the positions are $3, 2, 2$, and $v = abc$ then $\mathsf{wed}(u,v) = 3$: we delete $u[2] = b$ with weight 2, and insert c with weight 1.

▶ **Observation 1.** *Consider two words u, v, and assume that the weight of any position in u equals 1. We then have $\mathsf{ed}(u,v) \le \mathsf{wed}(u,v) \le 2\mathsf{ed}(u,v)$.*

▶ **Definition 2.** *The* weight distribution *over a word $u \in \Sigma^n$ is the probability distribution over $\{1,\dots n\}$ where the probability to sample a position i is equal to $\mathsf{weight}(u[i])/\mathsf{weight}(u)$.*

▶ **Definition 3.** *A* non-deterministic finite automaton (NFA) *is defined as a tuple $A = (\Sigma, Q, Q_{in}, Q_f, \Delta)$, where:*

- *Σ is a finite input alphabet,*
- *Q is a finite set of states $Q_{in} \subseteq Q$ of initial states, and $Q_f \subseteq Q$ of final states,*
- *$\Delta \subseteq Q \times \Sigma \times Q$ is the transition relation. For $(p,a,q) \in \Delta$, we write $p \xrightarrow{a}_\Delta q$.*

For a word $u = u[1] \dots u[n]$ and states $p, q \in Q$, we write $p \xrightarrow{u}_\Delta q$ if there exists a sequence of states $p = q_0, \dots, q_n = q$ such that $\forall i = 1, \dots, n \;\; q_{i-1} \xrightarrow{u[i]}_\Delta q_i$. The sequence $q_0, \dots, q_n$ is called a *run* labelled by u. A word u is recognized by A if there exists $(q_{in}, q_f) \in Q_{in} \times Q_f$ such that $q_{in} \xrightarrow{u}_\Delta q_f$. We write $L(A)$ for the set of all words recognized by A. A language $L \subseteq \Sigma^*$ is *regular* if there exists an NFA A such that $L = L(A)$.

Property testing model

For a given distance function d on Σ^*, we define the distance from a word $u \in \Sigma^*$ to L, $d(u, L) = \min_{w \in L} d(u, w)$. If a word u is weighted, we say that it is *ε-far* from L if $d(u, L) \geq \varepsilon \cdot \mathsf{weight}(u)$. In the unweighted case, if $d(u, L) \geq \varepsilon \cdot |u|$.

In the property testing model, we assume that we can query any symbol of the input according to the weight distribution over it in constant time. In the unweighted case, this is equivalent to constant-time random access.

▶ **Definition 4** (Property tester). *An ε-*property tester *for a language L for a distance d is a randomised algorithm that:*

- *accepts if $u \in L$ with probability 1,*
- *rejects with probability at least $2/3$ if u is ε-far from L under d,*
- *accepts or rejects otherwise.*

In addition to standard complexity measures, we are interested in so-called *query complexity* which is defined to be the number of symbols of u that a tester queries. The time complexity is defined as usual, and the space complexity of a tester is defined as space used beyond the space required to store the input.

Property testers can be *non-adaptive* (the symbols to query are selected offline) and *adaptive* (the position of the i-th queried symbol depends on the first $i - 1$ queried symbols). In this work, we focus on non-adaptive testers.

3 Property testing of regular languages

In this section, we show an improved upper bound and a matching lower bound on the query complexity of property testing of regular languages under weighted edit distance.

3.1 Upper bound

We start by showing the following theorem:

▶ **Theorem 5.** *Let $A = (\Sigma, Q, Q_{in}, Q_f, \Delta)$ be an NFA, $m = |Q|$, and k be the number of strongly connected components in the underlying graph. There exists an ε-property tester for membership of a weighted word u in the regular language $L = L(A)$ under the weighted edit distance. The query complexity of the tester is $\mathcal{O}(km \cdot \varepsilon^{-1} \log(m/\varepsilon))$ and the time complexity is $\mathcal{O}(km^3 \cdot \varepsilon^{-1} \log(m/\varepsilon))$, assuming constant-size alphabet.*

3.1.1 Combinatorics of blocking factors and fragments

▶ **Definition 6** (Blocking factor). *Consider a strongly connected component C of A. We say that a factor v of a word u is a C-*blocking *factor if for any two states p, q of C we have $p \overset{v}{\nrightarrow}_\Delta q$.*

We say that a sequence of components $P = (C_1, \dots, C_j)$ is a *component path* of A if for any $1 \le i \le j-1$ there is $p \in C_i, q \in C_{i+1}$ such that $p \xrightarrow{a}_\Delta q$ for some symbol $a \in \Sigma$. Note that $j \le k \le m$.

▶ **Definition 7** (P-partition, β-saturation). *Let $u \in \Sigma^*, \beta > 0$ and $P = (C_1, \dots, C_j)$ be a component path. We build the* P-partition *of u recursively. We start with $i = 1$. Let $u = vxu'$, where $v \in \Sigma^*$, $x \in \Sigma$, and vx is the shortest C_i-blocking prefix of u. If $\mathsf{weight}(x) > \beta \cdot \mathsf{weight}(u)$, we say that x is* heavy*, and otherwise we call it* light*. Consider three cases:*

- *If x is not C_i-blocking, add vx to the set $B_i(P)$;*
- *If x is C_i-blocking and heavy, add it to the set $H_i(P)$;*
- *If x is C_i-blocking and light, add it to the set $L_i(P)$.*

Recurse for $u = u'$ and the value of i computed as follows:

$$i = \begin{cases} \min_{i'}(\{i' \mid i' > i, \exists p \in C_{i'-1} \cup C_{i'}, q \in C_{i'}, p \xrightarrow{x}_\Delta q\} \cup \{j+1\}), & \text{if } x \text{ is heavy;} \\ i+1, & \text{if } x \text{ is light and } \big(|B_i(P)| \ge \beta \cdot \mathsf{weight}(u) \text{ or } \mathsf{weight}(L_i(P)) \ge \beta \cdot \mathsf{weight}(u)\big); \\ i, & \text{otherwise.} \end{cases}$$

Stop if $i = j+1$, if u is empty, or if there is no C_i-blocking factor in u. If we reach $i = j+1$, we say that u β-saturates *P. The sets $B_i(P), H_i(P), L_i(P), i \le j$ form the P-partition of u.*

We show that if a word is ε-far from L, then it β-saturates P for $\beta = \varepsilon/(6m)$. The proof is by contradiction: we show that if a word does not β-saturate P, then we can obtain a word in L by deleting the factors in $\cup_i L_i(P)$ and then inserting a small number of factors of length and weight at most m.

▶ **Lemma 8.** *Let $u \in \Sigma^*$ be such that $\mathsf{weight}(u) \ge 6km/\varepsilon$ and $P = (C_1, \dots, C_j)$ be a component path of A. Let $\beta = \varepsilon/(6m)$. If u is ε-far from L, then u β-saturates P.*

Proof. By contrapositive: assume that u does not β-saturate P, in other words, the algorithm that built the P-partition of u reached the end of u but $i \ne j+1$. We will show that there is a word $u' \in L$ such that the weighted edit distance between u and u' is at most $\varepsilon \cdot \mathsf{weight}(u)$. We first delete all the elements in $L_i(P)$, for all i. Note that $\mathsf{weight}(L_i(P)) \le 2\beta \cdot \mathsf{weight}(u)$ for any i, so the total weight of the deleted factors is at most $2k\beta \cdot \mathsf{weight}(u) = \varepsilon \cdot \mathsf{weight}(u)k/(3m)$. Next, we edit the elements of $B_i(P)$ as follows: let $v \in B_i(P)$, $v = v'x$ where $v' \in \Sigma^*, x \in \Sigma$. Since v is a minimal blocking prefix, v' is not C_i-blocking, and there exist $p, q \in C_i$ such that $p \xrightarrow{v'}_\Delta q$. Similarly, as $v \in B_i(P)$, x is not C_i-blocking and labels a run within C_i, from p' to q'. Therefore, we can edit v into $v'' = v'wx$, where $q \xrightarrow{w}_\Delta p'$ and $|w| \le |C_i|$. Since C_i is strongly connected, we can similarly insert factors of length at most $|C_i|$ between the elements of $B_i(P)$ to obtain a word w_i that labels a run within C_i. Note that we can choose the weights of the inserted symbols arbitrarily, and we put them equal to one. Hence, the cost of this step is at most $|B_i(P)| \cdot |C_i|$ for each i. Finally, we insert factors of length at most m in between the words w_i and the heavy blocking 1-factors in $H_i(P)$, as well as before and after the last factor, so that the resulting word u' belongs to L. Again, we put the weights of the symbols of the inserted factors equal to one, and hence the cost of this step is at most $2km$. We have the following bound on the weighted edit distance of u to L:

$$\begin{aligned} d(u, L) \le d(u, u') &\le \varepsilon \cdot \mathsf{weight}(u)k/(3m) + 2km + \sum_{i=1}^{j} 2 \cdot |B_i(P)| \cdot |C_i| \\ &\le \varepsilon(\mathsf{weight}(u)/3 + \mathsf{weight}(u)k/(3m)) + 2km \le \varepsilon \cdot \mathsf{weight}(u) \end{aligned}$$

A contradiction. ◀

Suppose that u β-saturates a component path $P = (C_1, \dots C_j)$, in other words, the algorithm that built the P-partition reached $i = j + 1$. For every $1 \leq i \leq j$, let $S_i(P) = B_i(P) \cup L_i(P) \cup H_i(P)$. We define indices $0 = a_0 \leq \cdots \leq a_j$ so that for each i either $S_i(P)$ is empty (because C_i was skipped by the algorithm of Definition 7 due to a heavy factor), or all the factors in it are the factors of $u[a_{i-1} + 1, a_i]$. For $i = 0$, we let $a_0 = 0$, and then for all i, $a_i = a_{i-1}$ if $S_i(P) = \emptyset$ and the largest index of a symbol in $S_i(P)$ otherwise. If $S_i(P) \neq \emptyset$, then by Definition 7, one of the following holds:

- The total weight of C_i-blocking 1-factors in $H_i(P) \cup L_i(P)$ is at least $\beta \cdot \mathsf{weight}(u)$;
- $|B_i(P)| \geq \beta \cdot \mathsf{weight}(u)$.

Blocking factors are witnesses of the fact that $u \notin L$. If there are many short blocking factors, as in the first case, we can sample them with a few queries. However, in the second case, the factors can be arbitrarily long. To develop an efficient tester, we must give a more accurate bound.

▶ **Corollary 9.** *Let $\gamma = \lceil 2/\beta \rceil = \lceil 12m/\varepsilon \rceil$. If $|B_i(P)| \geq \beta \cdot \mathsf{weight}(u)$, then $B_i(P)$ contains at least $\mathsf{weight}(u)/\gamma$ disjoint C_i-blocking factors of length at most γ.*

Proof. Since the factors in $B_i(P)$ are disjoint and their total weight is bounded by $\mathsf{weight}(u)$, at most $\mathsf{weight}(u)/\gamma$ of them can have weight (and length) larger than γ. Consequently, $B_i(P)$ contains at least $\beta \cdot \mathsf{weight}(u) - \mathsf{weight}(u)/\gamma \geq \mathsf{weight}(u)/\gamma$ factors of length $\leq \gamma$. ◀

Let us now introduce the notion of a fragment that will allow us to test the sampled factors efficiently.

▶ **Definition 10** (Fragment)**.** *Given a set of factors $u[i_r, j_r]$, $1 \leq r \leq t$, consider the decomposition of the set $S = \bigcup_{1 \leq r \leq t} [i_r, j_r]$ into maximal disjoint intervals, that is, $S = \sqcup_{1 \leq r \leq t'} [i'_r, j'_r]$, where $i'_1 \leq j'_1 < i'_2 \leq j'_2 < \dots i'_{t'} \leq j'_{t'}$. The* fragment F *formed by the factors is the word $F = * \, u[i'_1, j'_1] * u[i'_2, j'_2] * \cdots * u[i'_{t'}, j'_{t'}] \, *$, where "$*$" is a special symbol not in Σ. A word v* contains F as a fragment *if there exist words $w_1, \dots w_{t'+1} \in \Sigma^*$ such that by replacing the i-th symbol "$*$" with w_i in F, we obtain v.*

For example, for $u = abbabbab$ the fragment formed by factors $u[1, 2]$, $u[2, 3]$, $u[4, 4]$, and $u[6, 8]$, is $F = * \, u[1, 4] * u[6, 8] * = * \, abba * bab *$. The word $cabbadebabcde$ contains F.

▶ **Definition 11** (Blocking fragment)**.** *A fragment F is* A-blocking *if none of the words that contain F as a fragment belongs to $L(A)$.*

The following lemma relates C_i-blocking factors and A-blocking fragments and is crucial for correctness of our tester. To show it, we prove by induction that any run that starts in C_1 and is labelled by the prefix of the fragment containing the factors in $\cup_{1 \leq i \leq t} H_i(P)$ and at least one C_i-blocking factor from each $S_i \neq \emptyset$, $1 \leq i \leq t$, must end at a state in $\cup_{i \geq t+1} C_i$.

▶ **Lemma 12.** *If for any component path $P = (C_1, \dots, C_j)$ and i, the fragment F contains all factors in $\bigcup_i H_i(P)$, and a C_i-blocking factor from $u[a_{i-1} + 1, a_i]$ for each i such that $H_i(P) = \emptyset$ and $L_i(P) \cup B_i(P) \neq \emptyset$, then F is A-blocking.*

Proof. By contradiction, suppose that there is a word $w \in L(A)$ that contains F as a fragment. Since $w \in A$, there is a run in A labelled by w that goes through the connected components $C_1, \dots, C_j$. Consider the path $P = (C_1, \dots, C_j)$ and let $0 \leq a_0 \leq a_1 \cdots \leq a_j$ be the indices such that for each i either $S_i(P)$ is empty (because C_i was skipped by the algorithm of Definition 7 due to a heavy factor), or all the factors in it are the factors of $u[a_{i-1} + 1, a_i]$. Let w_t be the shortest prefix of w that contains the blocking factors from each

of the non-empty intervals $u[a_{i-1}+1, a_i]$, $1 \leq i \leq t$, as well as all factors in $\bigcup_{1\leq i\leq t} H_i(P)$. We show by induction on t that every run that starts at a state in C_1 and is labelled by w_t ends after C_t, i.e. in a state of $\bigcup_{t'>t} C_{t'}$.

As w_1 contains a C_1-blocking factor, any run labelled by w_1 that starts in C_1 exits C_1 and therefore ends after C_1. Suppose that the induction hypothesis holds for some $t' < j$. We show that there is no run labelled by $w_{t'+1}$ from C_1 to $C_{t'+1}$. There are two possibilities. First, $C_{t'+1}$ was skipped because of a heavy C_t-blocking factor, for some $t \leq t'$, which is therefore included in $w_{t'+1}$, and ends after $C_{t'+1}$. The other possibility is that $w_{t'+1}$ is $w_{t'}$ with an additional $C_{t'+1}$-blocking factor. If any run labelled by $w_{t'}$ ends after $C_{t'+1}$, so does the run labelled by $w_{t'+1}$, and we are done. If some of these runs end in $C_{t'+1}$, the $C_{t'+1}$-blocking factor that appears in $w_{t'+1}$ but not in $w_{t'}$ ensures that the run labelled by $w_{t'+1}$ ends after $C_{t'+1}$.

Adding symbols to w_j to obtain w, we get that there is no run labelled by w that starts in C_1 and ends in C_j, a contradiction. ◀

3.1.2 Tester

Our tester is as stated in Algorithm 1. We define the *l-factor sampling* over u as the distribution over factors v of u that have length at most l, where a position i is selected according to the weight distribution over u, and $v = u[i, \min(i+l-1, |u|)]$.

Algorithm 1 ε-property tester for regular languages.

$\beta \leftarrow \varepsilon/(6m)$, $\gamma \leftarrow 2/\beta$
if $\mathsf{weight}(u) \leq 6km/\varepsilon$ **then**
    Query all symbols of u and run the automaton A on it
    Reject if A rejects, else accept
else
    Query $\tau = \lceil 2\ln(9k \cdot 2^k)/\beta \rceil$ 1-factors of u
    for $t = 0$ to $T = \lceil \log(2\gamma) \rceil$ **do**
        $\ell_t \leftarrow 2^t, r_t \leftarrow \lceil 2\ln(9k \cdot 2^k)\gamma/\ell_t \rceil$
        Query $u[1, 2\ell_t]$
        Query r_t factors of u according to the $2\ell_t$-factor distribution
    Reject if the fragment formed by the sampled factors is A-blocking, else accept

If $\mathsf{weight}(u) \leq 6km/\varepsilon$, the length of u is at most $6km/\varepsilon$ as well, and hence Algorithm 1 queries $\mathcal{O}(6km/\varepsilon)$ symbols. Otherwise, Algorithm 1 first makes $\mathcal{O}(2\ln(9k \cdot 2^k)/\beta) = \mathcal{O}(km \cdot \varepsilon^{-1})$ queries to sample 1-factors, and then for each $t = 0, \ldots, T$, another $2^{t+1} \cdot (r_t + 1) = \mathcal{O}(4\ln(9k2^k)\gamma) = \mathcal{O}(km \cdot \varepsilon^{-1})$ queries. Hence, it has query complexity $O(km \cdot \varepsilon^{-1} \log(m/\varepsilon))$. To estimate the time complexity, we must explain how we check if the sampled fragment F is A-blocking. Given a fragment F and $S \subseteq Q$, let $\mathsf{reach}(F, S)$ denote the set of states that can be reached from a state of S when following a run labelled by some word v that contains F as a fragment, i.e.

$$\mathsf{reach}(F, S) = \{q \in Q \mid \exists p \in S, v \in \Sigma^* \ : \ p \xrightarrow{v}_{\Delta} q \text{ and } v \text{ contains the fragment } F\}$$

By definition, F is A-blocking if and only if $\mathsf{reach}(F, Q_{in}) \cap Q_f = \emptyset$.

▶ **Lemma 13.** *For any F, S, $\mathsf{reach}(F, S)$ can be computed in time $\mathcal{O}(|F| \cdot m^2)$.*

Proof. Recall that a constant-size alphabet is assumed. If $|F| = 1$, $\mathsf{reach}(F, S)$ can be computed in time and space $\mathcal{O}(m^2)$. If $F = a \in \Sigma$, $\mathsf{reach}(a, S) = \{q \in Q \mid \exists p \in S, p \xrightarrow{a}_\Delta q\}$. In this case, we can compute $\mathsf{reach}(a, S)$ by following every transition that is labelled by a and starts at a state $p \in S$. If $F = *$, $\mathsf{reach}(*, S)$ is the set of states q such that there exists a path from a state $p \in S$ to q. It can therefore be computed using a breadth-first traversal of the graph induced by the automaton, initialized at every $p \in S$. For any F, $|F| > 1$, we have $\mathsf{reach}(F, S) = \mathsf{reach}(F[2, |F|], \mathsf{reach}(F[1], S))$. Therefore, we can compute $\mathsf{reach}(F, S)$ in a recursive manner: let $S_0 = S$, and for every $1 \leq i \leq |F|$, $S_i = \mathsf{reach}(F[i], S_{i-1})$. By induction, we have $S_{|F|} = \mathsf{reach}(F, S)$. The algorithm makes $|F|$ calls to the reach function on a single symbol, which requires $\mathcal{O}(|F| \cdot m^2)$ time. ◀

The fragment F constructed in Algorithm 1 has size $\mathcal{O}(km \cdot \varepsilon^{-1} \log(m/\varepsilon))$, and therefore the time complexity of the tester is $\mathcal{O}(km^3 \cdot \varepsilon^{-1} \log(m/\varepsilon))$.

3.1.3 Correctness of the tester

In what follows, let $W := \mathsf{weight}(u)$. If $W \leq 6km/\varepsilon$, we query all symbols of u and run the automaton on u, the answer is correct with probability 1. Below we assume that $W > 6km/\varepsilon$.

If $u \in L$, there is no A-blocking fragment in u, and therefore the tester accepts with probability 1. We must now show that if u is ε-far from L, then the tester accepts with probability at most $1/3$, or in other words, the probability that F is not blocking is at most $1/3$. By Lemma 12, the probability that F is not blocking is smaller than the probability that there exist a component path $P = (C_1, \ldots, C_j)$ and an index $i, 1 \leq i \leq j$ such that F contains neither the factor from $H_i(P)$ (if $H_i(P) \neq \emptyset$), nor a factor from $L_i(P) \cup B_i(P)$ (if $H_i(P) = \emptyset$). Fix a path P and an index i. If $\mathsf{weight}(H_i(P)) \geq \beta \cdot W$ or $\mathsf{weight}(L_i(P)) \geq \beta \cdot W$, then by sampling independently $\ln(9k \cdot 2^k)/\beta$ factors of length 1 w.r.t. the 1-factor distribution, we miss such a factor with probability at most $1/(9k \cdot 2^k)$. Otherwise, if $S_i \neq \emptyset$, we have $|B_i(P)| \geq \beta \cdot W$.

▶ **Lemma 14.** *Assume $W > 6km/\varepsilon$ and $|B_i(P)| \geq \beta \cdot W$. Algorithm 1 fails to sample a C_i-blocking factor with probability at most $1/(9k2^k)$.*

Proof. Consider a fixed $t \in [0, T]$. If $u[1, 2\ell_t]$ contains a factor from $B_i(P)$, $u[1, 2\ell_t]$ is C_i-blocking and we are done. Assume that this is not the case.

We estimate the number of $2\ell_t$-factors that contain a factor from $B_i(P)$. For brevity, let $B' = \{v \in B_i(P) : |v| \leq \gamma\}$. Let $v_1, v_2, \ldots, v_{f_t}$ be the factors in B' of length at most ℓ_t, in the order of appearance in u. For all $j > 1$, the number of $2\ell_t$-factors such that v_j is the first factor appearing in them is equal to $\min(2\ell_t - |v_j| + 1, \mathsf{dist}(v_{j-1}, v_j))$, where $\mathsf{dist}(v_{j-1}, v_j)$ is the difference between the starting positions of v_{j-1} and v_j. Since $|v_j| \leq \ell_t$, we have $2\ell_t - |v_j| \geq \ell_t$. We also have $\mathsf{dist}(v_{j-1}, v_j) \geq |v_{j-1}|$, since the factors v_j and v_{j-1} are disjoint. Therefore, for all $j > 1$, $\min(2\ell_t - |v_j| + 1, \mathsf{dist}(v_{j-1}, v_j)) \geq \min(\ell_t, |v_{j-1}|) = |v_{j-1}|$. The first term corresponds to the case where there is no interval of length $2\ell_t$ that contains both v_{j-1} and v_j: if v_j starts at a position p in u, it ends at position $p + |v_j| - 1$, and the interval can start at any position p' such that $p' \leq p$ and $p' + 2\ell_t - 1 \geq p + |v_j| - 1$, i.e. $p - (2\ell_t - |v_j|) \leq p'$. The second term is equal to the number of positions between the start of v_{j-1} and the start of v_j in u: if an interval that contains v_j also contains v_{j-1}, then it does not contain v_j as its first blocking factor, and therefore it is associated with v_{j-1}. By summing over all j, we obtain that the number of $2\ell_t$-factors containing a factor from B' is at least

$$2\ell_t - |v_1| + \sum_{j=1}^{f_t} |v_{j-1}| \geq \ell_t + \sum_{j=0}^{f_t-1} |v_j| \geq l_{v_{f_t}} + \sum_{j=0}^{f_t-1} |v_j| \geq \sum_{j=0}^{f_t} |v_j|.$$

Let p_t be the probability that a factor of length $2\ell_t = 2^{t+1}$ sampled according to the $2\ell_t$-factor distribution is C_i-blocking. As any factor containing a factor from B' is C_i-blocking, from above we obtain that $p_t \geq \frac{1}{W}\sum_{j=0}^{f_t} |v_j| \geq \frac{1}{W}\sum_{v\in B':|v|\leq \ell_t} |v|$.

Consequently, for a fixed t, the probability that none of the r_t factors is C_i-blocking is at most $(1-p_t)^{r_t} \leq e^{-p_t r_t}$. By independence, the probability p that the algorithm failed to sample a C_i-blocking factor for every t satisfies:

$$p \leq \prod_{t=0}^{T} \exp(-p_t r_t) \leq \exp\left(\sum_{t=0}^{T} -p_t r_t\right) \leq \exp\left(-\sum_{t=0}^{T} \frac{2\ln(9k\cdot 2^k)\gamma 2^{-t}}{W} \sum_{v\in B':|v|\leq \ell_t} |v|\right).$$

We now show that $\sum_{t=0}^{T} 2^{-t} \sum_{v\in B':|v|\leq \ell_t} |v| \geq W/(2\gamma)$, which implies that $p \leq e^{-\ln(9k\cdot 2^k)} = 1/(9k2^k)$. We have:

$$\sum_{t=0}^{T} 2^{-t} \sum_{v\in B':|v|\leq \ell_t} |v| = \sum_{v\in B'} |v| \sum_{t=\lceil \log(|v|)\rceil}^{T} 2^{-t} = \sum_{v\in B'} |v| \frac{1}{2^{\lceil \log|v|\rceil}} \frac{1-2^{-(T-\lceil \log|v|\rceil+1)}}{1-1/2}$$
$$\geq \sum_{v\in B'} \left(1-2^{-(T-\lceil \log|v|\rceil+1)}\right) \geq \sum_{v\in B'} 1 - \frac{|v|}{2\gamma} \geq \sum_{v\in B'} 1/2 \geq W/(2\gamma),$$

where the last inequality holds because of Corollary 9. ◀

Hence, for fixed P and i, the probability that the fragment F built by Algorithm 1 contains neither the factor from $H_i(P)$ (if $H_i(P) \neq \emptyset$) nor a factor from $L_i(P) \cup B_i(P)$ (if $H_i(P) = \emptyset$) is bounded from above by $\left(\frac{1}{9k2^k} + \frac{1}{9k2^k} + \frac{1}{9k2^k}\right) \leq \frac{1}{3k2^k}$. By the union bound over all P and all k, and since there are at most 2^k component paths in A, we obtain that $\Pr[F \text{ is not blocking}] \leq \frac{1}{3}$. This concludes the proof of Theorem 5.

3.2 Lower bound

In this section we show that the query complexity of Theorem 5 is tight. Note that the indel distance is the weighted edit distance when all weights are equal to one, and hence it suffices to show the lower bound for the former.

▶ **Theorem 15.** *There exists a regular language L and constants $\varepsilon_0, C > 0$ such that on an input of length n and for any $1/n^{1/3} < \varepsilon < \varepsilon_0$, a non-adaptive ε-property tester for L under the indel distance has query complexity $\geq C\varepsilon^{-1}\log(1/\varepsilon)$.*

Note that by running a property tester twice, the error probability can be reduced from 1/3 to 1/9, while increasing the query complexity by a factor of two. From that and by Yao's minimax principle [23], it suffices to find a distribution $\mathcal{D}$ on $\{a,b,c,d\}^*$ and a constant $C > 0$ such that any *deterministic* algorithm A that makes at most $C\varepsilon^{-1}\log(1/\varepsilon)$ queries and accepts all inputs in L, errors with probability $> 1/9$.

Let n be the length of the input. Below, we fix arbitrarily a precision parameter $1/n^{1/3} \leq \varepsilon < 1/2^{-48}$. Consider regular languages $L_0 = \{u \in \{a,c\}^* : u \text{ contains an even number of } c\}$ and $L = (a \mid bL_0d)^*$. Define the distribution $\mathcal{D}$ as follows. Let r be a fair coin. For $\ell = \lceil 5/\varepsilon \rceil$,

divide $[1,n]$ into $z=\lfloor n/\ell \rfloor \geq 1$ intervals of size ℓ each, except for the last one that can be longer. Let $[a_j,b_j]$ be the j-th interval. We associate with it a random variable τ_j, distributed as follows:

$$\tau_j = \begin{cases} t, & \text{with probability } p_t = 12\varepsilon 2^t/\log(1/\varepsilon) \text{ for } t=1,2,\ldots,\lceil \log(1/\varepsilon)\rceil; \\ 0, & \text{with probability } p_0 = 1-\sum_{t=1}^{\lceil \log(1/\varepsilon)\rceil} p_t. \end{cases}$$

For any $\varepsilon < 2^{-48}$, we have $p_0 = 1-\sum_{t=1}^{\lceil \log(1/\varepsilon)\rceil} p_t > 0$, and hence the distribution is well-defined. The variable τ_j characterizes the length of the instances of L_0 that we will put in the j-th interval. If $\tau_j=0$, we put no instances at all: set $u[a_j,b_j]=aa\ldots a$. If $\tau_j=t>0$, set $u[a_j,b_j]=(bw_jd)^{\lceil 2^{-t}/\varepsilon\rceil}aa\ldots a$, where w_j is a word of length 2^t chosen uniformly at random from L_0 if $r=0$ and from $\{a,c\}^*\setminus L_0$ otherwise.

Notice that $\mathcal{D}$ produces a positive instance with probability at least $1/2$, since whenever $r=0$, $u\in L$. We prove in the following lemma that $\mathcal{D}$ also produces a word ε-far from L with constant probability.

▶ **Lemma 16.** *Let u be a word of length n sampled w.r.t. $\mathcal{D}$. With probability at least $1/4$, $\delta(u,L)\geq \varepsilon n$.*

Proof. Assume $r=1$ and let $\xi_j=\lceil 2^{-\tau_j}/\varepsilon\rceil$ if $\tau_j>0$, and 0 otherwise (ξ_j is the number of instances of L_0 in the j-th interval). The variables ξ_j are independent, and for every j, we have $1\leq \xi_j\leq 1/\varepsilon$. Let $\xi=\left(\sum_{j=1}^{z}\xi_j\right)/z$. We have:

$$\mathbb{E}[\xi]=\mathbb{E}[\xi_j]=\sum_{t>0}p_t\cdot\lceil 2^{-t}/\varepsilon\rceil=\sum_{t>0}\frac{12\varepsilon 2^t}{\log(1/\varepsilon)}\cdot\lceil 2^{-t}/\varepsilon\rceil\geq\sum_{t>0}\frac{12}{\log(1/\varepsilon)}\geq 12.$$

By Hoeffding's inequality, we have $\Pr[\xi\geq 6]\geq 1-e^{-\frac{72z^2}{z/\varepsilon^2}}=1-e^{-72\varepsilon^2 z}\geq 1/2$ and $z=\lfloor n/\lceil 5/\varepsilon\rceil\rfloor\geq \varepsilon n/6$, as $1/n^{1/3}\leq\varepsilon<2^{-48}$. Hence, $\Pr\left[\sum_j\xi_j\geq\varepsilon n\right]\geq\Pr[\xi\geq 6]\geq 1/2$.

It remains to show that conditioned on $r=1$, the indel distance between u and L is at least $\sum_j\xi_j$. As each word of form bw_jd, where $w_j\in\{a,c\}^*\setminus L_0$ requires at least one insertion or deletion to become a factor of a word in L, the bound follows by construction. Hence, $\Pr[\delta(u,L)\geq\varepsilon n]\geq\Pr[r=1]\cdot\Pr\left[\sum_j\xi_j\geq\varepsilon n\right]\geq 1/4$. ◀

▶ **Lemma 17.** *Let q_j be the number of symbols queried by A in the j-th interval $[a_j,b_j]$. Consider an input u sampled w.r.t. $\mathcal{D}$. If for all j such that $\tau_j>0$ we have $q_j<2^{\tau_j}$, then A must accept u.*

Proof. Assume that A rejects u. If $u\in L$, we immediately get a contradiction. If u is a negative instance, then for all j such that $\tau_j>0$ we have $u[a_j,b_j]=(bw_jd)^{\lceil 2^{-\tau_j}/\varepsilon\rceil}aa\ldots a$, where w_j is a word of length 2^{τ_j} not in L_0. Since A queries less than 2^{τ_j} symbols in $u[a_j,b_j]$, there is a symbol of w_j that A does not query. Hence, there is a word in L_0 that we denote by w'_j that has length 2^{τ_j} and does not differ from w_j on the queried symbols. We replace all copies of w_j by w'_j. Let u' denote the resulting word. Notice that $u'\in L$, and all the symbols for u and u' in the queried positions are equal and therefore A rejects it as well. The probability of u' under the distribution $\mathcal{D}$ is non-zero, and therefore there is a non-zero probability that A rejects a positive instance, a contradiction. ◀

Proof of Theorem 15. We finally derive that if A queries $\sum_j q_j < 1/(3\cdot 192\varepsilon)\log(1/\varepsilon)$ symbols, then it errs with probability $>1/9$, which implies the theorem.

Consider an input u of length n generated according to the distribution $\mathcal{D}$. Let $A(u)$ denote the output of A on u: 1 for accepting, 0 for rejecting. Let F denote the event "$\delta(u, L) \geq \varepsilon n$". The probability that A rejects inputs that are far from L is given by:

$$\begin{aligned}
\Pr[A(u) = 0 \mid F] &= \Pr[A(u) = 0 \wedge F] / \Pr[F] \leq \Pr[A(u) = 0] / \Pr[F] \\
&\leq 4 \cdot \Pr[A(u) = 0] \text{ (Lemma 16)} \\
&\leq 4 \cdot \Pr[\exists j : q_j \geq 2^{\tau_j} \wedge \tau_j > 0] \text{ (Lemma 17)} \\
&\leq 4 \cdot \sum_j \Pr[q_j \geq 2^{\tau_j} \wedge \tau_j > 0] \text{ (union bound)} \\
&\leq 4 \cdot \sum_j \sum_{t=1}^{\lceil \log q_j \rceil} p_t \leq 4 \sum_j \frac{12\varepsilon}{\log(1/\varepsilon)} \sum_{t=1}^{\lceil \log q_j \rceil} 2^t \leq \frac{192\varepsilon}{\log(1/\varepsilon)} \sum_j q_j < 1/3.
\end{aligned}$$

The probability that A gives an incorrect answer is at least the probability that it accepts a word that is ε-far. In other words, this probability is equal to $\Pr[A(u) = 1 \wedge F] = \Pr[A(u) = 1 \mid F] \cdot \Pr[F] > (2/3) \cdot (1/4) > 1/9$, concluding the proof. ◀

4 Streaming property testing of VPLs

Let us start by reminding the formal definitions of visibly pushdown languages and streaming property testers. Let $\Sigma = \Sigma_+ \sqcup \Sigma_= \sqcup \Sigma_-$. We refer to the symbols in Σ_+ as *push* symbols, and the symbols in Σ_- as *pop* symbols.

▶ **Definition 18** (Visibly pushdown automaton). *A visibly pushdown automaton (VPA) $\mathcal{A}$ over Σ is a tuple $(\Sigma, \Gamma, Q, Q_{in}, Q_f, \Delta)$, where*

- *Γ is a finite set of stack symbols,*
- *Q is a finite set of states, $Q_{in} \subseteq Q$ of initial states, $Q_f \subseteq Q$ of final states,*
- *$\Delta \subseteq (Q \times \Sigma_+ \times Q \times \Gamma) \cup (Q \times \Sigma_= \times Q) \cup (Q \times \Sigma_- \times \Gamma \times Q)$ is the transition relation.*

When running the automaton on a word $u \in \Sigma^n$, we maintain a stack. Let $\perp \notin \Gamma$ be a special symbol to denote the bottom of the stack. A configuration of a VPA $\mathcal{A}$ is a tuple $(\sigma, q) \in (\perp \cdot \Gamma^*) \times Q$. For $a \in \Sigma$, there is a transition from a configuration (σ, q) to (σ', q'), denoted $(\sigma, q) \xrightarrow{a}_\Delta (\sigma', q')$, in the following cases:

- *if $a \in \Sigma_+, \sigma' = \sigma \cdot \gamma, \gamma \in \Gamma$ and $(q, a, q', \gamma) \in \Delta$ (we write $q \xrightarrow{a}_\Delta (q', push(\gamma))$),*
- *if $a \in \Sigma_-, \sigma = \sigma' \cdot \gamma, \gamma \in \Gamma$ and $(q, a, \gamma, q') \in \Delta$ (we write $(q, pop(\gamma)) \xrightarrow{a}_\Delta q'$),*
- *if $a \in \Sigma_=$ and $(q, a, q') \in \Delta$ (we write $q \xrightarrow{a}_\Delta q'$).*

For a word $u \in \Sigma^n$, if for all $1 \leq i \leq n, (\sigma_{i-1}, q_{i-1}) \xrightarrow{u[i]}_\Delta (\sigma_i, q_i)$, we write $(\sigma_0, q_0) \xrightarrow{u}_\Delta (\sigma_n, q_n)$. A word u is accepted by an automaton $\mathcal{A}$ if there exists a initial state $q_{in} \in Q_{in}$ and a final state $q_f \in Q_f$ such that $(\perp, q_{in}) \xrightarrow{u}_\Delta (\perp, q_f)$. We denote the language of all words accepted by $L(\mathcal{A})$. A language $L \subseteq \Sigma^*$ is a visibly pushdown language *(VPL) if $L = L(\mathcal{A})$ for some VPA $\mathcal{A}$.*

▶ **Definition 19** (Streaming property testing algorithm). *A **streaming ε-property testing algorithm** for a language L under distance d is an algorithm that given streaming access to a word u:*

- *accepts if $u \in L$ with probability 1,*
- *rejects with probability at least p if u is ε-far from L w.r.t. d,*
- *accepts or rejects otherwise.*

If $p = 1$, we say that a streaming ε-property testing algorithm is deterministic. *If $p = 2/3$, we say that it is randomised. The* space complexity *of the algorithm is defined to be the total space used (in bits)* including *the space needed to store any information about the input.*

4.1 Upper bound

François et al. [5] showed that streaming property testing of visibly pushdown languages can be reduced to the problem of (approximately) encoding words as relationships in finite automata[3]. Consider a (non-deterministic) finite automaton $A = (\widehat{\Sigma}, \widehat{Q}, \widehat{Q}_{in}, \widehat{Q}_f, \widehat{\Delta})$. For $\widehat{\Sigma}' \subseteq \widehat{\Sigma}$, define a distance function $\mathsf{wed}_{\widehat{\Sigma}'}$ as the weighted edit distance where the insertions are restricted to symbols in $\widehat{\Sigma}'$. For a word u, let $R_u = \{(p,q) \mid p,q \in \widehat{Q}, p \xrightarrow{u}_{\widehat{\Delta}} q\}$.

▶ **Definition 20** (ξ-approximation). *A relation $R \subseteq Q^2$ is an (ξ, Σ')-approximation of R_u if the following two conditions are satisfied:*

- *For all p, q such that $p \xrightarrow{u}_{\widehat{\Delta}} q$, $(p,q) \in R$,*
- *If $(p,q) \in R$, then there exists a word v such that $\mathsf{wed}_{\widehat{\Sigma}'}(u,v) \leq \xi \cdot \mathsf{weight}(u)$ and $p \xrightarrow{v}_{\widehat{\Delta}} q$.*

We call A *$\widehat{\Sigma}'$-closed* if $p \xrightarrow{u}_{\widehat{Q}} q$ for some $u \in (\widehat{\Sigma})^*$ iff $p \xrightarrow{u'}_{\widehat{Q}} q$ for some $u' \in (\widehat{\Sigma}')^*$. The *$\widehat{\Sigma}'$-diameter* of A, denoted by d, is the maximum over all pairs of states p, q of $\min\{|u| : u \in (\widehat{\Sigma}')^*, p \xrightarrow{u}_{\widehat{\Delta}} q\}$, whenever this minimum is not over the empty set. Finally, for a fragment F and $S \subseteq \widehat{Q}$, let $\mathsf{reach}_{\widehat{\Sigma}'}(F, S)$ denote the set of states that can be reached from a state of S when following a run labelled by some word v that contains F as a fragment and such that all symbols in $v \setminus F$ belong to $\widehat{\Sigma}'$. By extending our property tester for regular languages, we obtain:

▶ **Lemma 21.** *Let A be $\widehat{\Sigma}'$-closed. Given a query access to a word u, Algorithm 2 computes a $(\xi, \widehat{\Sigma}')$-approximation of $R_u = \{(p,q) \mid p,q \in \widehat{Q}, p \xrightarrow{u}_{\widehat{Q}} q\}$ correctly with probability $\geq 1 - \mu$.*

Proof. For every $p, q \in \widehat{Q}$ and every fragment F of u, if $p \xrightarrow{u}_{\widehat{Q}} q$, then $q \in \mathsf{reach}_{\widehat{\Sigma}'}(F, \{p\})$. Therefore, the algorithm can err only if there exist $p, q \in \widehat{Q}$ such that $(p,q) \in R$ and for every word w such that $p \xrightarrow{w}_{\widehat{Q}} q$ there is $\mathsf{wed}_{\widehat{\Sigma}'}(w,u) \geq \xi \cdot \mathsf{weight}(u)$. This is equivalent to saying that $\mathsf{wed}_{\widehat{\Sigma}'}(u, L(A_{p,q})) \geq \xi \cdot \mathsf{weight}(u)$, where $A_{p,q}$ is the finite automaton A with a unique initial state p and a unique final state q. As A is $\widehat{\Sigma}'$-closed, an argument analogous to Theorem 5 shows that the algorithm errs for p, q with probability at most μ/m^2. The claim follows by the union bound. ◀

By plugging this result into the framework of François et al. [5], we obtain:

▶ **Corollary 22.** *Let $\varepsilon > 0$ be a constant, $\mathcal{A} = (Q, \Sigma, \Gamma, Q_{in}, Q_f, \Delta)$ be a VPA of constant size over $\Sigma = \Sigma_+ \sqcup \Sigma_= \sqcup \Sigma_-$, and $L = L(\mathcal{A})$. There is a randomised streaming property tester for L that uses $\mathcal{O}(\varepsilon^{-3} \log^5 n \log\log n)$ space.*

Proof. François et al. showed that property testing of visibly pushdown languages can be solved by running $\mathcal{O}(\varepsilon^{-1} \log^2 n)$ instances of the approximation algorithm of Lemma 21 with $\xi = \varepsilon/(6 \log n)$ and $\mu = 2/3n$, on an NFA of constant size [5, Theorem 5.4]. Furthermore, sampling factors of the input strings according to the ℓ-factor distribution can be imitated in streaming at an expense of $\mathcal{O}(\ell \cdot (\ell + \log n))$ space per sample [5, Fact 4.6]. We implement $\mathsf{reach}_{\widehat{\Sigma}'}$ similar to Lemma 13 using constant extra space. Therefore, an instance of the approximation algorithm of Lemma 21 requires space

$$\mathcal{O}(\xi^{-1} \log(1/\mu) + \sum_t r_t \ell_t \cdot (\ell_t + \log n)) =$$

$$= \mathcal{O}(\xi^{-2} \log^2(1/\mu) + \xi^{-1} \ln(1/\mu) \log(\xi^{-1} \ln(1/\mu)) \log n) = \mathcal{O}(\varepsilon^{-2} \log^3 n \log\log n).$$

The bound follows. ◀

[3] In this section, we refer to a more complete arXiv version of the paper.

Algorithm 2 Building an $(\xi, \widehat{\Sigma}')$-approximation R of R_u. Here k is the number of strongly connected components of A, and $m = |\widehat{Q}|$.

$\beta \leftarrow \xi/(6m)$, $\gamma \leftarrow 2/\beta$
if $\mathsf{weight}(u) \leq 6kmd/\xi$ **then**
  Read the whole input u
  $R = \{(p, q) \mid p \in \widehat{Q}, q \in \mathsf{reach}_{\widehat{\Sigma}'}(u, \{p\})\}$
else
  Query $\tau = \lceil 2\ln(3k2^k m^2/\mu)/\beta \rceil$ 1-factors of u
  for $t = 0$ to $T = \lceil \log(2\gamma) \rceil$ **do**
    $\ell_t \leftarrow 2^t, r_t \leftarrow \lceil 2\ln(3k2^k m^2/\mu)\gamma/\ell_t \rceil$
    Query $u[1, 2\ell_t]$
    Query r_t $2\ell_t$-factors of u according to the $2\ell_t$-factor distribution
  $R = \{(p, q) \mid p \in \widehat{Q}, q \in \mathsf{reach}_{\widehat{\Sigma}'}(F, \{p\})\}$
return R

4.2 Lower bound

▶ **Theorem 23.** *There exist a VPL L and a constant ε_0 such that for any $1/n \leq \varepsilon < \varepsilon_0$ any deterministic streaming ε-property tester for L under the edit distance uses space $\tilde{\Omega}(n(1 - 16\varepsilon\log(1/\varepsilon)))$. Any randomised streaming ε-property tester for VPLs under the edit distance uses space $\tilde{\Omega}(\max(\varepsilon^{-1}, \log n))$.*

Proof of Theorem 23. We first prove the deterministic bound, and then use it to derive the randomised one. Let L_{mirror} be a VPL over $\Sigma = \{0, 1, \bar{0}, \bar{1}\}$ defined as $L_{mirror} = \{w\bar{w}, w \in \{0,1\}\}$, where $\bar{w} = \overline{w[n]} \dots \overline{w[1]}$. For example, if $w = 1101$ then $\bar{w} = \bar{1}\bar{0}\bar{1}\bar{1}$. Recall that the indel distance $\delta(u, v)$ between two words u, v is equal to the minimum number of insertions and deletions needed to transform u into v and that a lower bound for the indel distance gives an asymptotically equal lower bound for the edit distance.

▶ **Lemma 24.** *Assume n is even. Let $u, v \in \Sigma^{n/2}$. If $\delta(u, v) > 2\varepsilon n$, then $\delta(u\bar{v}, L_{mirror}) > \varepsilon n$.*

Proof. We prove the claim by contrapositive. We assume that $\delta(u\bar{v}, L_{mirror}) \leq \varepsilon n$ and show that $\delta(u, v) \leq 2\varepsilon n$. Let $w\bar{w}$ be a word of L_{mirror} such that $\delta(u\bar{v}, L_{mirror}) = \delta(u\bar{v}, w\bar{w})$. By the triangle inequality, we have $\delta(u, v) \leq \delta(u, w) + \delta(w, v) = \delta(u, w) + \delta(\bar{w}, \bar{v})$.

Let us start with an auxiliary claim. Consider $y \in \Sigma^k, z \in \Sigma^l$. Let y_1 (resp. y_2) denote $y[1, \lceil k/2 \rceil]$ (resp. $y[\lceil k/2 \rceil + 1, k]$), and similarly for z. We show that if $\delta(y, z) = 2$, then $\delta(y_1, z_1) + \delta(y_2, z_2) \leq 4$.

We have either $k = l$ or $|k - l| = 2$. If $k = l$, then $|y_1| = |z_1|$, $|y_2| = |z_2|$, and the two edits are one insertion and one deletion. If the two edits occur in y_1, we have $\delta(y_1, z_1) \leq 2$ and $\delta(y_2, z_2) = 0$. The case when the two edits occur in y_2 is symmetric. If one edit occurs in y_1 and the other in y_2, then y_1 is transformed into a prefix of z of length $|y_1| - 1$ or $|y_1| + 1$, and y_2 into a suffix of z of length $|y_2| + 1$ or $|y_2| + 1$, respectively. Therefore, $\delta(y_1, z_1) \leq 2$ and $\delta(y_2, z_2) \leq 2$. Assume now $|k - l| = 2$. W.l.o.g., $k = l + 2$, and the two edit operations are deletions. Since $k = l + 2$, we have $|y_1| = |z_1| + 1$, $|y_2| = |z_2| + 1$. Consider two cases:

- One deletion occurs in y_1, and one deletion occurs in y_2. In this case, y_1 is transformed into z_1, and y_2 into z_2. Hence, $\delta(y_1, z_1) + \delta(y_2, z_2) = 2$.
- The two deletions occur in y_1 (the proof for y_2 is symmetrical). In this case, y_1 is transformed into a prefix of v of length $|y_1| - 2 = |z_1| - 1$, and y_2 is equal to the suffix of v of length $|y_2|$. Therefore, $\delta(y_1, z_1) \leq 3$ and $\delta(y_2, z_2) = 1$, and the claim follows.

For simplicity, let $T = \delta(u\bar{v}, w\bar{w})$. Note that T is even, as both words have even length. For every $0 \leq t \leq T$, let x_t be the word obtained by applying the first t edit operations to $u\bar{v}$ in the sequence that transforms $u\bar{v}$ into $w\bar{w}$, and let $x_1^t = x^t[1, \lceil |x^t|/2 \rceil]$ and $x_2^t = x_t[\lceil |x^t|/2 \rceil + 1, |x^t|]$. For all $t \leq T-2$, $\delta(x^t, x^{t+2}) = 2$, and therefore $\delta(x_1^t, x_1^{t+2}) + \delta(x_2^t, x_2^{t+2}) \leq 4$. Finally, we obtain

$$\begin{aligned}\delta(u,w) + \delta(\bar{v},\bar{w}) = \delta(x_1^0, x_1^T) + \delta(x_2^0, x_2^T) &\leq \sum_{t=0}^{T/2} \delta(x_1^{2t}, x_1^{2t+2}) + \sum_{t=0}^{T/2} \delta(x_2^{2t}, x_2^{2t+2}) \\ &\leq \sum_{t=0}^{T/2} \left(\delta(x_2^{2t}, x_2^{2t+2}) + \delta(x_2^{2t}, x_2^{2t+2})\right) \leq 4T/2 \leq 2\varepsilon n\end{aligned}$$

In short, we have $\delta(u,w) + \delta(\bar{v},\bar{w}) \leq 2\varepsilon n$, which completes the proof. ◀

▷ Claim 25. Let $u \in \{0,1\}^n$ and $B(u,k,n) = \{v \in \{0,1\}^n | \, \delta(u,v) \leq k\}$. For all $2/n \leq \alpha < 1$, we have $|B(u, \alpha n, n)| \leq 2^{\alpha n(1+2\log 8e/\alpha)/2}$.

Proof. Let $2t$ be the largest even number smaller or equal to αn. Let $v \in B(u,k,n)$. Since $|u| = |v| = n$, we can obtain v from u using t insertions and t deletions. (If $\delta(u,v) < 2t$, we can insert a symbol $(2t - \delta(u,v))/2$ times, and then delete it $(2t - \delta(u,v))/2$ times.)

W.l.o.g., assume that the insertions occur before the deletions. After t insertions, we obtain a word of size $n+t$. The number of possible ways to delete t symbols in this word is therefore $\binom{n+t}{t}$. We give an upper bound on the number of words that can be reached with t insertions from a word of length n the following way: in a word of length $n+t$, there are t symbols that have been inserted, and there are two options for each symbol. Therefore, there are at most $2^t\binom{n+t}{t}$ such words. The number of words in $B(u,k,n)$ is at most the number of words reached after a sequence of t insertions, multiplied by the number of sequences of t deletions. Overall, we obtain:

$$\begin{aligned}|B(u,\alpha n,n)| &\leq 2^t \binom{n+t}{t}^2 \leq 2^t \left(\frac{e(n+t)}{t}\right)^{2t} \leq 2^{(1+2\log e)t}(\frac{n}{t}+1)^{2t} \\ &\leq 2^{(1+2\log e)t}(8/\alpha)^{2t} \leq 2^{\alpha n(1+2\log e + 2\log(8/\alpha))/2} \\ &\leq 2^{\alpha n(1+2\log(8e/\alpha))/2}\end{aligned}$$

◁

▶ **Corollary 26.** *There exists a constant ε_0 such that for any $1/n \leq \varepsilon < \varepsilon_0$, any deterministic streaming ε-property tester for L_{mirror} under the indel distance uses space $\bar{\Omega}(n(1 - 16\varepsilon\log(1/\varepsilon)))$.*

Proof. Assume n is even. Consider the memory state of a tester after reading two words $u, v \in \{0,1\}^{n/2}$. Suppose that they give the same memory state. We can then continue the streams with $\bar{u}$, and since the algorithm must accept $u\bar{u} \in L_{mirror}$, it must also accept $v\bar{u}$. By the definition of a streaming ε-property tester and Lemma 24, we have $\delta(u,v) \leq 2\varepsilon n$. Therefore, the number of distinct memory states of the tester after reading $n/2$ symbols is at least $2^{n/2}/|B(u, 2\varepsilon n, n/2)|$. The space $s(n)$ used by the tester is at least the logarithm of the number of memory states. Therefore,

$$\begin{aligned}s(n) &\geq \log(2^{n/2}/|B(u,2\varepsilon n, n/2)|) \\ &\geq \log(2^{n/2}/2^{\varepsilon n(1+2\log(2e/\varepsilon))}) \text{ (applying Claim 25 for } \alpha = 4\varepsilon \text{ and size } n/2) \\ &\geq n/2 - \varepsilon n(1+2\log(2e/\varepsilon)) \geq \frac{n}{2}(1 - 16\varepsilon\log(1/\varepsilon))\end{aligned}$$

◀

The deterministic bound for the edit distance follows immediately. We now derive the randomised lower bound. We show $\bar{\Omega}(\log n)$ and $\bar{\Omega}(1/\varepsilon)$ space lower bounds separately, and then combine them to yield the theorem.

▶ **Corollary 27.** *Any randomised streaming ε-property tester for L_{mirror} under the edit distance uses space $\bar{\Omega}(\log n)$.*

Proof. A streaming ε-property tester for a VPL L with an input of length n can be viewed as an automaton A_n whose states are the memory states of the algorithm. Deterministic algorithms are deterministic automata, and randomised algorithms are probabilistic automata (see Rabin [22] for definitions).

Consider a randomised streaming ε-property testing algorithm A_n for L_{mirror} on inputs of length n. Let $L(A_n) = \{u \in \{0,1,\bar{0},\bar{1}\}^n \mid A_n \text{ accepts } u \text{ with probability } \geq 2/3\}$. By definition, a deterministic streaming algorithm that recognizes $L(A_n)$ is a streaming ε-property tester for L_{mirror}. By [7, Lemma 6.4], the space complexity of A_n is at least the logarithm of the space complexity of the deterministic one, that is $\bar{\Omega}(\log(n(1-16\varepsilon\log(1/\varepsilon)))) = \bar{\Omega}(\log n)$. ◀

We now show the $\bar{\Omega}(1/\varepsilon)$ bound. Consider a VPL $L_{disj} = \{x\bar{y} \mid x,y \in \{0,1\}^n \text{ and } \forall 1 \leq i \leq n, x[i]\cdot y[i] = 0\}$.

▷ Claim 28. Let $\alpha = \lfloor 1/\varepsilon \rfloor$ and assume that n is a multiple of α. Define a morphism $\phi : \Sigma^* \to \Sigma^*$ such that for any $a \in \Sigma$ we have $\phi(a) = a^{6n/\alpha}$. Consider a word $u = x\bar{y}$, where $x,y \in \Sigma^\alpha$. If $u \in L_{disj}$, then $\mathsf{ed}(\phi(u), L_{disj}) = 0$, and otherwise $\mathsf{ed}(\phi(u), L_{disj}) > \varepsilon n$.

Proof. The first part of the claim is obvious. The rest of the proof is devoted to the case $u \notin L_{disj}$. Assume by contradiction that $\mathsf{ed}(\phi(u), L_{disj}) \leq \varepsilon n$, in other words, that there exists a sequence of at most εn edits such that when applied to $\phi(u)$, we obtain a word in L_{disj}. W.l.o.g. assume that the edits are applied only to the first half of $\phi(u)$, i.e. to $\phi(x)$. Since $u \notin L_{disj}$, there exists i such that $x[i]\cdot y[i] = 1$. Consider the middle part of $\phi(x[i])$, i.e. the factor $w = \phi(u)[(i-1)\cdot(6n/\alpha) + 2n/\alpha + 1, (i-1)\cdot(6n/\alpha) + 4n/\alpha]$. After we apply the at most $\varepsilon n \leq n/\alpha$ edits to $\phi(x)$, there is at least one symbol of w that does not change and therefore is equal to 1, let it be $w[j]$. Moreover, the index of this symbol in the resulting word is between $(i-1)\cdot(6n/\alpha) + n/\alpha + 1$ and $(i-1)\cdot(6n/\alpha) + 5n/\alpha$. On the other hand, $\phi(y[i])$ can be shifted by at most εn positions to the left or to the right. Therefore, $w[j]$ will be aligned with a symbol in $\phi(y[i])$ equal to 1 as well, a contradiction. ◁

The $\bar{\Omega}(1/\varepsilon)$ bound follows immediately from the linear space lower bound for randomised streaming 0-property testing algorithms for L_{disj} [5, 15]. ◀

References

1. Noga Alon, Michael Krivelevich, Ilan Newman, and Mario Szegedy. Regular languages are testable with a constant number of queries. *SIAM Journal on Computing*, 30(6):1842–1862, 2001. doi:10.1137/S0097539700366528.
2. Ajesh Babu, Nutan Limaye, Jaikumar Radhakrishnan, and Girish Varma. Streaming algorithms for language recognition problems. *Theoretical Computer Science*, 494:13–23, 2013.
3. Eldar Fischer, Frédéric Magniez, and Michael de Rougemont. Approximate satisfiability and equivalence. *SIAM Journal on Computing*, 39(6):2251–2281, 2010.
4. Eldar Fischer, Frédéric Magniez, and Tatiana Starikovskaya. Improved bounds for testing Dyck languages. In *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1529–1544. SIAM, 2018. doi:10.1137/1.9781611975031.100.

5 Nathanaël François, Frédéric Magniez, Michel De Rougemont, and Olivier Serre. Streaming property testing of visibly pushdown languages. *CoRR*, abs/1505.03334, 2015. arXiv:1505.03334.

6 Nathanaël François, Frédéric Magniez, Michel de Rougemont, and Olivier Serre. Streaming property testing of visibly pushdown languages. In *Proceedings of the 24th Annual European Symposium on Algorithms*, volume 57 of *LIPIcs*, pages 43:1–43:17, 2016. doi:10.4230/LIPIcs.ESA.2016.43.

7 Moses Ganardi. *Language recognition in the sliding window model*. PhD thesis, University of Siegen, Germany, 2019.

8 Moses Ganardi. Visibly pushdown languages over sliding windows. In *Proceedings of the 36th International Symposium on Theoretical Aspects of Computer Science*, volume 126 of *LIPIcs*, pages 29:1–29:17. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2019. doi:10.4230/LIPIcs.STACS.2019.29.

9 Moses Ganardi, Danny Hucke, Daniel König, Markus Lohrey, and Konstantinos Mamouras. Automata theory on sliding windows. In *Proceedings of 35th International Symposium on Theoretical Aspects of Computer Science*, volume 96 of *LIPIcs*, pages 31:1–31:14, 2018. doi:10.4230/LIPIcs.STACS.2018.31.

10 Moses Ganardi, Danny Hucke, and Markus Lohrey. Querying regular languages over sliding windows. In *Proceedings of the 36th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science*, volume 65 of *LIPIcs*, pages 18:1–18:14, 2016. doi:10.4230/LIPIcs.FSTTCS.2016.18.

11 Moses Ganardi, Danny Hucke, and Markus Lohrey. Randomized sliding window algorithms for regular languages. In *Proceedings of the 45th International Colloquium on Automata, Languages, and Programming*, volume 107 of *LIPIcs*, pages 127:1–127:13, 2018. doi:10.4230/LIPIcs.ICALP.2018.127.

12 Moses Ganardi, Danny Hucke, Markus Lohrey, and Tatiana Starikovskaya. Sliding window property testing for regular languages. In *Proceedings of the 30th International Symposium on Algorithms and Computation*, volume 149 of *LIPIcs*, pages 6:1–6:13, 2019. doi:10.4230/LIPIcs.ISAAC.2019.6.

13 Moses Ganardi, Artur Jeż, and Markus Lohrey. Sliding windows over context-free languages. In *Proceedings of the 43rd International Symposium on Mathematical Foundations of Computer Science*, volume 117 of *LIPIcs*, pages 15:1–15:15, 2018. doi:10.4230/LIPIcs.MFCS.2018.15.

14 Rahul Jain and Ashwin Nayak. The space complexity of recognizing well-parenthesized expressions in the streaming model: The index function revisited. *IEEE Transactions on Information Theory*, 60(10):6646–6668, October 2014. doi:10.1109/TIT.2014.2339859.

15 Bala Kalyanasundaram and Georg Schintger. The probabilistic communication complexity of set intersection. *SIAM Journal on Discrete Mathematics*, 5(4):545–557, 1992.

16 Andreas Krebs, Nutan Limaye, and Srikanth Srinivasan. Streaming algorithms for recognizing nearly well-parenthesized expressions. In *Proceedings of the 36th International Symposium on Mathematical Foundations of Computer Science*, volume 6907 of *LNCS*, pages 412–423, 2011.

17 Frédéric Magniez and Michel de Rougemont. Property testing of regular tree languages. *Algorithmica*, 49(2):127–146, 2007. doi:10.1007/s00453-007-9028-3.

18 Frédéric Magniez, Claire Mathieu, and Ashwin Nayak. Recognizing well-parenthesized expressions in the streaming model. *SIAM J. Comput.*, 43(6):1880–1905, 2014. doi:10.1137/130926122.

19 Antoine Ndione, Aurélien Lemay, and Joachim Niehren. Approximate membership for regular languages modulo the edit distance. *Theor. Comput. Sci.*, 487:37–49, 2013. doi:10.1016/j.tcs.2013.03.004.

20 Antoine Ndione, Aurélien Lemay, and Joachim Niehren. Sublinear DTD validity. In *International Conference on Language and Automata Theory and Applications*, volume 8977 of *LNCS*, pages 739–751, 2015. doi:10.1007/978-3-319-15579-1_58.

21 Michal Parnas, Dana Ron, and Ronitt Rubinfeld. Testing membership in parenthesis languages. *Random Struct. Algorithms*, 22(1):98–138, 2003. doi:10.1002/rsa.10067.

22 Michael O. Rabin. Probabilistic automata. *Information and control*, 6(3):230–245, 1963.

23 Andrew Chi-Chih Yao. Probabilistic computations: Toward a unified measure of complexity. In *Proceedings of the 18th Annual Symposium on Foundations of Computer Science*, pages 222–227, 1977.

Datalog-Expressibility for Monadic and Guarded Second-Order Logic

Manuel Bodirsky
Institut für Algebra, TU Dresden, Germany

Simon Knäuer
Institut für Algebra, TU Dresden, Germany

Sebastian Rudolph
Computational Logic Group, TU Dresden, Germany

Abstract

We characterise the sentences in Monadic Second-order Logic (MSO) that are over finite structures equivalent to a Datalog program, in terms of an existential pebble game. We also show that for every class $\mathcal{C}$ of finite structures that can be expressed in MSO and is closed under homomorphisms, and for all $\ell, k \in \mathbb{N}$, there exists a *canonical* Datalog program Π of width (ℓ, k), that is, a Datalog program of width (ℓ, k) which is sound for $\mathcal{C}$ (i.e., Π only derives the goal predicate on a finite structure $\mathfrak{A}$ if $\mathfrak{A} \in \mathcal{C}$) and with the property that Π derives the goal predicate whenever *some* Datalog program of width (ℓ, k) which is sound for $\mathcal{C}$ derives the goal predicate. The same characterisations also hold for Guarded Second-order Logic (GSO), which properly extends MSO. To prove our results, we show that every class $\mathcal{C}$ in GSO whose complement is closed under homomorphisms is a finite union of constraint satisfaction problems (CSPs) of ω-categorical structures.

2012 ACM Subject Classification Theory of computation → Finite Model Theory

Keywords and phrases Monadic Second-order Logic, Guarded Second-order Logic, Datalog, constraint satisfaction, homomorphism-closed, conjunctive query, primitive positive formula, pebble game, ω-categoricity

Digital Object Identifier 10.4230/LIPIcs.ICALP.2021.120

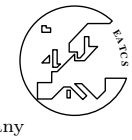
Category Track B: Automata, Logic, Semantics, and Theory of Programming

Related Version *Previous Version*: `https://arxiv.org/abs/2010.05677`

Funding *Manuel Bodirsky*: The author has received funding from the European Research Council (Grant Agreement no. 681988, CSP-Infinity).
Simon Knäuer: The author is supported by DFG Graduiertenkolleg 1763 (QuantLA).
Sebastian Rudolph: The author has received funding from the European Research Council (Grant Agreement no. 771779, DeciGUT).

1 Introduction

Monadic Second-order Logic (MSO) is an important logic in theoretical computer science. By Büchi's theorem, a formal language can be defined in MSO if and only if it is regular (see, e.g., [24]). MSO sentences can be evaluated in polynomial time on classes of structures whose treewidth is bounded by a constant; this is known as Courcelle's theorem [16]. The latter result even holds for the more expressive logic of *Guarded Second-order Logic (GSO)* [21, 18], which extends First-order Logic by second-order quantifiers over *guarded relations*. Guarded Second-order Logic contains *Guarded First-order Logic* (which itself captures many description logics [20]).



Another fundamental formalism in theoretical computer science, which is heavily studied in database theory, is *Datalog* (see, e.g., [24]). Every Datalog program can be evaluated on finite structures in polynomial time. Like MSO, Datalog strikes a good balance between expressivity and good mathematical and computational properties. Two important parameters of a Datalog program Π are the maximal arity ℓ of its auxiliary predicates (IDBs), and the maximal number k of variables per rule in Π. We then say that Π has *width* (ℓ, k), following the terminology of Feder and Vardi [19]. These parameters are important both in theory and in practice: ℓ closely corresponds to the exponent of the size of the memory space and k to the exponent of the number of computation steps needed when evaluating Π on a given structure (see, e.g., [4]).

In some scenarios we are interested in having the good computational properties of expressibility in Datalog *and* having the good computational properties of expressibility in MSO. A wide variety of popular query formalisms (among them (unions of) conjunctive queries, (2-way conjunctive) regular path queries, monadic Datalog, guarded Datalog, monadically defined queries, or nested monadically defined queries) are known to be both in Datalog and GSO [25]. Also, all these formalisms have favourable properties when it comes to static analysis, most notably decidable query containment [25]. Note that on the contrary, query containment in unrestricted Datalog is undecidable, as is query containment in unrestricted MSO / GSO. So it is really the interplay of the restrictions imposed by both formalisms that is required to ensure decidability of a central task in databases and that makes this fragment interesting and worthwhile investigating.

In this paper we investigate two questions that (perhaps surprisingly) turn out to be closely related:

1. Which classes of finite structures are simultaneously expressible in MSO and in Datalog?
2. Which *constraint satisfaction problems (CSPs)* can be expressed in MSO, or, more generally, in GSO?

For a structure $\mathfrak{B}$ with a finite relational signature τ, the *constraint satisfaction problem for* $\mathfrak{B}$ is the class of all finite τ-structures that homomorphically map to $\mathfrak{B}$. Every finite-domain constraint satisfaction problem can already be expressed in monotone monadic SNP (MMSNP; [19]), which is a small fragment of MSO. On the other hand, the constraint satisfaction problem for $(\mathbb{Q}; <)$, which is the class of all finite acyclic digraphs $(V; E)$, cannot be expressed in MMSNP [6], but can be expressed in MSO by the sentence

$$\forall X \neq \emptyset \; \exists x \in X \; \forall y \in X \colon \neg E(x, y).$$

The class of CSPs of arbitrary infinite structures $\mathfrak{B}$ is quite large; it is easy to see that a class $\mathcal{D}$ of finite structures with a finite relational signature τ is a CSP of a countably infinite structure if and only if

- it is closed under disjoint unions, and
- $\mathfrak{A} \in \mathcal{D}$ for any $\mathfrak{A}$ that maps homomorphically to some $\mathfrak{A}' \in \mathcal{D}$.

The second item can equivalently be rephrased as the *complement* of $\mathcal{D}$ (meant within the class of all finite τ-structures; this comment applies throughout and will be omitted in the following) being *closed under homomorphisms*: a class $\mathcal{C}$ is closed under homomorphisms if for any structure $\mathfrak{A} \in \mathcal{C}$ that maps homomorphically to some $\mathfrak{C}$ we have $\mathfrak{C} \in \mathcal{C}$. Examples of classes of structures that are closed under homomorphisms naturally arise from Datalog. We say that a class $\mathcal{C}$ of finite τ-structures *is definable in Datalog*[1] if there exists a Datalog

[1] Warning: Feder and Vardi [19] say that a CSP is in Datalog if its *complement* in the class of all finite τ-structures is in Datalog.

program Π with a distinguished predicate nullary goal such that Π derives goal on a finite τ-structure if and only if the structure is in $\mathcal{C}$; in this case, we write $[\![\Pi]\!]$ for $\mathcal{C}$. Every class of τ-structures in Datalog is closed under homomorphisms. However, not every class of finite structures in Datalog describes the complement of a CSP: consider for example, for unary predicates R and B, the class $\mathcal{C}_{R,B}$ of finite $\{R,B\}$-structures $\mathfrak{A}$ such that $R^{\mathfrak{A}}$ is empty or $B^{\mathfrak{A}}$ is empty. Clearly, $\mathcal{C}_{R,B}$ is not closed under disjoint unions. However, a finite structure is in $\mathcal{C}_{R,B}$ if and only if the Datalog program that consists of just one rule

$$\mathsf{goal} :- R(x), B(y)$$

does not derive goal on that structure.

An important class of CSPs is the class of CSPs for structures $\mathfrak{B}$ that are countably infinite and *ω-categorical*. A structure $\mathfrak{B}$ is *ω-categorical* if all countable models of the first-order theory of $\mathfrak{B}$ are isomorphic. A well-known example of an ω-categorical structure is $(\mathbb{Q};<)$, which is a result due to Cantor [15]. Constraint satisfaction problems of ω-categorical structures can be evaluated in polynomial time on classes of treewidth bounded by some constant $k \in \mathbb{N}$, by a result of Bodirsky and Dalmau [7]. The polynomial-time algorithm presented by Bodirsky and Dalmau is in fact a Datalog program of width $(k-1,k)$. A Datalog program Π is called *sound* for a class of τ-structures $\mathcal{C}$ if $[\![\Pi]\!] \subseteq \mathcal{C}$. Bodirsky and Dalmau showed that if $\mathcal{C}$ is the complement of the CSP of an ω-categorical τ-structure $\mathfrak{B}$ then there exists for all $\ell, k \in \mathbb{N}$ a *canonical Datalog program of width (ℓ,k) for $\mathcal{C}$*, i.e., a Datalog program Π of width (ℓ,k) such that

- Π is sound for $\mathcal{C}$, and
- $[\![\Pi']\!] \subseteq [\![\Pi]\!]$ for every Datalog program Π' of width (ℓ,k) which is sound for $\mathcal{C}$.

Moreover, whether the canonical Datalog program of width (ℓ,k) for $\mathcal{C}$ derives goal on a given τ-structure $\mathfrak{A}$ can be characterised in terms of the existential pebble game from finite model theory, played on $(\mathfrak{A},\mathfrak{B})$ [7]. The *existential ℓ,k pebble game* is played by two players, called *Spoiler* and *Duplicator* (see, e.g., [17, 19, 23]). Spoiler starts by placing k pebbles on elements $a_1,\dots,a_k$ of $\mathfrak{A}$, and Duplicator responds by placing k pebbles $b_1,\dots,b_k$ on $\mathfrak{B}$. If the map that sends $a_1,\dots,a_k$ to $b_1,\dots,b_k$ is not a partial homomorphism from $\mathfrak{A}$ to $\mathfrak{B}$, then the game is over and Spoiler wins. Otherwise, Spoiler removes all but at most ℓ pebbles from $\mathfrak{A}$, and Duplicator has to respond by removing the corresponding pebbles from $\mathfrak{B}$. Then Spoiler can again place all his pebbles on $\mathfrak{A}$, and Duplicator must again respond by placing her pebbles on $\mathfrak{B}$. If the game continues forever, then Duplicator wins. If $\mathfrak{B}$ is a finite, or more generally a countable ω-categorical structure then Spoiler has a winning strategy for the existential ℓ,k pebble game on $(\mathfrak{A},\mathfrak{B})$ if and only if the canonical Datalog program for $\mathrm{CSP}(\mathfrak{B})$ derives goal on $\mathfrak{A}$ (Theorem 19). This connection played an essential role in proving Datalog inexpressibility results, for example for the class of finite-domain CSPs [2] (leading to a complete classification of those finite structures $\mathfrak{B}$ such that the complement of $\mathrm{CSP}(\mathfrak{B})$ can be expressed in Datalog [3]).

Results and Consequences

We present a characterisation of those GSO sentences Φ that are over finite structures equivalent to a Datalog program. Our characterisation involves a variant of the existential pebble game from finite model theory, which we call the *(ℓ,k)-game*. This game is defined for a homomorphism-closed class $\mathcal{C}$ of finite τ-structures, and it is played by the two players Spoiler and Duplicator on a finite τ-structure $\mathfrak{A}$ as follows.

- Duplicator picks a countable τ-structure $\mathfrak{B}$ such that $\mathrm{CSP}(\mathfrak{B}) \cap \mathcal{C} = \emptyset$.
- The game then continues as the existential (ℓ,k) pebble game played by Spoiler and Duplicator on $(\mathfrak{A},\mathfrak{B})$.

In Section 4 we show that a GSO sentence Φ is over finite structures equivalent to a Datalog program of width (ℓ, k) if and only if

- $[\![\Phi]\!]$ is closed under homomorphisms, and
- Spoiler wins the existential (ℓ, k)-game for $[\![\Phi]\!]$ on $\mathfrak{A}$ if and only if $\mathfrak{A} \models \Phi$.

We also show that for every GSO sentence Φ whose class of finite models $\mathcal{C}$ is closed under homomorphisms and for all $\ell, k \in \mathbb{N}$ there exists a canonical Datalog program Π of width (ℓ, k) for $\mathcal{C}$ (Theorem 22). To prove these results, we first show that every class of finite structures in GSO whose complement is closed under homomorphisms is a finite union of CSPs that can also be expressed in GSO (Lemma 16; an analogous statement holds for MSO). Moreover, every CSP in GSO is the CSP of a countable ω-categorical structure (Corollary 10); this allows us to use results from [7] to make the link to existential pebble games. We also present an example of such a CSP which is even expressible in MSO and coNP-complete, and hence not the CSP of a reduct of a finitely bounded homogeneous structure, unless NP=coNP (Proposition 23). Note that our results imply that every class of finite structures that can be expressed both in in GSO and in Datalog is a finite intersection of the complements of CSPs for ω-categorical structures. In general, it is not true that a Datalog program describes a finite intersection of complements of CSPs (we present a counterexample in Example 18).

2 Preliminaries

In the entire text, τ denotes a finite signature containing relation symbols and sometimes also constant symbols. If $R \in \tau$ is a relation symbol, we write $ar(R)$ for its arity. If $\mathfrak{A}$ is a τ-structure we use the corresponding capital roman A letter to denote the domain of $\mathfrak{A}$; the domains of structures are assumed to be non-empty. If $R \in \tau$, then $R^{\mathfrak{A}} \subseteq A^{ar(R)}$ denotes the corresponding relation of $\mathfrak{A}$.

A *primitive positive τ-formula* (in database theory also *conjunctive query*) is a first-order τ-formula without disjunction, negation, and universal quantification. Every primitive positive formula is equivalent to a formula of the form

$$\exists x_1, \dots, x_n (\psi_1 \wedge \cdots \wedge \psi_m)$$

where $\psi_1, \dots, \psi_m$ are atomic τ-formulas, i.e., formulas built from relation symbols in τ or equality. An *existential positive τ-formula* is a first-order τ-formula without negation and universal quantification. We write $\psi(x_1 \dots, x_n)$ if the free variables of ψ are from $x_1, \dots, x_n$. If $\mathfrak{A}$ is a τ-structure and $\psi(x_1, \dots, x_n)$ is a τ-formula, then the relation

$$R := \{(a_1, \dots, a_n) \mid \mathfrak{A} \models \psi(a_1, \dots, a_n)\}$$

is called the relation *defined by ψ over $\mathfrak{A}$*; if ψ can be chosen to be primitive positive (or existential positive) then R is called *primitively positively definable* (or *existentially positively definable*, respectively).

For all logics over the signature τ considered in this text, we say that two formulas $\Phi(x_1, \dots, x_n)$ and $\Psi(x_1, \dots, x_n)$ are *equivalent (over finite structures)* if for all (finite) τ-structures $\mathfrak{A}$ and all $a_1, \dots, a_n \in A$ we have

$$\mathfrak{A} \models \Phi(a_1, \dots, a_n) \Leftrightarrow \mathfrak{A} \models \Psi(a_1, \dots, a_n).$$

It is easy to see that every existential positive τ-formula is a disjunction of primitive positive τ-formulas (and hence referred to as a *union of conjunctive queries* in database theory). Formulas without free variables are called *sentences*; in database theory, formulas are often called *queries* and sentences are often called *Boolean queries*. If Φ is a sentence, we write $[\![\Phi]\!]$ for the class of all finite models of Φ.

A *reduct* of a relational structure $\mathfrak{A}$ is a structure $\mathfrak{A}'$ obtained from $\mathfrak{A}$ by dropping some of the relations, and $\mathfrak{A}$ is called an *expansion* of $\mathfrak{A}'$.

2.1 Datalog

In this section we refer to the finite set of relation and constant symbols τ as *EDBs* (for *extensional database predicates*). Let ρ be a finite set of new relation symbols, called the *IDBs* (for *intensional database predicates*). A Datalog program is a set of rules of the form

$$\psi_0 :- \psi_1, \ldots, \psi_n$$

where ψ_0 is an atomic ρ-formula and $\psi_1, \ldots, \psi_n$ are atomic $(\rho \cup \tau)$-formulas; we also assume that every variable that appears in the head also appears in the body. If $\mathfrak{A}$ is a τ-structure, and Π is a Datalog program with EDBs τ and IDBs ρ, then a $(\tau \cup \rho)$-expansion $\mathfrak{A}'$ of $\mathfrak{A}$ is called a *fixed point of* Π *on* $\mathfrak{A}$ if $\mathfrak{A}'$ satisfies the sentence

$$\forall \bar{x}(\psi_0 \vee \neg\psi_1 \vee \cdots \vee \neg\psi_n)$$

for each rule $\psi_0 :- \psi_1, \ldots, \psi_n$. If $\mathfrak{A}_1$ and $\mathfrak{A}_2$ are two $(\rho \cup \tau)$-structures with the same domain A, then $\mathfrak{A}_1 \cap \mathfrak{A}_2$ denotes the $(\rho \cup \tau)$-structure with domain A such that $R^{\mathfrak{A}_1 \cap \mathfrak{A}_2} := R^{\mathfrak{A}_1} \cap R^{\mathfrak{A}_2}$. Note that if $\mathfrak{A}_1$ and $\mathfrak{A}_2$ are two fixed points of Π on $\mathfrak{A}$, then $\mathfrak{A}_1 \cap \mathfrak{A}_2$ is a fixed point of Π on $\mathfrak{A}$, too. Hence, there exists a unique smallest (with respect to inclusion) fixed point of Π on $\mathfrak{A}$, which we denote by $\Pi(\mathfrak{A})$. It is well-known that if $\mathfrak{A}$ is a finite structure then $\Pi(\mathfrak{A})$ can be computed in polynomial time in the size of $\mathfrak{A}$ [24]. If $R \in \rho$, we also say that Π *defines* $R^{\Pi(\mathfrak{A})}$ *on* $\mathfrak{A}$. A Datalog program together with a distinguished predicate $R \in \rho$ may also be viewed as a formula, which we also call a *Datalog query*, and which over a given τ-structure $\mathfrak{A}$ denotes the relation $R^{\Pi(\mathfrak{A})}$. If the distinguished predicate has arity 0, we often call it the *goal predicate*; we say that Π *derives* goal *on* $\mathfrak{A}$ if $\mathsf{goal}^{\Pi(\mathfrak{A})} = \{()\}$. The class $\mathcal{C}$ of finite τ-structures $\mathfrak{A}$ such that Π derives goal on $\mathfrak{A}$ is called *the class of finite τ-structures defined by* Π, and denoted by $[\![\Pi]\!]$. Note that this class $\mathcal{C}$ is definable in universal second-order logic (we have to express that in every expansion of the input by relations for the IDBs that satisfies all the rules of the Datalog program the goal predicate is non-empty).

2.2 Second-Order Logic

Second-order logic is the extension of first-order logic which additionally allows existential and universal quantification over relations; that is, if R is a relation symbol and ϕ is a second-order $\tau \cup \{R\}$-formula, then $\exists R\colon \phi$ and $\forall R\colon \phi$ are second-order τ-formulas. If $\mathfrak{A}$ is a τ-structure and Φ is a second-order τ-sentence, we write $\mathfrak{A} \models \Phi$ (and say that $\mathfrak{A}$ is a model of Φ) if $\mathfrak{A}$ satisfies Φ, which is defined in the usual Tarskian style. We write $[\![\Phi]\!]$ for the class of all finite models of Φ. A second-order formula is called *monadic* if all second-order variables are unary. We use syntactic sugar and also write $\forall x \in X\colon \psi$ instead of $\forall x(X(x) \Rightarrow \psi)$ and $\exists x \in X\colon \psi$ instead of $\exists x(X(x) \wedge \psi)$.

2.3 Guarded Second-Order Logic

Guarded Second-order Logic (GSO), introduced by Grädel, Hirsch, and Otto [21], is the extension of *guarded first-order logic* by second-order quantifiers. Guarded (first-order) τ-formulas are defined inductively by the following rules [1]:

1. all atomic τ-formulas are guarded τ-formulas;
2. if ϕ and ψ are guarded τ-formulas, then so are $\phi \wedge \psi$, $\phi \vee \psi$, and $\neg\phi$.
3. if $\psi(\bar{x}, \bar{y})$ is a guarded τ-formula and $\alpha(\bar{x}, \bar{y})$ is an atomic τ-formula such that all free variables of ψ occur in α then $\exists\bar{y}\big(\alpha(\bar{x}, \bar{y}) \wedge \psi(\bar{x}, \bar{y})\big)$ and $\forall\bar{y}\big(\alpha(\bar{x}, \bar{y}) \Rightarrow \psi(\bar{x}, \bar{y})\big)$ are guarded τ-formulas.

Guarded second-order formulas are defined similarly, but we additionally allow (unrestricted) second-order quantification; GSO generalises Courcelle's logic MSO_2 from graphs to general relational structures.

▶ **Definition 1.** *A second-order τ-formula is called* guarded *if it is defined inductively by the rules (1)–(3) for guarded first-order logic and additionally by second-order quantification.*

There are many semantically equivalent ways of introducing GSO [21]. Let $\mathfrak{B}$ be a τ-structure. Then $(t_1, \ldots, t_n) \in B^n$ is called *guarded in* $\mathfrak{B}$ if there exists an atomic τ-formula ϕ and $b_1, \ldots, b_k$ such that $\mathfrak{B} \models \phi(b_1, \ldots, b_k)$ and $\{t_1, \ldots, t_n\} \subseteq \{b_1, \ldots, b_k\}$. Note that (for $n = 1$) every element of B is guarded (because of the atomic formula $x = x$). A relation $R \subseteq B^n$ is called *guarded* if all tuples in R are guarded. Note that all unary relations are guarded. If Ψ is an arbitrary second-order sentence, we say that a finite structure $\mathfrak{A}$ *satisfies* Ψ *with guarded semantics*, in symbols $\mathfrak{A} \models_g \Phi$, if all second-order quantifiers in Ψ are evaluated over guarded relations only. Note that for MSO sentences, the usual semantics and the guarded semantics coincide.

▶ **Proposition 2** (see [21])**.** *Guarded Second-order Logic and full Second-order Logic with guarded semantics are equally expressive.*

It follows that GSO is at least as expressive as MSO. There are Datalog programs that are equivalent to a GSO sentence, but not to an MSO sentence. The proof is based on a variant of an example of a Datalog query in GSO given in [13] (Example 2).

▶ **Proposition 3.** *There is a Datalog query that can be expressed in GSO but not in MSO.*

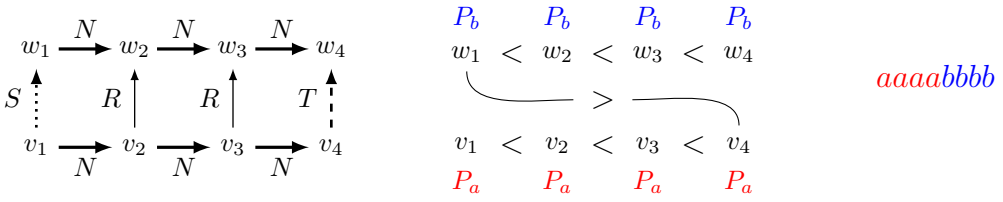
Proof. Let τ be the signature consisting of the binary relation symbols S, T, R, N, and let $\mathcal{C}$ be the class of finite τ-structures such that the following Datalog program with one binary IDB U derives goal.

$$
\begin{aligned}
U(x, y) &:\!- S(x, y) \\
U(x', y') &:\!- U(x, y), N(x, x'), N(y, y'), R(x', y') \\
\mathsf{goal} &:\!- U(x, y), T(x, y)
\end{aligned}
$$

◀

On the left of Figure 1 one can find an example of a $\{S, T, R, N\}$-structure $\mathfrak{B}$ where the given Datalog program derives goal. To show that $\mathcal{C}$ is not MSO definable, suppose for contradiction that there exists an MSO sentence Φ such that $[\![\Phi]\!] = \mathcal{C}$. We use Φ to construct an MSO sentence Ψ which holds on a finite word $w \in \{a, b\}^*$ (represented as a structure with signature $P_a, P_b, <$ in the usual way [24]) if and only if $w \in \{a^n b^n \mid n \geq 1\}$; this contradicts the theorem of Büchi-Elgot-Trakhtenbrot (see, e.g., [24]). Let Φ' be the MSO sentence obtained from Φ by replacing all subformulas of Φ of the form

- $S(x, y)$ by a formula $\phi_S(x, y)$ that states that x is the smallest element with respect to $<$, that $P_b(y)$, and that there is no $z < y$ in P_b;
- $T(x, y)$ by a formula $\phi_T(x, y)$ that states that $P_a(x)$, that there is no $z > x$ in P_a, and that y is the largest element with respect to $<$;
- $R(x, y)$ by the formula $\phi_R(x, y)$ given by $x < y$;
- $N(x, y)$ by a formula $\phi_N(x, y)$ stating that y is the next element after x with respect to $<$.



(a) Structure $\mathfrak{B}$ **(b)** Structure $\mathfrak{A}$ **(c)** Word $w_{\mathfrak{A}}$

Figure 1 An example of an $\{S, T, R, N\}$-structure $\mathfrak{B}$ in the class $\mathcal{C}$ of Proposition 3.

The resulting MSO sentence Ψ_1 has the signature $\{P_a, P_b, <\}$; let Ψ be the conjunction of Ψ_1 with the sentence Ψ_2 which states that for all $x, y \in A$, if $x < y$ and $P_a(y)$ then $P_a(x)$. We first show that if $\mathfrak{A}$ is a $\{<, P_a, P_b\}$-structure that represents a word $w_{\mathfrak{A}} \in \{a, b\}^*$, then $\mathfrak{A} \models \Psi$ if and only if $w_{\mathfrak{A}}$ is of the form $a^n b^n$ for some $n \geq 1$. Let $\mathfrak{B}$ be the $\{S, T, R, N\}$-structure such that for $X \in \{S, T, R, N\}$ we have $X^{\mathfrak{B}} := \{(x, y) \mid \mathfrak{A} \models \phi_X(x, y)\}$. See Figure 1 for an example of a structure $\mathfrak{A}$ such that $w_{\mathfrak{A}} = a^4 b^4$ and the corresponding $\{S, T, R, N\}$-structure $\mathfrak{B}$.

If $w_{\mathfrak{A}}$ is of the form $a^n b^n$ for some $n \geq 1$, then $\mathfrak{A}$ clearly satisfies Ψ_2. To show that it also satisfies Ψ_1, let $v_1, \ldots, v_n, w_1, \ldots, w_n \in A$ be such that $\{v_1, \ldots, v_n\} = P_a^{\mathfrak{A}}$ and $\{w_1, \ldots, w_n\} = P_b^{\mathfrak{A}}$ such that for all $i, j \in \{1, \ldots, n\}$, if $i < j$ then $v_i <^{\mathfrak{A}} v_j$ and $w_i <^{\mathfrak{A}} w_j$. Then

$$\begin{aligned} &(v_1, w_1) \in S^{\mathfrak{B}}, \qquad (v_n, w_n) \in T^{\mathfrak{B}}, \\ &(v_i, w_i) \in R^{\mathfrak{B}} \text{ for all } i \in \{2, \ldots, n-1\}, \\ &(v_i, v_{i+1}), (w_i, w_{i+1}) \in N^{\mathfrak{B}} \text{ for all } i \in \{1, \ldots, n-1\}. \end{aligned} \tag{1}$$

It follows that $\mathfrak{B}$ satisfies Φ and therefore $\mathfrak{A} \models \Psi$.

For the converse direction, suppose that $\mathfrak{A} \models \Psi$. Clearly, $w_{\mathfrak{A}} \in a^* b^*$ because $\mathfrak{A} \models \Psi_2$. Moreover, since $\mathfrak{A} \models \Psi_1$ we have that $\mathfrak{B} \models \Phi$, and hence there exist $n \in \mathbb{N}$ and elements $v_1, \ldots, v_n, w_1, \ldots, w_n \in A$ such that $\mathfrak{B}$ satisfies (1). We first prove that $P_a^{\mathfrak{A}} = \{v_1, \ldots, v_n\}$ and $|P_a^{\mathfrak{A}}| = n$. Since $(v_n, w_n) \in T^{\mathfrak{B}}$ we have $\phi_T(v_n, w_n)$ and hence $v_n \in P_a^{\mathfrak{A}}$. Since $\mathfrak{B} \models N(v_1, v_2), \ldots, N(v_{n-1}, v_n)$ we have that $v_1 < v_2 < \cdots < v_{n-1} < v_n$ holds in $\mathfrak{A}$ and it also follows that $|P_a^{\mathfrak{A}}| = n$. Then for every $i \in n$ we have that $v_i \in P_a^{\mathfrak{A}}$ because $v_i \leq v_n$, $v_n \in P_a^{\mathfrak{A}}$, and $w_{\mathfrak{A}} \in a^* b^*$. Now suppose for contradiction that there exists $x \in P_a^{\mathfrak{A}} \setminus \{v_1, \ldots, v_n\}$; choose x largest with respect to $<^{\mathfrak{A}}$. Since $(v_n, w_n) \in T^{\mathfrak{B}}$ and $x \in P_a^{\mathfrak{A}}$ we must have $x \leq v_n$, and hence $x < v_n$ since $x \notin \{v_1, \ldots, v_n\}$. Then there exists $y \in A$ such that $\phi_N(x, y)$ holds in $\mathfrak{A}$. Since $y \leq v_n$, $v_n \in P_a^{\mathfrak{A}}$, and $w_{\mathfrak{A}} \in a^* b^*$, we must have $P_a^{\mathfrak{A}}$. By the maximal choice of x we get that $y = v_i$ for some $i \in \{1, \ldots, n\}$. But then $\phi_N(x, v_i)$ implies that $x \in \{v_1, \ldots, v_{n-1}\}$, a contradiction. Similarly, one can prove that $P_b^{\mathfrak{A}} = \{w_1, \ldots, w_n\}$ and that $|P_b^{\mathfrak{A}}| = n$. This implies that $w_{\mathfrak{A}} = a^n b^n$.

We finally have to prove that $\mathcal{C}$ is in GSO. Let Φ be the GSO $\{S, T, R, N\}$ sentence with existentially quantified unary relations V, W, and existentially quntified binary relations $R' \subseteq R$ and $N' \subseteq N$, which states that

- there are elements $v_1, v_n \in V$ and $w_1, w_n \in W$ such that $S(v_1, w_1)$ and $T(v_n, w_n)$ hold;
- for every $x \in V \setminus \{v_1\}$ there exists a unique element $y \in V \setminus \{v_n\}$ such that $N'(y, x)$ holds;
- for every $x \in V \setminus \{v_n\}$ there exists a unique element $y \in V \setminus \{v_1\}$ such that $N'(x, y)$ holds;

- for every $x \in W \setminus \{w_1\}$ there exists a unique element $y \in W \setminus \{w_n\}$ such that $N'(y, x)$ holds;
- for every $x \in W \setminus \{w_n\}$ there exists a unique element $y \in W \setminus \{w_1\}$ such that $N'(x, y)$ holds;
- for all $v \in V$ and $w \in W$ we have that $N'(v_1, v) \wedge N'(w_1, w)$ implies $R'(v, w)$.
- for all $v, v' \in V \setminus \{v_1, v_n\}$ and $w, w' \in W \setminus \{w_1, w_n\}$ we have that $R'(v, w) \wedge N'(v, v') \wedge N'(w, w')$ implies $R'(v, w)$.
- For all $v \in V$ and $w \in W$ we have that $N'(v, v_n) \wedge N'(w, w_n)$ implies $R'(v, w)$.

Then Φ holds on a finite $\{S, T, R, N\}$-structure $\mathfrak{B}$ if and only if B has elements $v_1, \ldots, v_n, w_1, \ldots, w_n$ satisfying (1), which is the case if and only if $\mathfrak{B} \in \mathcal{C}$.

Sometimes, we will also use the term GSO (MSO, Datalog) to denote all problems (i.e., all classes of structures) that can be expressed in the formalism. In particular, this justifies to say that a certain CSP is *in* GSO (MSO, Datalog).

3 Homomorphism-Closed GSO

We prove that the class of finite models of a GSO sentence is a finite union of CSPs of ω-categorical structures whenever its complement is closed under homomorphisms. In particular, every CSP in GSO (and therefore every CSP in MSO) is the CSP of an ω-categorical structure. CSPs that can be formulated as the CSP of an ω-categorical structure have been characterised [10]; this characterisation will be recalled in the next section.

3.1 CSPs for Countably Categorical Structures

By the theorem of Ryll-Nardzewski, a countable structure $\mathfrak{B}$ is ω-categorical if and only if for every $n \in \mathbb{N}$ there are finitely many orbits of the componentwise action of the automorphism group of $\mathfrak{B}$ on B^n (see, e.g., [22]). We now present a condition that characterises classes of structures that are CSPs of ω-categorical structures. Let $\mathcal{C}$ be a class of finite τ-structures. Let Λ_n be the class of primitive positive τ-formulas with free variables $x_1, \ldots, x_n$ whose canonical database is in $\mathcal{C}$. We define $\sim_n^{\mathcal{C}}$ to be the equivalence relation on Λ_n such that $\phi_1 \sim_n^{\mathcal{C}} \phi_2$ holds if for all primitive positive τ-formulas $\psi(x_1, \ldots, x_n)$ we have that $\phi_1(x_1, \ldots, x_n) \wedge \psi(x_1, \ldots, x_n)$ is satisfiable in a structure from $\mathcal{C}$ if and only if $\phi_2(x_1, \ldots, x_n) \wedge \psi(x_1, \ldots, x_n)$ is satisfiable in a structure from $\mathcal{C}$. The *index* of an equivalence relation is the number of its equivalence classes.

▶ **Theorem 4** (Bodirsky, Hils, Martin [10], Theorem 4.27). *Let $\mathcal{C}$ be a constraint satisfaction problem. Then there is an ω-categorical structure $\mathfrak{B}$ such that $\mathcal{C} = \mathrm{CSP}(\mathfrak{B})$ iff $\sim_n^{\mathcal{C}}$ has finite index for all n. Moreover, the structure $\mathfrak{B}$ can be chosen so that for all $n \in \mathbb{N}$ the orbits of the componentwise action of the automorphism group of $\mathfrak{B}$ on B^n are primitively positively definable in $\mathfrak{B}$.*

▶ **Example 5.** The structure $\mathfrak{B}_1 := (\mathbb{Z}; <)$ is not ω-categorical. However, $\sim_n^{\mathrm{CSP}(\mathfrak{B}_1)}$ has finite index for all n, and indeed $\mathrm{CSP}(\mathbb{Z}; <) = \mathrm{CSP}(\mathbb{Q}; <)$ and $(\mathbb{Q}; <)$ is ω-categorical. On the other hand, for $\mathfrak{B}_2 := (\mathbb{Z}; \mathrm{Succ})$ we have that the index $\sim_2^{\mathrm{CSP}(\mathfrak{B}_2)}$ is infinite, and it follows that there is no ω-categorical structure $\mathfrak{B}$ such that $\mathrm{CSP}(\mathfrak{B}_2) = \mathrm{CSP}(\mathfrak{B})$; see [6].

A rich source of examples of ω-categorical structures are structures with finite relational signature that are *homogeneous*, i.e., every isomorphism between finite substructures can be extended to an automorphism. There are uncountably many countable homogeneous

digraphs with pairwise distinct CSP, and it follows that there are homogeneous digraphs with undecidable CSPs. A structure $\mathfrak{B}$ is called *finitely bounded* if there exists a finite set $\mathcal{F}$ of finite structures such that a finite structure $\mathfrak{A}$ embeds into $\mathfrak{B}$ if and only if no structure in $\mathcal{F}$ embeds into $\mathfrak{A}$.

It is well-known that if a structure is ω-categorical, then all of its *reducts* are ω-categorical as well [22]. Moreover, it is easy to see that the CSP of reducts of finitely bounded structures is in NP. It has been conjectured that the CSP of reducts of finitely bounded homogeneous structures is in P or NP-complete [12]; this conjecture generalises the finite-domain complexity dichotomy that was conjectured by Feder and Vardi [19] and proved by Bulatov [14] and by Zhuk [26].

3.2 Quantifier Rank

In order to construct ω-categorial structures for a given CSP in GSO, we need to verify the condition given in Theorem 4; in this context, it will be convenient to work with signatures that also contain constant symbols. The *quantifier rank* of a second-order τ-formula Φ is the maximal number of nested (first-order or second-order) quantifiers in Φ; for this definition, we view Φ as a second-order sentence with guarded semantics, just as in [5]. If $\mathfrak{A}$ and $\mathfrak{B}$ are τ-structures and $q \in \mathbb{N}$ we write $\mathfrak{A} \equiv_q^{\mathrm{GSO}} \mathfrak{B}$ if $\mathfrak{A}$ and $\mathfrak{B}$ satisfy the same GSO τ-sentences of quantifier rank at most q.

▶ **Lemma 6** (Proposition 3.3 in [5]). *Let $q \in \mathbb{N}$ and τ be a finite signature with relation and constant symbols. Then $\equiv_q^{\mathrm{GSO}}$ is an equivalence relation with finite index on the class of all finite τ-structures. Moreover, every class of $\equiv_q^{\mathrm{GSO}}$ can be defined by a single GSO sentence with quantifier rank q. The analogous statements hold for MSO as well.*

If $\mathfrak{A}$ is a τ-structure and $\bar{a}$ is a k-tuple of elements of A, then we write $(\mathfrak{A}, \bar{a})$ for a $\tau \cup \{c_1, \dots, c_k\}$-structure expanding $\mathfrak{A}$ where $c_1, \dots, c_k$ denote fresh constant symbols being mapped to the corresponding entries of $\bar{a}$. If $\mathfrak{A}$ and $\mathfrak{B}$ are τ-structures and $\bar{a} \in A^k$, $\bar{b} \in B^k$, and when writing $(\mathfrak{A}, \bar{a}) \equiv_q^{\mathrm{GSO}} (\mathfrak{B}, \bar{b})$ we implicitly assume that we have chosen the same constant symbols for $\bar{a}$ and for $\bar{b}$.

▶ **Lemma 7** (Proposition 3.4 in [5]). *Let $q \in \mathbb{N}$ and let $\mathfrak{A}$ and $\mathfrak{B}$ be τ-structures. Then $\mathfrak{A} \equiv_{q+1}^{\mathrm{GSO}} \mathfrak{B}$ if and only if the following properties hold:*

- *(first-order forth) For every $a \in A$, there exists $b \in B$ such that $(\mathfrak{A}, a) \equiv_q^{\mathrm{GSO}} (\mathfrak{B}, b)$.*
- *(first-order back) For every $b \in B$, there exists $a \in A$ such that $(\mathfrak{A}, a) \equiv_q^{\mathrm{GSO}} (\mathfrak{B}, b)$.*
- *(second-order forth) For every expansion $\mathfrak{A}'$ of $\mathfrak{A}$ by a guarded relation, there exists an expansion $\mathfrak{B}'$ of $\mathfrak{B}$ by a guarded relation such that $\mathfrak{A}' \equiv_q^{\mathrm{GSO}} \mathfrak{B}'$.*
- *(second-order back) For every expansion $\mathfrak{B}'$ of $\mathfrak{B}$ by a guarded relation, there exists an expansion $\mathfrak{A}'$ of $\mathfrak{A}$ by a guarded relation such that $\mathfrak{A}' \equiv_q^{\mathrm{GSO}} \mathfrak{B}'$.*

In the following, τ denotes a finite relational signature.

▶ **Definition 8.** *Let $\rho := \{c_1, \dots, c_n\}$ be a finite set of constant symbols. Then $\mathcal{D}_n$ is defined to be the set of all pairs $(\mathfrak{A}, \mathfrak{B})$ of finite $(\tau \cup \rho)$-structures such that*

- *$c^{\mathfrak{A}} = c^{\mathfrak{B}}$ for all constant symbols $c \in \rho$;*
- *$\{c_1^{\mathfrak{A}}, \dots, c_n^{\mathfrak{A}}\} = A \cap B = \{c_1^{\mathfrak{B}}, \dots, c_n^{\mathfrak{B}}\}$.*

We write $\mathfrak{A} \uplus \mathfrak{B}$ for the structure with domain $A \cup B$ such that $R^{\mathfrak{A} \uplus \mathfrak{B}} := R^{\mathfrak{A}} \cup R^{\mathfrak{B}}$ for each relation symbol $R \in \tau$ and $c^{\mathfrak{A} \uplus \mathfrak{B}} = c^{\mathfrak{A}} = c^{\mathfrak{B}}$ for each constant symbol $c \in \rho$.

The following theorem in the special case of $n = 0$ is Proposition 4.1 in [5].

▶ **Theorem 9.** *Let $q, n, r, s \in \mathbb{N}$, let $(\mathfrak{A}_1, \mathfrak{B}_1), (\mathfrak{A}_2, \mathfrak{B}_2) \in \mathcal{D}_n$, and let $\bar{a}_1 \in (A_1)^r$, $\bar{a}_2 \in (A_2)^r$, $\bar{b}_1 \in (B_1)^s$, $\bar{b}_2 \in (B_2)^s$ be such that $(\mathfrak{A}_1, \bar{a}_1) \equiv_q^{\mathrm{GSO}} (\mathfrak{A}_2, \bar{a}_2)$ and $(\mathfrak{B}_1, \bar{b}_1) \equiv_q^{\mathrm{GSO}} (\mathfrak{B}_2, \bar{b}_2)$. Then*

$$(\mathfrak{A}_1 \uplus \mathfrak{B}_1, \bar{a}_1, \bar{b}_1) \equiv_q^{\mathrm{GSO}} (\mathfrak{A}_2 \uplus \mathfrak{B}_2, \bar{a}_2, \bar{b}_2).$$

Proof. Our proof is by induction on q. Every quantifier-free formula is a Boolean combination of atomic formulas, so for $q = 0$ it suffices to consider atomic formulas ϕ. By symmetry, it suffices to show that if $(\mathfrak{A}_1 \uplus \mathfrak{B}_1, \bar{a}_1, \bar{b}_1) \models \phi$ then $(\mathfrak{A}_2 \uplus \mathfrak{B}_2, \bar{a}_2, \bar{b}_2) \models \phi$. Then ϕ is built using a relation symbol $R \in \tau$, and the tuple that witnesses the truth of ϕ in $\mathfrak{A}_1 \uplus \mathfrak{B}_1$ must be from $R^{\mathfrak{A}_1}$ or from $R^{\mathfrak{B}_1}$, by the definition of $\mathfrak{A}_1 \uplus \mathfrak{B}_1$. We first consider the former case; the latter case can be treated similarly. If a constant that appears in ϕ is from $A_1 \cap B_1$, then by the definition of $\mathcal{D}_n$ this element is denoted by a constant symbol $c \in \rho$, and therefore we may assume without loss of generality that ϕ is a formula over the signature of $(\mathfrak{A}_1, \bar{a}_1)$. Hence, $(\mathfrak{A}_1, \bar{a}_1) \models \phi$ and by assumption $(\mathfrak{A}_2, \bar{a}_2) \models \phi$. This in turn implies that $(\mathfrak{A}_2 \uplus \mathfrak{B}_2, \bar{a}_2, \bar{b}_2) \models \phi$.

For the inductive step, suppose that the claim holds for q, and that $(\mathfrak{A}_1, \bar{a}_1) \equiv_{q+1}^{\mathrm{GSO}} (\mathfrak{A}_2, \bar{a}_2)$ and $(\mathfrak{B}_1, \bar{b}_1) \equiv_{q+1}^{\mathrm{GSO}} (\mathfrak{B}_2, \bar{b}_2)$. By symmetry and Lemma 7 it suffices to verify the properties (first-order forth) and (second-order forth). Let $c_1 \in A_1 \cup B_1$. We may assume that $c_1 \in A_1$; the case that $c_1 \in B_1$ can be shown similarly. By Lemma 7, there exists $c_2 \in A_2$ such that $(\mathfrak{A}_1, \bar{a}_1, c_1) \equiv_q^{\mathrm{GSO}} (\mathfrak{A}_2, \bar{a}_2, c_2)$. By the inductive assumption, this implies that

$$(\mathfrak{A}_1 \uplus \mathfrak{B}_1, \bar{a}_1, c_1, \bar{b}_1) \equiv_q^{\mathrm{GSO}} (\mathfrak{A}_2 \uplus \mathfrak{B}_2, \bar{a}_2, c_2, \bar{b}_2)$$

and concludes the proof of (first-order forth).

Now let R be a guarded relation of $\mathfrak{A}_1 \uplus \mathfrak{B}_1$ of arity k. Let $\mathfrak{A}_1'$ be the expansion of $\mathfrak{A}_1$ by the guarded relation $R \cap A_1^k$, and $\mathfrak{B}_1'$ be the expansion of $\mathfrak{B}_1$ by the guarded relation $R \cap B_1^k$. By Lemma 7 there are expansions $\mathfrak{A}_2'$ of $\mathfrak{A}$ and $\mathfrak{B}_2'$ of $\mathfrak{B}_2$ by guarded relations such that $(\mathfrak{A}_1', \bar{a}_1) \equiv_q^{\mathrm{GSO}} (\mathfrak{A}_2', \bar{a}_2)$ and $(\mathfrak{B}_1', \bar{b}_1) \equiv_q^{\mathrm{GSO}} (\mathfrak{B}_2', \bar{b}_2)$. By the inductive assumption, this implies that $(\mathfrak{A}_1' \uplus \mathfrak{B}_1', \bar{a}_1, \bar{b}_1) \equiv_q^{\mathrm{GSO}} (\mathfrak{A}_2' \uplus \mathfrak{B}_2', \bar{a}_2, \bar{b}_2)$, which completes the proof of (second-order forth). ◀

▶ **Corollary 10.** *Let $\mathcal{C}$ be a CSP that can be expressed in GSO. Then there exists a countable ω-categorical structure $\mathfrak{B}$ such that $\mathcal{C} = \mathrm{CSP}(\mathfrak{B})$.*

Proof. Let τ be the signature of $\mathcal{C}$, and let Φ be a GSO τ-formula with quantifierrank q such that $\mathcal{C} = [\![\Phi]\!]$. By Theorem 4 it suffices to show that the equivalence relation $\sim_n^{\mathcal{C}}$ has finite index for every $n \in \mathbb{N}$. Let $\rho := \{c_1, \ldots, c_n\}$ be a set of new constant symbols. By Lemma 6, there exists an $m \in \mathbb{N}$ such that $\equiv_q^{\mathrm{GSO}}$ has m equivalence classes on $(\tau \cup \rho)$-structures. If $\phi(x_1, \ldots, x_n)$ is a primitive positive τ-formula, then define $\mathfrak{S}_\phi$ to be the $(\tau \cup \rho)$-structure whose elements are the equivalence classes of the smallest equivalence relation on the variables of ϕ that contains all pairs x, y such that ϕ contains the conjunct $x = y$, and such that $(C_1, \ldots, C_n) \in R^{\mathfrak{S}}$ for $R \in \tau$ if and only if there are $y_1 \in C_1, \ldots, y_n \in C_2$ such that $R(y_1, \ldots, y_n)$ is a conjunct of ϕ; finally, we set $c_i^{\mathfrak{S}_\phi} := [x_i]$ for all $i \in \{1, \ldots, n\}$.

We claim that if $\mathfrak{S}_\phi \equiv_q^{\mathrm{GSO}} \mathfrak{S}_\psi$, then $\phi \sim_n^{\mathcal{C}} \psi$. Let $\theta(x_1, \ldots, x_n)$ be a primitive positive τ-formula; we may assume that the existentially quantified variables of θ are disjoint from the existentially quantified variables of ϕ and of ψ, so that $(\mathfrak{S}_\phi, \mathfrak{S}_\theta), (\mathfrak{S}_\psi, \mathfrak{S}_\theta) \in \mathcal{D}_n$. Since $\mathfrak{S}_\phi \equiv_q^{\mathrm{GSO}} \mathfrak{S}_\psi$ and $\mathfrak{S}_\theta \equiv_q^{\mathrm{GSO}} \mathfrak{S}_\theta$, we have $\mathfrak{S}_\phi \uplus \mathfrak{S}_\theta \equiv_q^{\mathrm{GSO}} \mathfrak{S}_\psi \uplus \mathfrak{S}_\theta$ by Theorem 9. Now suppose that $\phi \wedge \theta$ is satisfiable in a model of Φ. This is the case if and only if $\mathfrak{S}_\phi \uplus \mathfrak{S}_\theta$ satisfies Φ, which in turn implies that $\mathfrak{S}_\psi \uplus \mathfrak{S}_\theta$ satisfies Φ since Φ has quantifierrank q. This in turn is the case if and only if $\psi \wedge \theta$ is satisfiable in a model of Φ, which proves the claim.

The claim implies that $\sim_n^{\mathcal{C}}$ has at most m equivalence classes, concluding the proof. ◀

▶ **Example 11.** Let Φ be the following MSO sentence.

$$\forall X \big(\exists x\colon X(x) \Rightarrow \exists x, y \in X\ \forall z \in X(\neg E(x,z) \vee \neg E(y,z))\big)$$

It is easy to see that $[\![\Phi]\!]$ is closed under disjoint unions and that its complement is closed under homomorphisms. Corollary 10 implies that there exists a countable ω-categorical structure with $\mathrm{CSP}(\mathfrak{B}) = [\![\Phi]\!]$.

3.3 Finite Unions of CSPs

In this section we prove that every class in GSO whose complement is closed under homomorphisms is a finite union of CSPs (Lemma 16); the statement announced at the beginning of Section 3 then follows (Corollary 17). Throughout this section, let $\mathcal{C}$ be a non-empty class of finite τ-structures whose complement is closed under homomorphisms. In particular, $\mathcal{C}$ contains the structure $\mathfrak{I}$ with only one element where all relations are empty.

Let $\sim$ be the equivalence relation defined on $\mathcal{C}$ by letting $\mathfrak{A} \sim \mathfrak{B}$ if for every $\mathfrak{C} \in \mathcal{C}$ we have $\mathfrak{A} \uplus \mathfrak{C} \in \mathcal{C}$ if and only if $\mathfrak{B} \uplus \mathfrak{C} \in \mathcal{C}$; here $\uplus$ denotes the usual disjoint union of structures, which is a special case of Definition 8 for $n = 0$. Note that the equivalence classes of $\sim$ are in one-to-one correspondence to the equivalence classes of $\sim_0^{\mathcal{C}}$. Also note that $\mathcal{C}$ is closed under disjoint unions if and only if $\sim$ has only one equivalence class.

If $\mathfrak{A} \in \mathcal{C}$, then we write $[\mathfrak{A}]$ for the equivalence class of $\mathfrak{A}$ with respect to $\sim$. The following observations are immediate consequences from the definitions:

1. each $\sim$-equivalence class is closed under homomorphic equivalence.
2. each $\sim$-equivalence class is closed under disjoint unions.
3. $\mathfrak{A} \in [\mathfrak{I}]$ if and only if $\mathfrak{A} \uplus \mathfrak{B} \in \mathcal{C}$ for all $\mathfrak{B} \in \mathcal{C}$.

▶ **Lemma 12.** *Let $\mathfrak{A} \in \mathcal{C}$ and let $\mathcal{D}$ be the smallest subclass of $\mathcal{C}$ that contains $[\mathfrak{A}]$ and whose complement is closed under homomorphisms. Then*

1. *$\mathcal{D}$ is a union of equivalence classes of $\sim$, and*
2. *if $\sim$ has more than one equivalence class, then $\mathcal{C} \setminus \mathcal{D}$ is non-empty.*

Proof. Let $\mathfrak{C} \in [\mathfrak{A}]$, let $\mathfrak{B}$ be a finite structure with a homomorphism to $\mathfrak{C}$, and let $\mathfrak{B}' \in [\mathfrak{B}]$. Since $\mathfrak{B} \uplus \mathfrak{C}$ and $\mathfrak{C}$ are homomorphically equivalent, we have that $\mathfrak{B} \uplus \mathfrak{C} \sim \mathfrak{C}$. We claim that $\mathfrak{B}' \uplus \mathfrak{C} \sim \mathfrak{C}$. To see this, let $\mathfrak{D} \in \mathcal{C}$. Then

$$\begin{aligned}
\mathfrak{C} \uplus \mathfrak{D} \in \mathcal{C} &\Leftrightarrow (\mathfrak{B} \uplus \mathfrak{C}) \uplus \mathfrak{D} \in \mathcal{C} && \text{(since } \mathfrak{B} \uplus \mathfrak{C} \sim \mathfrak{C}\text{)} \\
&\Leftrightarrow \mathfrak{B} \uplus (\mathfrak{C} \uplus \mathfrak{D}) \in \mathcal{C} \\
&\Leftrightarrow \mathfrak{B}' \uplus (\mathfrak{C} \uplus \mathfrak{D}) \in \mathcal{C} && \text{(since } \mathfrak{B} \sim \mathfrak{B}'\text{)} \\
&\Leftrightarrow (\mathfrak{B}' \uplus \mathfrak{C}) \uplus \mathfrak{D} \in \mathcal{C}
\end{aligned}$$

which shows the claim. So $\mathfrak{B}' \uplus \mathfrak{C} \in [\mathfrak{C}] = [\mathfrak{A}]$. Since $\mathfrak{B}'$ has a homomorphism to $\mathfrak{B}' \uplus \mathfrak{C}$ we obtain that $\mathfrak{B}' \in \mathcal{D}$; this proves the first statement.

To prove the second statement, first observe that the statement is clear if $\mathfrak{A} \in [\mathfrak{I}]$, since the complement of $[\mathfrak{I}]$ is closed under homomorphisms. The statement therefore follows from the assumption that $\sim$ has more than one equivalence class. Otherwise, if $\mathfrak{A} \notin [\mathfrak{I}]$, then there exists a structure $\mathfrak{B} \in \mathcal{C}$ such that $\mathfrak{A} \uplus \mathfrak{B} \notin \mathcal{C}$. Then $\mathfrak{B} \in \mathcal{C} \setminus \mathcal{D}$ can be shown indirectly as follows: otherwise $\mathfrak{B}$ would have a homomorphism to a structure $\mathfrak{A}' \in [\mathfrak{A}]$. Since $\mathfrak{B} \uplus \mathfrak{A}'$ is homomorphically equivalent to $\mathfrak{A}'$, we have $\mathfrak{B} \uplus \mathfrak{A}' \sim \mathfrak{A}' \sim \mathfrak{A}$ and in particular $\mathfrak{B} \uplus \mathfrak{A}' \in \mathcal{C}$. But $\mathfrak{B} \uplus \mathfrak{A}' \in \mathcal{C}$ if and only if $\mathfrak{B} \uplus \mathfrak{A} \in \mathcal{C}$ since $\mathfrak{A} \sim \mathfrak{A}'$. This is in contradiction to our assumption on $\mathfrak{B}$. ◀

▶ **Example 13.** We consider a signature $\tau := \{R_1, R_2, R_3\}$ of unary relation symbols. Define for every $i \in \{1,2,3\}$ the τ-structure $\mathfrak{S}_i$ to be a one-element structure where R_i is non-empty and R_j, for $j \neq i$, is empty. Let

$$\mathcal{C} := \mathrm{CSP}(\mathfrak{S}_1 \uplus \mathfrak{S}_2) \cup \mathrm{CSP}(\mathfrak{S}_2 \uplus \mathfrak{S}_3) \cup \mathrm{CSP}(\mathfrak{S}_3 \uplus \mathfrak{S}_1).$$

Clearly, the complement of $\mathcal{C}$ is closed under homomorphisms. The equivalence classes of $\sim$ can be described as follows. For distinct $i, j \in \{1,2,3\}$,

$$\begin{aligned}[\mathfrak{S}_i \uplus \mathfrak{S}_j] &= \mathrm{CSP}(\mathfrak{S}_i \uplus \mathfrak{S}_j) \setminus (\mathrm{CSP}(\mathfrak{S}_i) \cup \mathrm{CSP}(\mathfrak{S}_j)) \\ [\mathfrak{S}_i] &= \mathrm{CSP}(\mathfrak{S}_i) \setminus [\mathfrak{I}] \\ [\mathfrak{I}] &= \mathrm{CSP}(\mathfrak{I}).\end{aligned}$$

For the remainder of the section we fix a GSO τ-sentence Φ of quantifier rank q. Recall that Lemma 6 asserts that the equivalence relation $\equiv_q^{\mathrm{GSO}}$ on the class of finite τ-structures has finitely many equivalence classes $\mathcal{C}_1, \ldots, \mathcal{C}_m$, and that each of the equivalence classes $\mathcal{C}_i$ can be defined by a single GSO τ-sentence Ψ_i with quantifier rank q; we write $T_q^\tau := \{\Psi_1, \ldots, \Psi_m\}$ for this set of GSO sentences. Let $J \subseteq \{1, \ldots, m\}$ be such that $\{\Psi_j \in T_q^\tau \mid j \in J\}$ is exactly the set of all sentences in T_q^τ that imply Φ. Then $|J|$ is called the *degree* of Φ. It is easy to see that the degree of Φ is exactly the index of $\equiv_q^{\mathrm{GSO}}$ restricted to $[\![\Phi]\!]$. Let $\sim$ be the equivalence relation defined in the beginning of this section for the class $\mathcal{C} := [\![\Phi]\!]$.

▶ **Lemma 14.** *For every $\sim$-class $\mathcal{D}$ there exists $I \subseteq \{1, \ldots, m\}$ such that $\mathcal{D} = \bigcup_{i \in I} [\![\Psi_i]\!]$.*

Proof. As in the proof of Corollary 10 one can use Theorem 9 to show for all finite τ-structures $\mathfrak{A}, \mathfrak{B}$ that if $\mathfrak{A} \equiv_q^{\mathrm{GSO}} \mathfrak{B}$, then $\mathfrak{A} \sim \mathfrak{B}$. This means that $\mathcal{D}$ is a union of $\equiv_q^{\mathrm{GSO}}$-classes and therefore there exists $I \subseteq J \subseteq \{1, \ldots, m\}$ such that $\mathcal{D} = \bigcup_{i \in I} [\![\Psi_i]\!]$. ◀

▶ **Corollary 15.** *The index of $\sim$ is smaller than or equal to the degree of Φ.*

▶ **Lemma 16.** *If the complement of $[\![\Phi]\!]$ is closed under homomorphisms, then there are GSO τ-sentences $\Phi_1, \ldots, \Phi_t$ each of which describes a CSP such that Φ is equivalent to $\Phi_1 \vee \cdots \vee \Phi_t$. If Φ is an MSO sentence, then $\Phi_1, \ldots, \Phi_t$ can be be chosen to be MSO sentences as well.*

Proof. We prove the statement by induction on the degree n of Φ. By Lemma 15 the equivalence relation $\sim$ has at most n equivalence classes on τ-structures. Hence, if $n = 1$, then $[\![\Phi]\!]$ is closed under disjoint unions, and we are done.

Let $\mathfrak{A}_1, \ldots, \mathfrak{A}_s$ be τ-structures such that $\{[\mathfrak{A}_1], \ldots, [\mathfrak{A}_s]\}$ is the set of all equivalence classes of $\sim$ that are distinct from $[\mathfrak{I}]$. Let $\mathcal{D}_i$ be the smallest subclass of $[\![\Phi]\!]$ that contains $[\mathfrak{A}_i]$ and whose complement is closed under homomorphisms. Note that $[\![\Phi]\!] = \bigcup_{i \leq s} \mathcal{D}_i$ since $[\mathfrak{I}]$ is contained in $\mathcal{D}_i$ for all $i \leq s$. By Lemma 12 (1), each $\mathcal{D}_i$ is a union of $\sim$-classes which are themselves a union of $\equiv_q^{\mathrm{GSO}}$-classes by Lemma 14. It follows that there exists $I_i \subseteq \{1, \ldots, m\}$ such that $\mathcal{D}_i = \bigcup_{j \in I_i} [\![\Psi_j]\!]$. We define $\Phi_i := \bigvee_{j \in I_i} \Psi_j$. Note that the GSO sentence Φ_i is of quantifier rank q such that $\mathcal{D}_i = [\![\Phi_i]\!]$. Hence, Φ is equivalent to $\bigvee_{i \leq s} \Phi_i$. Lemma 12 (2) asserts that $[\![\Phi]\!] \setminus \mathcal{D}_i$ is non-empty, and hence the degree of Φ_i must be strictly smaller than n for all $i \in \{1, \ldots, s\}$. The statement now follows from the inductive assumption. The same argument applies to MSO as well. ◀

Lemma 16 together with Corollary 10 implies the following.

▶ **Corollary 17.** *Every GSO sentence which is closed under homomorphisms is equivalent to a finite conjunction of GSO sentences each of which describes the complement of a CSP of a countable ω-categorical structure. The analogous statement holds for MSO.*

Not every homomorphism-closed class of structures that can be expressed in Second-order Logic is a finite intersection of complements of CSPs. We even have an example of a class of finite τ-structures that can be expressed in Datalog but cannot be written in this form.

▶ **Example 18.** Let S and T be unary, and let R be a binary relation symbol. Let $\mathcal{C}$ be the class of all finite $\{S,T,R\}$-structures $\mathfrak{A}$ such that the following Datalog program Π with the binary IDB E derives goal on $\mathfrak{A}$.

$$\begin{aligned} E(x,y) &:\!- S(x), S(y) \\ E(x,y) &:\!- E(x',y'), R(x',x), R(y',y) \\ \mathsf{goal} &:\!- T(x), E(x,x'), R(x',y) \end{aligned}$$

For $n \in \mathbb{N}$, let $\mathfrak{P}_n$ be the $\{S,T,R\}$-structure on the domain $\{1,\dots,n\}$ with

$$S^{\mathfrak{P}_n} := \{1\} \qquad T^{\mathfrak{P}_n} := \{n\} \qquad R^{\mathfrak{P}_n} := \big\{(i,i+1) \mid i \in \{1,\dots,n-1\}\big\}.$$

It is easy to see that each of the structures in $\{\mathfrak{P}_n \mid n \geq 1\}$ is not contained in $\mathcal{C}$, and that the disjoint union of $\mathfrak{P}_i$ and $\mathfrak{P}_j$, for $i \neq j$, is contained in $\mathcal{C}$. It follows that $\mathcal{C}$ is not a finite intersection of complements of CSPs (and, by Corollary 17, cannot be expressed in GSO).

4 Canonical Datalog Programs

A remarkable fact about the expressive power of Datalog for constraint satisfaction problems over finite domains is the existence of *canonical Datalog programs* [19]; this has been generalised to CSPs for ω-categorical structures.

▶ **Theorem 19** (Bodirsky and Dalmau [7])**.** *Let $\mathfrak{B}$ be a countable ω-categorical τ-structure. Then for all $\ell, k \in \mathbb{N}$ there exists a canonical Datalog program Π of width (ℓ,k) for the complement of* CSP($\mathfrak{B}$)*. Moreover, for every finite τ-structure $\mathfrak{A}$ the following are equivalent:*

- *Π derives* goal *on $\mathfrak{A}$;*
- *Spoiler has a winning strategy for the existential (ℓ,k)-pebble game on $(\mathfrak{A},\mathfrak{B})$.*

We later need the following well-known fact.

▶ **Lemma 20.** *If $\mathcal{C}_1$ and $\mathcal{C}_2$ are in Datalog, then so are $\mathcal{C}_1 \cup \mathcal{C}_2$ and $\mathcal{C}_1 \cap \mathcal{C}_2$. If Π_1 and Π_2 are Datalog programs of width (ℓ,k), then there is a Datalog program Π of width (ℓ,k) for $[\![\Pi_1]\!] \cup [\![\Pi_2]\!]$ and for $[\![\Pi_1]\!] \cap [\![\Pi_2]\!]$.*

Proof. For union, let Π be obtained by taking the union of the rules of Π_1 and of Π_2, possibly after renaming IDB predicate names to make them disjoint except for goal. For intersection, we proceed similarly, but we first rename the symbol goal in Π_1 to goal_1 and the symbol goal in Π_2 to goal_2. Finally we add the new rule $\mathsf{goal} :\!- \mathsf{goal}_1, \mathsf{goal}_2$ to the union of Π_1 and Π_2. It is clear that these constructions preserve the width. ◀

▶ **Theorem 21.** *Let Φ be a GSO sentence such that $[\![\Phi]\!]$ is closed under homomorphisms. Let $\ell, k \in \mathbb{N}$. Then there exists a canonical Datalog program Π of width (ℓ,k) for $[\![\Phi]\!]$.*

Proof. By Corollary 17 there are GSO sentences $\Phi_1, \dots, \Phi_m$ and ω-categorical structures $\mathfrak{B}_1, \dots, \mathfrak{B}_m$ such that Φ is equivalent to $\Phi_1 \wedge \dots \wedge \Phi_m$ and $[\![\neg\Phi_i]\!] = \mathrm{CSP}(\mathfrak{B}_i)$. Let Π_i be the canonical Datalog program for $\mathrm{CSP}(\mathfrak{B}_i)$ which exists by Theorem 19. Then Lemma 20 implies that there exists a Datalog program Π such that $[\![\Pi]\!] = [\![\Pi_1]\!] \cap \dots \cap [\![\Pi_m]\!]$. It is clear that Π is sound for $[\![\Phi]\!]$. To see that Π is a canonical Datalog program for $[\![\Phi]\!]$, suppose that $\mathfrak{A}$ is such that some Datalog program Π' of width (ℓ, k) which is sound for $[\![\Phi]\!]$ derives goal on $\mathfrak{A}$. Since, for every $i \in \{1, \dots, m\}$, the program Π' is also sound for $[\![\Phi_i]\!]$, and Π_i is a canonical Datalog program for $[\![\Phi_i]\!]$, the program Π_i derives goal on $\mathfrak{A}$. Hence, $\mathfrak{A} \in [\![\Pi]\!] = [\![\Pi_1]\!] \cap \dots \cap [\![\Pi_m]\!]$. ◀

▶ **Theorem 22.** *Let Φ be a GSO sentence. Then $[\![\Phi]\!]$ can be defined in Datalog if and only if*

1. *$[\![\Phi]\!]$ is closed under homomorphisms, and*
2. *there exist $\ell, k \in \mathbb{N}$ such that for all finite structures $\mathfrak{A}$, Spoiler wins the (ℓ, k)-game for $[\![\Phi]\!]$ on $\mathfrak{A}$ if and only if $\mathfrak{A} \models \Phi$.*

Proof. First suppose that $[\![\Phi]\!]$ is in Datalog. That is, there exists $\ell, k \in \mathbb{N}$ and a Datalog program Π of width (ℓ, k) such that $[\![\Phi]\!] = [\![\Pi]\!]$. Then clearly $[\![\Phi]\!]$ is closed under homomorphisms, and by Lemma 16, there are GSO sentences $\Phi_1, \dots, \Phi_m$ such that Φ is equivalent to $\Phi_1 \wedge \dots \wedge \Phi_m$ and $[\![\Phi_i]\!]$ is the complement of a CSP, for each $i \in \{1, \dots, m\}$. Corollary 10 implies that there exists an ω-categorical structure $\mathfrak{B}_i$ such that $\mathrm{CSP}(\mathfrak{B}_i) = [\![\neg\Phi_i]\!]$. Now suppose that $\mathfrak{A}$ is a finite τ-structure such that $\mathfrak{A} \models \Phi$. Then Spoiler wins the (ℓ, k)-game as follows. Suppose that Duplicator plays the countable structure $\mathfrak{B}$ such that $\mathrm{CSP}(\mathfrak{B}) \cap [\![\Phi]\!] = \emptyset$. Then $\mathrm{CSP}(\mathfrak{B}) \cap [\![\Phi_i]\!] = \emptyset$ for some $i \in \{1, \dots, m\}$; otherwise, if there is a structure $\mathfrak{A}_i \in \mathrm{CSP}(\mathfrak{B}) \cap [\![\Phi_i]\!]$ for every $i \in \{1, \dots, m\}$, then the disjoint union of $\mathfrak{A}_1, \dots, \mathfrak{A}_m$ satisfies Φ_i since Φ_i is closed under homomorphisms, and is in $\mathrm{CSP}(\mathfrak{B})$ since $\mathrm{CSP}(\mathfrak{B})$ is closed under disjoint unions; but this is in contradiction to our assumption that $\mathrm{CSP}(\mathfrak{B}) \cap [\![\Phi]\!] = \emptyset$. Hence, $\mathrm{CSP}(\mathfrak{B}) \subseteq \mathrm{CSP}(\mathfrak{B}_i)$ and hence there is a homomorphism h from $\mathfrak{B}$ to $\mathfrak{B}_i$ (see [7]). Note that Π is sound for $\mathrm{CSP}(\mathfrak{B}_i)$, and Π derives goal on $\mathfrak{A}$, and hence Theorem 19 implies that Spoiler wins the existential (ℓ, k)-pebble game on $(\mathfrak{A}, \mathfrak{B}_i)$. But since $\mathfrak{B}$ homomorphically maps to $\mathfrak{B}_i$, this implies that Spoiler wins the existential (ℓ, k)-pebble game on $(\mathfrak{A}, \mathfrak{B}_i)$. Now suppose that $\mathfrak{A} \models \neg\Phi$. Hence, there exists $i \in \{1, \dots, m\}$ such that $\mathfrak{A} \models \neg\Phi_i$. Then Duplicator wins the (ℓ, k)-game as follows. She starts by playing $\mathfrak{B}_i$. Then $\mathfrak{A}$ homomorphically maps to $\mathfrak{B}_i$, and Duplicator can win the existential (ℓ, k) pebble game on $(\mathfrak{A}, \mathfrak{B}_i)$ by always playing along the homomorphism.

For the converse implication, suppose that 1. and 2. hold. Since $[\![\Phi]\!]$ is closed under homomorphisms, Corollary 17 implies that there are GSO sentences $\Phi_1, \dots, \Phi_m$ and ω-categorical structures $\mathfrak{B}_1, \dots, \mathfrak{B}_m$ such that Φ is equivalent to $\Phi_1 \wedge \dots \wedge \Phi_m$ and $[\![\neg\Phi_i]\!] = \mathrm{CSP}(\mathfrak{B}_i)$. By Theorem 19, for every $i \in \{1, \dots, m\}$ there exists a canonical Datalog program Π_i of width (ℓ, k) for $[\![\Phi_i]\!]$. Then Lemma 20 implies that there exists a Datalog program Π such that $[\![\Pi]\!] = [\![\Pi_1]\!] \cap \dots \cap [\![\Pi_m]\!]$. Since each Π_i is sound for $[\![\Phi_i]\!]$, it follows that Π is sound for $[\![\Phi]\!]$. Hence, it suffices to show that if $\mathfrak{A}$ is a finite τ-structure such that $\mathfrak{A} \models \Phi$, then Π derives goal on $\mathfrak{A}$. Since $\mathfrak{A} \models \Phi_i$ for all $i \in \{1, \dots, m\}$, the assumption implies that Spoiler wins the existential (ℓ, k) pebble game on $(\mathfrak{A}, \mathfrak{B}_i)$. By Theorem 19, it follows that Π_i derives goal on $\mathfrak{A}$. Hence, Π derives goal on $\mathfrak{A}$. ◀

5 A coNP-complete CSP in MSO

In this section we show that the class of CSPs in MSO is (under complexity-theoretic assumptions) larger than the class of CSPs for reducts of finitely bounded structures (see Section 3.1). Let $\mathcal{T} = \{\mathfrak{T}_2, \mathfrak{T}_3, \dots\}$ be the set of *Henson tournaments*: the tournament $\mathfrak{T}_n$, for $n \geq 2$, has vertices $0, 1, \dots, n+1$ and the following edges:

- $(i, i+1)$ for $i \in \{0, \dots, n\}$;
- $(0, n+1)$;
- (j, i) for $i+1 < j$ and $(i, j) \neq (0, n+1)$.

The class $\mathcal{C}$ of all finite loopless digraphs that do not embed any of the digraphs from $\mathcal{T}$ is an amalgamation class, and hence there exists a homogenous structure $\mathfrak{H}$ with age $\mathcal{C}$. It has been shown in [9] that CSP($\mathfrak{H}$) is coNP-complete.

▶ **Proposition 23.** CSP($\mathfrak{H}$) *can be expressed in MSO.*

Proof. We have to find an MSO sentence that holds on a given digraph $(V; E)$ if and only if $(V; E)$ does not embed any of the tournaments from $\mathcal{T}$. We specify an MSO $\{X, E\}$-sentence Φ, for a unary relation symbol X, that is true on a finite $\{X, E\}$-structure $\mathfrak{S}$ if and only if $(X^{\mathfrak{S}}; E^{\mathfrak{S}})$ is isomorphic to $\mathfrak{T}_n$, for some $n \geq 2$. In ϕ we existentially quantify over

- two vertices $s, t \in X$ (that stand for the vertex 0 and the vertex $n+1$ in $\mathfrak{T}_n$).
- a partition of $X \setminus \{s\}$ into two sets A and B (they stand for the set of even and the set of odd numbers in $\{1, \dots, n+1\}$).

The formula Φ has the following conjuncts:

1. a first-order formula that states that E defines a tournament on X;
2. a first-order formula that expresses that E is a linear order on A with maximal element a;
3. a first-order formula that expresses that E is a linear order on B with maximal element b;
4. $E(s, t)$, $E(s, a)$, $E(a, b)$, and $E(x, s)$ for all $x \in X \setminus \{a, t\}$;
5. a first-order formula that states that if there is an edge from an element $x \in A$ to an element $y \in B$ then there is precisely one element $z \in A$ such that $(y, z), (z, x) \in E$, unless $y = t$;
6. a first-order formula that states that if there is an edge from an element $x \in B$ to an element $y \in A$ then there is precisely one element $z \in B$ such that $(y, z), (z, x) \in E$, unless $y = t$.

We claim that the MSO sentence $\forall x \colon \neg E(x, x) \wedge \forall X \colon \neg\Phi$ holds on a finite digraph if and only if the digraph is loopless and does not embed $\mathfrak{T}_n$, for all $n \geq 3$. The forwards implication easily follows from the observation that if $(X; T)$ is isomorphic to $\mathfrak{T}_n$, for some $n \geq 2$, then ϕ holds; this is straightforward from the construction of Φ (and the explanations above given in brackets). Conversely, suppose that Φ holds. Then $(X; T)$ is a tournament. We construct an isomorphism f from $(X; T)$ to $\mathfrak{T}_{|X|-1}$ as follows. Define $f(s) := 0$, $f(a) := 1$, and $f(b) = 2$. Since $E(a, b)$, by item 5 there exists exactly one $a' \in A$ such that $E(b, a')$ and $E(a', a)$. Define $f(a') := 3$. If $a' = t$ then we have found an isomorphism with $\mathfrak{T}_2$. Otherwise, the partial map f defined so far is an embedding into $\mathfrak{T}_n$ for some $n \geq 3$. Item 6 and $E(b, a')$ imply that there exists exactly one $b' \in B$ such that $E(a', b')$ and $E(b', b)$, and we define $f(b') := 4$. Continuing in this manner, we eventually define f on all of X and find an isomorphism with $\mathfrak{T}_{|X|-1}$. ◀

This shows that CSP($\mathfrak{H}$) cannot be expressed, unless NP = coNP, as CSP($\mathfrak{B}$) for some reduct of a finitely bounded structure and such CSPs are in NP. We do not know how to show this statement without complexity-theoretic assumptions, even if we just want to rule out that CSP($\mathfrak{H}$) can be expressed as CSP($\mathfrak{B}$) for some reduct of a finitely bounded *homogeneous* structure.

6 Conclusion and Open Problems

We provided a game-theoretic characterisation of those problems in Guarded Second-order Logic that are equivalent to a Datalog program. We also proved the existence of canonical Datalog programs for GSO sentences whose models are closed under homomorphisms. To

prove these results, we showed that every class of finite τ-structures in GSO whose complement is closed under homomorphisms is a finite union of CSPs. We also showed that every CSP in GSO can be formulated as a CSP of an ω-categorical structure. These results also imply that the so-called universal-algebraic approach, which has eventually led to the classification of finite-domain CSPs in Datalog [3], can be applied to study problems that are simultaneously in Datalog and in GSO (also see [11]). Our results might also pave the way towards a syntactic characterisation of Datalog $\cap$ GSO. We close with two open problems.

1. *Nested monadically defined queries (Nemodeq)* have been introduced by Rudolph and Krötzsch [25]; they prove that Nemodeq is contained both in MSO and in Datalog. We ask wether conversely, every problem in MSO $\cap$ Datalog is expressible as a Nemodeq.
2. Is every CSP of a reduct of a finitely bounded homogeneous structure in GSO?

We are also confident that our results will advance the understanding of CSPs (the complements of) which are obtained as the homomorphism-closure of the set of some theory's finite models. For example, the homomorphism-closures of the model sets of guarded- and guarded-negation-theories have recently been found to be GSO-expressible [8] so, by virtue of our results, we immediately know they must be (complements of) ω-categorical CSPs.

References

1 Hajnal Andréka, István Németi, and Johan van Benthem. Modal languages and bounded fragments of predicate logic. *J. Philos. Log.*, 27(3):217–274, 1998. doi:10.1023/A:1004275029985.

2 Albert Atserias, Andrei A. Bulatov, and Anuj Dawar. Affine systems of equations and counting infinitary logic. *Theoretical Computer Science*, 410(18):1666–1683, 2009.

3 Libor Barto and Marcin Kozik. Constraint satisfaction problems solvable by local consistency methods. *Journal of the ACM*, 61(1):3:1–3:19, 2014.

4 Christoph Berkholz. Lower bounds for existential pebble games and k-consistency tests. *Log. Methods Comput. Sci.*, 9(4), 2013. doi:10.2168/LMCS-9(4:2)2013.

5 Achim Blumensath. Monadic second-order logic. Lecture Notes, 2020.

6 Manuel Bodirsky. Complexity of infinite-domain constraint satisfaction. To appear in the LNL Series, Cambridge University Press, 2021.

7 Manuel Bodirsky and Víctor Dalmau. Datalog and constraint satisfaction with infinite templates. *Journal on Computer and System Sciences*, 79:79–100, 2013. A preliminary version appeared in the proceedings of the Symposium on Theoretical Aspects of Computer Science (STACS'05).

8 Manuel Bodirsky, Thomas Feller, Simon Knäuer, and Sebastian Rudolph. On logics and homomorphism closure. In *Proceedings of the Symposium on Logic in Computer Science (LICS)*, 2021. Preprint arXiv:2104.11955.

9 Manuel Bodirsky and Martin Grohe. Non-dichotomies in constraint satisfaction complexity. In Luca Aceto, Ivan Damgard, Leslie Ann Goldberg, Magnús M. Halldórsson, Anna Ingólfsdóttir, and Igor Walukiewicz, editors, *Proceedings of the International Colloquium on Automata, Languages and Programming (ICALP)*, Lecture Notes in Computer Science, pages 184–196. Springer Verlag, July 2008.

10 Manuel Bodirsky, Martin Hils, and Barnaby Martin. On the scope of the universal-algebraic approach to constraint satisfaction. In *Proceedings of the Symposium on Logic in Computer Science (LICS)*, pages 90–99. IEEE Computer Society, July 2010.

11 Manuel Bodirsky, Wied Pakusa, and Jakub Rydval. Temporal constraint satisfaction problems in fixed-point logic. In Holger Hermanns, Lijun Zhang, Naoki Kobayashi, and Dale Miller, editors, *LICS '20: 35th Annual ACM/IEEE Symposium on Logic in Computer Science, Saarbrücken, Germany, July 8-11, 2020*, pages 237–251. ACM, 2020. doi:10.1145/3373718.3394750.

12 Manuel Bodirsky, Michael Pinsker, and András Pongrácz. Projective clone homomorphisms. *Journal of Symbolic Logic*, pages 1–13, 2019. doi:10.1017/jsl.2019.23.

13 Pierre Bourhis, Markus Krötzsch, and Sebastian Rudolph. Reasonable highly expressive query languages - IJCAI-15 distinguished paper (honorary mention). In Qiang Yang and Michael J. Wooldridge, editors, *Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence, IJCAI 2015, Buenos Aires, Argentina, July 25-31, 2015*, pages 2826–2832. AAAI Press, 2015.

14 Andrei A. Bulatov. A dichotomy theorem for nonuniform CSPs. In *58th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2017, Berkeley, CA, USA, October 15-17*, pages 319–330, 2017.

15 Georg Cantor. Über unendliche, lineare Punktmannigfaltigkeiten. *Mathematische Annalen*, 23:453–488, 1884.

16 Bruno Courcelle and Joost Engelfriet. *Graph Structure and Monadic Second-Order Logic: A Language-Theoretic Approach*. Cambridge University Press, 40 W. 20 St. New York, NY, United States, 2012.

17 Victor Dalmau, Phokion G. Kolaitis, and Moshe Y. Vardi. Constraint satisfaction, bounded treewidth, and finite-variable logics. In *Proceedings of the International Conference on Principles and Practice of Constraint Programming (CP)*, pages 310–326, 2002.

18 Michael Elberfeld, Martin Grohe, and Till Tantau. Where first-order and monadic second-order logic coincide. *CoRR*, abs/1204.6291, 2012. `arXiv:1204.6291`.

19 Tomás Feder and Moshe Y. Vardi. The computational structure of monotone monadic SNP and constraint satisfaction: a study through Datalog and group theory. *SIAM Journal on Computing*, 28:57–104, 1999.

20 Erich Grädel. Description logics and guarded fragments of first order logic. In Enrico Franconi, Giuseppe De Giacomo, Robert M. MacGregor, Werner Nutt, and Christopher A. Welty, editors, *Proceedings of the 1998 International Workshop on Description Logics (DL'98), IRST, Povo - Trento, Italy, June 6-8, 1998*, volume 11 of *CEUR Workshop Proceedings*. CEUR-WS.org, 1998. URL: `http://ceur-ws.org/Vol-11/graedel.ps`.

21 Erich Grädel, Colin Hirsch, and Martin Otto. Back and forth between guarded and modal logics. *ACM Trans. Comput. Log.*, 3(3):418–463, 2002.

22 Wilfrid Hodges. *A shorter model theory*. Cambridge University Press, Cambridge, 1997.

23 Phokion G. Kolaitis and Moshe Y. Vardi. On the expressive power of Datalog: Tools and a case study. *Journal of Computer and System Sciences*, 51(1):110–134, 1995.

24 Leonid Libkin. *Elements of Finite Model Theory*. Springer, 2004.

25 Sebastian Rudolph and Markus Krötzsch. Flag & check: Data access with monadically defined queries. In *Proc. 32nd Symposium on Principles of Database Systems (PODS'13)*, pages 151–162. ACM, June 2013. `doi:10.1145/2463664.2465227`.

26 Dmitriy N. Zhuk. A proof of CSP dichotomy conjecture. In *58th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2017, Berkeley, CA, USA, October 15-17*, pages 331–342, 2017. `arXiv:1704.01914`.

Beyond PCSP(1-in-3, NAE)

Alex Brandts
Department of Computer Science, University of Oxford, UK

Stanislav Živný
Department of Computer Science, University of Oxford, UK

Abstract

The promise constraint satisfaction problem (PCSP) is a recently introduced vast generalisation of the constraint satisfaction problem (CSP) that captures approximability of satisfiable instances. A PCSP instance comes with two forms of each constraint: a strict one and a weak one. Given the promise that a solution exists using the strict constraints, the task is to find a solution using the weak constraints. While there are by now several dichotomy results for fragments of PCSPs, they all consider (in some way) symmetric PCSPs.

1-in-3-SAT and Not-All-Equal-3-SAT are classic examples of Boolean symmetric (non-promise) CSPs. While both problems are NP-hard, Brakensiek and Guruswami showed [SODA'18] that given a satisfiable instance of 1-in-3-SAT one can find a solution to the corresponding instance of (weaker) Not-All-Equal-3-SAT. In other words, the PCSP template (**1-in-3**, **NAE**) is tractable.

We focus on *non-symmetric* PCSPs. In particular, we study PCSP templates obtained from the Boolean template (**t-in-k**, **NAE**) by either adding tuples to **t-in-k** or removing tuples from **NAE**. For the former, we classify all templates as either tractable or not solvable by the currently strongest known algorithm for PCSPs, the combined basic LP and affine IP relaxation of Brakensiek and Guruswami [SODA'20]. For the latter, we classify all templates as either tractable or NP-hard.

2012 ACM Subject Classification Theory of computation → Problems, reductions and completeness; Theory of computation → Constraint and logic programming

Keywords and phrases promise constraint satisfaction, PCSP, polymorphisms, algebraic approach

Digital Object Identifier 10.4230/LIPIcs.ICALP.2021.121

Category Track B: Automata, Logic, Semantics, and Theory of Programming

Related Version *Full Version*: `https://arxiv.org/abs/2104.12800` [14]

Funding This project has received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreement No 714532).
Alex Brandts: Royal Society Enhancement Award and NSERC PGS Doctoral Award.
Stanislav Živný: Royal Society University Research Fellowship.

1 Introduction

How hard is it to find a 6-colouring of a graph if it is promised to be 3-colourable? We do not know but believe it to be NP-hard. Despite sustained effort, this so-called *approximate graph colouring* problem has been elusive since it was considered by Garey and Johnson almost 50 years ago [22]. The current state of the art, established in 2019, is NP-hardness of finding a 5-colouring of a 3-colourable graph [17]. Approximate graph colouring is an example of the very general promise constraint satisfaction problem, which is the focus of this paper. We start with (non-promise) constraint satisfaction problems to set the stage.

Constraint satisfaction. While deciding whether a graph is 2-colourable is solvable in polynomial time, deciding 3-colourability is NP-complete [26]. The *constraint satisfaction problem* (CSP) is a general framework that captures graph colourings and many other

fundamental computational problems. Feder and Vardi initiated a systematic study of so-called fixed-template decision CSPs. Let $\mathbf{A}$ be a fixed finite relational structure, called the *template* or constraint language; i.e., $\mathbf{A}$ consists of a finite universe A and finitely many relations on A, each of possibly different arity. The fixed-template CSP over $\mathbf{A}$, denoted by $\mathrm{CSP}(\mathbf{A})$, is the class of CSPs in which all constraint relations come from $\mathbf{A}$. In more detail, $\mathrm{CSP}(\mathbf{A})$ denotes the following computational problem: Given a structure $\mathbf{X}$ over the same signature as $\mathbf{A}$, is there a homomorphism from $\mathbf{X}$ to $\mathbf{A}$, denoted by $\mathbf{X} \to \mathbf{A}$? (Formal definitions can be found in Section 2.) If $\mathbf{A} = K_3$ is a clique on 3 vertices then $\mathrm{CSP}(\mathbf{A})$ is precisely the standard graph 3-colouring problem.

A classic result of Schaefer shows that, for any $\mathbf{A}$ on a 2-element set, $\mathrm{CSP}(\mathbf{A})$ is either solvable in polynomial time or NP-complete. The non-trivial tractable cases from Schaefer's classification are taught in undergraduate algorithms courses: 2-SAT, (dual) Horn-SAT, and linear equations over $\{0,1\}$. Two concrete CSPs that are NP-hard by Schaefer's result are the (positive) 1-in-3-SAT and (positive) Not-All-Equal-3-SAT. For both problems, the instance is a list of triples of variables. In 1-in-3-SAT, the task is to find a mapping from the variables to $\{0,1\}$ so that in each triple exactly one variable is set to 1. Formally, 1-in-3-SAT is $\mathrm{CSP}(\mathbf{A})$, where $\mathbf{A} = (\{0,1\}; \{(1,0,0),(0,1,0),(0,0,1)\})$. In Not-All-Equal-3-SAT, the task is to find a mapping from the variables to $\{0,1\}$ so that in each triple not all variables are assigned the same value. Formally, Not-All-Equal-3-SAT is $\mathrm{CSP}(\mathbf{A})$, where $\mathbf{A} = (\{0,1\}; \{0,1\}^3 \setminus \{(0,0,0),(1,1,1)\})$.

If $\mathbf{A}$ is a graph (i.e., a single symmetric binary relation) then, as shown by Hell and Nešetřil [24], $\mathrm{CSP}(\mathbf{A})$ is either solvable in polynomial time or NP-complete. In this case, essentially the only non-trivial tractable case is graph 2-colouring.

Based on these two examples and a connection to logic, Feder and Vardi famously conjectured [20] that, for any finite $\mathbf{A}$, $\mathrm{CSP}(\mathbf{A})$ is either solvable in polynomial time or NP-complete. Bulatov [16], and independently Zhuk [31], proved the conjecture in the affirmative, both relying on the algebraic approach to CSPs [25, 15, 6]. In this case, the tractable cases are complicated and hard to describe in an elementary way.

Promise constraint satisfaction. Austrin, Guruswami, and Håstad [2] and Brakensiek and Guruswami [8, 9] initiated the investigation of the *promise constraint satisfaction problem* (PCSP), which is a vast generalisation of the CSP. Let $\mathbf{A}$ and $\mathbf{B}$ be two relational structures such that $\mathbf{A} \to \mathbf{B}$. The fixed-template PCSP over $\mathbf{A}$ and $\mathbf{B}$, denoted by $\mathrm{PCSP}(\mathbf{A},\mathbf{B})$, is the following computational problem: Given $\mathbf{X}$ such that $\mathbf{X} \to \mathbf{A}$, find a homomorphism from $\mathbf{X}$ to $\mathbf{B}$ (which exists by the composition of the promised homomorphism from $\mathbf{X}$ to $\mathbf{A}$ and the homomorphism from $\mathbf{A}$ to $\mathbf{B}$). If we take $\mathbf{A} = K_3$ to be a clique on 3 vertices and $\mathbf{B} = K_6$ to be a clique on 6 vertices, then $\mathrm{PCSP}(\mathbf{A},\mathbf{B})$ is an instance of the approximate graph colouring problem mentioned at the beginning of this article. Actually, what we described is the *search* version of the PCSP. The *decision* version is as follows: Given $\mathbf{X}$, return YES if $\mathbf{X} \to \mathbf{A}$ and return NO if $\mathbf{X} \not\to \mathbf{B}$. (The promise in the decision version is that it does not happen that $\mathbf{X} \not\to \mathbf{A}$ but $\mathbf{X} \to \mathbf{B}$.) It is well known that the decision version reduces to the search version but it is not known whether there is a reduction the other way [5]. In most results (including ours), hardness is established for the decision version and tractability for the search version.

If $\mathbf{A} = \mathbf{B}$ then $\mathrm{PCSP}(\mathbf{A},\mathbf{B})$ is the same as $\mathrm{CSP}(\mathbf{A})$ and thus PCSPs indeed generalise CSPs. For CSPs, the decision and search versions are known to be equivalent [15].

Building on the result of Barto, Opršal, and Pinsker [7] that the complexity of $\mathrm{CSP}(\mathbf{A})$ is captured by certain types of identities of higher-order symmetries (called polymorphisms) of $\mathbf{A}$, Barto, Bulín, Krokhin, and Opršal showed that the basics of the algebraic approach developed for CSPs [7] can be generalised to PCSPs [17, 5], thus introducing a general methodology for investigating the computational complexity of PCSPs.

Related work. Motivated by the goal to understand the computational complexity of all fixed-template PCSPs, a recent line of research has focused on restricted classes of templates, with the main directions being Boolean templates (i.e., templates on a two-element set) and symmetric templates (i.e., all relations in the template satisfy that if a tuple belongs to a relation then so do all of its permutations).

Austrin, Guruswami, and Håstad [2] considered the $(1, g, k)$-SAT problem: Given an instance of k-SAT with the promise that there is an assignment satisfying at least g literals in each clause, find an assignment that satisfies at least one literal in each clause. They showed that this problem is NP-hard if $\frac{g}{k} < \frac{1}{2}$, and polynomial-time solvable otherwise. $(1, g, k)$-SAT is a Boolean PCSP with a (symmetric) template that includes the binary disequality relation and a relation containing all tuples of particular Hamming weights. The NP-hardness in [2] was proved via reduction from the label cover problem using the idea of polymorphisms lifted from CSPs to PCSPs. Building on the algebraic theory from [17, 5], Brandts, Wrochna, and Živný [13] extended the classification of $(1, g, k)$-SAT to arbitrary finite domains.

Brakensiek and Guruswami [9] managed to classify all PCSPs over symmetric Boolean templates with the disequality relation as NP-hard or solvable in polynomial time. Ficak, Kozik, Olšák, and Stankiewicz [21] extended this result to all symmetric Boolean templates.

In very recent work, Barto, Battistelli, and Berg [4] explored symmetric PCSPs on three- and four-element domains.

While the approximate graph colouring problem remains open, hardness was proved under stronger assumptions (namely Khot's 2-to-1 Conjecture [27] for k-colourings with $k \geq 4$ and its non-standard variant for 3-colourings) by Dinur, Mossel, and Regev [18]. Guruswami and Sandeep [23] recently established this result under a weaker assumption, the so-called d-to-1 conjecture for any fixed $d \geq 2$. For approximate *hypergraph* colouring, another important PCSP, NP-hardness was established by Dinur, Regev, and Smyth [19]. There has been some recent progress on approximate graph colourings [30] and related PCSPs, e.g. approximate graph homomorphism problems [28, 30], and rainbow vs. normal hypergraph colourings [1].

Unlike most previous works, which focused on symmetric PCSPs, we investigate non-symmetric PCSPs. Our first motivation is that a classification of more concrete PCSP templates is needed to improve and extend the general algebraic theory from [17, 5]. At the moment, even an analogue of Schaefer's result, i.e., classifying all Boolean PCSPs, seems out of reach. Our second motivation is the pure beauty of the template $(\mathbf{1\text{-}in\text{-}3}, \mathbf{NAE})$. While PCSP$(\mathbf{1\text{-}in\text{-}3}, \mathbf{NAE})$ admits a polynomial-time algorithm [9, 10], Barto showed [3, 5] that this tractability result cannot be obtained via an algebraic reduction to tractable finite-domain CSPs.

Contributions. Consider the Boolean PCSP$(\mathbf{t\text{-}in\text{-}k}, \mathbf{NAE})$, which is a natural generalisation of PCSP$(\mathbf{1\text{-}in\text{-}3}, \mathbf{NAE})$. This is a symmetric, tractable PCSP. When can we add tuples to **t-in-k** to keep the PCSP tractable? When can we remove tuples from **NAE** to keep the PCSP tractable? Note that these changes generally do not give symmetric templates.

For the second question, we give a complete answer in Theorem 11: If t is odd, k is even, and tuples of only even Hamming weight are removed from **NAE**, the resulting PCSP is solvable in polynomial time. In all other cases, the resulting PCSP is NP-hard.

For the first question, we give a second-best possible answer in Theorem 10: If t is odd, k is even, and tuples of only odd Hamming weight are added to **t-in-k**, the resulting PCSP is tractable. In all other cases, the resulting PCSP is not solved by the combined basic LP and affine IP relaxation of Brakensiek and Guruswami [11], the currently strongest known algorithm for PCSPs. The power of this relaxation has recently been characterised by Brakensiek, Guruswami, Wrochna, and Živný [12].

2 Preliminaries

We denote by $[n]$ the set $\{1, 2, \ldots, n\}$. For a k-tuple $\mathbf{x}$, we write $\mathbf{x} = (x_1, \ldots, x_k)$. We denote by $\leq_p$ a polynomial-time many-one reduction.

A *relational structure* is a tuple $\mathbf{A} = (A; R_1, \ldots, R_p)$, where A is a finite set called the *domain* of $\mathbf{A}$, and each R_i is a relation of arity $\mathrm{ar}(R_i) \geq 1$, that is, R_i is a non-empty subset of $A^{\mathrm{ar}(R_i)}$. A relational structure is *symmetric* if each relation in it is invariant under any permutation of coordinates. Two relational structures $\mathbf{A} = (A; R_1, \ldots, R_p)$ and $\mathbf{B} = (B; S_1, \ldots, S_q)$ have the same *signature* if $p = q$ and $\mathrm{ar}(R_i) = \mathrm{ar}(S_i)$ for every $i \in [p]$. In this case, a mapping $\phi : A \to B$ is called a *homomorphism* from $\mathbf{A}$ to $\mathbf{B}$, denoted by $\phi : \mathbf{A} \to \mathbf{B}$, if ϕ preserves all relations; that is, for every $i \in [p]$ and every tuple $\mathbf{x} \in R_i$, we have $\phi(\mathbf{x}) \in S_i$, where ϕ is applied component-wise. The existence of a homomorphism from $\mathbf{A}$ to $\mathbf{B}$ is denoted by $\mathbf{A} \to \mathbf{B}$. A PCSP *template* is a pair $(\mathbf{A}, \mathbf{B})$ of relational structures over the same signature such that $\mathbf{A} \to \mathbf{B}$.

▶ **Definition 1.** *Let* $(\mathbf{A}, \mathbf{B})$ *be a PCSP template. The* decision version *of* $\mathrm{PCSP}(\mathbf{A}, \mathbf{B})$ *is the following problem: Given as input a relational structure* $\mathbf{X}$ *over the same signature as* $\mathbf{A}$ *and* $\mathbf{B}$*, output* YES *if* $\mathbf{X} \to \mathbf{A}$ *and* NO *if* $\mathbf{X} \not\to \mathbf{B}$*. The* search version *of* $\mathrm{PCSP}(\mathbf{A}, \mathbf{B})$ *is the following problem: Given as input a relational structure* $\mathbf{X}$ *over the same signature as* $\mathbf{A}$ *and* $\mathbf{B}$ *such that* $\mathbf{X} \to \mathbf{A}$*, find a homomorphism from* $\mathbf{X}$ *to* $\mathbf{B}$*.*

We call $\mathrm{PCSP}(\mathbf{A}, \mathbf{B})$ *tractable* if any instance of $\mathrm{PCSP}(\mathbf{A}, \mathbf{B})$ can be solved in polynomial time in the size of the input structure $\mathbf{X}$. It is easy to show that the decision version reduces to the search version [5]. Our hardness results will be for the decision version and our tractability results for the search version. For a relational structure $\mathbf{A}$, the constraint satisfaction problem with the template $\mathbf{A}$, denoted by $\mathrm{CSP}(\mathbf{A})$, is $\mathrm{PCSP}(\mathbf{A}, \mathbf{A})$.

The following notion of polymorphisms is at the heart of the algebraic approach to (P)CSPs. Intuitively, an arity m polymorphism of a PCSP template $(\mathbf{A}, \mathbf{B})$ is a homomorphism from the m-th Cartesian power of $\mathbf{A}$ to $\mathbf{B}$.

▶ **Definition 2.** *Let* $(\mathbf{A}, \mathbf{B})$ *be a PCSP template. A function* $f : A^m \to B$ *is a* polymorphism *of arity* m *of* $(\mathbf{A}, \mathbf{B})$ *if for each pair of corresponding relations* R_i *and* S_i *from* $\mathbf{A}$ *and* $\mathbf{B}$*, respectively, the following holds: For any* $(k \times m)$ *matrix* M *whose columns are tuples in* R_i*, the application of* f *to rows of* M *gives a tuple in* S_i*. We denote by* $\mathrm{Pol}(\mathbf{A}, \mathbf{B})$ *the set of all polymorphisms of* $(\mathbf{A}, \mathbf{B})$*.*

In a PCSP template $(\mathbf{A}, \mathbf{B})$ we view tuples from $\mathbf{A}$ and $\mathbf{B}$ as columns. When writing tuples in text we may write them as rows to simplify notation but they should still be understood as columns. For a k-ary relation R on the set $[t]$, we denote by $R^c = [t]^k \setminus R$ the complement of R. For a relational structure $\mathbf{A}$, we denote by $\mathbf{A}^c$ the structure with relations R^c for each relation R in $\mathbf{A}$. Most of our relational structures will be on the Boolean domain $\{0, 1\}$ and contain a single relation of arity k. The (Hamming) weight of a tuple $\mathbf{x} \in \{0, 1\}^k$, denoted throughout by d, is the number of 1's in $\mathbf{x}$. For $1 \leq t < k$, the Boolean relational structure **t-in-k** consists (of one relation consisting) of all k-tuples with weight t. The Boolean relational structure **NAE** contains all k-tuples except 0^k and 1^k.

We need a definition and some notation to state existing results on Boolean (P)CSPs.

▶ **Definition 3.** *A function* $f : \{0, 1\}^m \to \{0, 1\}$ *is*

- *an* OR_m *(*AND_m*) if it returns the logical OR (respectively logical AND) of its arguments;*
- *an alternating threshold* AT_m *if* $m \geq 1$ *is odd and*

$$f(x_1, \ldots, x_m) = \mathbb{1}[x_1 - x_2 + x_3 - \cdots + x_m] > 0;$$

- *a parity function* XOR_m *if* $f(x_1,\dots,x_m) = x_1 + \cdots + x_m \mod 2$*;*
- *a q-threshold* $\mathrm{THR}_{q,m}$ *(for* q *a rational between 0 and 1 and* mq *not an integer) if* $f(x_1,\dots,x_m) = 0$ *if* $\sum_{i=1}^{m} x_i < mq$ *and* 1 *otherwise;*
- *a majority* MAJ_m *if* f *is a* $\frac{1}{2}$*-threshold.*

We denote by OR *and* AND *the set of all* OR_m *and* AND_m *functions, respectively, for* $m \geq 2$*. We denote by* AT *and* XOR *the set of all* AT_m *and* XOR_m *functions, respectively, for odd* $m \geq 1$*. Finally,* THR_q *denotes the set of all* $\mathrm{THR}_{q,m}$ *functions for* $qm \notin \mathbb{Z}$*.*

Define $\overline{f}$*, the* inversion of f*, as the function* $x \mapsto 1 - f(x)$*, and for a family of functions* F*, define the* inversion of F *by* $\overline{F} = \{\overline{f} | f \in F\}$*.*

Schaefer's celebrated dichotomy theorem classified all Boolean CSP templates.

▶ **Theorem 4** ([29])**.** *Let* $\mathbf{B}$ *be a Boolean CSP template. If* $\mathrm{Pol}(\mathbf{B})$ *contains a constant,* AND_2*,* OR_2*,* MAJ_3*, or* XOR_3*, then* $\mathrm{CSP}(\mathbf{B})$ *is tractable. Otherwise,* $\mathrm{CSP}(\mathbf{B})$ *is NP-hard.*

Ficak et al. classified all symmetric Boolean PCSP templates [21].

▶ **Theorem 5** ([21])**.** *Let* $(\mathbf{A},\mathbf{B})$ *be a symmetric Boolean PCSP template. If* $\mathrm{Pol}(\mathbf{A},\mathbf{B})$ *contains a constant or at least one of* OR*,* AND*,* XOR*,* AT*,* THR_q *(for some* q*) or their inversions, then* $\mathrm{PCSP}(\mathbf{A},\mathbf{B})$ *is tractable. Otherwise,* $\mathrm{PCSP}(\mathbf{A},\mathbf{B})$ *is NP-hard.*

The only possibly unresolved promise templates are those with NP-hard CSP templates.

▶ **Proposition 6.** *Let* $(\mathbf{A},\mathbf{B})$ *be a promise template such that at least one of* $\mathrm{CSP}(\mathbf{A})$*,* $\mathrm{CSP}(\mathbf{B})$ *is tractable. Then* $\mathrm{PCSP}(\mathbf{A},\mathbf{B})$ *is tractable.*

Theorem 4 established NP-hardness of two natural CSPs: $\mathrm{CSP}(\mathbf{1\text{-}in\text{-}3})$ and $\mathrm{CSP}(\mathbf{NAE})$. Interestingly, $\mathrm{PCSP}(\mathbf{1\text{-}in\text{-}3},\mathbf{NAE})$ is solvable in polynomial-time, as first shown by Brakensiek and Guruswami [9]. (Note that this shows that the converse of Proposition 6 is false.) A natural generalisation of **1-in-3** is **t-in-k**. Theorem 4 implies that $\mathrm{CSP}(\mathbf{t\text{-}in\text{-}k})$ is NP-hard. Theorem 5 implies that the tractability of $\mathrm{PCSP}(\mathbf{1\text{-}in\text{-}3},\mathbf{NAE})$ also holds for $\mathrm{PCSP}(\mathbf{t\text{-}in\text{-}k},\mathbf{NAE})$; both observations are proved in the full version [14].

▶ **Proposition 7.** *For* $k \geq 3$ *and* $1 \leq t < k$*,* $\mathrm{CSP}(\mathbf{t\text{-}}\boldsymbol{in}\mathbf{\text{-}k})$ *is NP-hard.*

▶ **Proposition 8.** *For* $k \geq 2$ *and* $1 \leq t < k$*,* $\mathrm{PCSP}(\mathbf{t\text{-}}\boldsymbol{in}\mathbf{\text{-}k},\mathbf{NAE})$ *is tractable.*

We now give a characterisation of the power of a certain convex relaxation useful for tractability of (the decision version of) PCSPs. The relaxation is called the *combined basic LP and affine IP relaxation* (BLP+AIP) and was introduced in [11]; see also [10]. All known tractable PCSPs are solvable by either BLP+AIP or a reduction to a tractable CSP.

A $(2m+1)$-ary function f is called 2-block-symmetric if its $2m+1$ coordinates of f can be partitioned into two blocks of size $m+1$ and m in such a way that the value of f is invariant under any permutation of coordinates within each block. Without loss of generality, we will assume that the two blocks are the odd and even coordinates of f.

▶ **Theorem 9** ([12, Theorem 5.1])**.** *Let* $(\mathbf{A},\mathbf{B})$ *be a PCSP template. Then (the decision version of)* $\mathrm{PCSP}(\mathbf{A},\mathbf{B})$ *is tractable via BLP+AIP if and only if* $\mathrm{Pol}(\mathbf{A},\mathbf{B})$ *contains 2-block-symmetric functions of all odd arities.*

In the proof of one of our results (Theorem 10) we will use Theorem 9 to rule out solvability by BLP+AIP. However, for our tractability results, we will only need a characterisation result (in terms of polymorphisms) of a weaker relaxation, namely the affine IP relaxation (AIP) [5], which also works for the search version of PCSPs; details can be found in [14].

3 Our results

Our results are concerned with templates that arise from $(\mathbf{t\text{-}in\text{-}k}, \mathbf{NAE})$ by either adding tuples to $\mathbf{t\text{-}in\text{-}k}$ or removing tuples from $\mathbf{NAE}$. For a set of tuples $S \subseteq \{0,1\}^k$, we write $\mathbf{t\text{-}in\text{-}k} \cup \mathbf{S}$ for the relational structure whose (only) relation contains all k-tuples of weight t and the tuples from S, and similarly for $\mathbf{NAE} \setminus \mathbf{S}$.

▶ **Theorem 10** (Main #1). *Let $k \geq 3$ and $\emptyset \neq S \subseteq (\mathbf{t\text{-}in\text{-}k})^c \cap \mathbf{NAE}$. If t is odd, k is even, and S contains tuples of only odd weight, then* $\mathrm{PCSP}(\mathbf{t\text{-}in\text{-}k} \cup \mathbf{S}, \mathbf{NAE})$ *is tractable. Otherwise,* $\mathrm{PCSP}(\mathbf{t\text{-}in\text{-}k} \cup \mathbf{S}, \mathbf{NAE})$ *is not solved by BLP+AIP.*

The tractability part of Theorem 10 follows easily from existing work, cf. [14]. Our contribution is ruling out the applicability of BLP+AIP from [11]. Using Theorem 9, we prove the second part of Theorem 10 for $t = 1$ in Section 4 and for general t in [14].

▶ **Theorem 11** (Main #2). *Let $k \geq 3$ and $\emptyset \neq S \subseteq (\mathbf{t\text{-}in\text{-}k})^c \cap \mathbf{NAE}$. If t is odd, k is even, and S contains tuples of only even weight, then* $\mathrm{PCSP}(\mathbf{t\text{-}in\text{-}k}, \mathbf{NAE} \setminus \mathbf{S})$ *is tractable. Otherwise,* $\mathrm{PCSP}(\mathbf{t\text{-}in\text{-}k}, \mathbf{NAE} \setminus \mathbf{S})$ *is NP-hard.*

Again, the tractability part of Theorem 11 follows easily from existing work. Our contribution is establishing the hardness. It essentially follows from the following result.

▶ **Theorem 12.** *Let $k \geq 3$ and let $\mathbf{T} \subseteq \{0,1\}^k$ be a relation such that* $\mathrm{CSP}(\mathbf{T})$ *is NP-hard. Then* $\mathrm{PCSP}(\mathbf{t\text{-}in\text{-}k}, \mathbf{T})$ *is tractable if and only if $\mathbf{T} = \mathbf{NAE}$.*

Theorem 12 is proved in Section 5 and relies on Theorem 5, as well as a symmetrisation trick (Proposition 14, observed independently in [4]) and the following simple observation.

▶ **Proposition 13.** *Let $t, t' \geq 1$ and let R be a symmetric relation on $[t]$. For any function $f : [t] \to [t']$, the component-wise image of R under f, denoted $f(R)$, is also symmetric.*

Proof. Suppose that $\mathbf{y} \in f(R)$, so $\mathbf{y} = f(\mathbf{x})$ for some $\mathbf{x} \in R$. We must show that $\pi(\mathbf{y}) \in f(R)$ for an arbitrary permutation π. But since R is symmetric, we have $\pi(\mathbf{x}) \in R$, and so $f(\pi(\mathbf{x})) = \pi(\mathbf{y})$ since f is applied component-wise. ◀

▶ **Proposition 14.** *Let $(\mathbf{A}, \mathbf{B})$ be a PCSP template with $\mathbf{A}$ symmetric. For each relation $R \in \mathbf{B}$, let R' be the largest symmetric relation contained in R. Let $\mathbf{B}'$ be the relational structure with the same domain as $\mathbf{B}$ but with relations R' instead of R. Then* $\mathrm{PCSP}(\mathbf{A}, \mathbf{B})$ *is polynomial-time equivalent to* $\mathrm{PCSP}(\mathbf{A}, \mathbf{B}')$.

Proof. We must first check that $(\mathbf{A}, \mathbf{B}')$ is a valid PCSP template, i.e., that there is a homomorphism $\mathbf{A} \to \mathbf{B}'$. Let ϕ be a homomorphism from $\mathbf{A}$ to $\mathbf{B}$. By Proposition 13, $\phi(\mathbf{A})$ is symmetric, and since $\mathbf{B}'$ is the largest symmetric relational structure contained in $\mathbf{B}$, we have $\phi(\mathbf{A}) \subseteq \mathbf{B}'$. Therefore $(\mathbf{A}, \mathbf{B}')$ is a valid PCSP template.

The reduction $\mathrm{PCSP}(\mathbf{A}, \mathbf{B}) \leq_p \mathrm{PCSP}(\mathbf{A}, \mathbf{B}')$ is trivial since $\mathbf{B}' \subseteq \mathbf{B}$.

To see that $\mathrm{PCSP}(\mathbf{A}, \mathbf{B}') \leq_p \mathrm{PCSP}(\mathbf{A}, \mathbf{B})$, suppose that a relation $R \in \mathbf{B}$ gives rise to the symmetric relation $R' \in \mathbf{B}'$ as described above. For each constraint $C = R'(x_1, ..., x_k)$ of the symmetrised instance of $\mathrm{PCSP}(\mathbf{A}, \mathbf{B}')$, we create $k!$ constraints by taking all coordinate permutations of C. Homomorphisms to $\mathbf{A}$ are preserved since $\mathbf{A}$ is symmetric. Conversely, suppose that we have a homomorphism ψ from the created instance to $\mathbf{B}$ and suppose that the tuple $\mathbf{x}$ was removed from R to create R'. Then no constraint is assigned $\mathbf{x}$ under ψ, since by our construction this would force all permutations of $\mathbf{x}$ to appear in constraints, violating membership in the relation R. Therefore, for any removed tuple $\mathbf{x}$, applying ψ does not produce $\mathbf{x}$ in any constraint, so ψ is a homomorphism from the original input to $\mathbf{B}'$. To complete the reduction, we repeat this construction for each $R \in \mathbf{B}$. ◀

4 Adding tuples for $t = 1$

We will rule out 2-block-symmetric polymorphisms of certain odd arities for PCSP templates of the form $(\mathbf{1\text{-}in\text{-}k} \cup \{\mathbf{x}\}, \mathbf{NAE})$ such that the weight of $\mathbf{x}$ is even if k is even. This implies the second part of Theorem 10 with $t = 1$. The more general case stated in Theorem 10, for any $t \geq 1$, can be found in [14].

Any 2-block-symmetric function $f : \{0,1\}^{2m+1} \to \{0,1\}$ of odd arity $2m+1$ is determined by the values on the $(m+2)(m+1)$ possible combinations of weights of the two blocks. Thus we will represent f by $f(x,y)$, where x and y are the number of 1's on the odd and even coordinates, respectively.

▶ **Proposition 15.** *Let $k \geq 3$ and $2 \leq d \leq \frac{k+1}{2}$ be such that if k is even then so is d. Let $\mathbf{x}$ be a tuple of weight d. Then,* $\mathrm{Pol}(\mathbf{1\text{-}in\text{-}k} \cup \{\mathbf{x}\}, \mathbf{NAE})$ *has no 2-block-symmetric function of arity $2(k-d+1)+1$.*

Proof. Since **NAE** is symmetric, permuting the rows of a matrix of inputs to a polymorphism permutes the values of the output tuple and does not affect membership in **NAE**. Hence we will assume that $\mathbf{x} = 1^d 0^{k-d}$. Let f be a 2-block-symmetric function of arity $2(k-d+1)+1$. Thus the odd block is of size $k-d+2$ and the even block is of size $k-d+1$. We exhibit a set of tableaux such that for any f, one of these tableaux prevents f from being a polymorphism of $(\mathbf{1\text{-}in\text{-}k} \cup \{\mathbf{x}\}, \mathbf{NAE})$. Each tableau in this set contains the same construction on its even coordinates: the tuple $\mathbf{x}$ as its first column, followed by the $(k-d) \times (k-d)$ identity matrix below and to the right of $\mathbf{x}$, so that every row has exactly one 1. An illustration is given in Figure (1a).

$$\begin{pmatrix} 1&0&0&0&0&0\\ 1&0&0&0&0&0\\ 1&0&0&0&0&0\\ 1&0&0&0&0&0\\ 0&1&0&0&0&0\\ 0&0&1&0&0&0\\ 0&0&0&1&0&0\\ 0&0&0&0&1&0\\ 0&0&0&0&0&1 \end{pmatrix} \qquad \begin{pmatrix} 1&0&0&0&0&0&0\\ 0&1&0&0&0&0&0\\ 0&0&1&0&0&0&0\\ 0&0&0&1&0&0&0\\ 0&0&0&0&1&0&0\\ 0&0&0&0&0&1&0\\ 0&0&0&0&0&0&1\\ 0&0&0&0&0&0&0\\ 0&0&0&0&0&0&0 \end{pmatrix}$$

(a) Even block. **(b)** Odd block if $f(0,1) = f(1,1)$.

Figure 1 Example with $k = 9$ and $d = 4$.

It remains to give only the description of each tableau's odd coordinates.

If $f(0,1) = f(1,1)$, then the block of odd coordinates consists of the $(k-d+2) \times (k-d+2)$ identity matrix on top of $d-2$ rows of zeros. Since $2 \leq d < k$, the dimensions of the identity matrix are between 3 and k. An illustration is given in Figure (1b).

If $f(1,1) = f(2,1)$, then we obtain the block of odd coordinates by adding any tuple of weight 1 to the block of even coordinates.

Otherwise we have $f(0,1) \neq f(1,1) \neq f(2,1)$, so $f(0,1) = f(2,1)$.

If the number of odd coordinates $k-d+2$ is even, then we place two copies of the $\frac{k-d+2}{2} \times \frac{k-d+2}{2}$ identity matrix in the first $\frac{k-d+2}{2}$ rows, followed by zeros in the remaining rows. There are always enough rows to accommodate these matrices since $\frac{k-d+2}{2} \leq k \Leftrightarrow k+d \geq 2$. An illustration is given in Figure (2a).

$$\begin{pmatrix} 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

(a) Odd block with $d = 5$ and $k - d + 2 = 6$ even.

$$\begin{pmatrix} 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

(b) Odd block with $d = 4$ and $k - d + 2 = 7$ odd.

Figure 2 Example with $k = 9$ and $f(0,1) = f(2,1)$.

If $k - d + 2$ is odd then so is $k - d$. By the assumption "even k implies even d", we must have k odd and d even. The construction consists of the tuple $\mathbf{x}$ with the $d \times d$ identity matrix to its right. To fill the remaining $k - 2d + 1$ columns (assuming $d \leq \frac{k+1}{2}$), we place two copies of the $\frac{k-2d+1}{2} \times \frac{k-2d+1}{2}$ identity matrix starting from row $d+1$ and column $d+2$, and fill any remaining rows with zeros. There are always enough rows to accommodate the identity matrices since $\frac{k-2d+1}{2} \leq k - d$. An illustration is given in Figure (2b). ◀

▶ **Proposition 16.** *Let $k \geq 3$ and $\frac{k+1}{2} < d < k$ be such that if k is even then so is d. Let $\mathbf{x}$ be a tuple of weight d. Then,* $\mathrm{Pol}(\mathbf{1\text{-}in\text{-}k} \cup \{\mathbf{x}\}, \mathbf{NAE})$ *has no 2-block-symmetric function of arity $2k + 1$.*

Proof. Without loss of generality let $\mathbf{x} = 1^d 0^{k-d}$. The proof is similar to the proof of Proposition 15: We will again exhibit a set of tableaux such that for any f, one of these tableaux prevents f from being a polymorphism of $(\mathbf{1\text{-}in\text{-}k} \cup \{\mathbf{x}\}, \mathbf{NAE})$. Each tableau in this set has the $k \times k$ identity matrix as its even coordinates, so it remains only to give a description of each tableau's odd coordinates. Consider the values $f(1,1)$, $f(2,1)$, and $f(3,1)$: At least two of these must be equal since f takes only the values 0 and 1.

If $f(1,1) = f(2,1)$, then taking the odd coordinates to be the $k \times k$ identity matrix along with any other tuple of weight 1 prevents f from being a polymorphism.

If $f(1,1) = f(3,1)$, the odd coordinates are as follows: the first column is the tuple $\mathbf{x}$, followed immediately below and to the right by the $(k-d) \times (k-d)$ identity matrix, and then by two copies of the $\frac{d}{2} \times \frac{d}{2}$ identity matrix in the upper-right corner. The tuple $\mathbf{x}$ and the $(k-d) \times (k-d)$ matrix occupy $k - d + 1$ columns, leaving d columns to be filled by the two $\frac{d}{2} \times \frac{d}{2}$ identity matrices, and since d is even, $\frac{d}{2}$ is always an integer. An illustration is given in Figure (3a).

Finally, if $f(2,1) = f(3,1)$, the odd coordinates are as follows: two columns of $\mathbf{x}$, followed immediately below and to the right by two copies of the $(k-d) \times (k-d)$ identity matrix, and then by the $(2d-k-1) \times (2d-k-1)$ identity matrix in the upper-right corner. This fills all the columns. To place the two $(k-d) \times (k-d)$ identity matrices requires $2(k-d)$ columns after the $\mathbf{x}$'s. This is always possible since $2(k-d) \leq k-1 \Leftrightarrow d \geq \frac{k+1}{2}$. After placing these, there remain $2d - k - 1$ columns for the final identity matrix, and since $(2d-k-1) \leq k \Leftrightarrow d \leq k + \frac{1}{2}$, the number of available rows is never exceeded. An illustration is given in Figure (3b). ◀

$$\begin{pmatrix} 1&0&0&0&1&0&0&1&0&0\\ 1&0&0&0&0&1&0&0&1&0\\ 1&0&0&0&0&0&1&0&0&1\\ 1&0&0&0&0&0&0&0&0&0\\ 1&0&0&0&0&0&0&0&0&0\\ 1&0&0&0&0&0&0&0&0&0\\ 0&1&0&0&0&0&0&0&0&0\\ 0&0&1&0&0&0&0&0&0&0\\ 0&0&0&1&0&0&0&0&0&0 \end{pmatrix}$$

(a) Odd block with $f(1,1) = f(3,1)$.

$$\begin{pmatrix} 1&1&0&0&0&0&0&0&1&0\\ 1&1&0&0&0&0&0&0&0&1\\ 1&1&0&0&0&0&0&0&0&0\\ 1&1&0&0&0&0&0&0&0&0\\ 1&1&0&0&0&0&0&0&0&0\\ 1&1&0&0&0&0&0&0&0&0\\ 0&0&1&0&0&1&0&0&0&0\\ 0&0&0&1&0&0&1&0&0&0\\ 0&0&0&0&1&0&0&1&0&0 \end{pmatrix}$$

(b) Odd block with $f(2,1) = f(3,1)$.

Figure 3 Example with $k = 9$ and $d = 6$.

5 Removing tuples

In this section, we will prove Theorem 12 and show how it implies Theorem 11.

▶ **Theorem** (Theorem 12 restated). *Let $k \geq 3$ and let $\mathbf{T} \subseteq \{0,1\}^k$ be a relation such that* CSP($\mathbf{T}$) *is NP-hard. Then* PCSP(**t-*in*-k**, $\mathbf{T}$) *is tractable if and only if* $\mathbf{T} = \mathbf{NAE}$.

Proof. By Proposition 14, we can assume that $\mathbf{T}$ is symmetric. If $\mathbf{T} = \mathbf{NAE}$ then PCSP(**t-in-k**, $\mathbf{T}$) is tractable by Proposition 8. Otherwise, we show that Pol(**t-in-k**, $\mathbf{T}$) does not contain any of the tractable polymorphism families identified in the symmetric Boolean PCSP dichotomy (Theorem 5), and therefore PCSP(**t-in-k**, $\mathbf{T}$) is NP-hard.

The families are constants, OR, AND, XOR, AT, and THR_q for $q \in \mathbb{Q}$, as well as their inversions. We deal first with the non-inverted families. Since $0^k \notin \mathbf{T}$ and $1^k \notin \mathbf{T}$, Pol(**t-in-k**, $\mathbf{T}$) does not contain constants. Let C_k^t be the $k \times k$ matrix containing the k cyclic shifts of the column 1^t0^{k-t}. Then C_k^t prevents the polymorphism families OR, AND, XOR (if k is odd), and THR_q for all $q \neq \frac{t}{k}$. For all k, it remains to show that Pol(**t-in-k**, $\mathbf{T}$) contains neither $\mathrm{THR}_{\frac{t}{k}}$ nor AT. For even k it remains to show that Pol(**t-in-k**, $\mathbf{T}$) excludes XOR when t is even, and likewise when t is odd and $\mathbf{T}$ is missing a tuple of odd weight.

$\mathrm{THR}_{\frac{t}{k}}$: Suppose that $\mathbf{T}$ does not contain the tuple $\mathbf{x} = 1^d0^{k-d}$ of weight d where $1 \leq d < k$. Note that we cannot have $d = t$ as this would violate **t-in-k** $\to \mathbf{T}$, so either $t < d$ or $t > d$.

If $t < d$, then the $k \times d$ matrix M formed by placing C_d^t on top of the $(k-d) \times d$ zero matrix returns the forbidden tuple $\mathbf{x}$ when $\mathrm{THR}_{\frac{t}{k},d}$ is applied. An illustration is given in Figure (4a).

We must check that $\mathrm{THR}_{\frac{t}{k},d}$ is indeed in $\mathrm{THR}_{\frac{t}{k}}$, which requires that $d \times \frac{t}{k} \notin \mathbb{Z}$. This is equivalent to $dt \not\equiv 0 \pmod{k}$. However this is not always true, for example when $t = 2$, $k = 6$ and $d = 3$. When $dt \equiv 0 \pmod{k}$, we take instead $\mathrm{THR}_{\frac{t}{k},2d+1}$ and the $k \times (2d+1)$ input matrix M' consisting of two side-by-side copies of M and any extra column from M. In the first d rows of M, t of the d entries are 1's, so $\mathrm{THR}_{\frac{t}{k},d}$ returns 1 since $\frac{t}{d} > \frac{t}{k}$. Alternatively, in the first d rows of M', at least $2t$ of the $2d+1$ entries are 1's, so $\mathrm{THR}_{\frac{t}{k},2d+1}$ returns 1 since $\frac{2t}{2d+1} > \frac{t}{k}$. Both $\mathrm{THR}_{\frac{t}{k},d}$ and $\mathrm{THR}_{\frac{t}{k},2d+1}$ return 0 when applied to any of the last $k-d$ rows of M and M', respectively, since these rows contain only 0's. This completes the case $t < d$.

If $t > d$, let M be the $d \times (k-d)$ 1's matrix on top of C_{k-d}^{t-d}. Then applying $\mathrm{THR}_{\frac{t}{k},k-d}$ $\mathrm{THR}_{\frac{t}{k}}$ to M returns the forbidden tuple $\mathbf{x}$. An illustration is given in Figure (4b).

$$\mathrm{THR}_{\frac{3}{7},5}\left(\begin{array}{ccccc}1&0&0&1&1\\1&1&0&0&1\\1&1&1&0&0\\0&1&1&1&0\\0&0&1&1&1\\\hline 0&0&0&0&0\\0&0&0&0&0\end{array}\right)=\begin{array}{c}1\\1\\1\\1\\1\\\hline 0\\0\end{array}$$

(a) $t = 3$ and $d = 5$.

$$\mathrm{THR}_{\frac{4}{7},5}\left(\begin{array}{ccccc}1&1&1&1&1\\1&1&1&1&1\\\hline 1&0&0&0&1\\1&1&0&0&0\\0&1&1&0&0\\0&0&1&1&0\\0&0&0&1&1\end{array}\right)=\begin{array}{c}1\\1\\\hline 0\\0\\0\\0\\0\end{array}$$

(b) $t = 4$ and $d = 2$.

Figure 4 Example with $k = 7$.

Again we have to check that $(k-d)\times\frac{t}{k}\notin\mathbb{Z}$, or equivalently, that $(k-d)t\not\equiv 0 \pmod k$. If $(k-d)t\equiv 0 \pmod k$, then $d\neq 1$, and we use instead $\mathrm{THR}_{\frac{t}{k},k-d+1}$ and the input matrix M' which is equal to M but with any one column repeated. Both $\mathrm{THR}_{\frac{t}{k},k-d}$ and $\mathrm{THR}_{\frac{t}{k},k-d+1}$ return 1 when applied to the first d rows of M and M', respectively. In the last $k-d$ rows of M, $t-d$ of the $k-d$ entries are 1's, so $\mathrm{THR}_{\frac{t}{k},k-d}$ returns 0 as $\frac{t-d}{k-d}<\frac{t}{k}$. In the last $k-d+1$ rows of M', at most $t-d+1$ of the $k-d+1$ entries are 1's, so $\mathrm{THR}_{\frac{t}{k},k-d+1}$ returns 0 since $\frac{t-d+1}{k-d+1}<\frac{t}{k}$ (note that $d>1$ in this case). This completes the proof for $t>d$, and thus we conclude that $\mathrm{THR}_{\frac{t}{k}}\not\subseteq \mathrm{Pol}(\textbf{t-in-k},\mathbf{T})$.

AT: Again we split into the cases $t < d$ and $t > d$.

Let $t < d$ and suppose that T does not contain the tuple $\mathbf{x} = 1^d0^{k-d}$ of weight d where $1\leq d<k$. We construct an input that returns $\mathbf{x}$ when a specific AT function is applied. This is done in two stages. First, we construct a matrix M such that for some AT function f, $f(M)$ agrees with $\mathbf{x}$ on all coordinates but one. Next we pad the input M with a matrix P such that for another AT function f', we have $f'(M|P|\cdots|P)=\mathbf{x}$.

To ease readability, we will separate the odd and even columns into two contiguous blocks, with the odd columns appearing first in the matrices. We take $f=\mathrm{AT}_{2d-1}$ and define M to be the following $k\times(2d-1)$ matrix. In the d odd columns, we fill the first d rows with the matrix C_d^t. We fill the remaining $k-d$ rows with 0's. In the $d-1$ even columns, we place on top a row of 1's, followed by $d-1$ rows of 0's, then $t-1$ rows of 1's, and finally $k-d-t+1$ rows of 0's. An illustration is given in Figure (5a).

$$\mathrm{AT}_9\left(\begin{array}{ccccc|cccc}1&0&0&1&1&1&1&1&1\\1&1&0&0&1&0&0&0&0\\1&1&1&0&0&0&0&0&0\\0&1&1&1&0&0&0&0&0\\0&0&1&1&1&0&0&0&0\\\hline 0&0&0&0&0&1&1&1&1\\0&0&0&0&0&1&1&1&1\\\hline 0&0&0&0&0&0&0&0&0\end{array}\right)=\begin{array}{c}0\\1\\1\\1\\1\\\hline 0\\0\\\hline 0\end{array}$$

(a) AT_9.

$$P=\left(\begin{array}{ccc|ccc}1&1&1&1&1&0\\1&1&1&1&0&1\\1&1&1&0&1&1\\\hline 0&0&0&0&0&0\\0&0&0&0&0&0\\0&0&0&0&0&0\\0&0&0&0&0&0\\\hline 0&0&0&1&1&1\end{array}\right)$$

(b) P.

Figure 5 $t = 3$, $k = 8$, and $d = 5$.

Notice that $\mathrm{AT}_{2d-1}(M)$ agrees with $\mathbf{x}$ everywhere except the first coordinate. This happens because the first row of M has too many 1's in the even columns; we would require at least $(d-1)-t+1 = d-t$ more 1's in the odd columns for f to output 1 on the first row. We can achieve this by padding M with a matrix P that increases the proportion of 1's in the odd coordinates of the first row but does not affect the output of AT functions on the other rows.

Take $f' = \mathrm{AT}_{2d-1+2t(d-t)}$ and define the $k \times 2t$ matrix P as follows. In the t odd columns of P, we place t rows of 1's followed by $k-t$ rows of 0's. In the even columns of P, we place C_t^{t-1} in the first t rows, followed by $k-t-1$ rows of 0's, and then by a final row of 1's. An illustration is given in Figure (5b).

By padding M with copies of P, we increase the proportion of 1's in odd columns in the first $t < d$ rows, and in particular in the first row. The 0's in rows $t+1, \ldots, k-1$ do not affect the output, and the 1's in the last row do not affect the output since they are in even columns and the last coordinate of x is always zero since $d < k$. Therefore $f'(M|P|\cdots|P) = \mathbf{x}$, where P appears $d-t$ times. This completes the case $t < d$.

Now let $t > d$ and again suppose that T does not contain the tuple $\mathbf{x} = 1^d 0^{k-d}$ of weight d. Define the $k \times (2d-1)$ input matrix M to AT_{2d-1} as follows. In the d odd columns of M, we first place d rows of 1's. Then in next $t-d+1$ rows, we place side-by-side as many copies of C_{t-d+1}^{t-d} as we can, removing columns of the last copy if necessary. We fill the remaining rows with 0's. In the $d-1$ even columns, we place a row of 0's, then t rows of 1's, and then fill the remaining rows with 0's. An illustration is given in Figure (6).

$$\mathrm{AT}_7 \left(\begin{array}{cccc|ccc} 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ \hline 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ \hline 1 & 1 & 0 & 1 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 0 & 1 & 1 & 1 \\ \hline 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{array}\right) = \begin{array}{c} 1 \\ 1 \\ 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \end{array}$$

Figure 6 $t = 6$, $k = 8$, and $d = 4$.

We have $\mathrm{AT}_{2d-1}(M) = x$, which concludes the case $t > d$. Therefore $\mathrm{AT} \not\subseteq \mathrm{Pol}(\textbf{t-in-k}, \mathbf{T})$.

XOR: Let t and k be even. Applying XOR_k to the matrix C_k^t returns the tuple 0^k, so applying XOR_{k-1} to the first $k-1$ columns of C_k^t returns the last column $1^{t-1}0^{k-t}1$. We can "fill in" the 0's in the output by swapping 0-1 pairs of values in the input matrix. In particular, in the columns $k-1, k-3, \ldots, t+1$, we swap the entries in the pairs of rows $(k-1, k-2), (k-3, k-4), \ldots, (t+1, t)$, respectively. The resulting $k \times (k-1)$ matrix M then satisfies $\mathrm{XOR}_{k-1}(M) = 1^k$ and the arity $k-1$ is odd as required. An example with swapped values in bold is illustrated in Figure (7a).

Now let t be odd, k be even, and suppose that T does not contain the tuple $\mathbf{x} = 1^d 0^{k-d}$ of odd weight d. If $t < d$, then XOR_d applied to the input matrix C_d^t padded with $k-d$ rows of 0's returns $\mathbf{x}$. If $t > d$, let M be the $k \times (t-d+1)$ matrix with d rows of 1's followed by the matrix C_{t-d+1}^{t-d}. Fill any extra rows with 0's. Then $\mathrm{XOR}_{t-d+1}(M) = \mathbf{x}$. An illustration is given in Figure (7b). Therefore $\mathrm{XOR} \not\subseteq \mathrm{Pol}(\textbf{t-in-k}, \mathbf{T})$.

$$\mathrm{XOR}_7 \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & \mathbf{1} & 0 & 0 \\ 0 & 1 & 1 & 1 & \mathbf{0} & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & \mathbf{1} \\ 0 & 0 & 0 & 1 & 1 & 1 & \mathbf{0} \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 \end{pmatrix} = \begin{matrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \end{matrix}$$

(a) $t = 4$ and $k = 8$.

$$\mathrm{XOR}_3 \left(\begin{array}{ccc} 1 & 1 & 1 \\ \hline 1 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 1 & 1 \\ \hline 0 & 0 & 0 \\ 0 & 0 & 0 \end{array}\right) = \begin{matrix} 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{matrix}$$

(b) $t = 3$, $k = 6$, and $d = 1$.

Figure 7 XOR.

Inversions: Let F be a family of functions. We reduce the task of showing $\overline{F} \not\subseteq \mathrm{Pol}(\mathbf{t\text{-}in\text{-}k}, \mathbf{T})$ to the already completed task of showing $F \not\subseteq \mathrm{Pol}(\mathbf{t\text{-}in\text{-}k}, \mathbf{T})$. Let $\mathbf{x} \in \{0,1\}^k \setminus \mathbf{T}$, let $f \in F$ be a function of arity m, and let M be a $k \times m$ matrix of inputs to f whose columns are **t-in-k** tuples. We established $F \not\subseteq \mathrm{Pol}(\mathbf{t\text{-}in\text{-}k}, \mathbf{T})$ by finding f and M with $f(M) = \mathbf{x}$, and in the remaining cases we must find $\overline{f} \in \overline{F}$ and M such that $\overline{f}(M) = \mathbf{x}$. But since $\overline{f}(M) = \mathbf{x} \Leftrightarrow f(M) = \overline{\mathbf{x}}$, it suffices to find $f \in F$ such that $f(M) = \overline{\mathbf{x}}$, where $\overline{\mathbf{x}} = (1 - x_1, \ldots, 1 - x_k)$ if $\mathbf{x} = (x_1, \ldots, x_k)$.

The families $\overline{\mathrm{AND}}$, $\overline{\mathrm{OR}}$, and $\overline{\mathrm{XOR}}$ (except when t is odd and k is even) are excluded from $\mathrm{Pol}(\mathbf{t\text{-}in\text{-}k}, \mathbf{T})$ in the same way as AND, OR, and XOR, with the same matrices serving as counterexamples. To see that $\overline{\mathrm{AT}}$ and $\overline{\mathrm{THR}_{\frac{t}{k}}}$ are also excluded, let $\mathbf{x} \notin \mathbf{T}$ be a tuple of weight $d, d \neq t, 1 \leq d < k$. Then the tuple $\overline{\mathbf{x}}$ of weight $k - d$ can be returned by an AT function and a $\mathrm{THR}_{\frac{t}{k}}$ function by the arguments above. If $k - d = t$, then the AT and $\mathrm{THR}_{\frac{t}{k}}$ functions of arity 1 output $\overline{\mathbf{x}}$ on input $\overline{\mathbf{x}}$.

Finally, when t is odd and k is even, and $\mathbf{T}$ does not contain the tuple $\mathbf{x}$ of odd weight d, the XOR argument above applies since $\overline{\mathbf{x}}$ also has odd weight $k - d$. Again, if $k - d = t$, then the XOR function of arity 1 outputs $\overline{\mathbf{x}}$ on input $\overline{\mathbf{x}}$. ◀

Schaefer's dichotomy theorem (Theorem 4) allows us to obtain a simple description of all $\mathbf{T}$ with $\mathrm{CSP}(\mathbf{T})$ tractable and $\mathbf{t\text{-}in\text{-}k} \to \mathbf{T}$; a proof can be found in [14].

▶ **Proposition 17.** *Let $k \geq 3$, $1 \leq t < k$, and suppose that $\mathbf{t\text{-}in\text{-}k} \to \mathbf{T}$. Then $\mathrm{CSP}(\mathbf{T})$ is tractable if and only if*

1. *$0^k \in \mathbf{T}$ or $1^k \in \mathbf{T}$, or*
2. *t is odd, k is even, and $\mathbf{T}$ contains all tuples of odd weight.*

Observe that Proposition 17 in particular implies Proposition 7, NP-hardness of $\mathrm{CSP}(\mathbf{t\text{-}in\text{-}k})$.

With Proposition 17 in hand, we can prove Theorem 11.

▶ **Theorem** (Theorem 11 restated). *Let $k \geq 3$ and $\emptyset \neq S \subseteq (\mathbf{t\text{-}in\text{-}k})^c \cap \mathbf{NAE}$. If t is odd, k is even, and S contains tuples of only even weight, then $\mathrm{PCSP}(\mathbf{t\text{-}in\text{-}k}, \mathbf{NAE} \setminus \mathbf{S})$ is tractable. Otherwise, $\mathrm{PCSP}(\mathbf{t\text{-}in\text{-}k}, \mathbf{NAE} \setminus \mathbf{S})$ is NP-hard.*

Proof. The tractability in the first statement of the theorem is proved in [14]. Otherwise, t is even, or k is odd, or S contains a tuple of odd weight. Take $\mathbf{T} = \mathbf{NAE} \setminus \mathbf{S}$. Observe that case (1) of Proposition 17 does not apply as neither 0^k nor 1^k is part of the template. Moreover,

case (2) of Proposition 17 does not apply either: If t is odd and k is even then S contains a tuple of odd weight and hence $\mathbf{NAE} \setminus \mathbf{S}$ cannot have all odd weight tuples. Thus, by Proposition 17, CSP($\mathbf{T}$) is NP-hard. Then, by Theorem 12, PCSP($\mathbf{t\text{-}in\text{-}k}, \mathbf{T}$) = PCSP($\mathbf{t\text{-}in\text{-}k}, \mathbf{NAE} \setminus \mathbf{S}$) is NP-hard. ◀

References

1. Per Austrin, Amey Bhangale, and Aditya Potukuchi. Improved inapproximability of rainbow coloring. In *Proceedings of the 31st Annual ACM-SIAM Symposium on Discrete Algorithms (SODA'20)*, pages 1479–1495, 2020. doi:10.1137/1.9781611975994.90.
2. Per Austrin, Venkatesan Guruswami, and Johan Håstad. (2+ϵ)-Sat is NP-hard. *SIAM J. Comput.*, 46(5):1554–1573, 2017. doi:10.1137/15M1006507.
3. Libor Barto. Promises Make Finite (Constraint Satisfaction) Problems Infinitary. In *Proceedings of the 34th Annual ACM/IEEE Symposium on Logic in Computer Science (LICS'19)*, pages 1–8. IEEE, 2019. doi:10.1109/LICS.2019.8785671.
4. Libor Barto, Diego Battistelli, and Kevin M. Berg. Symmetric Promise Constraint Satisfaction Problems: Beyond the Boolean Case. In *Proceedings of the 38th International Symposium on Theoretical Aspects of Computer Science (STACS'21)*, volume 187 of *LIPIcs*, pages 10:1–10:16. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPIcs.STACS.2021.10.
5. Libor Barto, Jakub Bulín, Andrei A. Krokhin, and Jakub Opršal. Algebraic approach to promise constraint satisfaction. *J. ACM*, 2018. arXiv:1811.00970.
6. Libor Barto, Andrei Krokhin, and Ross Willard. Polymorphisms, and how to use them. In Andrei Krokhin and Stanislav Živný, editors, *The Constraint Satisfaction Problem: Complexity and Approximability*, volume 7 of *Dagstuhl Follow-Ups*, pages 1–44. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl, Germany, 2017. doi:10.4230/DFU.Vol7.15301.1.
7. Libor Barto, Jakub Opršal, and Michael Pinsker. The wonderland of reflections. *Israel Journal of Mathematics*, 223(1):363–398, February 2018. doi:10.1007/s11856-017-1621-9.
8. Joshua Brakensiek and Venkatesan Guruswami. New hardness results for graph and hypergraph colorings. In Ran Raz, editor, *31st Conference on Computational Complexity (CCC 2016)*, volume 50 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 14:1–14:27, Dagstuhl, Germany, 2016. Schloss Dagstuhl – Leibniz-Zentrum fuer Informatik. doi:10.4230/LIPIcs.CCC.2016.14.
9. Joshua Brakensiek and Venkatesan Guruswami. Promise Constraint Satisfaction: Structure Theory and a Symmetric Boolean Dichotomy. In *Proceedings of the 29th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA'18)*, pages 1782–1801. SIAM, 2018. doi:10.1137/1.9781611975031.117.
10. Joshua Brakensiek and Venkatesan Guruswami. An Algorithmic Blend of LPs and Ring Equations for Promise CSPs. In *Proceedings of the 30th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA'19)*, pages 436–455. SIAM, 2019. doi:10.1137/1.9781611975482.28.
11. Joshua Brakensiek and Venkatesan Guruswami. Symmetric polymorphisms and efficient decidability of promise CSPs. In *Proceedings of the 2020 ACM-SIAM Symposium on Discrete Algorithms, SODA 2020, Salt Lake City, UT, USA, January 5-8, 2020*, pages 297–304, 2020. doi:10.1137/1.9781611975994.18.
12. Joshua Brakensiek, Venkatesan Guruswami, Marcin Wrochna, and Stanislav Živný. The power of the combined basic LP and affine relaxation for promise CSPs. *SIAM Journal on Computing*, 49:1232–1248, 2020. doi:10.1137/20M1312745.
13. Alex Brandts, Marcin Wrochna, and Stanislav Živný. The Complexity of Promise SAT on Non-Boolean Domains. In *Proceedings of the 47th International Colloquium on Automata, Languages, and Programming (ICALP'20)*, volume 168, pages 17:1–17:13. Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPIcs.ICALP.2020.17.

14 Alex Brandts and Stanislav Živný. Beyond PCSP(1-in-3,NAE), 2021. arXiv:2104.12800.

15 Andrei Bulatov, Peter Jeavons, and Andrei Krokhin. Classifying the complexity of constraints using finite algebras. *SIAM Journal on Computing*, 34(3):720–742, 2005. doi:10.1137/S0097539700376676.

16 Andrei A. Bulatov. A dichotomy theorem for nonuniform CSPs. In *Proceedings of the 58th IEEE Annual Symposium on Foundations of Computer Science (FOCS)*, pages 319–330, 2017. doi:10.1109/FOCS.2017.37.

17 Jakub Bulín, Andrei Krokhin, and Jakub Opršal. Algebraic approach to promise constraint satisfaction. In *Proceedings of the 51st Annual ACM SIGACT Symposium on the Theory of Computing (STOC '19)*, New York, NY, USA, 2019. ACM. doi:10.1145/3313276.3316300.

18 Irit Dinur, Elchanan Mossel, and Oded Regev. Conditional hardness for approximate coloring. *SIAM J. Comput.*, 39(3):843–873, 2009. doi:10.1137/07068062X.

19 Irit Dinur, Oded Regev, and Clifford Smyth. The hardness of 3-uniform hypergraph coloring. *Combinatorica*, 25(5):519–535, 2005. doi:10.1007/s00493-005-0032-4.

20 Tomás Feder and Moshe Y. Vardi. The Computational Structure of Monotone Monadic SNP and Constraint Satisfaction: A Study through Datalog and Group Theory. *SIAM Journal on Computing*, 28(1):57–104, 1998. doi:10.1137/S0097539794266766.

21 Miron Ficak, Marcin Kozik, Miroslav Olšák, and Szymon Stankiewicz. Dichotomy for Symmetric Boolean PCSPs. In *Proceedings of the 46th International Colloquium on Automata, Languages, and Programming (ICALP'19)*, volume 132, pages 57:1–57:12. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2019. doi:10.4230/LIPIcs.ICALP.2019.57.

22 M. R. Garey and David S. Johnson. The complexity of near-optimal graph coloring. *J. ACM*, 23(1):43–49, 1976. doi:10.1145/321921.321926.

23 Venkatesan Guruswami and Sai Sandeep. d-To-1 Hardness of Coloring 3-Colorable Graphs with O(1) Colors. In *Proceedings of the 47th International Colloquium on Automata, Languages, and Programming (ICALP'20)*, volume 168 of *LIPIcs*, pages 62:1–62:12. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPIcs.ICALP.2020.62.

24 Pavol Hell and Jaroslav Nešetřil. On the Complexity of H-coloring. *Journal of Combinatorial Theory, Series B*, 48(1):92–110, 1990. doi:10.1016/0095-8956(90)90132-J.

25 Peter Jeavons, David A. Cohen, and Marc Gyssens. Closure properties of constraints. *J. ACM*, 44(4):527–548, 1997. doi:10.1145/263867.263489.

26 Richard M. Karp. Reducibility Among Combinatorial Problems. In *Proceedings of a Symposium on the Complexity of Computer Computations*, pages 85–103, 1972. URL: http://www.cs.berkeley.edu/%7Eluca/cs172/karp.pdf.

27 Subhash Khot. On the power of unique 2-prover 1-round games. In *Proceedings of the 34th Annual ACM Symposium on Theory of Computing (STOC'02)*, pages 767–775. ACM, 2002. doi:10.1145/509907.510017.

28 Andrei Krokhin and Jakub Opršal. The complexity of 3-colouring H-colourable graphs. In *2019 IEEE 60th Annual Symposium on Foundations of Computer Science (FOCS'19)*, pages 1227–1239, 2019. doi:10.1109/FOCS.2019.00076.

29 Thomas Schaefer. The complexity of satisfiability problems. In *Proceedings of the tenth Annual ACM Symposium on the Theory of Computing (STOC '78)*, pages 216–226, 1978. doi:10.1145/800133.804350.

30 Marcin Wrochna and Stanislav Živný. Improved hardness for H-colourings of G-colourable graphs. In *Proceedings of the Fourteenth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA'20)*, pages 1426–1435, 2020. doi:10.1137/1.9781611975994.86.

31 Dmitriy Zhuk. A proof of the CSP dichotomy conjecture. *J. ACM*, 67(5):30:1–30:78, 2020. doi:10.1145/3402029.

Computational Characterization of Surface Entropies for $\mathbb{Z}^2$ Subshifts of Finite Type

Antonin Callard
Université Paris-Saclay, ENS Paris-Saclay, Département Informatique, 91190 Gif-sur-Yvette, France

Pascal Vanier
Normandie Univ, UNICAEN, ENSICAEN, CNRS, GREYC, 14000 Caen, France

Abstract

Subshifts of finite type (SFTs) are sets of colorings of the plane that avoid a finite family of forbidden patterns. In this article, we are interested in the behavior of the growth of the number of valid patterns in SFTs. While entropy h corresponds to growths that are squared exponential 2^{hn^2}, surface entropy (introduced in Pace's thesis in 2018) corresponds to the eventual linear term in exponential growths. We give here a characterization of the possible surface entropies of SFTs as the Π_3 real numbers of $[0, +\infty]$.

2012 ACM Subject Classification Mathematics of computing → Discrete mathematics; Theory of computation → Models of computation

Keywords and phrases surface entropy, arithmetical hierarchy of real numbers, 2D subshifts, symbolic dynamics

Digital Object Identifier 10.4230/LIPIcs.ICALP.2021.122

Category Track B: Automata, Logic, Semantics, and Theory of Programming

Funding This research was partially funded by ANR JCJC 2019 19-CE48-0007-01.

Acknowledgements The authors would like to thank Ronnie Pavlov for answering their many questions about surface entropy when they started this work, and Benjamin Hellouin de Menibus for the relecturing. Finally, we warmly thank the anonymous reviewers for their many helpful remarks and improvements.

1 Introduction

For a finite alphabet of colors, a two dimensional subshift is the set of all colorings of the plane $\mathbb{Z}^2$ that respect some local constraints. These constraints are usually given as a family of forbidden patterns. The most studied class of subshift are *Subshifts of Finite Type* (SFTs), subshifts that can be defined by a finite family of forbidden patterns. A famous special case is defined by Wang tilesets (unit squares with colored edges that may be placed side by side only when the colors on the edges match) are a famous special: the set of all tilings by some Wang tileset is always an SFT.

Subshifts were introduced in order to discretize continuous dynamical systems and are themselves dynamical systems. While it has long been known that most problems concerning Wang tiles (and thus subshifts) are undecidable [2, 5, 6], the role of computability has since shifted from an obstacle to a major tool in the study of SFTs and other related classes of subshifts. An aperiodic subshift has for instance been constructed [4] based on Kleene's *fixed-point theorem* [11], a classical theorem of computability theory. Many conjugacy invariants have been characterized using computability or complexity classes. The first such characterization was for *topological entropy*, which measures the exponential growth of the number of valid colorings of finite patterns. For a subshift X, its (topological) entropy is defined by:

$$h(X) = \lim \frac{\log N_X(n)}{n^2}$$

where $N_X(n)$ denotes the number of $n \times n$ patterns appearing in colorings of X. Having entropy h corresponds to having $N_X = 2^{hn^2+o(n^2)}$. It turns out that the possible entropies for $\mathbb{Z}^2$ are exactly the upper semi-computable [8] real numbers. Another invariant related to growth, called *entropy dimension*, was then characterized using the arithmetical hierarchy of real numbers [13]. Many other invariants have since been linked to computability: for instance, there is a relationship between periodic colorings and computational complexity classes [10], subactions [7, 1, 3] can be characterized through recursively enumerable forbidden patterns, and many more.

We focus here on the notion of *surface entropy*, a notion which was introduced in Dennis Pace's thesis [14] in order to quantify the linear term inside exponential growth functions. Indeed, topological entropy cannot distinguish between the two following behaviors of the complexity function:

$$N_X(n) \approx 2^{hn^2} \qquad \text{and} \qquad N_X(n) \approx 2^{hn^2+2sn}.$$

Surface entropy corresponds roughly to the s term in the second behavior and will be introduced more formally in Subsection 2.2. In his thesis, Pace realizes Π_1 and Σ_1 numbers of the arithmetical hierarchy of real numbers and conjectures that surface entropies are exactly the Π_3 numbers. This is exactly what we prove here:

▶ **Theorem 1.** *The class of surface entropies of $\mathbb{Z}^2$ SFTs is $[0, +\infty] \cap \Pi_3$.*

In fact, surface entropies still cover the whole class of Π_3 real numbers for sofic subshifts, which are the letter-by-letter projections of SFTs, and for effective subshifts, which are the subshifts that can be defined with a recursively enumerable family of forbidden patterns:

▶ **Theorem 2.** *The class of surface entropies of $\mathbb{Z}^2$ sofic subshifts, and of $\mathbb{Z}^2$ effective subshifts, is $[0, +\infty] \cap \Pi_3$.*

The paper is organized as follows. The next section recalls some background and useful definitions. Section 3 and Section 4 focus on the proof of Theorem 1, and provide in particular a construction which creates $\mathbb{Z}^2$ SFTs with arbitrary Π_3 surface entropies. Some open questions are then discussed in Section 5.

2 Preliminaries

2.1 Subshifts

This subsection introduces some standard definitions and facts about subshifts. The reader may consult [12] for more details.

Let Σ be a finite alphabet of colors. A $\mathbb{Z}^d$ *configuration* (in this paper, $d = 1$ or $d = 2$) is a coloring $x : \mathbb{Z}^d \mapsto \Sigma$, and the value of x at position z is noted x_z. A *(d-dimensional) pattern* is a coloring $p : D \mapsto \Sigma$, with $D \subseteq \mathbb{Z}^d$ a finite domain. For a configuration x, we say that a pattern p appears in x (noted $p \sqsubseteq x$) if there exists some position $t \in \mathbb{Z}^d$ such that for all $z \in D$, $p_z = x_{t+z}$. A *subshift* is a set of colorings/configurations defined by some family of forbidden patterns. Each family of forbidden patterns $\mathcal{F}$ defines a subshift, possibly empty:

$$X_{\mathcal{F}} = \{(x : \mathbb{Z}^d \mapsto \Sigma) : \forall p \in \mathcal{F}, p \not\sqsubseteq x\}.$$

A subshift is *effective* if it can be defined by a recursively enumerable family of forbidden patterns. A subshift is *of finite type* (SFT) if it can be defined by a finite family of forbidden patterns. A subshift Y on Σ is *sofic* if there exists an SFT X on an alphabet Σ' and a letter-by-letter projection $\pi : \Sigma' \mapsto \Sigma$ that sends X on Y.

Finally, for any $\mathbb{Z}$ subshift X_1, the $\mathbb{Z}^2$ *lift of* X_1 is the subshift Y_2 whose configurations are vertically repeated configurations of X_1:

$$Y_2 = \{(y : \mathbb{Z}^2 \mapsto \Sigma) : \exists x \in X_1, \forall i, j \in \mathbb{Z}, y_{(i,j)} = x_i\}.$$

2.2 Complexity function of $\mathbb{Z}^2$ subshifts

Given a $\mathbb{Z}^2$ subshift X, its *complexity function* $N_X(m, n)$ (for $m, n \in \mathbb{N}$) is the number of different patterns that appear in a rectangle of size $m \times n$ in the configurations of X:

$$N_X(m, n) = \left|\left\{p \in \Sigma^{m\times n} : \exists x \in X, p \sqsubseteq x\right\}\right|.$$

This complexity function can be used to define the (topological) entropy $h(X)$:

$$h(X) = \lim_{n\to+\infty} \frac{\log N_X(n, n)}{n^2}.$$

This led Dennis Pace to introduce in [14] the notion of *surface entropy*, which corresponds to the "linear term" of the complexity function. Here, we define *surface entropy with eccentricity* α as:

$$h_s(X, \alpha) = \limsup_{n\to+\infty} \frac{\log N_X(pn, qn) - pqn^2 h(X)}{(p + q)n} \qquad \text{with } \alpha = \frac{p}{q} \in \mathbb{Q}^+,$$

where p, q are relatively prime integers. This definition slightly differs from the one of [14]; this will be further discussed in Section 5.

Note that in the definition of the topological entropy, only square patterns are used. In fact, any rectangular patterns would generate the same value. Interestingly, this is no longer the case with surface entropy: the *eccentricity* (ratio of the patterns' widths to heights) affects the value of the calculations. This explains why h_s is a function of both a subshift X and a rational parameter $\alpha = p/q$. The study in [14] focuses on both the realizability of specific surface entropies, and the behavior of surface entropies as functions of their eccentricities.

While surface entropy is not a conjugacy invariant, it was proved in [14] that the finiteness of surface entropy is invariant under conjugacy. For more details or examples about surface entropies, one may refer to [14, Section 3.2].

Links with other growth-type invariants

Links with (topological) entropy. If X_1 is a $\mathbb{Z}$ subshift, and Y_2 its $\mathbb{Z}^2$ lift (ie. the subshift whose configuration are vertically repeated configurations of X_1), then $h_s(Y_2, \alpha) = \frac{\alpha}{1+\alpha} h(X_1)$.

Links with entropy dimension. In [13], a growth-type invariant called *entropy dimension* was introduced as $h_d(X) = \limsup_{n\to+\infty}(\log\log N_X(n))/\log n$. It roughly represents the exponent α if $\log N_X(n) \simeq n^\alpha$. For any $\mathbb{Z}^2$ subshift X, $0 \leq h_d(X) \leq 2$.

Entropy dimension and surface entropies are linked as follows:

- If $h_d(X) = 2$, then $h_s(X)$ can be either finite or infinite (similarly to $h_d(X) = 1$);
- If $1 < h_d(X) < 2$, then $h_s(X) = +\infty$;
- If $h_d(X) = 1$, $h_s(X)$ can either be finite (if $\log N_X(n) = O(n)$, see [14, Example 3.2.14]) or infinite (for example, if $\log N_X(n) \simeq n \log n$, see [14, Example 3.2.4]);
- If $0 \leq h_d(X) < 1$, then $h_s(X) = 0$.

2.3 Arithmetical hierarchy of real numbers

In order to state our main result, this section recalls from [15] the arithmetical hierarchy of real numbers, which classifies elements of the real line according to their computational properties. Denote by $\Gamma_{\mathbb{Q}}$ the set of total computable functions $f : \mathbb{N}^k \mapsto \mathbb{Q}$ for any k. For $n \geq 1$, the classes of real numbers Σ_n, Π_n and Δ_n are defined as follows:

$$\Sigma_n = \{ \sup_{i_1} \inf_{i_2} \sup_{i_3} \dots f(i_1, \dots, i_n) \mid f \in \Gamma_{\mathbb{Q}} \}$$

$$\Pi_n = \{ \inf_{i_1} \sup_{i_2} \inf_{i_3} \dots f(i_1, \dots, i_n) \mid f \in \Gamma_{\mathbb{Q}} \}$$

$$\Delta_n = \Sigma_n \cap \Pi_n.$$

It is known that for any $n \geq 1$, the inclusions $\Pi_n \subset \Sigma_{n+1}$ and $\Sigma_n \subset \Pi_{n+1}$ are proper. One may refer to [15] for more details.

In this paper, we will be interested in the third level of the hierarchy, and one of its equivalent characterization proved in [15]:

$x \in \Pi_3$ iff there exists $f \in \Gamma_{\mathbb{Q}}$ such that $x = \limsup_i \inf_j f(i, j)$.

3 Arithmetical restrictions of surface entropies

In this section, we prove the first and easiest direction of Theorem 1:

▶ **Theorem 3.** *For any $\mathbb{Z}^2$ SFT X and $\alpha \in \mathbb{Q}^+$:*

$$h_s(X, \alpha) \in [0, +\infty] \cap \Pi_3$$

Proof. Let X be a $\mathbb{Z}^2$ an SFT. We prove that $N_X(m, n)$ is a Π_1 real number (it is not computable in X: indeed, $N_X(m, n) = 0$ if and only if $X = \emptyset$; the latter is well-known for being undecidable). A pattern is said to be *admissible* if it does not contain a pattern which is forbidden: any valid pattern of X is admissible, but the converse is false. For $j \geq m, n$, define $N_X^{(j)}(m, n)$ as the number of $m \times n$ patterns that appear at the center of admissible patterns of size $j \times j$ (ie. the squares of size $j \times j$ in which none of the forbidden patterns of X appear).

By compactness, if a pattern is not valid in X, there exists some j such that it does not appear in any admissible pattern bigger than $j \times j$: this proves $N_X(m, n) = \inf_{j \geq m,n} N_X^{(j)}(m, n)$. In particular, this implies that $\log N_X(m, n)$ is a Π_1 real number.

It follows that $h(X)$ is a Π_1 real number (as proved in [8]). As the difference of two Π_1 real numbers is a Δ_2 real number, this proves that for any $\alpha = p/q \in \mathbb{Q}^+$, and for every $n \in \mathbb{N}$, the following is a Δ_2 real number:

$$\frac{\log N_X(pn, qn) - pqn^2 h(X)}{(p + q)n}$$

which then leads to $h_s(X, \alpha) \in \Pi_3$. Finally, by sub-additivity $h_s(X, \alpha) \geq 0$ (see [14]). ◀

A very similar reasoning proves that this is still the case for sofic or effective subshifts:

▶ Remark 4. For any $\mathbb{Z}^2$ sofic or effective subshift X and $\alpha \in \mathbb{Q}^+$, $h_s(X, \alpha) \in [0, +\infty] \cap \Pi_3$.

Proof. As sofic subshifts are effective, we can focus on the latter. For X an effective subshift, we slightly alter the definition of $N_X^{(j)}(m,n)$: it will now count the number of patterns of size $m \times n$ which appear in the squares of size $j \times j$ in which *none of the first j enumerated forbidden patterns* of X appear. Then we still have $N_X(m,n) = \inf_{j \geq m,n} N_X^{(j)}(m,n)$, and this still leads to $\log N_X(m,n) \in \Pi_1$ and $h(X) \in \Pi_1$.

After this, the end of the proof is the same. ◀

4 Realization of Π_3 real numbers as surface entropies

In this section, we prove the other (and harder) direction of Theorem 1. Because [14] provides examples of SFTs with infinite surface entropy, we now prove:

▶ **Theorem 5.** *For any $x \in [0, +\infty) \cap \Pi_3$, there exists an SFT X with surface entropy:*

$$\forall \alpha \in \mathbb{Q}^+, \qquad h_s(X, \alpha) = \frac{\min(\alpha, 1)}{1+\alpha} \cdot x.$$

Theorem 1 and 2 are consequences of the previous section together with this statement.

A naive construction could consist in lifting $\mathbb{Z}$ subshifts to obtain $\mathbb{Z}^2$ SFTs with specific surface entropies. However, as the topological entropy of a $\mathbb{Z}$ effective subshift is a Π_1 real number (by ideas similar to Section 3), lifts can only have Π_1 surface entropies (cf. the link between $\mathbb{Z}$ entropies and $\mathbb{Z}^2$ surface entropies in Section 2.2).

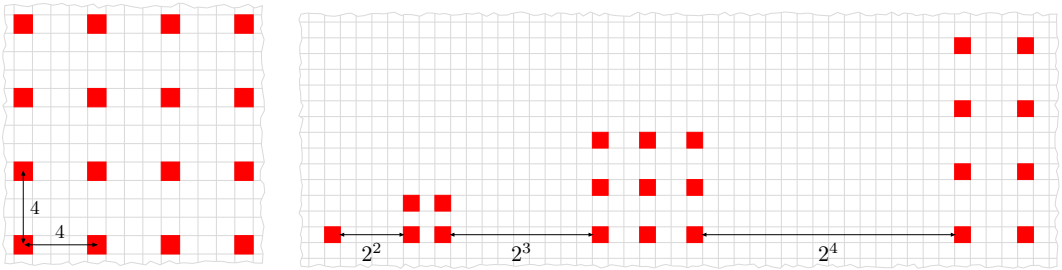
Any method to overcome this limitation would need some creative spatial distribution. Hence the *sparse squares* we develop below.

Proof. Let $x \in [0, +\infty) \cap \Pi_3$ be a Π_3 real number.

Let $e \in \mathbb{N}$ and $x' \in [0, 1)$ be such that $x = e + x'$. Because x' is also a Π_3 real number, there exists a computable function $f : \mathbb{N}^2 \mapsto \mathbb{Q}$ such that $x' = \limsup_k \inf_l f(k,l)$ (see the characterization in Subsection 2.3). We can assume that f only takes values in $[0, 1)$.

In the following subsections, we create an SFT X which verifies the property of Theorem 5. The proof is organized as follows:

1. Subsection 4.1 introduces our "sparse squares" construction, which aims at creating a set of colorings of the plane with controlled surface entropy. All the following sections focus on implementing this geometrical construction into an actual SFT X.
2. Subsection 4.2 recalls the Toeplitz sequences, which are sequences of uniform densities. They will be used in the sparse square layout to control the density of each square.
3. In Subsection 4.3, we create a $\mathbb{Z}$ effective subshift X_1. Effectiveness gives us a lot of room to control its patterns, and we will use X_1 as the foundation of X.
4. We use the *fixed-point* construction of [3] to create a $\mathbb{Z}^2$ SFT which simulates X_1. Subsection 4.4 provides an intermediary lemma about the entropy and the surface entropy of this construction: in our case, it proves that the intermediary construction has surface entropy zero.
5. In Subsection 4.5, we then create the desired SFT X, which arranges the sparse squares on the plane (with the help of the previous points).
6. Finally, in Subsection 4.6 and Subsection 4.7 we create and compute the surface entropy of X. This proves that X is a valid example for Theorem 5.



(a) 4^{th} sparse square. **(b)** (Portion of) the sparse square layout.

Figure 1 Presentation of the sparse square layout.

4.1 The sparse squares and the sparse square layout

To understand the idea behind this construction, consider the *full shift* over $\mathbb{Z}^2$ on the alphabet $\{0,1\}$. Configurations are full grids of *free bits*, ie. bits that are allowed to vary freely in $\{0,1\}$. It is not difficult to see that for the full shift, $\log N_{\text{full}}(n,n) = n^2$. In particular, its complexity function is quadratic in n, and its entropy is 1.

To realize specific surface entropies, we first need to figure out a way to contribute *linearly* to $\log N_X$, instead of quadratically. To do this, we create a sequence of *sparse squares*. A sparse square is, roughly, a finite piece taken from the full shift, but whose points are moved apart from one another: the square is *sparsified*.

More precisely, the *sparse square of index* k (see Figure 1a) is a set of positions that form a finite grid. Any position not in this set of points is blank. In the grid, there are k columns (the distance between two columns is also k), and in each column there are k points (the distance between two points in a column is also k).

The *sparse square layout* (see Figure 1b) makes the sparse squares sit next to one another on a single line, according to their indices. We set the distance between the square of index $k-1$ and the square of index k to 2^k.

The key feature of this geometrical layout lies in its linearity: the k^{th} square has edges of size $k(k-1)+1$: thus it has an area which is roughly $\left(k^2\right)^2$, while it also contains k^2 points. In addition, as the distance between two points increases as one considers squares of greater indices, compactness will only lead to degenerate configurations that contain at most one point. As such, they will not contribute significantly to the complexity function.

For now, the set of positions in the sparse square layout is not very interesting. In order to increase the complexity function, we will allow free bits to vary at each position inside a sparse square. Additionally, to create a surface entropy related to x, the density of free bits in each square will be related to x.

More precisely, define $x'_k = \inf_l f(k,l)$ (recall that $x = e + x'$, and $x' = \limsup_k \inf_l f(k,l)$). For each k, let t_k be a word of size k over the alphabet $\{\text{On}, \text{Off}\}$, whose density of On is x'_k. Then, define the subshift X' as (the closure of) the following configurations (see Figure 2):

- These configurations follow the sparse square layout.
- Each position not marked in the sparse square layout is blank (ie. marked with $\square$).
- The word t_k is written in each row of the k^{th} square.

On the positions marked by On, we then allow free bits to vary. On every position of the sparse square, we also allow free letters to vary in $\{1,\dots,2^e\}$. With this, the square of index k contributes to $\log N_{X'}$ with a term $k^2(x'_k + e)$. More precisely:

▶ **Lemma 6.** *The k^{th} sparse square contributes to $\log N_{X'}$ more than $k^2(e + x'_k) + O(k)$.*

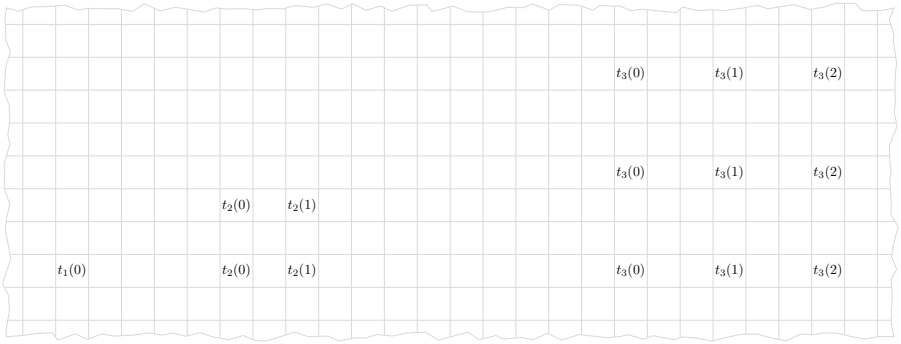


Figure 2 A sequence of words (t_k) written in (a finite piece of) the sparse square layout.

Proof. As this is a lower bound, consider the position of the k^{th} sparse square fixed, and only look at the contribution induced by its "free bits" and "free letters".

Free bits: (For now, we assume the "free letters" are fixed). A word t_k of density x'_k is written in this square, and the square contains $k|t_k|_{\text{ON}}$ free bits: indeed, there are k identical lines, and in each line there are $|t_k|_{\text{ON}}$ positions marked with ON. Additionally, one has $|t_k|_{\text{ON}} = kx'_k + O(1)$ (because the density is x'_k). This implies that all the free bits of the k^{th} sparse square (fixed at this position), when marked with t_k, contribute to the complexity function with a term $\exp_2(k|t_k|_{\text{ON}}) = \exp_2(k^2x'_k + O(k))$.

Free letters: Recall that free letters vary in the alphabet $\{1, \dots, 2^e\}$ at each point in the k^{th} sparse squares. There are k^2 points in this square, so the previous contribution is multiplied by $\exp_2(k^2e)$. This concludes the proof. ◀

This provides a lower bound for the complexity function: for a given size of pattern, we only count the biggest square that fits in it. If the squares do not interfere too much, one should expect the surface entropy to "converge" towards $\limsup_k x'_k + e = x' + e = x$.

The previous paragraphs were a draft of a geometrical construction. Below, we create an actual $\mathbb{Z}^2$ SFT X which implements this subshift (with some additional construction lines). Then, we formally prove that X has the desired surface entropy.

4.2 Effective $\mathbb{Z}$ upper-density Toeplitz subshifts

In order to create specific densities of letters in a subshift, we recall the useful Toeplitz sequences from [9]. Let $0 \leq y = \sum_{i=1}^{+\infty} y_i 2^{-i} \leq 1$ be a real number. A *Toeplitz sequence* associated to y is a bi-infinite sequence $b \in \{0,1\}^{\mathbb{Z}}$ such that:

$$\forall k \geq 1, \exists j_k \notin \left(\sum_{i=1}^{k-1} j_i + 2^i \mathbb{Z} \right), \ b_{j_k + 2^k \mathbb{Z}} = y_k$$

i.e. one bit in two is y_1; on the remaining bits, one bit in two is y_2; etc...

For any $y \in [0,1)$, we define the *upper-density Toeplitz subshift* $\mathcal{T}(y)$ *associated to* y:

$$\mathcal{T}(y) = \{(b) \in \{0,1\}^{\mathbb{Z}} : \exists 0 \leq y' \leq y, (b) \text{ is a Toeplitz sequence associated to } y'\}.$$

In the following subsection, we will use subwords of Toeplitz sequences on the alphabet $\{\text{ON}, \text{OFF}\}$ (rather than $\{1,0\}$). They have high regularity and tightly controlled densities. Indeed, assume that t_n is a factor of length n which appears in $\mathcal{T}(y)$. Then the number of letters ON in t_n is bounded by $0 \leq |t_n|_{\text{ON}} \leq ny + O(1)$.

We will use these subshits in the context of Π_1 real numbers. Indeed:

▶ **Lemma 7.** *If $y \in \Pi_1 \cap [0,1)$, then $\mathcal{T}(y)$ is an effective $\mathbb{Z}$ subshift.*

Proof. If $y \in \Pi_1$, there exists a computable total function $f : \mathbb{N} \mapsto \mathbb{Q}$ such that $y = \inf_n f(n)$. It is possible to recursively enumerate the different values of f, to computably forbid patterns of $\mathcal{T}(y)$ which do not respect the structure of Toeplitz sequences, and to computably forbid patterns of $\mathcal{T}(y)$ that respect the structure of Toeplitz sequences but whose density is too high. ◀

4.3 Building the base line

Consider the line on which all the sparse squares sit in the previous layout. We call it the *base line*. In this section, we create a $\mathbb{Z}$ effective subshift X_1 on the alphabet $\Sigma = \{\#, S, E, B, \text{On}, \text{Off}\}$ which implements the base line: its configurations will be all the possible values for this base line.

Base of the k^{th} sparse square. The base of the k^{th} sparse square is composed of k points, each separated by $k-1$ blanks (in X_1, these blanks will be marked by B). In each point, a letter On or Off is written, and the word t_k composed by these k letters is a Toeplitz subword of $\mathcal{T}(x'_k)$, where $x'_k = \inf_l f(k,l)$.

This naturally leads to defining the set of all possible bases for the k^{th} sparse square:

$$\mathcal{W}_k = \{t_k(1)\ B^{k-1}\ t_k(2)\ B^{k-1}\ \dots\ B^{k-1}\ t_k(k) \mid t_k \text{ is a toeplitz subword in } \mathcal{T}(x'_k)\}.$$

Prefixes for the squares. In order to implement the sparse square layout, we will build the sparse squares on top of the base line. In order to build the squares properly, we set a specific prefix before the base of each square. These prefixes have no meaning in terms of surface entropy, and can be considered as construction lines.

In the layout, the positions of these prefixes were blank; in the implementation, they are marked with letters S, B and E, which will have different roles when building other construction lines in Subsection 4.5.

These prefixes are $s_1 = S$, and for $k \geq 2$, $s_k = SB^{k-2}E$.

The whole subshift. We now create a whole base line with these considerations: for any k, the base of the k^{th} sparse square is a word of $\mathcal{W}_k$; before the base of the k^{th} sparse square, we add the word s_k; and the other positions are marked with blanks $\#$.

For the rest of the paper, we denote X_1 as the closure of the following configurations:

$$X_1 = \text{cl}\{\#^\infty s_1 w_1\ \#^{2^2-2}\ s_2 w_2\ \dots\ s_{k-1} w_{k-1}\ \#^{2^k-k}\ s_k w_k\ \cdots \mid \forall k, w_k \in \mathcal{W}_k\}$$

(for ease of understanding, we highlight the base of each sparse square in red).

The construction below relies on the fact that X_1 (ie. the set of all possible base lines) is a $\mathbb{Z}$ effective subshift. Indeed, it is possible to computably enumerate all the patterns in which at least two letters of the set $\{\text{On}, \text{Off}\}$ appear, and to forbid each of these patterns that do not respect the structure of the configurations above. Furthermore, by definition each $x'_k = \inf_l f(k,l)$ is a Π_1 real number, and its upper-density Toeplitz subshift $\mathcal{T}(x'_k)$ is effective. We conclude that X_1 is an effective $\mathbb{Z}$ subshift.

4.4 Entropy of the fixed-point realization of effective $\mathbb{Z}$ subshifts as subactions of $\mathbb{Z}^2$ SFTs

We now have a $\mathbb{Z}$ subshift X_1 which can be used as a foundation in order to implement the whole sparse square layout into an actual SFT. In this section, we recall a particular method (from [3]) which transforms $\mathbb{Z}$ effective subshifts into $\mathbb{Z}^2$ SFTs. Additionally, we compute how much this method impacts the surface entropy.

The construction of *fixpoint-based tile sets* was originally introduced in [4]. One particular application of this construction, explained in [3], is the following theorem: for any $\mathbb{Z}$ effective subshift X_1, the $\mathbb{Z}^2$ lift Y_2 of X_1 (ie. the subshift whose configurations are the configurations of X_1 repeated vertically) is sofic.

It was also proved in [1] with a different method. These constructions improve the original construction of [7], which realized $\mathbb{Z}$ effective subshifts as $\mathbb{Z}^3$ sofic subshifts. We use the construction of [3] in order to prove:

▶ **Lemma 8.** *Let X_1 be a $\mathbb{Z}$ effective subshift. There exists a $\mathbb{Z}^2$ SFT X_2 composed of two superimposed layers of tilings such that:*

1. *The projection of X_2 on its first layer is the $\mathbb{Z}^2$ lift Y_2 of X_1, whose configurations are the configurations of X_1 repeated vertically.*
2. *The second layer of X_2 is composed of tilings of a fixpoint based tile set.*
3. (New) *For any p, q relatively prime,*

$$h(X_2) = h(Y_2) = 0 \qquad \text{and} \qquad h_s(X_2, \alpha) = h_s(Y_2, \alpha) = \frac{\alpha}{1+\alpha} h(X_1).$$

Proof. Points 1 and 2 of this lemma come directly from the construction of Theorem 1 in [3]. We prove point 3, and use notations from [3]. The reader should feel free to skip this proof and go directly to the next section if they are more interested in the geometrical construction.

Let X_1 be some $\mathbb{Z}$ effective subshift, and X_2 (resp. Y_2) be the SFT (resp. the sofic subshift) given by the first two points of Lemma 8. Below, we compute the entropies and the surface entropies of X_2 and Y_2.

First, we prove that $\log N_{X_2}(pn, qn) = pnh(X_1) + o(n)$.

By definition of $\mathbb{Z}$ entropy, $\log N_{X_1}(n) = nh(X_1) + o(n)$. As any configuration of Y_2 is entirely determined by a single line, one has $\log N_{Y_2}(pn, qn) = \log N_{X_1}(pn) = pnh(X_1) + o(n)$. The complexity function of X_2 is at least the contribution of its first layer, which leads to $\log N_{X_2}(pn, qn) \geq \log N_{Y_2}(pn, qn) = pnh(X_1) + o(n)$.

On the other hand, one can find an upper bound of $\log N_{X_2}(pn, qn)$ by considering the contributions of the two layers independently. As the contribution of the first layer is the contribution of Y_2, we now focus on the contribution of all the tilings obtained from the fixpoint-based tile set. Here, we use notions and notations of [3].

In the basic construction of a self-simulating tile set, each macro-tile of level i (ie. of size N_i) is entirely determined by its four macro-colors, which fit in $O(\log N_i)$ bits. In the construction used in [3] to transform $\mathbb{Z}$ effective subshifts into $\mathbb{Z}^2$ SFTs, these macro-colors contain additional data: the level of the macro-tile ($\log N_i$ bits), one segment of l_i and three segments of l_{i+1} letters from configurations of X_1, and the position in the grand-father macro-tile ($\log N_{i+2}$ bits). By taking (as in [3]) $N_i = 2^{C2^i}$ with C being a constant, $L_i = \prod_{j=0}^{i-1} N_i$ and $l_i = \log \log L_i$, we obtain that these macro-colors still fit in $O(\log N_i)$ bits.

For any n big enough, there exists i verifying $L_i \leq pn \leq L_{i+1}$ and $qn \leq L_{i+2}$ (indeed, $\lim_{i \to +\infty} L_{i+2}/L_{i+1} = +\infty$). In this context, a pattern of size $pn \times qn$ can partially cover at most four macro-tiles of level $i+2$. These macro-tiles are entirely determined by their

four macro-colors; each macro-tile of level $i+1$ entirely determines the macro-tiles of inferior levels that compose it; and by the previous paragraph each macro-color fits on $O(\log N_{i+2})$ bits (and the constant in the O does not depend on i): all these considerations imply that the number of patterns of size $pn \times qn$ on the second layer is at most polynomial in N_{i+2}.

Considering now the contribution of the two layers independently, one obtains that $\log N_{X_2}(pn, qn) \leq \log N_{Y_2} + \log poly(N_{i+2})$. We have just proved:

$$\log N_{X_2}(pn, qn) = pn\ h(X_1) + o(n) \qquad \text{and} \qquad \log N_{Y_2}(pn, qn) = pn\ h(X_1) + o(n).$$

This immediately leads to:

$$h(X_2) = \lim_{n \to +\infty} \frac{nh(X_1) + o(n)}{n^2} = 0 = h(Y_2)$$
$$h_s(X_2, p/q) = \limsup_{n \to +\infty} \frac{pnh(X_1) + o(n)}{(p+q)n} = \frac{p}{p+q} h(X_1) = h_s(Y_2, p/q). \qquad \blacktriangleleft$$

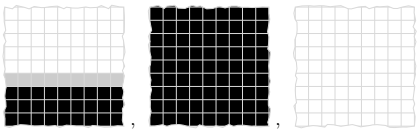
4.5 Building the sparse square layout

In this section, we use the $\mathbb{Z}$ subshift X_1 to build the whole sparse square layout.

First, apply Lemma 8 from the previous section: there exists a $\mathbb{Z}^2$ SFT X_2 with two superimposed layers, such that its first layer (**Base Layer 1**) contains all the vertical replications of configurations of X_1, and the second layer (**Computation Layer 2**) contains some embedded computations. We then create a $\mathbb{Z}^2$ SFT X_3 by superimposing a third layer to X_2, which we describe below.

This third layer (**Square Layer 3**) is itself a superimposition of several sub-layers:

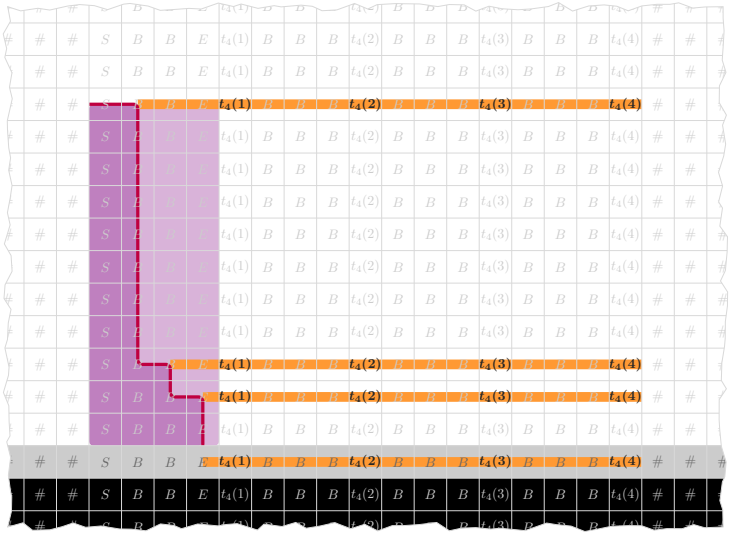
1. First, one must choose a line to be the base line on *Base Layer 1.* It is composed of the same line repeated vertically: we choose one. To do so, we add a [Layer 3a] with three colors (black, white and gray) whose only type of configurations are the following three:



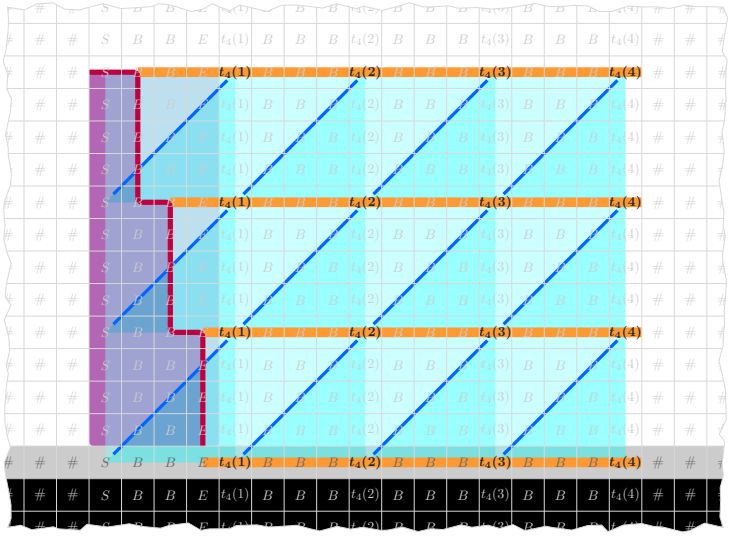
The base line will appear in gray (if it exists).

▶ **Important.** *The other markings of the Square Layer 3 (Layers 3b to 3d) will only be applied on white and gray areas. Additionally, they are not applied on areas marked by # on Base Layer 1.*

2. Add [Layer 3b] with purple construction lines (see Figure 3a). Any E marked in gray (on [Layer 3a]) starts a line at its top, and this line goes up.
   Purple lines have the ability to "start" an orange line (on [Layer 3c]). Each time they start an orange line, they move to their left (it ensures that there are at most k orange lines in the k^{th} sparse square).
   Purple lines can only end on an S.
   Because compactness might lead to surprising results (like infinitely many purple lines behaving erratically), we add colored areas below and above these lines. In these areas of color, we forbid any other purple line to exist: this ensures that there are exactly one purple line per square.
3. Add [Layer 3c] with orange construction lines (see Figure 3a). *These lines will highlight the rows of the sparse squares.* They can only be started by the purple line at their left, and end just before a # on the right. Additionally, an E colored in gray (on [Layer 3a]) must start an orange line.



(a) Base Layer 1 and 3a–3c. Layer 3b ensures there are at most 4 orange lines on Layer 3c.



(b) Base Layer 1 and 3a–3d. Layer 3d enforces the positions of the orange lines.

Figure 3 Behavior of the construction for a pattern which contains the 4^{th} sparse square. These figures highlight the effect of each layer. (Positions of the sparse square are highlighted for the convenience of the reader).

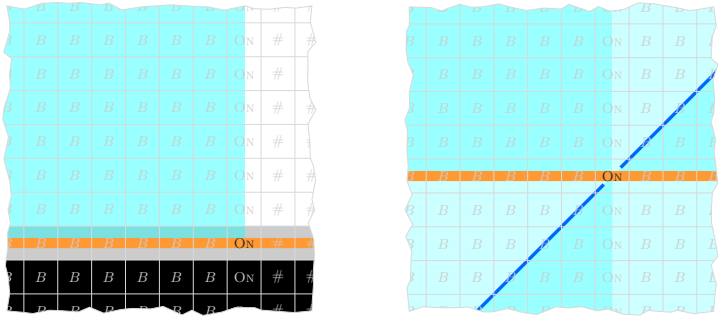


Figure 4 Two examples of degenerate configurations.

(In the figure, we highlight in bold the $t_k(i)$ marked by an orange line. These lines mark the future rows of the sparse squares, and these letters On, Off marked by an orange line will be at the exact non-blank positions of the sparse squares layout.)

4. Add [Layer 3d] with blue construction lines (see Figure 3b). These lines are diagonals. Each $t_k(i)$ colored in orange (on [Layer 3c]) and not in gray (on [Layer 3a]) should start a line that goes diagonally down and left. Additionally, these lines should end either on a letter S (colored in purple on [Layer 3b]), or on the next column marked with $t_k(i-1)$ at a position which is colored in orange (on [Layer 3c]).
   We also impose with colored areas that at each horizontal position, there can be only one blue line between two orange lines. These blue lines ensure the structure of the sparse squares, and constrain the behavior of degenerate configurations (see Figure 4 for examples of these degenerate configurations).

4.6 Contributing to the entropy

The subshift X_3 reflects the sparse square layout in the following way: by considering letters $t_k(i)$ marked in orange on [Layer 3c], we obtain a set of positions that respects the layout.

The final step of the construction consists in adding a fourth layer (**Entropy Layer 4**) to X_3 with *free letters* (ie. letters in the alphabet $1, \dots, 2^e$) and *free bits* (ie. bits in $\{0,1\}$) to obtain an SFT X with the right surface entropy. More precisely:

- *Free letters.* At each site in the sparse square layout (ie. for each position marked by On or Off on *Base Layer 1*, and which is marked in orange on [Layer 3c], we add a free letter which varies in $\{1, \dots, 2^e\}$.
- *Free bits.* At each activated position in the sparse square layout (ie. for each position marked by On on *Base Layer 1*, and which is marked in orange on [Layer 3c]), we add a free bit which varies in $\{0,1\}$.

Let X be the SFT composed of these four superimposed layers. We prove in Subsection 4.7:

▶ **Lemma 9.** *For any $\alpha = p/q \in \mathbb{Q}^+$ (for p and q relatively prime), and for any $n \in \mathbb{N}$, if k is the integer such that $k(k-1)+1 \leq \min(p,q)n < (k+1)k+1$, then the complexity function of the SFT X (defined in Section 4) behaves as follows:*

$$\min(p,q)n\,(e + x'_k) + o(n) \leq \log N_X(pn, qn) \leq \min(p,q)n \left(e + \sup_{\log n \leq i} x'_i\right) + o(n).$$

In particular, this first implies that $h(X) = 0$, then that:

$$h_s(X, \alpha) = \limsup_{n \to +\infty} \frac{\log N_X(pn, qn)}{(p+q)n} = \frac{\min(p,q)}{p+q}\left(e + \limsup_{n \to +\infty} x'_n\right) = \frac{\min(p,q)}{p+q} x$$

which concludes the proof of Theorem 5. ◀

4.7 Computation of the complexity function

This section solely proves Lemma 9. The reader not interested in the precise computation of the complexity function N_X should feel free to skip these pages.

Let $\alpha = p/q$ (for p and q relatively prime) be a positive rational number. In this section, we compute the complexity function $N_X(pn, qn)$ of the SFT X introduced in the previous section. We recall that $x = e + x'$ is a Π_3 real number with $x' \in \Pi_3 \cap [0,1)$, given by $x' = \limsup_i \inf_l f(i,l)$, and that we defined $x'_i = \inf_l f(i,l)$.

We now prove Lemma 9. In this statement, the value $k(k-1)+1$ corresponds to the length of the edges of the k^{th} sparse square.

Proof. As we already proved the lower bound (see Lemma 6), we focus on the upper bound. Here is the structure of the proof:

1. We first consider the contribution of "degenerate configurations" (ie. configurations that do not respect the structure of the sparse square layout: these are obtained when defining X_1 as a closure) to $\log N_X(pn, qn)$, and prove that they contribute only as $o(n)$ (subsection 4.7.1).
2. Then, we consider how many sparse squares can appear simultaneously in a pattern of size $pn \times qn$ (subsection 4.7.2), and:
   a. We provide an upper bound for the sparse square that appear simultaneously in a pattern of size $pn \times qn$ (subsection 4.7.3). As these squares have very low indices, they only contribute as $o(n)$.
   b. We provide an upper bound for the sparse squares that appear alone (subsection 4.7.4; they contribute with the significant term of the upper bound). To obtain it, we bound the number of positions that can appear simultaneously (in a pattern of size $pn \times qn$) of such a sparse square.

Before we begin, in the whole proof we denote by k the integer such that $k(k-1)+1 \leq \min(p, q)n < (k+1)k+1$. (As mentioned above, $k(k-1)+1$ is the length of the edges of the k^{th} sparse square). In the rest of the proof, we look for an upper bound of $\log N_X(pn, qn)$.

4.7.1 Contribution of degenerate configurations

We call *degenerate* the configurations that do not respect the structure of the sparse square layout. There are two possible sources for these configurations: some were obtained when X_1 was defined as a closure (first kind); and some are obtained if *Base Layer 1* respects the structure of a line in the sparse square layout, but [Layer 3a] does not choose a base line for the squares to sit on (second kind).

▷ Claim 10. The contribution of these configurations to $\log N_X(pn, qn)$ is $o(n)$.

Proof. First, we consider the case of a degenerate configurations of the first kind: *Base Layer 1* is not a base line. The number of such patterns only depends on the position of the gray line on [Layer 3a] (if it exists at all), of the position of the letter On or Off in the pattern, etc... There exists at most one varying bit in this pattern (there can be at most one letter On or Off colored in orange in such a degenerate configuration, because of the colored areas of the blue line on [Layer 3d]), which only multiplies the number of patterns by two.

All these patterns depend on finitely many parameters that range from 0 to $\max(p, q)n$. These configurations contribute polynomially in n to $N_X(pn, qn)$, so they contribute $O(\log n)$ to $\log N_X(pn, qn)$.

Consider then the case of degenerate configurations of the second kind: [Layer 3a] does not have a gray line (ie. it does not choose a line for the squares to sit on). If [Layer 3a] is a full black configuration, there are no markings at all on *Square Layer 3* or *Entropy Layer 4*, and the number of patterns depends only on *Base Layer 1*. If [Layer 3a] is a full white configuration, then purple lines on [Layer 3b] can only go up, and never go left: if they did, there would be an orange line on [Layer 3c], which is impossible because of the blue areas of color on [Layer 3d]. This implies that the number of patterns again only depends on *Base Layer 1* (and by Lemma 8, it contributes $o(n)$ to $\log N_X(pn, qn)$).

All in all, degenerate configurations contribute $o(n)$ to $\log N_X(pn, qn)$. ◁

In the rest of the proof, we assume that we consider non-degenerate configurations, ie. configurations that respect the sparse square layout.

We also assume that the gray line of [Layer 3a] is fixed at the bottom of the pattern of size $pn \times qn$ that we consider: this maximizes the number of free bits/free letters in the pattern. Furthermore, we will happily forget to count the different horizontal positions of the squares in the patterns.

Indeed, all these other patterns can be taken into account by translating the figure/varying some parameters which range between 0 and $\max(p,q)n$: these considerations only multiply the complexity function by a polynomial in n, or in other words only add a $o(n)$ to $\log N_X(pn, qn)$.

We also define the following set of words for any $l \in \mathbb{N}$ and $y \in [0,1)$ (usually $\mathcal{T}(y)$ is defined on the alphabet $\{0,1\}$ rather than $\{\text{On}, \text{Off}\}$; otherwise, there is no difference):

$$\mathcal{T}_l(y) = \{t_l \in \{\text{On}, \text{Off}\}^l : \ t_l \text{ is a subword of } \mathcal{T}(y)\}.$$

As we are looking at non-degenerate configurations, [Layers 3b–3d] (construction layers) are fixed by the line chosen by [Layer 3a]. This means that we can now focus the different Toeplitz words written on *Base Layer 1*, and on the contribution from the free bits and free letters that appear on *Entropy Layer 4*. To count these patterns, we mainly have to compute how many free bits/letters can fit in a pattern at the same time.

4.7.2 Which sparse squares can only appear alone in a pattern?

To find an upper bound of the complexity function, we ask the following question: how many sparse squares can fit in a pattern of size $pn \times qn$?

▷ Claim 11. For $n \geq p$, if at least two different sparse squares appear (maybe partially) in a pattern of size $pn \times qn$, then their indices are below $2 \log n$.

Proof. Assume that a range of squares from i to j, with $i < j$, appear (maybe partially) in a pattern of size $pn \times qn$. Then the horizontal space before the square of index j is entirely contained in the pattern, ie $2^j < pn$. Then for any $n \geq p$, one has $j \leq 2 \log n$. ◁

Reciprocally,

▷ Claim 12. If a sparse square can only appear alone in a pattern of size $pn \times qn$, then its index is greater than $\log n$.

Proof. Assume that a square of index j can "only" appear alone in a pattern of size $pn \times qn$. This means that the space before the sparse square, and the space after the sparse square, are bigger than the horizontal size of the pattern pn. In other words, $2^j \geq pn$, which becomes $j \geq \log n + \log p \geq \log n$. ◁

4.7.3 Contribution of simultaneously appearing sparse squares

▷ Claim 13. The sparse squares that can appear grouped with others contribute as $M_1 = o(n)$ to $\log N_X(pn, qn)$.

Proof. Assume that a range (between i and j, $i < j$) of sparse squares appear (partially) in a pattern of size $pn \times qn$. For n big enough, one has $j \leq 2 \log n$ by Claim 11. Additionally, because we are interested in an upper bound of $\log N_X(pn, qn)$, we can freely assume that all the free bits of the sparse squares of index i and j appear in this pattern.

If $C_{i,j}$ denotes the contribution to $N_X(pn, qn)$ of this slice of squares between i and j, then an upper bound on $C_{i,j}$ is (we count all the Toeplitz subwords written in the squares on *Base Layer 1*, and then their free bits and free letters on *Entropy Layer 4*):

$$\begin{aligned}
C_{i,j} &\leq \sum_{t_i \in \mathcal{T}_i(x'_i), \dots, t_j \in \mathcal{T}_j(x'_j)} \exp_2\left(\sum_{r=i}^{j} r|t_r|_{\text{ON}} + e\sum_{t=i}^{j} r^2\right) \\
&\leq \sum_{t_i \in \mathcal{T}_i(x'_i), \dots, t_j \in \mathcal{T}_j(x'_j)} \exp_2\left(\sum_{r=1}^{j} (r|t_r|_{\text{ON}} + er^2)\right) \\
&\leq \sum_{t_i \in \mathcal{T}_i(x'_i), \dots, t_j \in \mathcal{T}_j(x'_j)} \exp_2(j^3(1+e)) \\
&\leq \sum_{t_1, \dots, t_j \in \{\text{ON},\text{OFF}\}^{1+\dots+j}} \exp_2((2\log n)^3(1+e)) \\
&\leq \exp_2(j^2 + o(j^2)) \exp_2((2\log n)^3(1+e)) \\
&\leq \exp_2(O((\log n)^3)).
\end{aligned}$$

As there are less than $(2\log n)^2$ different tuples of $i, j \leq 2\log n$, the contribution M_1 of these sparse squares to $\log N_X(pn, qn)$ verifies:

$$\begin{aligned}
M_1 \leq \log\left(\sum_{i<j\leq 2\log n} C_{i,j}\right) &\leq \log\left[(4(\log n)^2)\exp_2(O((\log n)^3))\right] \\
&\leq \log \exp_2(O((\log n)^3))) \\
&\leq o(n).
\end{aligned}$$

◁

4.7.4 Contribution of the other sparse squares

The other sparse squares can only appear alone (maybe partially) in a pattern of size $pn \times qn$. We distinguish two cases for them:

- The sparse squares of indices $i \leq k$. As they can fit entirely in a pattern of size $pn \times qn$ (recall that $k(k+1) - 1 \leq \min(p, q)n \leq (k+1)k + 1$), we assume they do (this maximizes the number of free bits that appear simultaneously).
- The sparse squares of indices $i > k$. They can only fit partially in a pattern of size $pn \times qn$, and we need to "count" the number of their free bits/letters that can appear simultaneously.

Contribution of the sparse squares of index $i \leq k$.

▷ Claim 14. The sparse squares that can only appear alone in a pattern of size $pn \times qn$, and of indices $i \leq k$, contribute to $\log N_X(pn, qn)$ with a term:

$$M_2 \leq \min(p, q)n\left(e + \max_{\log n \leq i \leq k} x'_i\right) + o(n).$$

Proof. As $\#\mathcal{T}_i(x'_i) \leq 2^i$, each of these squares contribute with a term (again, we count the words of $\mathcal{T}_i(x'_i)$ on *Base Layer 1*, and free letters and free bits on *Entropy Layer 4*):

$$\begin{aligned} C_i &\leq (2^e)^{i^2} \times \sum_{t_i \in \mathcal{T}_i(x'_i)} \exp_2(i|t_i|_{\text{ON}}) \\ &\leq \exp_2(ei^2) \times \exp_2(i) \exp_2(i^2 x'_i + O(i)) \\ &\leq \exp_2(i^2(e + x'_i) + O(i)). \end{aligned}$$

By Claim 12, such a square must be of index $\geq \log n$. This implies that all these squares contribute to $\log N_X(pn, qn)$ with a term:

$$\begin{aligned} M_2 &\leq \log \left(\sum_{i=\log n}^{k} C_i \right) \\ &\leq \log \left(\sum_{i=\log n}^{k} \exp_2(i^2(e + x'_i) + O(i)) \right) \\ &\leq \log \left(k \exp_2 \left(k^2 \left(e + \max_{\log n \leq i \leq k} x'_i \right) + O(k) \right) \right) \\ &\leq \log \exp_2 \left(k^2 \left(e + \max_{\log n \leq i \leq k} x'_i \right) + O(k) \right) \\ &\leq k^2 \left(e + \max_{\log n \leq i \leq k} x'_i \right) + O(k) \\ &\leq \min(p, q) n \left(e + \max_{\log n \leq i \leq k} x'_i \right) + o(n). \end{aligned}$$

◁

Contribution of the sparse squares of index $i \geq k$. Finally, we consider the sparse squares of indices $i \geq k$: these square only appear partially in a pattern of size $pn \times qn$. They contribute significantly to $\log N_X(pn, qn)$, as explained in the following claim:

▷ Claim 15. The sparse squares of indices $i \geq k$ contribute to $\log N_X(pn, qn)$ with a term:

$$M_3 \leq \min(p, q) n \left(e + \max_{k \leq i \leq \max(p,q)(k+1)^2} x'_i \right) + o(n).$$

Proof. This proof is organized as follows:

1. We provide an upper bound on the contribution C_i of the i^{th} sparse square to $N_X(pn, qn)$, for $i \geq k$, according to the number of free bits/free letters of these squares that can appear simultaneously in a pattern of size $pn \times qn$.
2. Then, we provide an upper bound on the contribution M_3 of all these sparse squares of indices $i \geq k$ to $\log N_X(pn, qn)$.
3. By studying the variations of two functions $h : \mathbb{N} \mapsto \mathbb{N}$ and $v : \mathbb{N} \mapsto \mathbb{N}$ we prove that for any $i \geq k$ the number of simultaneously appearing free bits/free letters of the i^{th} sparse square is $h(i)v(i) \leq \min(p, q)n + o(n)$. This will conclude the proof.

Contribution C_i of the sparse square of index i. First, we need to answer the following question: how many bits can appear simultaneously in a pattern of size $pn \times qn$? Recall that there are exactly i bits in the square of index i per row (and per column). If $h(i)$ denotes the number of horizontal bits that can appear simultaneously in a slice of width pn (and height 1), and $v(i)$ the number of vertical bits in a slice of height qn (and width 1), then:

$$h(i) = \min\left(i, \left\lfloor \frac{pn-1}{i} \right\rfloor + 1\right) \qquad v(i) = \min\left(i, \left\lfloor \frac{qn-1}{i} \right\rfloor + 1\right).$$

h and v are eventually decreasing, and $\lim_{i\to+\infty} h(i) = \lim_{i\to+\infty} v(i) = 1$. There exists an integer $J_k \leq \max(p,q)(k+1)^2$ such that for any $i \geq J_k$, one has $v(i) = h(i) = 1$.

We now count free letters and free bits to compute an upper bound on the contribution C_i of a square of index $i \geq k$. As opposed to the previous cases, in which the squares appeared entirely in the pattern of size $pn \times qn$, here we can only see at once $h(i)$ different column: but thanks to the use of Toeplitz sequences, which have a very uniform distribution, the density of a Toeplitz subword of size $h(i)$ is still less than or equal to $h(i)x'_i + O(1)$:

$$\begin{aligned} C_i &\leq \sum_{t_i \in \mathcal{T}(x'_i)h(i)} \exp_2\left[h(i)v(i)e + v(i)|t_i|_{0N})\right] \\ &\leq \exp_2\left[h(i)v(i)(e+x'_i) + O(v(i))\right]. \end{aligned}$$

Additionally, as $v(i) \leq \frac{qn-1}{i} + 1$, $i \geq k$ and $k = \Theta(\sqrt{n})$, one has $O(v(i)) = O(\sqrt{n})$.

Contribution M_3 of the sparse square of index $i \geq k$. This implies that the contribution M_3 of all these squares of indices $i \geq k$ is:

$$\begin{aligned} M_3 &\leq \log\left(\sum_{i=k}^{J_k} \exp_2[h(i)v(i)(e+x'_i) + o(h(i)v(i))]\right) \\ &\leq \log\left(J_k \exp_2\left(\left[\max_{k\leq i\leq J_k} h(i)v(i) + o(h(i)v(i))\right] \times \left[e + \max_{k\leq i\leq J_k} x'_i\right]\right)\right) \\ &\leq \left[\max_{k\leq i\leq J_k} h(i)v(i) + o(h(i)v(i))\right] \times \left[e + \max_{k\leq i\leq J_k} x'_i\right] + o(n). \end{aligned}$$

Study of the product $h(i)v(i)$ for $k \geq i \geq J_k$. We now have to study the product $h(i)v(i)$ for $k \leq i \leq J_k$. Below, we will prove that $\forall k \leq i \leq J_k, h(i)v(i) \leq \min(p,q)n + O(\sqrt{n})$.

Without any loss of generality, assume $q \geq p$. We prove that:

$$h(i)v(i) \leq pn + O(\sqrt{n}), \qquad \text{i.e.} \qquad M_3 \leq pn\left(e + \max_{k\leq i\leq \max(p,q)(k+1)^2} x'_i\right) + o(n).$$

As $p = \min(p,q)$, one has $h(i) = \left\lfloor \frac{pn-1}{i} \right\rfloor + 1$ and $v(i) = \min\left(i, \left\lfloor \frac{qn-1}{i} \right\rfloor + 1\right)$. For the first values of i, h is an increasing function, and it then decreases for i large enough: below, we study these variations and conclude about the product $h(i)v(i)$.

- For any $i \leq \lfloor\sqrt{qn-1}\rfloor$, one has $\left\lfloor \frac{qn-1}{i} \right\rfloor + 1 \geq i$ (which implies $v(i) = i$). Indeed,
$$\begin{aligned} \left\lfloor \frac{qn-1}{i} \right\rfloor + 1 &\geq \left\lfloor \sqrt{qn-1} \right\rfloor + 1 \\ &\geq i. \end{aligned}$$
- For any $k \leq i \leq \lfloor\sqrt{qn-1}\rfloor$, one has $h(i)v(i) \leq pn + O(\sqrt{n})$. Indeed,
$$\begin{aligned} h(i)v(i) = h(i)i &\leq \left(\left\lfloor \frac{pn-1}{i} \right\rfloor + 1\right) i \\ &\leq i\left\lfloor \frac{pn-1}{i} \right\rfloor + i \\ &\leq pn - 1 + i \\ &\leq pn + O(\sqrt{n}). \end{aligned}$$

- For any $i \geq \lfloor \sqrt{qn-1} \rfloor + 1$, one has $i \geq \left\lfloor \frac{qn-1}{i} \right\rfloor + 1$ (which implies $v(i) = \left\lfloor \frac{qn-1}{i} \right\rfloor + 1$). Indeed,
$$\begin{aligned}\left\lfloor \frac{qn-1}{i} \right\rfloor + 1 &\leq \left\lfloor \frac{qn-1}{\sqrt{qn-1}} \right\rfloor + 1 \\ &\leq \left\lfloor \sqrt{qn-1} \right\rfloor + 1 \\ &\leq i.\end{aligned}$$
- For any $\lfloor \sqrt{qn-1} \rfloor + 1 \leq i \leq J_k$, one has $h(i)v(i) \leq pn + O(\sqrt{n})$. Indeed,
$$\begin{aligned}h(i)v(i) = h(i)\left(\left\lfloor \frac{qn-1}{i} \right\rfloor + 1\right) &= \left\lfloor \frac{pn-1}{i} + 1 \right\rfloor \left\lfloor \frac{qn-1}{i} + 1 \right\rfloor \\ &\leq \left\lfloor \frac{pn-1}{\sqrt{qn-1}} + 1 \right\rfloor \left\lfloor \frac{qn-1}{\sqrt{qn-1}} + 1 \right\rfloor \\ &\leq \left\lfloor \frac{pn-1}{\sqrt{qn-1}} + 1 \right\rfloor \left(\sqrt{qn-1} + 1\right) \\ &\leq pn + O(\sqrt{n}).\end{aligned}$$
With all of these computations, we conclude that
$$\max_{k \leq i \leq J_k} h(i)v(i) \leq pn + o(n).$$
In the case $p \geq q$, the computations are completely symmetric and one obtains:
$$\max_{k \leq i \leq J_k} h(i)v(i) \leq qn + O(\sqrt{n}).$$ ◁

4.7.5 End of the proof

We can now conclude the proof of Lemma 9 about the bounds of $\log N_X(pn, qn)$:

$$\min(p,q)n\,(e + x'_k) + o(n) \leq \log N_X(pn, qn) \leq \min(p,q)n\left(e + \sup_{\log n \leq i} x'_i\right) + o(n).$$

The lower bound comes from Lemma 6. Indeed, if $k(k-1)+1 \leq \min(p,q)n \leq k(k+1)+1$, then $k^2 = \min(p,q)n + o(n)$, which leads to the desired lower bound. In order to compute the upper bound, we can count the contributions of the different layers independently.

Base Layer 1 and *Computation Layer 2* (the 1D subshift composed of the base line repeated vertically, and the tilings obtained with self-simulating tile sets) do not contribute to the surface entropy by Lemma 8. Indeed, the 1D entropy of the base line is $h(X_1) = 0$. In other words, these two layers add $o(n)$ to $\log N_X(pn, qn)$.

Degenerate configurations, *Square Layer 3*, along with considerations on the different shifts of the configurations, also contribute as $o(n)$ to $\log N_X(pn, qn)$ (they indeed contribute polynomially in n to $N_X(pn, qn)$, see the remark at the end of "Contribution of degenerate configurations"). Finally, in the previous sections, we provided 3 quantities M_1, M_2, M_3 whose sum is greater than the contribution of *Entropy Layer 4* (and which take into account the different written words on non-degenerate configurations of *Base Layer 1*).

With these considerations, we conclude that:

$$\begin{aligned}\log N_X(pn, qn) &\leq o(n) + M_1 + M_2 + M_3 \\ &\leq o(n) + o(n) + \min(p,q)n\left(e + \max_{\log n \leq i \leq \max(p,q)(k+1)^2} x'_i\right) + o(n) \\ &\leq \min(p,q)n\left(e + \max_{\log n \leq i \leq \max(p,q)(k+1)^2} x'_i\right) + o(n).\end{aligned}$$ ◀

5 Conclusive remarks and open questions

Many questions remain open about the notion of surface entropy.

Computational behavior of the definition in [14]. The definition of *surface entropy* used in this paper differs from the original notion of surface entropy in Pace's thesis [14], which was: for any eccentricity $\alpha \in \mathbb{R}^+$,

$$h_s(X,\alpha) = \sup_{\substack{(x_n,y_n)\in(\mathbb{N}^2)^{\mathbb{N}}:\\ x_n,y_n\to+\infty,\ \frac{x_n}{y_n}\to\alpha}} \limsup_{n\to+\infty} \frac{\log N_X(x_n,y_n) - x_n y_n h(X)}{x_n+y_n}.$$

This definition was chosen in [14] because it provides a unified approach for rational (ie. $\alpha \in \mathbb{Q}^+$) and irrational eccentricities. However, we are currently unsure of how the supremum over all sequences impacts the computational characterization of surface entropies. Our construction still realizes any Π_3 surface entropy with the definition of [14], but surface entropies may not be Π_3 real numbers anymore. For all we know, they may not be at any level of the arithmetical hierarchy.

Equivalence between the two definitions. Furthermore, as we modified [14]'s definition of surface entropy, a natural question is whether our new definition coincides with it in the case of rational eccentricities. In other words, can the supremum over all sequences be removed when the eccentricity is a rational number. We are not sure of the answer at the time of writing.

Arbitrary topological entropy with an arbitrary surface entropy. Finally, in the main section of this paper, we created SFTs with zero topological entropy and any Π_3 surface entropy. It was proved in [8] that the class of entropies of $\mathbb{Z}^2$ SFTs is exactly the class of Π_1 real numbers. This led us to wonder whether we could create a family of $\mathbb{Z}^2$ SFTs with arbitrary Π_1 entropy and arbitrary Π_3 surface entropy.
As we do not know the surface entropies of the main constructions of Π_1 entropies, we could not answer this problem with the straightforward solution (ie. a Cartesian product of our construction with one for Π_1 entropies).

References

1 Nathalie Aubrun and Mathieu Sablik. Simulation of effective subshifts by two-dimensional subshifts of finite type. *Acta Applicandae Mathematicae*, 126(1):35–63, 2013. doi:10.1007/s10440-013-9808-5.

2 Robert Berger. *The Undecidability of the Domino Problem*. Number 66 in Memoirs of the American Mathematical Society. The American Mathematical Society, 1966.

3 Bruno Durand, Andrei Romashchenko, and Alexander Shen. Effective Closed Subshifts in 1D Can Be Implemented in 2D. In *Fields of Logic and Computation*, number 6300 in Lecture Notes in Computer Science, pages 208–226. Springer, 2010. doi:10.1007/978-3-642-15025-8_12.

4 Bruno Durand, Andrei Romashchenko, and Alexander Shen. Fixed-point tile sets and their applications. *Journal of Computer and System Sciences*, 78(3):731–764, May 2012. doi:10.1016/j.jcss.2011.11.001.

5 Yuri Gurevich and I Koryakov. Remarks on Berger's paper on the domino problem. *Siberian Math. Journal*, pages 319–320, 1972.

6 David Harel. Recurring Dominoes: Making the Highly Undecidable Highly Understandable. *Annals of Discrete Mathematics*, 24:51–72, 1985.

7 Michael Hochman. On the dynamics and recursive properties of multidimensional symbolic systems. *Inventiones Mathematicae*, 176(1):2009, April 2009.

8 Michael Hochman and Tom Meyerovitch. A characterization of the entropies of multidimensional shifts of finite type. *Annals of Mathematics*, 171(3):2011–2038, May 2010. doi:10.4007/annals.2010.171.2011.

9 Konrad Jacobs and Michael Keane. 0-1-sequences of Toeplitz type. *Zeitschrift für Wahrscheinlichkeitstheorie und Verwandte Gebiete*, 13(2):123–131, 1969. doi:10.1007/BF00537017.

10 Emmanuel Jeandel and Pascal Vanier. Characterizations of periods of multidimensional shifts. *Ergodic Theory and Dynamical Systems*, 35(2):431–460, 2015. doi:10.1017/etds.2013.60.

11 S.C. Kleene. *Two Papers on the Predicate Calculus*, chapter Finite Axiomatizability of Theories in the Predicate Calculus Using Additional Predicate Symbols, pages 31–71. Number 10 in Memoirs of the American Mathematical Society. American Mathematical Society, 1952.

12 Douglas A. Lind and Brian Marcus. *An Introduction to Symbolic Dynamics and Coding*. Cambridge University Press, New York, NY, USA, 1995.

13 Tom Meyerovitch. Growth-type invariants for $\mathbb{Z}^d$ subshifts of finite type and arithmetical classes of real numbers. *Inventiones Mathematicae*, 184(3), 2010. doi:10.1007/s00222-010-0296-1.

14 Dennis Pace. *Surface Entropy of Shifts of Finite Type*. PhD thesis, University of Denver, 2018.

15 Xizhong Zheng and Klaus Weihrauch. Arithmetical hierarchy of real numbers. In *Mathematical Foundations of Computer Science (MFCS)*, pages 23–33, 1999. doi:10.1007/3-540-48340-3_3.

Optimal Transformations of Games and Automata Using Muller Conditions

Antonio Casares
LaBRI, Université de Bordeaux, France

Thomas Colcombet
CNRS, IRIF, Université de Paris, France

Nathanaël Fijalkow
CNRS, LaBRI, Université de Bordeaux, France
The Alan Turing Institute of Data Science, London, UK

Abstract

We consider the following question: given an automaton or a game with a Muller condition, how can we efficiently construct an equivalent one with a parity condition? There are several examples of such transformations in the literature, including in the determinisation of Büchi automata.

We define a new transformation called the alternating cycle decomposition, inspired and extending Zielonka's construction. Our transformation operates on transition systems, encompassing both automata and games, and preserves semantic properties through the existence of a locally bijective morphism. We show a strong optimality result: the obtained parity transition system is minimal both in number of states and number of priorities with respect to locally bijective morphisms.

We give two applications: the first is related to the determinisation of Büchi automata, and the second is to give crisp characterisations on the possibility of relabelling automata with different acceptance conditions.

2012 ACM Subject Classification Theory of computation → Automata over infinite objects

Keywords and phrases Automata over infinite words, Omega regular languages, Determinisation of automata

Digital Object Identifier 10.4230/LIPIcs.ICALP.2021.123

Category Track B: Automata, Logic, Semantics, and Theory of Programming

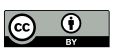
Related Version *Full Version*: `https://arxiv.org/pdf/2011.13041.pdf` [4]

Funding *Thomas Colcombet*: Supported by the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreement No.670624) and the DeLTA ANR project (ANR-16-CE40-0007).

Acknowledgements We want to thank Philipp Meyer and Salomon Sickert for their comments and for spotting a mistake on a previous version of this paper.

1 Introduction

Games and automata form the theoretical basis for the verification and synthesis of reactive systems; we refer to the recent Handbook [5] for a broad exposition of this research area, in particular Chapters 2 and 27. A milestone objective is the synthesis of reactive systems specified in *Linear Temporal Logic* (LTL). The original approach of Pnueli and Rosner [24] using automata and games devised more than four decades ago is today at the heart of the state of the art synthesis tools [8, 16, 20, 21]. The bottleneck is the determinisation of Büchi automata: given a non-deterministic Büchi automaton, construct an equivalent parity automaton. This problem has a long history; it was originally solved by McNaughton [18], and the first asymptotically optimal construction is due to Safra [25], see also [15] for a recent



exposition. Most of the recent theoretical and practical solutions of this problem are based on the construction of Piterman [23]. Schewe's [26] enlightening perspective on this construction is to decompose it into two steps: first construct a deterministic Muller automaton, and then transform it into an equivalent deterministic parity automaton. Piterman and Schewe's determinisation procedure is one of many examples of constructions using as an intermediate step (subclasses of) Muller conditions before transforming them into parity conditions, either working with automata models or games models.

The objective of this work is to focus on this particular step and study transformations from Muller to parity. We work with general transition systems to seamlessly encompass both automata and games models.

There are several existing constructions transforming subclasses of Muller conditions to parity. The first is the Latest Appearance Record (LAR) [9], which applies to all Muller conditions. It was proved to be optimal in the worst case [17]: *there exists* a family of Muller automata for which the obtained parity automata are minimal. Many refinements of the LAR have been constructed for subclasses of Muller conditions, *e.g.* [17, 13].

The starting point of our work is the notion of a Zielonka tree of a Muller condition, which was introduced in [30] and shown to capture the exact memory requirements of Muller games [7]. In the long version of [7], it implicitly appears that the Zielonka tree of a Muller condition can be used to construct a parity automaton recognising this Muller condition. Our first observation is to show a *strong* optimality result: *for all* Muller conditions, the parity automaton obtained from the Zielonka tree of a Muller condition is minimal both in the number of states and in the number of priorities. This result has also been obtained in the independent unpublished work [19]. This optimality result is much stronger than the worst case optimality result of the LAR transformation; in essence, it shows that the Zielonka tree of a Muller condition precisely captures the properties of the Muller condition, whereas for instance the LAR only depends on the number of colours.

Our first insight is to note that all existing constructions, including the one based on Zielonka trees, only consider the Muller condition but do not take into account the structure of the underlying transition system. In other words, all transformations work at the level of conditions: they transform a Muller condition into a parity condition, and ignore the interplay between the condition and the transition structure.

Our main contribution is to construct a new transformation called the alternating cycle decomposition (ACD) which captures this interplay: the ACD transforms a Muller transition system $\mathcal{T}$ into a parity transition system $\mathcal{P}_{\mathcal{ACD}(\mathcal{T})}$, extending Zielonka trees by considering the alternation of accepting and rejecting cycles in $\mathcal{T}$.

Our second insight is to introduce the notion of locally bijective morphisms to capture the notion of a "transformation", preserving many natural semantic properties (such as language equivalence, being deterministic, unambiguous, or good for game in the context of automata, and the winner for games). We use this notion to state and prove a strong optimality result for the ACD transformation: $\mathcal{P}_{\mathcal{ACD}(\mathcal{T})}$ is minimal both in the number of states and in the number of priorities amongst parity transition systems admitting a locally bijective morphism into $\mathcal{T}$.

We present two applications. The first is an improvement in the determinisation of Büchi automata: the second step of the Piterman and Schewe construction is a locally bijective transformation of some deterministic Muller automaton into a deterministic parity automaton; we show that our ACD transformation yields in all cases smaller (and in some sense minimal) automata, and in many cases strictly smaller. The second application is a set of crisp characterisations for relabelling transition systems with different classes of

acceptance conditions: for instance, given a transition system with a Rabin condition, does there exist a parity condition on the same structure yielding an equivalent transition system? This unifies and extends results from [1, 30].

The outline of the paper follows the narration of this introduction. We show in Section 3 how the Zielonka tree yields a parity automaton recognising the Muller condition, inducing a transformation at the level of conditions. We then lift this transformation from conditions to transition systems: we introduce the alternating cycle decomposition and its transformation in Section 4. Our two applications are discussed in Section 5.

2 Notations and definitions

The symbol ω denotes the ordered set of non-negative integers. For $i, j \in \omega$, $i \leq j$, the notation $[i, j]$ stands for $\{i, i+1, \dots, j-1, j\}$. For a set Σ, a *word* over Σ is a sequence of elements from Σ. The length of a word u is $|u|$. The set of words of finite length (resp. of length ω) over Σ will be written Σ^* (resp. Σ^ω). We let $\Sigma^\infty = \Sigma^* \cup \Sigma^\omega$. For a word $u \in \Sigma^\infty$ we write u_i to represent the i-th letter of u. If $u = v \cdot w$ for $v \in \Sigma^*, u, w \in \Sigma^\infty$, we say that v is a *prefix* of u and we write $v \sqsubseteq u$ (it induces a partial order on Σ^*). For a finite word $u \in \Sigma^*$ we write $\mathit{First}(u) = u_1$ and $\mathit{Last}(u) = u_{|u|}$. For a word $u \in \Sigma^\infty$, we let $\mathit{Inf}(u) = \{a \in \Sigma \ : \ u_i = a \text{ for infinitely many } i \in \omega\}$ and $\mathit{Occ}(u) = \{a \in \Sigma \ : \ \exists i \in \omega \text{ such that } u_i = a\}$. Given a map $\alpha : A \to B$, we implicitly extend α to words component-wise, i.e., $\alpha : A^\infty \to B^\infty$ will be defined as $\alpha(a_1 a_2 \dots) = \alpha(a_1)\alpha(a_2)\dots$. A *directed graph* is a tuple $(V, E, \mathit{Source}, \mathit{Target})$ where V is a set of vertices, E a set of edges and $\mathit{Source}, \mathit{Target} : E \to V$ are maps indicating the source and target for each edge. A *path* is a word $\varrho \in E^*$ such that $\mathit{Source}(\varrho_{i+1}) = \mathit{Target}(\varrho_i)$ for $i < |\varrho|$. A graph is *strongly connected* if there is a path connecting each pair of vertices. A *subgraph* of $(V, E, \mathit{Source}, \mathit{Target})$ is a graph $(V', E', \mathit{Source}', \mathit{Target}')$ such that $V' \subseteq V$, $E' \subseteq E$ and Source' and Target' are the restriction to E' of Source and Target, respectively. A *strongly connected component* is a maximal strongly connected subgraph. For a subset of vertices $A \subseteq V$ we write: $\mathit{In}(A) = \{e \in E \ : \ \mathit{Target}(e) \in A\}$ and $\mathit{Out}(A) = \{e \in E \ : \ \mathit{Source}(e) \in A\}$.

Transition systems. A *transition system graph* $\mathcal{T}_G = (V, E, \mathit{Source}, \mathit{Target}, I_0)$ is a directed graph with a non-empty set of initial vertices $I_0 \subseteq V$. We will also refer to vertices and edges as *states* and *transitions*, respectively. We will suppose that every vertex has at least one outgoing edge. A *transition system* $\mathcal{T}$ is obtained from a transition system graph $\mathcal{T}_G$ by adding:

- A function $\gamma : E \to \Gamma$. The set Γ will be called a *set of colours* and the function γ a *colouring function*.
- An *acceptance condition* $\mathit{Acc} \subseteq \Gamma^\omega$.

For technical convenience we use transition-labelled systems: acceptance conditions are defined over edges instead of over states. These can be easily transformed into state-labelled systems. We will usually take $\Gamma = E$ and γ the identity function. In that case we will omit γ in the description of $\mathcal{T}$. We let $|\mathcal{T}|$ denote $|V|$, for V the set of vertices.

A (finite or infinite) *run* from $q \in V$ on a transition system graph $\mathcal{T}$ is a path $\varrho = e_1 e_2 \dots \in E^\infty$ starting at q. For $A \subseteq V$ we let $\mathcal{R}un_{\mathcal{T},A}$ denote the set of runs on $\mathcal{T}$ starting from some $q \in A$, and $\mathcal{R}un_{\mathcal{T}} = \mathcal{R}un_{\mathcal{T},I_0}$ the set of runs starting from some initial vertex. A run $\varrho \in \mathcal{R}un_{\mathcal{T}}$ is *accepting* if $\gamma(\varrho) \in \mathit{Acc}$, and rejecting otherwise. In this work we suppose that only infinite runs can be accepted.

We say that a vertex $v \in V$ is *accessible* if there exists a finite run $\varrho \in \mathcal{R}un_{\mathcal{T}}$ ending in v. A set of vertices $B \subseteq V$ is accessible if every vertex $v \in B$ is accessible. The *accessible part* of a transition system is the set of accessible vertices.

We might want to add additional information to a transition system (as illustrated in the following paragraphs). For this purpose we introduce labelled transition system: a *vertex-labelled* (resp. edge-labelled) transition system is a transition system $\mathcal{T}$ with a labelling function $l_V : V \to L_V$ (resp. $l_E : E \to L_E$) from vertices (resp. edges) into a set of labels.

Automata as transition systems. An *automaton* is an edge-labelled transition system $\mathcal{A} = (V, E, \mathit{Source}, \mathit{Target}, I_0, \mathit{Acc}, l_E)$ where $l_E : E \to \Sigma$, for Σ a finite set called the *input alphabet* (we say that $\mathcal{A}$ is an automaton over Σ). Given a word $w \in \Sigma^\omega$, a *run over* w is an infinite run $\varrho \in \mathcal{R}un_{\mathcal{T}}$ such that $l_E(\varrho_i) = w_i$ for every $i > 0$. The word $w \in \Sigma^\omega$ is accepted by the automaton $\mathcal{A}$ if there exists an accepting run over w in $\mathcal{A}$. The *language accepted* by an automaton $\mathcal{A}$ is the set $\mathcal{L}(\mathcal{A}) := \{u \in \Sigma^\omega \ : \ u \text{ is accepted by } \mathcal{A}\}$.

We say that an automaton $\mathcal{A}$ is *deterministic* if $|I_0| = 1$ and for every $q \in V$ and every $a \in \Sigma$ there is exactly one edge $e \in \mathit{Out}(v)$ such that $l_E(e) = a$. In this case, we write $\delta(q, a)$ for the only state reachable from q taking the transition labelled with a. We extend the function $\delta(q, -)$ to finite words in the natural way. If $\mathcal{A}$ is deterministic then there is a single run over w for each $w \in \Sigma^\omega$, written $\mathcal{A}(w)$.

Games as transition systems. A *game* $\mathcal{G}_{v_0} = (V, E, \mathit{Source}, \mathit{Target}, v_0, \mathit{Acc}, l_V)$ is a vertex-labelled transition system with a single initial vertex v_0 and vertices labelled by a function $l_V : V \to \{\mathit{Eve}, \mathit{Adam}\}$ that induces a partition of V into vertices controlled by two different players. A *play* is an infinite run produced by moving a token along edges: the player controlling the current vertex chooses what transition to take. It is *winning* for Eve if it is accepting, and winning for Adam otherwise. We say that player $P \in \{\mathit{Eve}, \mathit{Adam}\}$ *wins* the game $\mathcal{G}_{v_0}$ if P can force to always produce a winning play. The *winning region* for player P is the set of vertices $v \in V$ such that P wins the game $\mathcal{G}_v$ obtained by setting the initial vertex to v.

Classes of acceptance conditions. We present the main classes of *ω-regular conditions*. Let Γ be a finite set of *colours*, it will usually be the set of edges of a transition system.

Büchi A *Büchi condition* Acc_B is represented by a subset $B \subseteq \Gamma$. An infinite word $u \in \Gamma^\omega$ belongs to Acc_B if some colour from B appears infinitely often in u.

Rabin A *Rabin condition* Acc_R is represented by a family of *Rabin pairs*, $R = \{(E_1, F_1), \ldots, (E_r, F_r)\}$, where $E_i, F_i \subseteq \Gamma$. A word $u \in \Gamma^\omega$ belongs to Acc_R if $\mathit{Inf}(u) \cap E_i \neq \emptyset$ and $\mathit{Inf}(u) \cap F_i = \emptyset$ for some index $i \in \{1, \ldots, r\}$.

Streett A word $u \in \Gamma^\omega$ belongs to the *Streett condition* Acc_S associated to the family $S = \{(E_1, F_1), \ldots, (E_r, F_r)\}$, $E_i, F_i \subseteq \Gamma$ if $\mathit{Inf}(u) \cap E_i \neq \emptyset \ \rightarrow \ \mathit{Inf}(u) \cap F_i \neq \emptyset$ for every $i \in \{1, \ldots, r\}$.

Parity To define a *parity condition* we suppose that Γ is a finite subset of $\mathbb{N}$. A word $u \in \Gamma^\omega$ belongs to the condition Acc_P if $\min \mathit{Inf}(u)$ is even. The elements of Γ are called *priorities* in this case. We associate to a parity condition the interval $[\mu, \eta]$, where $\mu = \min \Gamma$ and $\eta = \max \Gamma$.

Muller A *Muller condition* $\mathit{Acc}_{\mathcal{F}}$ is given by a family $\mathcal{F} \subseteq \mathcal{P}(\Gamma)$. A word $u \in \Gamma^\omega$ is accepted if the colours appearing infinitely often in u form a set of the family $\mathcal{F}$.

Equivalent conditions. Two different acceptance conditions over a set Γ are *equivalent* if they define the same set $Acc \subseteq \Gamma^\omega$. Given a transition system graph $\mathcal{T}_G$, two representations $\mathcal{R}_1, \mathcal{R}_2$ of acceptance conditions are equivalent over $\mathcal{T}_G$ if they define the same accepting subset of runs of $\mathcal{R}un_T$. We write $(\mathcal{T}_G, \mathcal{R}_1) \simeq (\mathcal{T}_G, \mathcal{R}_2)$ in that case.

If $\mathcal{A}$ is the transition system graph of an automaton and $\mathcal{R}_1, \mathcal{R}_2$ are two representations of acceptance conditions such that $(\mathcal{A}, \mathcal{R}_1) \simeq (\mathcal{A}, \mathcal{R}_2)$, then they recognise the same language: $\mathcal{L}(\mathcal{A}, \mathcal{R}_1) = \mathcal{L}(\mathcal{A}, \mathcal{R}_2)$. However, the converse only holds for deterministic automata.

▶ **Proposition 2.1.** *Let $\mathcal{A}$ be the the transition system graph of a deterministic automaton over the alphabet Σ and let $\mathcal{R}_1, \mathcal{R}_2$ be two representations of acceptance conditions such that $\mathcal{L}(\mathcal{A}, \mathcal{R}_1) = \mathcal{L}(\mathcal{A}, \mathcal{R}_2)$. Then, both conditions are equivalent over $\mathcal{A}$, $(\mathcal{A}, \mathcal{R}_1) \simeq (\mathcal{A}, \mathcal{R}_2)$.*

▶ Remark. A parity condition given by $\Gamma \subseteq \mathbb{N}$ is equivalent to Rabin and Streett conditions over Γ. Any of the previous conditions over a set Γ is equivalent to a Muller condition.

Trees. A *tree* is a set of sequences of non-negative integers $T \subseteq \omega^*$ that is prefix-closed: if $\tau \cdot i \in T$, for $\tau \in \omega^*, i \in \omega$, then $\tau \in T$. In this paper we will only consider finite trees.

The elements of T are called *nodes*. A *subtree* of T is a tree $T' \subseteq T$. The empty sequence ε belongs to every non-empty tree and it is called the *root* of the tree. A node of the form $\tau \cdot i$, $i \in \omega$, is called a *child* of τ, and τ is called its *parent*. We let $Children(\tau)$ denote the set of children of a node τ. Two different children σ_1, σ_2 of τ are called *siblings*, and we say that σ_1 is *older* than σ_2 if $Last(\sigma_1) < Last(\sigma_2)$. If two nodes τ, σ verify $\tau \sqsubseteq \sigma$, then τ is called an *ancestor* of σ, and σ a *descendant* of τ (we add the adjective "strict" if in addition they are not equal). A node is called a *leaf* of T if it is a maximal sequence of T. A *branch* of T is the set of prefixes of a leaf. The set of branches of T is denoted $Branch(T)$. We order the set of branches from left to right.

For a node $\tau \in T$ we define $Subtree_T(\tau)$ as the subtree consisting on the set of nodes that appear below τ, or above it in the same branch: $Subtree_T(\tau) = \{\sigma \in T \ : \ \sigma \sqsubseteq \tau \text{ or } \tau \sqsubseteq \sigma\}$.

Given a node τ of a tree T, the *depth* of τ in T is defined as the length of τ, $Depth(\tau) = |\tau|$. The *height of a tree* T, written $Height(T)$, is defined as the maximal depth of a leaf of T plus 1. The *height of the node* $\tau \in T$ is $Height(T) - Depth(\tau)$.

A *labelled tree* is a pair (T, ν), where T is a tree and $\nu : T \to \Lambda$ is a labelling function into a set of labels Λ.

3 An optimal transformation of Muller into parity conditions

In this section we show how to use the Zielonka tree of a Muller condition to construct a deterministic parity automaton recognising the Muller condition. This can be seen as an extension of the existing constructions transforming Muller conditions into parity conditions such as the LAR [9] or the Index Appearance Record (IAR) [13, 17]. We prove that for all Muller conditions, the parity automaton has a minimal number of states (Theorem 3.7) and a minimal number of priorities (Proposition 3.6).

3.1 The Zielonka tree automaton

▶ **Definition 3.1** (Zielonka tree of a Muller condition [30]). *Let Γ be a finite set of colours and $\mathcal{F} \subseteq \mathcal{P}(\Gamma)$ a Muller condition over Γ. The* Zielonka tree *of $\mathcal{F}$, written $T_\mathcal{F}$, is a tree labelled with subsets of Γ via the labelling $\nu : T_\mathcal{F} \to \mathcal{P}(\Gamma)$, defined inductively as:*

- $\nu(\varepsilon) = \Gamma$
- *If τ is a node already constructed labelled with $S = \nu(\tau)$, we let $S_1, \dots, S_k$ be the maximal subsets of S verifying the property $S_i \in \mathcal{F} \ \Leftrightarrow \ S \notin \mathcal{F}$, for $i \in \{1, \dots, k\}$. For each $i \in \{1, \dots, k\}$ we add a child to τ labelled with S_i.*

We say that the condition $\mathcal{F}$ and the tree $T_{\mathcal{F}}$ are *even* (resp. *odd*) if $\Gamma \in \mathcal{F}$ (resp. $\Gamma \notin \mathcal{F}$). To each node τ of the Zielonka tree we associate the priority $p_Z(\tau) = Depth(\tau)$, and we add 1 to it if $T_{\mathcal{F}}$ is odd.

This way, $p_Z(\tau)$ is even if and only if $\nu(\tau) \in \mathcal{F}$. We represent nodes $\tau \in T_{\mathcal{F}}$ such that $p_Z(\tau)$ is even as a *circle* (*round nodes*), and those for which $p_Z(\tau)$ is odd as a *square*.

▶ **Example 3.2.** Let $\Gamma_1 = \{a, b, c\}$ and $\mathcal{F}_1 = \{\{a\}, \{b\}\}$. The Zielonka tree $T_{\mathcal{F}_1}$ is shown in Figure 1. It is odd.

Let $\Gamma_2 = \{a, b, c, d\}$ and $\mathcal{F}_2 = \{\{a, b, c, d\}, \{a, b, d\}, \{a, c, d\}, \{b, c, d\}, \{a, b\}, \{a, d\}, \{b, c\}, \{b, d\}, \{a\}, \{b\}, \{d\}\}$. The Zielonka tree $T_{\mathcal{F}_2}$ is even and it is shown on Figure 2.

On the right of each tree there are the priorities assigned to the nodes of the corresponding level. We have named the branches of the Zielonka trees with greek letters and we indicate the names of the nodes in violet.

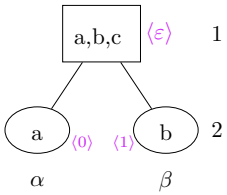


Figure 1 Zielonka tree $T_{\mathcal{F}_1}$.

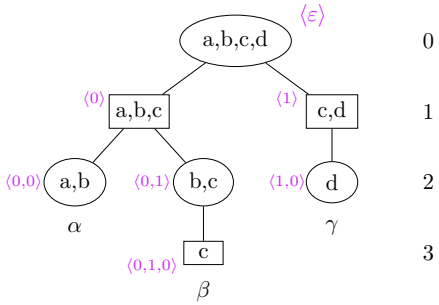


Figure 2 Zielonka tree $T_{\mathcal{F}_2}$.

We show next how to use the Zielonka tree of $\mathcal{F}$ to build a deterministic automaton recognizing the Muller condition $\mathcal{F}$. This automaton can be implicitly found in [7].

For a branch $\beta \in Branch(T_{\mathcal{F}})$ and a colour $a \in \Gamma$ we define $Supp(\beta, a) = \tau$ as the deepest node (maximal for $\sqsubseteq$) in β such that $a \in \nu(\tau)$.

Given a node $\tau \in \beta$, if τ is not a leaf then it has a unique child σ_β such that $\sigma_\beta \in \beta$. In this case, we let $Nextchild(\beta, \tau)$ be the next sibling of σ_β on its right, or the smallest child of τ if σ_β is the biggest one.

We define $Nextbranch(\beta, \tau)$ as the leftmost branch in T below $Nextchild(\beta, \tau)$, if τ is not a leaf, and we let $Nextbranch(\beta, \tau) = \beta$ if τ is a leaf of T.

▶ **Definition 3.3** (Zielonka tree automaton). *Given a Muller condition $\mathcal{F}$ over Γ with Zielonka tree $T_{\mathcal{F}}$, we define the* Zielonka tree automaton *$\mathcal{Z}_{\mathcal{F}}$ as a deterministic automaton over Γ using a parity acceptance condition given by $p : E \to [\mu, \eta]$, where*

- *$Q = Branch(T_{\mathcal{F}})$, the set of states is the set of branches of $T_{\mathcal{F}}$.*
- *The initial state q_0 is irrelevant, we pick the leftmost branch of $T_{\mathcal{F}}$.*
- *The transitions are: $\delta(\beta, a) = Nextbranch(\beta, Supp(\beta, a))$, for $\beta \in Branch(T_{\mathcal{F}})$ and $a \in \Gamma$.*
- *$\mu = 0$, $\eta = Height(T_{\mathcal{F}}) - 1$ if $\mathcal{F}$ is even; $\mu = 1$, $\eta = Height(T_{\mathcal{F}})$ if $\mathcal{F}$ is odd.*
- *$p(\beta, a) = p_Z(Supp(\beta, a))$.*

The transitions of the automaton are determined as follows: if we are in a branch β and we read a colour a, then we move up in the branch β until we reach a node τ that contains the colour a in its label. Then we pick the child of τ just on the right of the branch β (in a cyclic way) and we move to the leftmost branch below it. We produce the priority corresponding to the depth of τ.

▶ **Example 3.4.** Let us consider the conditions of Example 3.2. The Zielonka tree automaton for the Muller condition $\mathcal{F}_1$ is shown in Figure 3, and that for $\mathcal{F}_2$ in Figure 4.

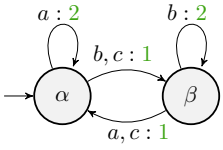


Figure 3 The Zielonka tree automaton $\mathcal{Z}_{\mathcal{F}_1}$.

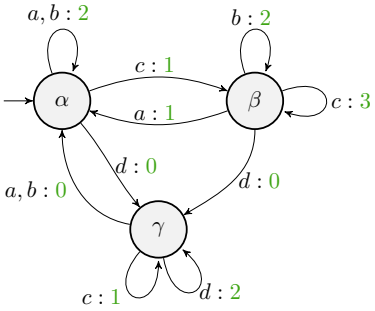


Figure 4 The Zielonka tree automaton $\mathcal{Z}_{\mathcal{F}_2}$.

▶ **Proposition 3.5** (Correctness). *Let $\mathcal{F} \subseteq \mathcal{P}(\Gamma)$ be a Muller condition over Γ. Then, a word $u \in \Gamma^\omega$ verifies $Inf(u) \in \mathcal{F}$ if and only if u is accepted by $\mathcal{Z}_{\mathcal{F}}$.*

3.2 Optimality of the Zielonka tree automaton

We prove in this section the strong optimality of the Zielonka tree automaton, both for the number of priorities (Proposition 3.6) and for the size (Theorem 3.7). These results have been obtained independently in a recent unpublished work by Meyer and Sickert [19].

▶ **Proposition 3.6** (Optimal number of priorities, independently proved in [19]). *The Zielonka tree $\mathcal{Z}_{\mathcal{F}}$ uses the optimal number of priorities for recognizing a Muller condition $\mathcal{F}$. More precisely, if $[\mu, \eta]$ are the priorities used by $\mathcal{Z}_{\mathcal{F}}$ and $\mathcal{P}$ is another parity automaton recognizing $\mathcal{F}$, then $\mathcal{P}$ uses at least $\eta - \mu + 1$ priorities, and in case of equality, its smallest priority has the same parity as μ.*

▶ **Theorem 3.7** (Optimal size of the Zielonka tree automaton, independently proved in [19]). *Every deterministic parity automaton $\mathcal{P}$ accepting a Muller condition $\mathcal{F}$ over Γ verifies $|\mathcal{Z}_{\mathcal{F}}| \leq |\mathcal{P}|$.*

The proof of both results appear in the full version of this paper [4], and Proposition 3.6 can also be deduced from the results of [22]. We sketch the proof of Theorem 3.7: for a set of letters $X \subseteq \Sigma$ we define an X-SCC of an automaton $\mathcal{A}$ over Σ as a strongly connected component of the graph obtained restricting the transitions of $\mathcal{A}$ to those labelled with letters from X. We prove that if A and B are the labels of two siblings in the Zielonka tree $T_{\mathcal{F}}$, and $\mathcal{P}$ is a parity automaton recognising the Muller condition $\mathcal{F}$, then A-SCCs and B-SCCs of $\mathcal{P}$ must be disjoint. Finding such disjoints X-SCC for the children of the nodes of the Zielonka tree allows us to conclude the proof by induction.

4 An optimal transformation of Muller into parity transition systems

In the previous section we have shown how the Zielonka tree yields a transformation of a Muller condition into a parity condition, through the construction of a deterministic parity automaton. This can be naturally lifted to transition systems by composing the automaton with the transition system. However this approach is oblivious to the transition system, meaning it does not consider the possibly fruitful interplay between the transition structure and the condition. All existing transformations follow this approach.

In this section we present our main contribution: an optimal transformation of Muller transition systems into parity transition systems. The key novelty is that it precisely captures the way the transition structure interacts with the condition. In the seminal work [28], Wagner introduces the *alternating chains of loops* of an automaton. This idea has been successfully applied to determine the complexity of computing the Rabin index of different types of ω-automata [2, 14, 22, 29]. Inspired by the notion of Zielonka trees and Wagner's alternating chains, we define a data structure called the alternating cycle decomposition (ACD) analysing the alternating chains of accepting and rejecting cycles of the transition system. We arrange this information in a collection of Zielonka trees obtaining a data structure, the alternating cycle decomposition, that subsumes all the structural information of the transition system necessary to determine whether a run is accepted or not.

We start in Subsection 4.1 by defining the notion of "transformations" using locally bijective morphisms. This will allow us to state the strong optimality result of Proposition 4.8 and Theorem 4.10: for all Muller transition system $\mathcal{T}$, the parity transition system $\mathcal{P}_{\mathcal{ACD}(\mathcal{T})}$ is minimal both in number of states and number of priorities amongst parity transition systems admitting a locally bijective morphism into $\mathcal{T}$.

4.1 Locally bijective morphisms as witnesses of transformations

▶ **Definition 4.1.** *Let $\mathcal{T} = (V, E, Source, Target, I_0, Acc)$, $\mathcal{T}' = (V', E', Source', Target', I_0', Acc')$ be two transition systems. A* morphism of transition systems, *written $\varphi : \mathcal{T} \to \mathcal{T}'$, is a pair of maps $(\varphi_V : V \to V', \varphi_E : E \to E')$ such that:*

- *$\varphi_V(v_0) \in I_0'$ for every $v_0 \in I_0$ (initial states are preserved).*
- *$Source'(\varphi_E(e)) = \varphi_V(Source(e))$ for every $e \in E$ (origins of edges are preserved).*
- *$Target'(\varphi_E(e)) = \varphi_V(Target(e))$ for every $e \in E$ (targets of edges are preserved).*
- *For every run $\varrho \in \mathcal{R}un_{\mathcal{T}}$, $\varrho \in Acc \Leftrightarrow \varphi_E(\varrho) \in Acc'$ (acceptance condition is preserved).*

For labelled transition systems, we say that φ is a morphism of labelled transition systems *if it also preserves the labels.*

We will denote both maps by φ whenever no confusion arises.

▶ **Definition 4.2.** *Given two transition systems $\mathcal{T}$ and $\mathcal{T}'$, a morphism of transition systems $\varphi : \mathcal{T} \to \mathcal{T}'$ is called* locally bijective *if for every $v \in V$ the restriction of φ_E (resp. φ_V) to $Out(v)$ (resp. I_0) is a bijection into $Out(\varphi(v))$ (resp. I_0').*

This is a very similar concept to the usual notion of bisimulation. The main difference is that locally bijective morphisms treat the acceptance of a run as a whole, allowing us to compare transition systems using different classes of acceptance conditions.

▶ **Observation 4.3.** *If $\varphi : \mathcal{T} \to \mathcal{T}'$ is a locally bijective morphism, then φ induces a bijection between the runs in $\mathcal{R}un_{\mathcal{T}}$ and $\mathcal{R}un_{\mathcal{T}'}$ that preserves their acceptance.*

Intuitively, if we transform a transition system $\mathcal{T}_1$ into $\mathcal{T}_2$ "without adding non-determinism", we will have a locally bijective morphism $\varphi : \mathcal{T}_2 \to \mathcal{T}_1$. In particular, if we take the product $\mathcal{T}_2 = \mathcal{T}_1 \times \mathcal{B}$ of $\mathcal{T}_1$ by some deterministic automaton $\mathcal{B}$, the projection over $\mathcal{T}_1$ yields a locally bijective morphism.

The existence of a locally bijective morphism is a witness of the fact that two systems share the same semantic properties: languages recognised by automata are preserved, as well as winning regions of games. Moreover, other important semantic properties of automata, such as being *unambiguous* or *good for games* (notions studied, respectively, in [3] and [10]) are preserved too. We refer to the full version for details [4].

4.2 The alternating cycle decomposition

In the following we will consider Muller transition systems $\mathcal{T} = (V, E, Source, Target, I_0, \mathcal{F})$ with the Muller acceptance condition using edges as colours. We can always suppose this, however, the size of the representation of the condition $\mathcal{F}$ might change. Making this assumption corresponds to considering what are called *explicit Muller conditions*. In particular, solving Muller games with explicit Muller conditions is in PTIME [11], while solving general Muller games is PSPACE-complete [12].

▶ **Example 4.4.** We will use the transition system $\mathcal{T}$ in Figure 5 as a running example. Its Muller condition is given by $\mathcal{F} = \{\{c, d, e\}, \{e\}, \{g, h, i\}, \{l\}, \{h, i, j, k\}, \{j, k\}\}$.

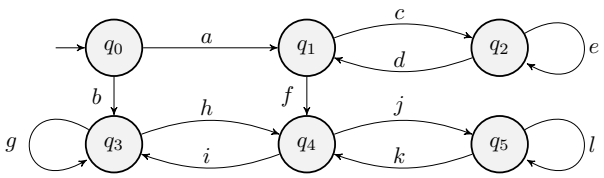


Figure 5 Transition system $\mathcal{T}$.

Given a transition system $\mathcal{T}$, a *loop* is a subset of edges $l \subseteq E$ such that exists $v \in V$ and a finite run $\varrho \in \mathcal{R}un_{T,v}$ starting and ending in v and $Occ(\varrho) = l$. The set of loops of $\mathcal{T}$ is denoted $\mathcal{L}oop(\mathcal{T})$. For a loop $l \in \mathcal{L}oop(\mathcal{T})$ we write $States(l) := \{v \in V \ : \ \exists e \in l, \ Source(e) = v\}$.

There is a natural partial order in the set $\mathcal{L}oop(\mathcal{T})$ given by set inclusion. The maximal loops of $\mathcal{L}oop(\mathcal{T})$ are disjoint and in one-to-one correspondence with the strongly connected components of $\mathcal{T}$.

In the system $\mathcal{T}$ in Figure 5, examples of loops are $l_1 = \{c, d, e\}$ or $l_2 = \{j, k\}$, with $States(l_1) = \{q_1, q_2\}$ and $States(l_2) = \{q_4, q_5\}$. The loop l_1 is maximal.

▶ **Definition 4.5** (Alternating cycle decomposition)**.** *Let $\mathcal{T}$ be a Muller transition system with acceptance condition given by $\mathcal{F} \subseteq \mathcal{P}(E)$. The* alternating cycle decomposition *of $\mathcal{T}$, noted $\mathcal{ACD}(\mathcal{T})$, is a family of labelled trees $(t_1, \nu_1), \dots, (t_r, \nu_r)$ with nodes labelled by loops in $\mathcal{L}oop(\mathcal{T})$, $\nu_i : t_i \to \mathcal{L}oop(\mathcal{T})$. We define it inductively as follows:*

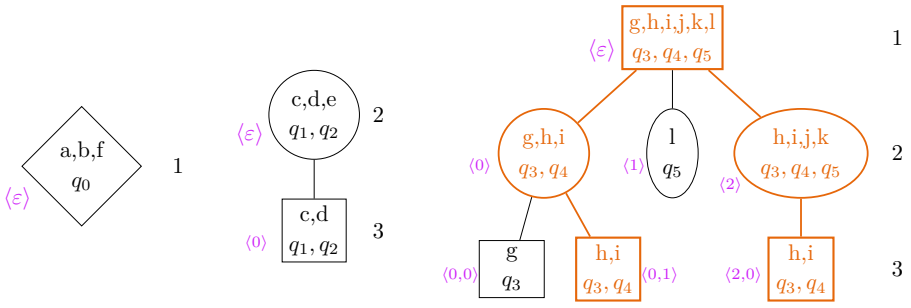
- *Let $\{l_1, \dots, l_r\}$ be the set of maximal loops of $\mathcal{L}oop(\mathcal{T})$. For each $i \in \{1, \dots, r\}$ we consider a tree t_i and define $\nu_i(\varepsilon) = l_i$.*
- *Given an already defined node τ of a tree t_i we consider the maximal loops of the set $\{l \subseteq \nu_i(\tau) \ : \ l \in \mathcal{L}oop(\mathcal{T})$ and $l \in \mathcal{F} \Leftrightarrow \nu_i(\tau) \notin \mathcal{F}\}$ and for each of these loops l we add a child to τ in t_i labelled by l.*

For notational convenience we add a special tree (t_0, ν_0) with a single node ε labelled with the edges not appearing in any other tree of the forest, i.e., $\nu_0(\varepsilon) = E \setminus \bigcup_{i=1}^{r} l_i$. We define $States(\nu_0(\varepsilon)) := V \setminus \bigcup_{i=1}^{r} States(l_i)$.

We call the trees $t_1, \dots, t_r$ the proper trees *of the alternating cycle decomposition of $\mathcal{T}$. Given a node τ of t_i, we note $States_i(\tau) := States(\nu_i(\tau))$.*

The ACD of $\mathcal{T}$ is shown in Figure 6. It consists of two proper trees, t_1 and t_2, corresponding to the strongly connected components of $\mathcal{T}$ and the tree t_0 that corresponds to the edges not appearing in the strongly connected components.

▶ Remark. The Zielonka tree for a Muller condition $\mathcal{F}$ can be seen as a special case of this construction, for an automaton with a single state.



Tree t_0. Tree t_1. Tree t_2. In bold orange, the subtree t_{q_4}.

Figure 6 Alternating cycle decomposition of $\mathcal{T}$. The priority assigned to the nodes of each level of the trees is indicated on the right. Nodes with an even priority are drawn as circles and those with an odd priority as rectangles (excepting the special node forming the root of t_0). Each node τ is labelled with $\nu_i(\tau)$ and with $States_i(\tau)$. In violet the names of the nodes.

Since each state and edge of $\mathcal{T}$ appears in exactly one of the trees of $\mathcal{ACD}(\mathcal{T})$, we can define the *index* of a state $q \in V$ (resp. of an edge $e \in E$) in $\mathcal{ACD}(\mathcal{T})$ as the only number $j \in \{0, 1, \ldots, r\}$ such that $q \in States_j(\varepsilon)$ (resp. $e \in \nu_j(\varepsilon)$).

For each state $q \in V$ of index j we define the *subtree associated to the state* q as the subtree t_q of t_j consisting in the set of nodes $\{\tau \in t_j \ : \ q \in States_j(\tau)\}$.

In Figure 6, state q_4 has index 2, and the subtree associated to q_4 is shown in bold orange.

For each proper tree t_i of $\mathcal{ACD}(\mathcal{T})$ we say that t_i is *even* if $\nu_i(\varepsilon) \in \mathcal{F}$ and that it is *odd* if $\nu_i(\varepsilon) \notin \mathcal{F}$. We say that $\mathcal{ACD}(\mathcal{T})$ is *odd* if all the trees of maximal height of $\mathcal{ACD}(\mathcal{T})$ are odd.

For each $\tau \in t_i$, $i = 1, \ldots, r$, we define the *priority* of τ in t_i as $p_i(\tau) = Depth(\tau)$, adding 1 if t_i is odd. In the case where $\mathcal{ACD}(\mathcal{T})$ is odd we add 2 to nodes on even trees in order to use an optimal number of priorities. We assign to $p_0(\varepsilon)$ the minimal priority appearing in other trees (0 or 1).

We proceed to show how to use the alternating cycle decomposition of a Muller transition system to obtain a parity one.

▶ **Definition 4.6** (ACD-transformation). *Let $\mathcal{T}$ be a Muller transition system with alternating cycle decomposition $\mathcal{ACD}(\mathcal{T}) = \{(t_0, \nu_0), (t_1, \nu_1), \ldots, (t_r, \nu_r)\}$. We define its* ACD-transformation *$\mathcal{P}_{\mathcal{ACD}(\mathcal{T})} = (V_P, E_P, Source_P, Target_P, I'_0, p : E_P \to \mathbb{N})$ as follows:*

For each state $q \in \mathcal{T}$ we consider the subtree t_q consisting of the nodes with q in its label, and we add a state for each branch of this subtree. For each initial state in $\mathcal{T}$, we choose one of its corresponding states in $\mathcal{P}_{\mathcal{ACD}(\mathcal{T})}$ and we set it as initial (the leftmost branch of t_q).

To define transitions in $\mathcal{P}_{\mathcal{ACD}(\mathcal{T})}$ we move simultaneously in $\mathcal{T}$ and in $\mathcal{ACD}(\mathcal{T})$. When we take a transition e in $\mathcal{T}$ that goes from q to q', while being in a branch β, we climb the branch β searching the lowest node τ with e and q' in its label (the support*). We produce the priority corresponding to the level reached. If no such node exists in the branch β, we jump to the root of the tree containing q', producing the priority assigned to this root. After having reached the support τ, we move to the next child of τ on the right of β in the tree $t_{q'}$, and we pick the leftmost branch under it in $t_{q'}$. If we had jumped to the root of $t_{q'}$ from a different tree, we just pick the leftmost branch of $t_{q'}$.*

For a formal definition we refer the reader to the full version of this paper [4].

In Figure 7 we show the ACD-transformation $\mathcal{P}_{\mathcal{ACD}(\mathcal{T})}$ of $\mathcal{T}$. States are labelled with the corresponding state q_j in $\mathcal{T}$, the tree of its index and a node $\tau \in t_i$ that is a leaf in t_{q_j}.

We have tagged the edges of $\mathcal{P}_{\mathcal{ACD}(\mathcal{T})}$ with names of edges from $\mathcal{T}$, in order to indicate the image of the edges by the morphism $\varphi : \mathcal{P}_{\mathcal{ACD}(\mathcal{T})} \to \mathcal{T}$.

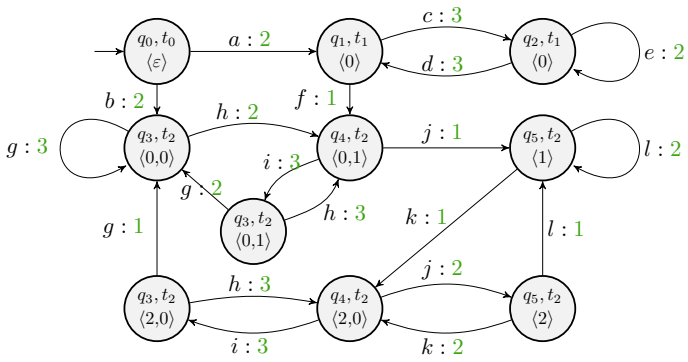


Figure 7 Transition system $\mathcal{P}_{\mathcal{ACD}(\mathcal{T})}$.

▶ **Proposition 4.7** (Correctness). *Let $\mathcal{T}$ be a (possibly labelled) Muller transition system and $\mathcal{P}_{\mathcal{ACD}(\mathcal{T})}$ its ACD-transformation. Then, there exists a locally bijective morphism (of labelled transition systems) $\varphi : \mathcal{P}_{\mathcal{ACD}(\mathcal{T})} \to \mathcal{T}$.*

4.3 Optimality of the alternating cycle decomposition transformation

▶ **Proposition 4.8** (Optimality of the number of priorities). *Let $\mathcal{T}$ be a Muller transition system and let $\mathcal{P}_{\mathcal{ACD}(\mathcal{T})}$ be its ACD-transition system. If $\mathcal{P}$ is another parity transition system such that there is a locally bijective morphism $\varphi : \mathcal{P} \to \mathcal{T}$, then $\mathcal{P}$ uses at least the same number of priorities than $\mathcal{P}_{\mathcal{ACD}(\mathcal{T})}$.*

In the case of deterministic automata, the results from [22] imply this proposition:

▶ **Proposition 4.9.** *If $\mathcal{A}$ is a deterministic Muller automaton, then $\mathcal{P}_{\mathcal{ACD}(\mathcal{A})}$ uses the optimal number of priorities to recognize $\mathcal{L}(\mathcal{A})$.*

Finally, we state the optimality of $\mathcal{P}_{\mathcal{ACD}(\mathcal{A})}$ for size.

▶ **Theorem 4.10** (Optimality of the number of states). *Let $\mathcal{T}$ be a Muller transition system and let $\mathcal{P}_{\mathcal{ACD}(\mathcal{T})}$ be its ACD-transition system. If $\mathcal{P}$ is another parity transition system such that there is a locally bijective morphism $\varphi : \mathcal{P} \to \mathcal{T}$, then $|\mathcal{P}_{\mathcal{ACD}(\mathcal{T})}| \leq |\mathcal{P}|$.*

The proof of Theorem 4.10 follows the same lines as for Theorem 3.7, we refer to the full version of this paper [4]. We note that from the hypothesis of Theorem 4.10 we cannot deduce that there is a morphism from $\mathcal{P}$ to $\mathcal{P}_{\mathcal{ACD}(\mathcal{T})}$ or vice-versa.

5 Applications

Determinisation of Büchi automata

The best theoretical bounds for the determinisation of Büchi automata are achieved by Piterman's construction [23]. In [26], Schewe revisits this construction and presents it as two consecutive steps: a first one producing a deterministic Rabin automaton $\mathcal{R}_{\mathcal{B}}$, and a second one transforming $\mathcal{R}_{\mathcal{B}}$ into a parity automaton $\mathcal{P}_{\mathcal{B}}$. This second step induces a locally

bijective morphism from $\mathcal{P}_{\mathcal{B}}$ to $\mathcal{R}_{\mathcal{B}}$, therefore, thanks to Theorem 4.10 it is guaranteed that the ACD-transformation $\mathcal{P}_{\mathcal{ACD}(\mathcal{R}_{\mathcal{B}})}$ always yields a smaller deterministic parity automaton that uses less priorities. In particular, by Proposition 4.9 the number of priorities used by $\mathcal{P}_{\mathcal{ACD}(\mathcal{R}_{\mathcal{B}})}$ are the optimal one for recognising $\mathcal{L}(\mathcal{B})$ (that is, $\mathcal{ACD}(\mathcal{R}_{\mathcal{B}})$ gives the *parity index* of the language).

In many cases, the gain in both size and number of priorities is strict (we refer to the full version for one example [4]). However, both steps of Piterman Schewe's construction are already optimal in the worst case [6, 27], and applying the ACD-transformation in this worst-case example would generate the same parity automaton.

Relabelling of transition systems by acceptance conditions

We use the information provided by the alternating cycle decomposition to obtain results about the possibility of relabelling Muller transition systems with parity, Rabin and Streett conditions. The results presented here lift the seminal results of [30, Section 5] from conditions to transition systems.

Given a Zielonka tree $T_{\mathcal{F}}$, we say that it has *Rabin shape* (resp. *parity shape*) if every node with an even (reps. even or odd) priority assigned has at most one child. Given a Muller transition system $\mathcal{T}$, we say that its alternating cycle decomposition $\mathcal{ACD}(\mathcal{T})$ is a *Rabin ACD* (resp. *parity ACD*) if for every state $q \in V$, the tree t_q has Rabin shape (resp. parity shape).

▶ **Theorem 5.1.** *Let $\mathcal{T}$ be a Muller transition system. The following conditions are equivalent:*

1. *We can define a Rabin (resp. parity) condition that is equivalent to $\mathcal{F}$ over $\mathcal{T}$.*
2. *For every pair of loops $l_1, l_2 \in \mathcal{L}oop(\mathcal{T})$, if $l_1 \notin \mathcal{F}$ and $l_2 \notin \mathcal{F}$ (resp. l_1 and l_2 are both in $\mathcal{F}$ or both in $\mathcal{P}(\Gamma) \setminus \mathcal{F}$), then $l_1 \cup l_2 \notin \mathcal{F}$ (resp. $l_1 \cup l_2 \in \mathcal{F} \Leftrightarrow l_1 \in \mathcal{F}$).*
3. *$\mathcal{ACD}(\mathcal{T})$ is a Rabin ACD (resp. parity ACD).*

By duality, a symmetric result of the Rabin case holds for Streett conditions.

Similar results can be obtained for weak automata, see the full version for details [4].

▶ **Corollary 5.2.** *Given a transition system graph $\mathcal{T}_G$ and a Muller condition $\mathcal{F} \subseteq \mathcal{P}(E)$, we can define a parity condition $p : E \to \mathbb{N}$ equivalent to $\mathcal{F}$ over $\mathcal{T}_G$ if and only if we can define both Rabin and Streett conditions over $\mathcal{T}_G$, R and S, such that $(\mathcal{T}_G, \mathcal{F}) \simeq (\mathcal{T}_G, R) \simeq (\mathcal{T}_G, S)$.*

The previous results are stated for non-labelled transition systems. We must be careful when translating these results to non-deterministic automata [1, Section 4]. However, Proposition 2.1 allows us to obtain analogous results for deterministic automata.

▶ **Corollary 5.3** (First proven in [1, Theorem 7])**.** *Let $\mathcal{A}$ be a deterministic automaton such that there are a Rabin condition R and a Streett condition S over $\mathcal{A}$ such that $\mathcal{L}(\mathcal{A}, R) = \mathcal{L}(\mathcal{A}, S)$. Then, there exists a parity condition p over $\mathcal{A}$ such that $\mathcal{L}(\mathcal{A}, p) = \mathcal{L}(\mathcal{A}, R) = \mathcal{L}(\mathcal{A}, S)$.*

6 Discussions

In this work we have introduced the alternating cycle decomposition of a transition system, uncovering the interplay between a transition system and its acceptance condition. In order to formalise the notion of a "transformation" we have introduced locally bijective morphisms, which open new lines of research concerning questions such as the complexity of minimising automata with respect to these morphisms. We formulate the following conjecture, which implies that lower bounds established for Muller, Rabin or Streett automata [6] yield lower bounds for parity automata.

▶ **Conjecture 6.1.** *If $\mathcal{A}$ is a minimal deterministic Muller (resp. Rabin) automaton recognising $\mathcal{L}(\mathcal{A})$, then $\mathcal{P}_{\mathcal{ACD}(\mathcal{A})}$ is a minimal deterministic parity automaton recognising $\mathcal{L}(\mathcal{A})$.*

References

1 Udi Boker, Orna Kupferman, and Avital Steinitz. Parityizing Rabin and Streett. In *FSTTCS*, pages 412–423, 2010. doi:10.4230/LIPIcs.FSTTCS.2010.412.

2 Olivier Carton and Ramón Maceiras. Computing the Rabin index of a parity automaton. *RAIRO: Theoretical Informatics and Applications*, pages 495–506, 1999. doi:10.1051/ita:1999129.

3 Olivier Carton and Max Michel. Unambiguous Büchi automata. *Theoretical Computer Science*, 297(1):37–81, 2003. doi:10.1016/S0304-3975(02)00618-7.

4 Antonio Casares, Thomas Colcombet, and Nathanaël Fijalkow. Optimal transformations of Muller conditions. *CoRR*, abs/2011.13041, 2020. arXiv:2011.13041.

5 Edmund M. Clarke, Thomas A. Henzinger, Helmut Veith, and Roderick Bloem, editors. *Handbook of Model Checking*. Springer, Cham, 2018. doi:10.1007/978-3-319-10575-8_2.

6 Thomas Colcombet and Konrad Zdanowski. A tight lower bound for determinization of transition labeled Büchi automata. In *ICALP*, pages 151–162, 2009. doi:10.1007/978-3-642-02930-1_13.

7 Stefan Dziembowski, Marcin Jurdziński, and Igor Walukiewicz. How much memory is needed to win infinite games? In *LICS*, pages 99–110, 1997. doi:10.1109/LICS.1997.614939.

8 Javier Esparza, Jan Křetínský, Jean-François Raskin, and Salomon Sickert. From LTL and limit-deterministic Büchi automata to deterministic parity automata. In *TACAS*, pages 426–442, 2017. doi:10.1007/978-3-662-54577-5_25.

9 Yuri Gurevich and Leo Harrington. Trees, automata, and games. In *STOC*, pages 60–65, 1982. doi:10.1145/800070.802177.

10 Thomas Henzinger and Nir Piterman. Solving games without determinization. In *CSL*, pages 395–410, 2006. doi:10.1007/11874683_26.

11 Florian Horn. Explicit Muller games are PTIME. In *FSTTCS*, pages 235–243, 2008. doi:10.4230/LIPIcs.FSTTCS.2008.1756.

12 Paul Hunter and Anuj Dawar. Complexity bounds for regular games. In *MFCS*, pages 495–506, 2005. doi:10.1007/11549345_43.

13 Jan Křetínský, Tobias Meggendorfer, Clara Waldmann, and Maximilian Weininger. Index appearance record for transforming Rabin automata into parity automata. In *TACAS*, pages 443–460, 2017. doi:10.1007/978-3-662-54577-5_26.

14 Sriram C. Krishnan, Anuj Puri, and Robert K. Brayton. Structural complexity of omega-automata. In *STACS*, pages 143–156, 1995. doi:10.1007/3-540-59042-0_69.

15 Christof Löding and Anton Pirogov. Determinization of Büchi automata: Unifying the approaches of Safra and Muller-Schupp. In *ICALP*, pages 120:1–120:13, 2019. doi:10.4230/LIPIcs.ICALP.2019.120.

16 Michael Luttenberger, Philipp J. Meyer, and Salomon Sickert. Practical synthesis of reactive systems from LTL specifications via parity games. *Acta Informatica*, pages 3–36, 2020. doi:10.1007/s00236-019-00349-3.

17 Christof Löding. Optimal bounds for transformations of ω-automata. In *FSTTCS*, page 97–109, 1999. doi:10.1007/3-540-46691-6_8.

18 Robert McNaughton. Testing and generating infinite sequences by a finite automaton. *Information and control*, 9:521–530, 1966.

19 Philipp Meyer and Salomon Sickert. On the optimal and practical conversion of Emerson-Lei automata into parity automata. *Personal Communication*, 2021.

20 Thibaud Michaud and Maximilien Colange. Reactive synthesis from LTL specification with Spot. In *SYNT@CAV*, 2018.

21 David Müller and Salomon Sickert. LTL to deterministic Emerson-Lei automata. In *GandALF*, pages 180–194, 2017. doi:10.4204/EPTCS.256.13.

22 Damian Niwiński and Igor Walukiewicz. Relating hierarchies of word and tree automata. In *STACS*, pages 320–331, 1998. doi:10.1007/BFb0028571.

23 Nir Piterman. From nondeterministic Büchi and Streett automata to deterministic parity automata. In *LICS*, pages 255–264, 2006. doi:10.1109/LICS.2006.28.

24 Amir Pnueli and Roni Rosner. On the synthesis of a reactive module. In *POPL*, page 179–190, 1989. doi:10.1145/75277.75293.

25 Schmuel Safra. On the complexity of ω-automata. In *FOCS*, page 319–327, 1988. doi:10.1109/SFCS.1988.21948.

26 Sven Schewe. Tighter bounds for the determinisation of Büchi automata. In *FoSSaCS*, pages 167–181, 2009. doi:10.1007/978-3-642-00596-1_13.

27 Sven Schewe and Thomas Varghese. Determinising parity automata. In *MFCS*, pages 486–498, 2014. doi:10.1007/978-3-662-44522-8_41.

28 Klaus Wagner. On ω-regular sets. *Information and Control*, 43(2):123–177, 1979. doi:10.1016/S0019-9958(79)90653-3.

29 Thomas Wilke and Haiseung Yoo. Computing the Rabin index of a regular language of infinite words. *Information and Computation*, pages 61–70, 1996. doi:10.1006/inco.1996.0082.

30 Wiesław Zielonka. Infinite games on finitely coloured graphs with applications to automata on infinite trees. *Theoretical Computer Science*, 200(1-2):135–183, 1998. doi:10.1016/S0304-3975(98)00009-7.

Faster Algorithms for Bounded Liveness in Graphs and Game Graphs

Krishnendu Chatterjee
IST Austria, Klosterneuburg, Austria

Monika Henzinger
Faculty of Computer Science, University of Vienna, Austria

Sagar Sudhir Kale
Faculty of Computer Science, University of Vienna, Austria

Alexander Svozil
Faculty of Computer Science, University of Vienna, Austria

Abstract

Graphs and games on graphs are fundamental models for the analysis of reactive systems, in particular, for model-checking and the synthesis of reactive systems. The class of ω-regular languages provides a robust specification formalism for the desired properties of reactive systems. In the classical infinitary formulation of the liveness part of an ω-regular specification, a "good" event must happen eventually without any bound between the good events. A stronger notion of liveness is bounded liveness, which requires that good events happen within d transitions. Given a graph or a game graph with n vertices, m edges, and a bounded liveness objective, the previous best-known algorithmic bounds are as follows: (i) $O(dm)$ for graphs, which in the worst-case is $O(n^3)$; and (ii) $O(n^2d^2)$ for games on graphs. Our main contributions improve these long-standing algorithmic bounds. For graphs we present: (i) a randomized algorithm with one-sided error with running time $O(n^{2.5}\log n)$ for the bounded liveness objectives; and (ii) a deterministic linear-time algorithm for the complement of bounded liveness objectives. For games on graphs, we present an $O(n^2d)$ time algorithm for the bounded liveness objectives.

2012 ACM Subject Classification Theory of computation → Modal and temporal logics

Keywords and phrases Graphs, Game Graphs, Büchi

Digital Object Identifier 10.4230/LIPIcs.ICALP.2021.124

Category Track B: Automata, Logic, Semantics, and Theory of Programming

Funding *Krishnendu Chatterjee*: Supported by the ERC CoG 863818 (ForM-SMArt).
Monika Henzinger: Supported by the Austrian Science Fund (FWF) and netIDEE SCIENCE project P 33775-N.
Sagar Sudhir Kale: Partially supported by the Vienna Science and Technology Fund (WWTF) through project ICT15-003.
Alexander Svozil: Fully supported by the Vienna Science and Technology Fund (WWTF) through project ICT15-003.

1 Introduction

Graphs and games on graphs. Graphs and two-player games played on graphs provide a general mathematical framework for a wide range of problems in computer science: in particular, for the analysis of reactive systems, where the vertices of the graph represent the states of a reactive system and the edges represent the transitions between the states. The classical synthesis problem (the problem of Church) asks for the construction of a winning strategy in a game played on the graph [13, 21, 20] and the fundamental model-checking problem is an algorithmic graph problem [14].

Omega-regular specifications: strength and weakness. In the analysis of reactive systems, the desired temporal properties that the system should satisfy constitute the specification. The class of ω-regular languages provides a robust specification formalism [18, 20]. Every ω-regular objective can be decomposed into a safety part and a liveness part [3]. The safety part ensures that the system will not do anything "bad" (such as violating an invariant) within any finite number of transitions. The liveness part ensures that the system will do something "good" (such as proceed or respond) in the long-run. Liveness can be violated only in the limit, by infinite sequences of transitions, as no bound is specified on when a "good" event must happen. This infinitary formulation has several strengths, such as robustness and simplicity [18, 23]. However, there is also a weakness of the classical definition of liveness: it can be satisfied by systems that are unsatisfactory because no bound can be put between the occurrence of desired events.

Stronger notion of liveness. For the weakness of the infinitary formulation of liveness, alternative and stronger formulations of liveness have been proposed. The first formulation is *bounded liveness* which ensures, given a bound d, that eventually, good events happen within d transitions. The second formulation is *finitary liveness* which requires the existence of a bound such that eventually good events happen within the bound. Finitary liveness was proposed in [4] and has been widely studied; e.g., games on graphs with finitary ω-regular objectives [12], and logics such as PromptLTL based on finitary liveness [17]. The notion of bounded liveness has also been investigated in many contexts, such as MSO with bounding quantifiers [7], bounded model-checking [6], and "bounded until" in logics such as RTCTL [15].

Algorithmic questions for bounded liveness. In this work, we consider graphs and games on graphs with bounded liveness objectives. Consider a graph with n vertices, m edges, and a bounded liveness objective with bound d. A basic algorithmic approach is to reduce the bounded liveness objective to a liveness objective on a larger graph (that we call the auxiliary graph) that explicitly keeps track of the number of transitions since the last good event. This basic approach yields the following bounds: (a) an $O(dm)$-time algorithm for graphs (applying the linear-time algorithm for liveness objectives on graphs), and (b) an $O(n^2d^2)$-time algorithm for games on graphs (applying the current best-known $O(n^2)$-time algorithm for games on graphs with liveness objectives [11]). A fundamental algorithmic question is whether the above bounds can be improved.

Our contributions. In this work, our main contributions are improved algorithmic bounds for bounded liveness on graphs and games on graphs.

- In graphs, there are two relevant semantics: (a) an existential semantic that asks whether there exists a path to satisfy the objective, and (b) a universal semantic that asks whether all paths satisfy the objective. The answer to the universal semantics with bounded liveness is "Yes" if and only if the answer is "No" for existential semantics with the complementary bounded coliveness objective. We consider graphs with the existential semantics and bounded liveness and bounded coliveness objectives. For bounded liveness objectives, all previous algorithmic approaches yield an $O(n^3)$ worst-case time-bound (where $d = O(n)$) and we present a randomized algorithm with one-sided error whose worst-case time-bound is $O(n^{2.5} \log n)$. For bounded coliveness objectives, we present a deterministic linear-time algorithm.
- For games on graphs with bounded liveness objectives, we present an $O(n^2d)$-time algorithm that improves the previous $O(n^2d^2)$-time algorithm.

Significance of the contributions. On the technical front, it is threefold.

1. To break the $O(n^3)$-time barrier for graphs, we exploit randomization to estimate for all pairs of good events how far they are from each other. Using this information along with a suitably modified auxiliary graph results in the faster $O(n^{2.5} \log n)$-time algorithm.

To get the improved time bound of $O(n^2 d)$ for game graphs:

2. we construct an auxiliary game graph (similar to the graph case) and make a crucial observation that this game graph after each iteration has a lot of structure, a property we call *induced symmetry*;
3. we strategically introduce as many "layover" vertices as there are good events; in combination with induced symmetry, this enables us to prove that a significant chunk of the auxiliary game graph is deleted after each iteration.

Furthermore, there are several important implications of our contributions. First, for graphs with bounded liveness objectives, the previous worst-case time-bound is $O(n^3)$. In recent years, many such algorithmic problems with $O(n^3)$ bound have been shown to be conditionally optimal with a reduction from classical problems such as BMM (boolean matrix multiplication) [1, 2, 8, 9, 10, 24]. Our new algorithm breaks the $O(n^3)$ barrier and shows that such conditional lower bound approaches do not apply for bounded liveness in graphs. Second, for graphs with bounded coliveness objectives our linear-time bound shows that there is a very efficient algorithm for the complement of the bounded liveness objectives. Finally, we show that the basic algorithmic approach for games on graphs can also be improved. Given our results improve the bounds for graphs and games on graphs with bounded liveness objectives, there are several interesting questions for future work. Whether the bounds can be further improved or a deterministic sub-cubic time algorithm can be obtained for graphs with bounded liveness objectives are the most interesting algorithmic open questions.

2 Preliminaries

Since the notation and definitions are standard, we base this section on the definitions section by Chatterjee and Henzinger [11].

Game graphs and graphs. A game graph $\Gamma = ((V, E), \langle V_1, V_2 \rangle)$ is a directed graph, where V is a finite set of vertices, E is a finite set of edges, and $\langle V_1, V_2 \rangle$ is a partition of V into player-1 vertices V_1 and the adversarial player-2 vertices V_2. Graphs are a special case of game graphs with $V_2 = \emptyset$. Define $Out(v) = \{u \in V \mid (v, u) \in E\}$ to be the set of vertices to which v has an outgoing edge and $In(v) = \{u \in V \mid (u, v) \in E\}$ to be the set of vertices from which v has an incoming edge. As is standard, we assume that there are no self-loops and that every vertex has an outgoing edge. Let $n = |V|$ be the number of vertices and $m = |E|$ be the number of edges.

Plays. A *play* $\langle v_0, v_1, v_2, \ldots \rangle$ is an infinite sequence of vertices in Γ such that each $(v_{i-1}, v_i) \in E$ for all $i \geqslant 1$. We denote by Ω the set of all plays. A *finite play* V^* is a prefix of a play.

Strategies. A player-ρ strategy tells which edge to follow next given a finite play that ends in a player-ρ vertex. More formally, a player-1 *strategy* is a function $\sigma : V^* \cdot V_1 \mapsto V$ such that for $\omega \in V^* \cdot V_1$ and v being the last vertex, $(v, \sigma(\omega)) \in E$. A player-2 strategy is defined in the same way. We denote by Σ the set of all player-1 strategies and by Π the set of all player-2 strategies.

Outcome of strategies. Given a starting vertex v and the strategies $\sigma \in \Sigma$ and $\pi \in \Pi$, there is a unique play $\omega(v, \sigma, \pi) = \langle v_0, v_1, v_2, \ldots \rangle$, which is defined as follows: $v_0 = v$; for all $i > 0$ if $v_i \in V_1$ then $\sigma(\langle v_0, \ldots, v_i \rangle) = v_{i+1}$, and if $v_i \in V_2$, then $\pi(\langle v_0, \ldots, v_i \rangle) = v_{i+1}$.

Objectives. An objective $\Phi \subseteq \Omega$ is a set of "winning" plays. The main objectives of this paper are the bounded Büchi objective for player 1 and the complementary bounded coBüchi objective for player 2. For a play ω, we define by $Inf(\omega)$ the set of vertices that occur infinitely often in ω. More formally, if $\omega = \langle v_0, v_1, v_2, \ldots \rangle \in \Omega$, then $Inf(\omega) = \{v \in V \mid \forall i \geqslant 0 \exists j > i : v_j = v\}$. We also need the reachability, safety, Büchi and the coBüchi objectives for the analyses. In the following definitions, assume that we are given a game graph Γ.

1. *Reachability and Safety objectives.* For $T \subseteq V$, the reachability objective states that *at least one* vertex in T be visited, and dually, the safety objective states that *only* vertices in C be visited. Formally, $\text{Reach}(T, \Gamma) = \{\langle v_0, v_1, v_2, \ldots \rangle \in \Omega \mid \exists k \geqslant 0 : v_k \in T\}$ and $\text{Safety}(C, \Gamma) = \{\langle v_0, v_1, v_2, \ldots \rangle \in \Omega \mid \forall k \geqslant 0 : v_k \in C\}$. The two objectives are dual, i.e., $\text{Reach}(T, \Gamma) = \Omega \setminus \text{Safety}(V \setminus T, \Gamma)$.
2. *Büchi and coBüchi objectives.* Given a set of *Büchi* vertices, the Büchi objective states that some Büchi vertex be visited infinitely often, and dually, the coBüchi objective states that only vertices in a given set C be visited infinitely often. Formally, given $B \subseteq V$, define $\text{Büchi}(B, \Gamma) = \{\omega \in \Omega \mid Inf(\omega) \cap B \neq \emptyset\}$ and given $C \subseteq V$, define $\text{coBüchi}(C, \Gamma) = \{\omega \in \Omega \mid Inf(\omega) \subseteq C\}$. The two objectives are dual, i.e., $\text{Büchi}(B, \Gamma) = \Omega \setminus \text{coBüchi}(V \setminus B, \Gamma)$.
3. *Bounded Büchi and bounded coBüchi objectives.* Given a set of Büchi vertices and an integer $d \geqslant 0$, the bounded Büchi objective states that from some point on, the distance between any two consecutive Büchi vertices is at most d. Dually, given $C \subseteq V$, the bounded coBüchi objective requires that there are at least d consecutive vertices in C infinitely often. Formally, the sets of winning plays are $\text{boundedBüchi}(B, d, \Gamma) = \{\omega \in \Omega \mid \exists i \geqslant 0 \forall j \geqslant i : \{v_j, v_{j+1}, \ldots, v_{j+d-1}\} \cap B \neq \emptyset\}$ and $\text{boundedcoBüchi}(C, d, \Gamma) = \{\omega \in \Omega \mid \forall i \geqslant 0 \ \exists j \geqslant i \text{ s.t. } \{v_j, v_{j+1}, \ldots, v_{j+d-1}\} \subseteq C\}$. These are also dual, i.e., $\text{boundedBüchi}(B, d, \Gamma) = \Omega \setminus \text{boundedcoBüchi}(V \setminus B, d, \Gamma)$.

When studying bounded Büchi (and bounded coBüchi) objectives, one can assume without loss of generality that $d \leqslant n$, because otherwise they are equivalent to Büchi objectives. We omit Γ from the definition of the objectives if it is obvious on which game graph the objectives are defined.

For an objective Φ, a strategy $\sigma \in \Sigma$ is a *winning strategy* for player 1 from vertex v if for all player-2 strategies $\pi \in \Pi$ the resulting play $\omega(v, \sigma, \pi) \in \Phi$, and the *set of winning vertices for player 1* is $W_1(\Phi) = \{v \in V \mid \exists \sigma \in \Sigma \text{ s.t. } \forall \pi \in \Pi : \omega(v, \sigma, \pi) \in \Phi\}$. Player-2 winning strategies and winning vertices are defined in the same way.

Remark about determinacy. The following theorem shows that every vertex in V either belongs to the winning set of bounded Büchi objectives of player 1 or to the winning set of bounded coBüchi objectives for player 2. The same holds for Büchi and coBüchi objectives. We say that a vertex is either *winning for player 1* or *winning for player 2*.

▶ **Theorem 1** (Determinancy [19]). *For all game graphs Γ, all (bounded) Büchi objectives Φ for player 1 and the complementary (bounded) coBüchi objectives $\Psi = \Omega \setminus \Phi$ for player 2 we have $W_1(\Phi) = V \setminus W_2(\Psi)$.*

Observe that for (bounded) Büchi objectives Φ for player 1 and the (bounded) coBüchi objectives $\Psi = \Omega \setminus \Phi$, by definition, we have $V \setminus W_2(\Psi) = \{v \in V \mid \forall \pi \in \Pi \ \exists \sigma \in \Sigma \text{ s.t. } \omega(v, \sigma, \pi) \in \Phi\}$. Theorem 1 allows to change existential and universal quantifiers, i.e.,

$V \setminus W_2(\Psi) = \{v \in V \mid \exists \sigma \in \Sigma \text{ s.t. } \forall \pi \in \Pi\ \omega(v, \sigma, \pi) \in \Phi\} = W_1(\Phi)$. If for every strategy π of player 2, there exists a strategy σ for player 1 that wins from vertex v, then there exists a (unique) strategy σ for player 1 that wins against every strategy π of player 2.

The computational problem. Given a game graph with bounded Büchi objective Φ the goal is to compute the set $W_1(\Phi)$. The focus of this paper is on bounded Büchi and bounded coBüchi objectives, and when we mention winning vertices or winning strategies, we mean winning for bounded Büchi objectives, unless stated otherwise.

Closed Sets. A set $U \subseteq V$ of vertices is a closed set for player 1 if $\forall u \in (U \cap V_1) : Out(u) \subseteq U$ and $\forall u \in (U \cap V_2) : Out(u) \cap V_2 \neq \emptyset$. We define player-2 closed sets analogously. Observe that every closed set U induces a subgame graph denoted $G \restriction U$.

A connection between closed sets, winning for safety, reachability and coBüchi objectives in the following proposition.

▶ **Proposition 2** ([11, Proposition 2.2]). *Consider a game graph Γ, and a closed set U for player 1. Then, the following assertions hold:*

1. *Player 2 has a winning strategy for the objective Safety(U) for all vertices in U, that is, player 2 can ensure that if the play starts in U, then the play never leaves the set U.*
2. *If $U \cap B = \emptyset$ (i.e., there is no Büchi vertex in U), then every vertex in U is winning for player 2 for the coBüchi objective.*

Attractors. For a set of "target" vertices $T \subseteq V$, the set of vertices from which player ρ can reach T against all strategies of the other player, is called the *player-ρ attractor* of T; formally [25, 23], $attr_\rho(T, \Gamma) = W_\rho(\text{Reach}(T, \Gamma))$. An attractor $A = attr_\rho(T, \Gamma)$ can be computed in $O(m)$ time [5, 16].

The following observation stipulates the connection between closed sets and attractors.

▶ **Observation 3** ([11]). *For all game graphs Γ, all players $\rho \in \{1, 2\}$, and all sets $U \subseteq V$ we have the following: The set $V \setminus attr_\rho(U, \Gamma)$ is a closed set for player ρ, i.e., no player-ρ vertex in $V \setminus attr_\rho(U, \Gamma)$ has an edge to $attr_\rho(U, \Gamma)$ and every vertex of the other player in $V \setminus attr_\rho(U, \Gamma)$ has an edge in $V \setminus attr_\rho(U, \Gamma)$.*

3 Algorithms for Graphs

Graphs are a special case of game graphs with $V_2 = \emptyset$. Hereon, we will call this "the graph case" as opposed to "the game graph case" (where $V_1 \neq \emptyset$ and $V_2 \neq \emptyset$). The objectives we consider are *prefix independent*, i.e., if $\omega \in \Omega$, then any play obtained by adding or removing a finite prefix to or from ω is also in Ω. Hence, with respect to computing winning vertices, it is enough to focus on strongly connected graphs. The reasoning is as follows.

In the input graph, we call a strongly connected component (SCC) S *good* if the graph restricted to S has a winning vertex. Due to prefix independence, all vertices in a good SCC and those from which you can reach a good SCC are winning. We will prove that such vertices are exactly the winning vertices, and that this set can be computed by the following procedure:

- Compute the SCCs of the input graph (can be done in linear time [22]).
- Determine for each SCC if it is good (this step depends on the objective).
- Consider the set of all vertices belonging to a good SCC. Perform reachability to this set. (This can also be done in linear time.)

▶ **Lemma 4.** *A vertex v is a winning vertex if and only if it there is path from v to some vertex in a good SCC.*

Proof. As mentioned before, due to prefix independence, if v has a path to some vertex in a good SCC, then it is winning. Next, we show the converse.

If v is winning, then there is a winning play ω starting at v. Since SCCs themselves form a directed acyclic graph (DAG), ω must eventually enter an SCC S and stay there. Again, due to prefix independence, the vertices visited by ω in S are also winning, i.e., S is a good SCC. ◀

By Lemma 4 and the procedure described above it, the problem of computing the winning vertices is reduced to determining, given a strongly-connected graph, whether there is a winning vertex or not. More formally, we get the following lemma.

▶ **Lemma 5.** *Let $S_1, S_2, \ldots$ be SCCs of the graph $G = (V, E)$. When $V_2 = \emptyset$, i.e., in the graph case, for a prefix independent objective, the set of winning vertices can be computed in time $O(m + \sum_i t(S_i))$ time, where $m = |E|$ and $t(S_i)$ is the time required to compute whether S_i is a good SCC or not.*

In this paper, we consider bounded Büchi and bounded coBüchi objectives.

3.1 The Bounded Büchi Objective

We are given a graph $G = (V, E)$, a set B of Büchi vertices, and a positive integer d. A cyclic-walk in G is a walk $(v_1, v_2, \ldots, v_\ell)$ such that $v_1 = v_\ell$. We say that a cyclic-walk C is *feasible* if it has at least one Büchi vertex and the number of edges in C between any two consecutive Büchi vertices is at most d. We assume that G is strongly connected, and our goal is to determine if there is a winning vertex in G. Then, using Lemma 5, we generalize the result to a graph that might not be strongly connected. The following lemma reduces this problem to finding a feasible cyclic-walk in G.

▶ **Lemma 6.** *The strongly-connected input graph G has a winning vertex with respect to the bounded Büchi objective if and only if it has a feasible cyclic-walk.*

Proof. If G has a winning vertex, say v, then there is a winning play ω that starts at v. Let $\omega = \langle v_0 = v, v_1, v_2, \ldots \rangle$; so by the definition of winning play, $\exists i \geqslant 1$ such that $\forall j \geqslant i : \{v_j, v_{j+1}, \ldots, v_{j+d-1}\} \cap B \neq \emptyset$. Consider the set $Inf(\omega)$ of vertices that appear infinitely often in ω. Since ω is winning, $Inf(\omega) \cap B \neq \emptyset$. Thus, we can choose a $j' \geqslant i$ such that $v_{j'} \in Inf(\omega) \cap B$. Since $v_{j'}$ appears infinitely often, for some $j'' > j'$, we have that $v_{j''} = v_{j'}$. Thus $(v_{j'}, v_{j'+1}, \ldots, v_{j''} = v_{j'})$ is a feasible cyclic-walk because $j' \geqslant i$ and, as mentioned earlier, $\forall j \geqslant i : \{v_j, v_{j+1}, \ldots, v_{j+d-1}\} \cap B \neq \emptyset$.

In the other direction, if G has a feasible cyclic-walk, then we can keep traversing it to construct a winning play, which means G has a winning vertex. ◀

An $O(dm)$-time algorithm for bounded Büchi

Next, we recall the basic $O(dm)$-time algorithm to determine if there is a feasible cyclic-walk. This algorithm tries to trace a feasible cycle by maintaining a counter with each possible non-Büchi vertex denoting how far away we are from the last visit to a Büchi vertex. We construct a $(d+1)$-layered auxiliary graph $G^* = (V^*, E^*)$, where $V^* = (B \times \{0\}) \cup ((V \setminus B) \times \{1, \ldots, d\})$. We define a more general graph here that we also use in Section 4. We illustrate an example in Figure 1. So, for $(v, \ell) \in V^*$, the integer ℓ corresponds to the aforementioned counter. We

call the vertices in $B \times \{0\}$ Büchi vertices and the vertices in $(V \setminus B) \times \{1, \ldots, d\}$ non-Büchi vertices. The edge set E^* is constructed by Algorithm 1. The last layer of the auxiliary graph is actually not needed for the graph case but is needed for the game graph case later. Observe that the auxiliary graph is also a game graph. (The ownership of the vertices will be defined later in a natural way.)

Algorithm 1 Construction of the auxiliary graph G^* from G, B, and d. It is easy to see that the running time of this algorithm is $O(dm)$ and G^* has at most dm edges.

procedure ConstructAuxiliaryGraph($G = (V, E), B \subseteq V$, d)
  $V^* \leftarrow (B \times \{0\}) \cup ((V \setminus B) \times \{1, \ldots, d\})$ and $E^* \leftarrow \emptyset$.
  for $(u, v) \in E$ such that $v \notin B$ (add counter-incrementing edges) **do**
    if $u \notin B$ **then**
      for $i \in \{1, \ldots, d-1\}$ **do**
        Add $((u, i), (v, i+1))$ to E^*.
      Add $((u, d), (v, d))$ to E^* (edges in the last layer to $V \setminus B$ stay in the last layer).
    else Add $((u, 0), (v, 1))$ to E^*.
  for $(u, v) \in E$ such that $v \in B$ (add counter-resetting edges) **do**
    if $u \notin B$ **then**
      for $i \in \{1, \ldots, d\}$ **do**
        Add $((u, i), (v, 0))$ to E^*.
    else Add $((u, 0), (v, 0))$ to E^*.
  return $G^* = (V^*, E^*)$
procedure AuxiliaryGraph-d-Layers($G = (V, E), B \subseteq V$, d)
  $G^* \leftarrow$ ConstructAuxiliaryGraph($G = (V, E), B \subseteq V, d$)
  Return the graph resulted by removing layer-d from G^*, called $G' = (V', E')$.

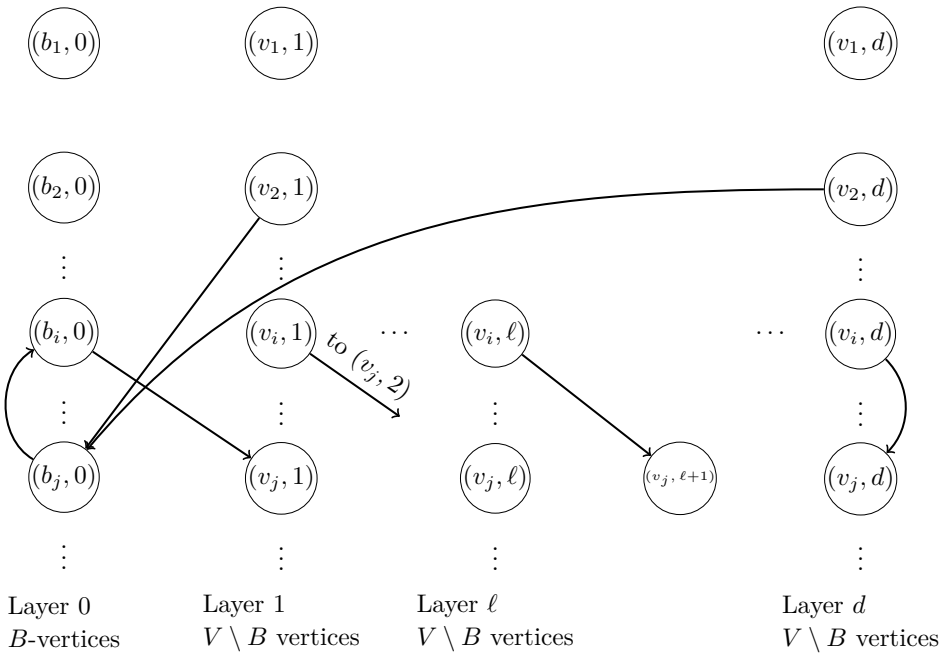


Figure 1 An illustration of how the auxiliary layered graph is constructed. If G contains the edges (b_j, b_i), (b_i, v_j), (v_2, b_j), and (v_i, v_j), then the auxiliary layered graph G^* will have shown edges.

▶ **Lemma 7.** *The running time of the procedures* ConstructAuxiliaryGraph *and* AuxiliaryGraph-d-Layers *in Algorithm 1 is* $O(dm)$.

Proof. In ConstructAuxiliaryGraph, each of the outer for loop runs for at most m iterations, and each of the inner for loops runs for at most d iterations. AuxiliaryGraph-d-Layers just calls ConstructAuxiliaryGraph and removes the last layer, which takes $O(dm)$ time. ◀

For the graph case, we are interested in G^* induced on layers-$\{0, 1, \ldots, d-1\}$. Let G' denote this graph.

▶ **Lemma 8.** *The strongly-connected input graph G has a feasible cyclic-walk if and only if G' has a cycle.*

Proof. Let $C = (b_1, v_{1,1}, \ldots, v_{1,\ell_1}, b_2, v_{2,1}, \ldots, v_{2,\ell_2}, b_3, \ldots, b_1)$, where each $b_i \in B$, each $v_{i,j} \in V \setminus B$, and each $\ell_i \leqslant d-1$, be a feasible cyclic-walk in G. There is a corresponding cyclic-walk C' in G':

- for each $(b_i, v_{i,1}) \in C$, the edge $((b_i, 0), (v_{i,1}, 1)) \in E'$,
- for each $(v_{i,j}, v_{i,j+1}) \in C$, the edge $((v_{i,j}, j), (v_{i,j+1}, j{+}1)) \in E'$,
- for each $(v_{i,\ell_j}, b_{i+1}) \in C$, the edge $((v_{i,\ell_j}, \ell_j), (b_{i+1}, 0)) \in E'$, and
- for the final edge $(v_{i,\ell_j}, b_1) \in C$, the edge $((v_{i,\ell_j}, \ell_j), (b_1, 0)) \in E'$.

If C' consists of union of cycles can be short-cut to get a cycle in G'.

In the other direction, consider a cycle in G'. A projection of this cycle on the first coordinate of the vertices, by construction, gives a feasible cyclic-walk in G, because the number of edges between consecutive Büchi vertices is at most d. ◀

Thus, by Lemmas 6 and 8, we get Algorithm 2.

Algorithm 2 This algorithm determines if the strongly-connected input graph has a winning vertex with respect to the bounded Büchi objective.

```
procedure BoundedBüchi(G = (V, E), B ⊆ V, d)
  G' ← AuxiliaryGraph-d-Layers(G, B, d)
  Run depth-first search on G' to determine if it has a cycle.
  if G' has a cycle then
    return "G has a winning vertex."
  else
    return "G does not have a winning vertex."
```

▶ **Lemma 9.** *Algorithm 2 determines if the strongly-connected input graph G has a winning vertex with respect to the bounded Büchi objective in $O(dm)$ time.*

Proof. By Lemmas 6 and 8, G has a winning vertex if and only if G' has a cycle. Since a depth-first search finds if there is a cycle in G', the correctness of the algorithm is established. By Lemma 7, AuxiliaryGraph-d-Layers takes $O(dm)$ time, and a depth-first search on G' takes time $O(dm)$, because the number of edges in G' is $O(dm)$. ◀

Thus, by Lemma 5, we get the following theorem.

▶ **Theorem 10.** *The set of winning vertices for the bounded Büchi objective in the graph case can be computed in time $O(dm)$.*

Proof. Let $S_1, S_2, \ldots$ be SCCs of the input graph $G = (V, E)$. Let $m_1, m_2, \ldots$ be the number of edges in the SCCs $S_1, S_2, \ldots$. Then, by Lemma 9, for $i = 1, 2, \ldots$, we can determine in time $O(dm_i)$ whether S_i is good. Since $m \geqslant \sum_i m_i$, the proof is complete by Lemma 5. ◀

An $O(|B|m)$-time algorithm for bounded Büchi

Now, we briefly discuss an $O(|B|m)$-time algorithm for bounded Büchi. Given $G = (V, E)$ and B, consider the graph $G' = (B, E')$ such that $(b, b') \in E'$ if the distance from b to b' in G is at most d. We allow self loops in G'. It is easy to see that G has a feasible-cyclic walk if and only if G' has a cycle. To construct G', we perform $|B|$ breadth-first searches, one starting from each vertex in B. This takes time $O(|B|m)$. Then, by a similar argument as in the proof of Theorem 10, we get the following theorem.

▶ **Theorem 11.** *The set of winning vertices for the bounded Büchi objective in the graph case can be computed in time $O(|B|m)$.*

▶ Remark 12. Note that both algorithms that we have seen so far can take $\Theta(n^3)$ time if $m = \Theta(n^2)$ and B and d are $\Theta(n)$. The next algorithm we see is combinatorial and has running time $O(n^{2.5} \log n)$ for the worst setting of the parameters and breaks the cubic barrier. This also rules out any conditional lower bound approaches to get an $\Omega(n^3)$ lower bound for combinatorial algorithms.

An $O((m + |B|^2)\sqrt{n} \log n)$-time algorithm for bounded Büchi

In this section, we present an $O((m+|B|^2)\sqrt{n} \log n)$-time algorithm for bounded Büchi in the graph case. This is one of our main contributions. Here, we give a procedure that computes distances between all pairs of Büchi vertices if the distance is at least $\sqrt{N}$, where $N \geqslant |V|$ is a parameter that we will fix later. This information can be used to reduce the number of layers in the auxiliary graph to $\sqrt{N}$. By dist, we denote the distance function with respect to G. For any $u, v \in V$, if $u \neq v$, then $\text{dist}(u, v)$ denotes the length of a shortest path from u to v, and for any $u \in V$, $\text{dist}(u, u)$ denotes the length of a shortest cycle through u.

Algorithm 3 This algorithm determines if the strongly-connected input graph has a winning vertex with respect to the bounded Büchi objective.

procedure RandBoundedBüchi($G = (V, E), B \subseteq V$, d, N)
  if $d < \sqrt{N}$ **then**
    return BoundedBüchi($G = (V, E), B \subseteq V$, d)
  Sample $4\sqrt{N} \ln N$ vertices uniformly at random, independently, and with replacement.
  $S \leftarrow$ the set of sampled vertices.
  for $s \in S$ **do**
    Perform incoming and outgoing breadth-first search (BFS) to and from s.
    Compute distances $\text{dist}(b, s)$ and $\text{dist}(s, b)$ for each $b \in B$ during the BFSs.
  $G' \leftarrow$ AuxiliaryGraph-d-Layers($G, B, \sqrt{N} - 1$)
  for $b \in B$ **do**
    for $b' \in B$ **do**
      $\text{dist}^S(b, b') \leftarrow \infty$
      for $s \in S$ **do**
        $\text{dist}^S(b, b') \leftarrow \min\{\text{dist}^S(b, b'), \text{dist}(b, s) + \text{dist}(s, b')\}$
      if $\text{dist}^S(b, b') \leqslant d$ **then**
        Add $((b, 0), (b', 0))$ to E' (this would be a self-loop if $b = b'$).
  Run depth-first search on G' to determine if it has a cycle.
  if G' has a cycle **then**
    return "G has a winning vertex."
  else
    return "G does not have a winning vertex."

▶ **Lemma 13.** *Let $N \geqslant |V|$. Algorithm 3 determines with probability at least $1 - 1/N^2$ if the strongly-connected input graph G has a winning vertex with respect to the bounded Büchi objective in $O((m + |B|^2)\sqrt{N} \log N)$ time. It never returns a* false positive*, i.e., if it outputs that G has a winning vertex, then it is correct with probability* 1*. Its running time is $O((m + |B|^2)\sqrt{N} \log N)$.*

Proof. If $d < \sqrt{N}$, then we are done by Lemma 9. Thus, we assume for the rest of the proof that $d \geqslant \sqrt{N}$.

For any $b, b' \in B$, by $T(b, b')$, we denote a fixed shortest cycle through b if $b = b'$ or a fixed shortest path from b to b' otherwise. Let the event that a vertex $v(b, b') \in T(b, b')$ is sampled into S be denoted by $\mathcal{E}(b, b')$. Since $v(b, b') \in T(b, b')$, we have that $\mathrm{dist}(b, b') = \mathrm{dist}(b, v(b, b')) + \mathrm{dist}(v(b, b'), b')$. This implies that if $\mathcal{E}(b, b')$ occurs, then $\mathrm{dist}(b, v(b, b'))$ and $\mathrm{dist}(v(b, b'), b')$ are computed by the algorithm using the incoming and outgoing BFS at $v(b, b')$, and hence $\mathrm{dist}^S(b, b') = \mathrm{dist}(b, b')$. Let $\mathcal{E}^c(b, b')$ be the complement of $\mathcal{E}(b, b')$. Now, $\Pr[\mathcal{E}^c(b, b')] = (1 - \mathrm{dist}(b, b')/|V|)^{4\sqrt{N} \ln N}$, because $1 - \mathrm{dist}(b, b')/|V|$ is the probability that a fixed sample does not contain a vertex of $T(b, b')$ and we draw $4\sqrt{N} \ln N$ independent samples.

For any $b, b' \in B$, where $\mathrm{dist}(b, b') \geqslant \sqrt{N}$, we denote the event that $\mathrm{dist}^S(b, b') = \mathrm{dist}(b, b')$ by $\mathcal{E}'(b, b')$. As noted earlier, $\mathrm{dist}^S(b, b') = \mathrm{dist}(b, b')$ if $\mathcal{E}(b, b')$ occurs, hence:

$$
\begin{aligned}
\Pr[\mathcal{E}'(b, b')] &\geqslant \Pr[\mathcal{E}(b, b')] = 1 - \Pr[\mathcal{E}^c(b, b')] && \mathcal{E}(b, b') \text{ is a subevent of } \mathcal{E}'(b, b'),\\
&= 1 - \left(1 - \frac{\mathrm{dist}(b, b')}{|V|}\right)^{4\sqrt{N} \ln N} && \text{by the argument earlier,}\\
&\geqslant 1 - \left(1 - \frac{1}{\sqrt{N}}\right)^{4\sqrt{N} \ln N} && \text{because } \mathrm{dist}(b, b')/|V| \geqslant 1/\sqrt{N},\\
&\geqslant 1 - \frac{1}{N^4} && \text{by well-known fact } (1 - 1/x)^x \leqslant 1/e.
\end{aligned}
$$

Since $N \geqslant |B|$, by the union bound and because the $\mathcal{E}'(b, b')$ are independent, we have $\Pr[\forall (b, b') \in B \times B : \mathcal{E}'(b, b')] \geqslant 1 - 1/N^2$. Let us condition on the event that for all $(b, b') \in B \times B : \mathcal{E}'(b, b')$, and let G' be the auxiliary graph constructed by the algorithm.

Suppose G has a winning vertex. By Lemma 6, there is a feasible cyclic-walk C in G. Then for any consecutive Büchi vertices b and b' in C, either $\mathrm{dist}(b, b') \geqslant \sqrt{N}$, in which case there is an edge $((b, 0), (b', 0))$ or $\mathrm{dist}(b, b') < \sqrt{N}$, in which case there exists a cycle $((b, 0), (u_1, 1), (u_2, 2), \ldots, (u_\ell, \ell), (b', 0))$ in G', where $\ell < \sqrt{N} - 1$. Thus, C induces a cycle in G'.

On the other hand, if there is a cycle C' in G', then a projection of C' on the first coordinate of the vertices, by construction of G', gives a feasible cyclic-walk in G after replacing all edges in C' of the form $((b, 0), (b', 0))$ by corresponding paths of length at most d that certify $\mathrm{dist}^S(b, b')$. By Lemma 6, G has a winning vertex.

Also, if the algorithm does return that G has a winning vertex, then G' has a cycle, and existence of a feasible cyclic-walk in G can be shown in the same way as above. This shows that the algorithm never returns a false positive.

Running time. Incoming and outgoing BFSs from the vertices in S take time $O(m\sqrt{N} \log N)$. Auxiliary-Graph-d-Layers takes $O(m\sqrt{N})$ time. Computing dist^S takes time $O(|B|^2\sqrt{N} \log N)$. DFS on G' takes time $O(|B|^2 + m\sqrt{N})$. In total, Algorithm 3 has running time $O((m + |B|^2)\sqrt{N} \log N)$. ◀

Finally, we use Lemma 5 to generalize the above to a graph that may not be strongly connected. Fix N to be n in Algorithm 3 when running it for each SCC. Then, by a similar argument as in the proof of Theorem 10, we get the following theorem.

▶ **Theorem 14.** *The set of winning vertices for the bounded Büchi objective can be computed with probability at least* $1-1/n$ *in time* $O((m+|B|^2)\sqrt{n}\log n)$ *which is* $O(n^{2.5}\log n)$*. Moreover, the algorithm never returns a* false positive*, i.e., each vertex in the set it outputs is a winning vertex with probability* 1*.*

Proof. Let $S_1, S_2, \ldots$ be SCCs of the input graph $G = (V, E)$. Let $m_1, m_2, \ldots$ be the number of edges and by $\beta_1, \beta_2, \ldots,$ be the number of Büchi vertices in the SCCs $S_1, S_2, \ldots$, respectively. Then, by Lemma 13, for $i = 1, 2, \ldots$, the algorithm outputs in time $O((m_i + \beta_i^2)\sqrt{n}\log n)$ whether S_i is good. Since $m \geqslant \sum_i m_i$ and $|B|^2 = (\sum_i \beta_i)^2 \geqslant \sum_i \beta_i^2$, the running time bound is proved.

The probability bound is obtained by a union bound over at most n SCCs. Moreover, the algorithm never returns a *false positive* by Lemma 13. ◀

3.2 The Bounded coBüchi Objective

Given a graph $G = (V, E)$, a set C of vertices, and a positive integer d, a walk W is called a *feasible* walk if $W \subseteq C$ and the number of vertices in W is at least d. Let $G[C]$ be the graph induced by C. The bounded coBüchi problem reduces to finding a feasible walk, which further reduces to finding whether there is a cycle in $G[C]$ (can be done in linear time), and if not $G[C]$ is a directed acyclic graph (DAG), so it reduces to determining whether the length of a longest path in the DAG $G[C]$ is at least d (also can be done in linear time). This gives us the following theorem.

▶ **Theorem 15.** *The set of winning vertices for the bounded coBüchi objective in the graph case can be computed in time* $O(m)$*.*

4 Algorithms for Game Graphs

In this section, we present algorithms for the bounded Büchi objective in game graphs. We first introduce the auxiliary *game graph* similar to the auxiliary graph defined earlier. We then show that we can compute in $O(n^2d^2)$ time the winning set of a given bounded Büchi objective on game graphs by computing the winning set of a coBüchi objective on the auxiliary game graph. Finally, we show how to improve the running time to $O(n^2d)$ by using structural properties of the auxiliary game graph and adapting a known technique for solving Büchi Games [11].

The Auxiliary Game Graph. Given a game graph $\Gamma = (V, E, \langle V_1, V_2\rangle)$ with n vertices, m edges and a bounded Büchi objective boundedBüchi(B, d), we first construct the auxiliary graph by calling ConstructAuxiliaryGraph$((V, E), B, d)$ in Algorithm 1 and additionally partition the vertices of the auxiliary graph V^* into player-1 vertices V_1^* and player-2 vertices V_2^*, i.e., for each $(v, \ell) \in V^*$ we get $(v, \ell) \in V_1^*$ if $v \in V_1$ and $(v, \ell) \in V_2^*$ if $v \in V_2$. The auxiliary game graph has $O(nd) = O(n^2)$ vertices and $O(md) = O(mn)$ edges. We say that a vertex $(v, \ell) \in V^*$ is a *layer-ℓ vertex* and v is its *first component*.

For any play λ, we denote by λ_k the kth vertex of the play. If a play has a superscript, it denotes the starting vertex of the play, e.g λ^v means that the play λ starts at v. By λ_k^v we refer to the kth vertex of the play λ^v which starts at v. Given a finite feasible play

$\lambda^{(w,\ell)}$ in Γ^* starting at (w,ℓ), we define $\mathrm{Proj}(\lambda^{(w,\ell)})$ to be the projection of $\lambda^{(w,\ell)}$ on the first component of the vertices in it; by definition, this finite play starts at w and is feasible in Γ. Analogously, given a finite feasible play λ^w in Γ, we define $\mathrm{Lift}(\lambda^w,\ell)$ to be the unique finite feasible play in Γ^* starting at (w,ℓ) such that the first component of $\mathrm{Lift}(\lambda^w,\ell)_k$ is the same as λ^w_k. For (u,v) in E such that $(u,j)\in V^*$ define (the appropriate next layer number if you followed the copy of (u,v) starting in layer j)

$$\mathrm{NxtLyr}(u,v,j)=\begin{cases} j+1 \text{ if } j<d \text{ and } v\notin B\\ d \text{ if } j=d \text{ and } v\notin B\\ 0 \text{ if } v\in B\,. \end{cases}$$

Now, define $\mathrm{Lift}(\lambda^w,\ell)_1=(w,\ell)$, and for $k>1$, given $\mathrm{Lift}(\lambda^w,\ell)_{k-1}=(\lambda^w_{k-1},j)$ define $\mathrm{Lift}(\lambda^w,\ell)_k=(\lambda^w_k,\mathrm{NxtLyr}(\lambda^w_{k-1},\lambda^w_k,j))$. Similarly, given the finite feasible play $\lambda^{(w,\ell)}$ in Γ^*, we define $\mathrm{Shift}(\lambda^{(w,\ell)},\ell')$ to be the finite play that starts at (w,ℓ') in Γ^* such that, for any k, the first components of $\lambda^{(w,\ell)}_k$ and $\mathrm{Shift}(\lambda^{(w,\ell)},\ell')_k$ are the same. By construction of Γ^* the finite play $\mathrm{Shift}(\lambda^{(w,\ell)},\ell')$ is well-defined because (1) edges going from layer-i vertices to layer-$(i+1)$ vertices ($1\leqslant i\leqslant d-1$) exist in all layers with the same respective first components except in layer-d where these edges go again to layer-d, (2) edges going to layer-0 vertices exist in all layers ($1\leqslant i\leqslant d$) and (3) because edges originating from layer-0 vertices implies that both plays are currently visiting the same layer-0 vertex.

In comparison, the goal of the two operations $\mathrm{Proj}(\cdot)$ and $\mathrm{Lift}(\cdot)$ is to map finite plays between Γ^* and Γ such that the finite play in Γ^* has, for all vertices, the same first component as the corresponding finite play in Γ and vice versa. In contrast, $\mathrm{Shift}(\lambda^{(w,\ell)},\ell')$ maps a finite play in Γ^* to a finite play also in Γ^* which has the same first component but a "shifted" starting vertex.

4.1 An $O(n^2d^2)$-time Algorithm for Bounded Büchi in Games

In this section, we show that we can compute the winning set of a given *bounded Büchi objective* on game graphs by computing the winning set of a *coBüchi objective* on the auxiliary game graph. Then we apply the best-known algorithm for computing the winning set of a Büchi objective on the auxiliary game graph to get the desired result.

In the following lemma, we prove that computing $W_1(\mathrm{boundedBüchi}(B,d,\Gamma))$ is the same as computing $W_1(\mathrm{coBüchi}(C^*,\Gamma^*))$ where C^* are the vertices in layers-$\{0,1,\ldots,d-1\}$. Intuitively, when a play ϕ in $\mathrm{coBüchi}(C^*,\Gamma^*)$ stays in layers-$\{0,1,\ldots,d-1\}$, it reaches a vertex in layer 0 every at most d steps by construction of Γ^*. The layer-0 vertices correspond to the vertices in B which means that a play ϕ' in Γ defined as the projection on the first component of the vertices in ϕ visits a vertex in B every at most d steps which implies that $\phi'\in\mathrm{boundedBüchi}(B,d,\Gamma)$. On the other hand, when player 1 has a strategy in Γ to visit a vertex in B every at most d steps, a similar strategy which visits the same vertices in the first component in Γ^* allows player 1 to stay in the first d layers of the auxiliary graph.

▶ **Lemma 16.** *Let $\Gamma=(V,E,\langle V_1,V_2\rangle)$ be a game graph with bounded Büchi objective boundedBüchi(B,d), let $\Gamma^*=(V^*,E^*,\langle V_1^*,V_2^*\rangle)$ be the corresponding auxiliary game graph, and let C^* be the vertices in the first d layers of the auxiliary graph, i.e., $C^*=\{(v,i)\in V^*\mid 0\leqslant i\leqslant d-1\}$. Then $\{w\mid (w,i)\in W_1(\mathit{coBüchi}(C^*,\Gamma^*)),\ \mathit{for\ some}\ 0\leqslant i\leqslant d\}=W_1(\mathit{boundedBüchi}(B,d,\Gamma))$.*

Proof. We first prove that $\{w \mid (w,i) \in W_1(\text{coBüchi}(C^*, \Gamma^*)), \text{ for some } 0 \leqslant i \leqslant d\} \subseteq W_1(\text{boundedBüchi}(B,d,\Gamma))$. Let $(w,i) \in W_1(\text{coBüchi}(C^*,\Gamma^*))$. Then player 1 has a winning strategy σ^* in Γ^* such that for all player-2 strategies π^*, we have that $\omega((w,i),\sigma^*,\pi^*) \in \text{coBüchi}(C^*,\Gamma^*)$.

Whenever player 1 makes a move in Γ^*, we define the corresponding move in Γ as follows: For any finite play λ^w in Γ that ends in a player-1 vertex, define $\sigma(\lambda^w)$ to be the first component of $\sigma^*(\text{Lift}(\lambda^w, i))$. (It does not matter how we define σ for plays that do not start at w.)

Next, we argue why σ is a winning player-1 strategy for $\text{boundedBüchi}(B,d,\Gamma)$ starting at w. Let π be an arbitrary player-2 strategy in Γ. We define a corresponding player-2 strategy π^* in Γ^*: for $\lambda^{(w,i)}$ that ends in a player-2 vertex (u,j), let $v = \pi(\text{Proj}(\lambda^{(w,i)}))$ and define $\pi^*(\lambda^{(w,i)}) = (v, \text{NxtLyr}(u,v,j))$.

Now, it is straightforward to show that the first component of $\omega((w,i),\sigma^*,\pi^*)_k$ is equal to $\omega(w,\sigma,\pi)_k$ by induction on k.

Since the play $\omega((w,i),\sigma^*,\pi^*) \in \text{coBüchi}(C^*,\Gamma^*)$, it stays in C^* after a finite number of steps. Note that to stay in C^* means to visit a layer-0 vertex after every at most d steps because there are only d layers in C^* and each step that does not go to a layer-0 vertex increases the layer counter. Since the first component of each layer-0 vertex is in B, the play $\omega(w,\sigma,\pi)$ visits a vertex in B every at most d steps after a finite number of steps and is in $\text{boundedBüchi}(B,d,\Gamma)$.

The other direction, $W_1(\text{boundedBüchi}(B,d,\Gamma)) \subseteq \{w \mid (w,i) \in W_1(\text{coBüchi}(C^*,\Gamma^*)), \text{for some } 0 \leqslant i \leqslant d\}$ can be shown with a similar argument. ◀

To compute $W_1(\text{coBüchi}(C^*))$ in Γ^*, we observe that, by Theorem 1, $W_1(\text{coBüchi}(C^*)) = V^* \setminus W_2(\text{Büchi}(V^* \setminus C^*)) = V^* \setminus W_2(\text{Büchi}(\{(v,d) \in V^*\}))$. Since, traditionally, we always compute the player-1 winning set of a given objective, we swap player-1 and player-2 vertices in Γ^*. Then we compute $W = W_1(\text{Büchi}(\{(v,d) \in V^*\}))$ using the algorithm of Chatterjee and Henzinger [11], which is the fastest algorithm for Büchi games known, and project $V^* \setminus W$ on the first coordinate. We illustrate the details in Algorithm 4.

Algorithm 4 Determine $W_1(\text{boundedBüchi}(B,d))$, given a game graph Γ.

```
1: procedure BoundedBüchiGames(Γ = (V, E, ⟨V1, V2⟩), B, d)
2:     (V*, E*) ← ConstructAuxiliaryGraph((V, E))
3:     V1* ← {(v,i) ∈ V* | v ∈ V1}, V2* ← {(v,i) ∈ V* | v ∈ V2}
4:     Γ* ← (V*, E*, V1*, V2*); B* ← {(v,d) ∈ V* | v ∈ V \ B}
5:     W ← BüchiGamesFast(Γ* = (V*, E*, ⟨V2*, V1*⟩), B*) ([11], Algorithm 5)
6:     return {x | (x,i) ∈ V* \ W for some 0 ⩽ i ⩽ d}
```

The correctness of Algorithm 4 is due to the correctness of the fast Büchi games algorithm [11, Theorem 2.14], the argument above, and Lemma 16. The argument for the running time of Algorithm 4 is as follows. We first construct Γ^* in $O(md)$ time and then compute the winning set of $\text{coBüchi}(C^*)$ in time $O(|V^*|^2)$ [11, Theorem 2.14]. As $|V^*| = O(nd)$ and $d = O(n)$, we get the following theorem.

▶ **Theorem 17.** *The set of winning vertices for the bounded Büchi objectives in games can be computed in time $O(n^2d^2) = O(n^4)$.*

4.2 An $O(n^2d)$-time Algorithm for Bounded Büchi in Games

In this section, we give a refined running time analysis of Algorithm 4 giving us an $O(n^2d)$-time algorithm for bounded Büchi games. We first describe the fastest algorithm for Büchi Games [11] for completeness. Then, we identify key ideas of the refined running time analysis when the input is an auxiliary game graph and prove the improved running time formally.

4.2.1 The Büchi Games Algorithm of [11]

Given a game graph $\Gamma = (V, E, \langle V_1, V_2\rangle)$ and a set B of Büchi vertices[1], we fix an order on the edges. In this fixed order, the edges (u, v) where u is a non-Büchi player-2 vertex, i.e., $u \in (V_2 \setminus B)$, come before all other edges. We call them priority-1 edges. All the other edges are priority-0 edges.

▶ **Definition 18.** *Given a game graph $\Gamma = (V, E, \langle V_1, V_2\rangle)$, let $\Gamma_i = (V, E_i, \langle V_1, V_2\rangle)$ for $1 \leqslant i \leqslant \log n$ be a subgraph of Γ which we define as follows: For all $u \in V$, the set E_i contains the following edges:*

1. *If the outdegree of u in E is at most 2^i, E_i contains all edges of the form (u, v), i.e., if $|Out(u)| \leqslant 2^i$ then the set $\{(u, v) \mid v \in Out(u)\} \subseteq E_i$.*
2. *If the edge (v, u) belongs to the first 2^i inedges of vertex u in E, we have $(v, u) \in E_i$ ("first" means with respect to the fixed order we specified above).*

Note that $E_{i-1} \subseteq E_i$ since the order of the edges is fixed. We form a partition of V in Γ_i by giving each vertex a color:

- Blue*: A player-1 vertex v in Γ_i is blue if the outdegree of v is greater than 2^i.*
- Red*: A player-2 vertex u in Γ_i is red if it has no outedge in E_i.*[2]
- *All other vertices are* white*.*

Thus, if a player-1 vertex is white then all its outedges are in E_i, and if a player-2 vertex is white then it has at least one outgoing edge in E_i.

Algorithm description. The input of Algorithm 5 is a game graph Γ and a set of Büchi vertices B. Recall that every vertex in a player-1 closed set S without Büchi vertices cannot be in the player-1 winning set of the given Büchi objective $W_1(\text{Büchi}(B))$ (Proposition 2 (2)). We repeatedly find such a set S by removing from V the player-1 attractor of the set B (Proposition 3) and forming S from all the remaining vertices. Then we remove the player-2 attractor of S. In the algorithm, we identify such a set S_j at Line 11 and remove the attractor at Line 15. Note that a naive algorithm would take $O(nm)$ time, as the attractor of S could always be of size 1 and computing the attractor is in $O(m)$ time. To obtain a quadratic-time (in the number of vertices) algorithm, the improved algorithm of Chatterjee and Henzinger constructs, for $i = 1, \ldots, \log n$, the graph Γ_i which has at most 2^i edges. Due to the properties of Γ_i, it can be shown that the set S_j has size of at least 2^{i-1}. In this way, the attractor computation take time proportional to the removed vertices. Since player-1 vertices with missing outgoing edges or player-2 vertices with no outgoing edge in Γ^i, i.e., non-white vertices might still be able to reach a vertex in B, we compute the player-1 attractor of the non-white vertices combined with the vertices in B. We illustrate the details in Algorithm 5.

---

[1] not to be confused with the input for the bounded Büchi problem in the previous and later sections

[2] In the algorithm of Chatterjee and Henzinger [11] red vertices are player-2 vertices where an edge of E is missing. We change this definition slightly, i.e., without changing their algorithm or correctness argument, by saying that player-2 vertices are red if they do not have any outedges in E_i.

Algorithm 5 Determine $W_1(\text{Büchi}(B))$, given a game graph Γ [11].

```
1: procedure BüchiGamesFast(Γ = (V, E, ⟨V_1, V_2⟩), B)
2:     Let j ← 0; U ← ∅; Y_0 ← attr_1(B, Γ); S_0 ← V \ Y_0; D_0 ← attr_2(S_0, Γ); Γ^j ← Γ;
3:     j ← j + 1;
4:     while D_{j−1} ≠ ∅ do
5:         Remove the vertices in D_{j−1} from Γ^{j−1} to obtain Γ^j; and U ← U ∪ D_{j−1};
6:         i ← 1;
7:         repeat
8:             Construct Γ_i^j from Γ^j as described in Definition 18.
9:             Let Z_i^j be the vertices of V^j that are either red or blue;
10:            Y_i^j ← attr_1(B^j ∪ Z_i^j, Γ_i^j);
11:            S_j ← V^j \ Y_i^j;
12:            i ← i + 1
13:        until S_j is nonempty or i ⩾ 1 + log n
14:        if S_j ≠ ∅ then
15:            D_j ← attr_2(S_j, Γ^j)
16:        else
17:            return V \ U
18:        j ← j + 1
```

The definition of a *separating cut* further refines the definition of the winning regions for player 2 in this regard.

Separating cut. A set S of vertices induces a separating cut in a game graph Γ_i or Γ_i^j in Algorithm 5 if

1. the only edges from S to $V \setminus S$ come from player-2 vertices in S
2. every player-2 vertex in S has an edge to another vertex in S
3. every player-1 vertex in S is white and
4. $B \cap S = \emptyset$.

Thus, a separating cut S is a player-1 closed set where (i) player-1 vertices are white and which (ii) does not contain a vertex in B.

The following lemmas are needed to establish the improved running time guarantees in the next section. Detailed proofs can be found in the paper by Chatterjee and Henzinger [11].

Lemma 19 below says that the set S_j is indeed a separating cut in Γ^j (not only in Γ_i^j) and that due to the careful construction of Γ_i^j from the game graph Γ^j in iteration j, S_j does not include a vertex of the player-1 attractor of the Büchi vertices in Γ^j.

▶ **Lemma 19** ([11, Lemma 2.9]). *Let S_j be the non-empty set computed by Algorithm 5 in iteration j. Then, (1) S_j is a separating cut in Γ^j; and (2) $S_j \cap attr_1(B^j, \Gamma^j) = \emptyset$.*

Lemma 20 establishes that the separating cut found in Γ_i^j is indeed the maximum separating cut in Γ_i^j. Also, if Γ_i^j contains a separating cut, Algorithm 5 finds it.

▶ **Lemma 20** ([11, Lemma 2.11]). *Let Γ_i^j be the game graph in iteration j of the outer loop and iteration i of the inner loop. If S induces a separating cut in Γ_i^j, then $S \subseteq S_j$.*

Lemma 21 says that the set S_j is a separating cut in Γ_i^j. This does not follow from Lemma 19(1) because Γ_i^j might have less edges than Γ^j and separating cuts are not preserved if we only consider a subset of edges in Γ^j (property 2 might be violated).

▶ **Lemma 21** ([11, Lemma 2.12]). *Consider an iteration j of the outer loop of Algorithm 5 such that the algorithm stops the inner loop at value i and identifies a non-empty set S_j. Then, S_j is a separating cut in Γ_i^j.*

4.2.2 Faster Algorithm for Bounded Büchi Games

In this section, we give the refined running time analysis of Algorithm 4. We note that Γ^* gets redefined to be $(V^*, E^*, \langle V_2^*, V_1^* \rangle)$ in Algorithm 4 on Line 4. Therefore, from hereon, when we say player 1 (respectively player 2), we mean the player controlling the vertices in V_2^* (respectively, those in V_1^*).

Distinct vertices. We call a set of vertices S in Γ^* *distinct* if, for each pair of vertices $(v, \ell), (v', \ell') \in S$, we have $v \neq v'$.

Copies of a vertex. Let $\mathit{Copies}(v)$ denote the set of "copies" of a vertex $v \in V^*$, i.e., for a layer-0 vertex $(v, 0)$ we have that $\mathit{Copies}((v, 0)) = \{(v, 0)\}$ and for a vertex (v, ℓ), where $\ell > 0$, we have $\mathit{Copies}((v, \ell)) = \{(v, 1), \ldots, (v, d)\}$.

The improved running time guarantee is due to two key ideas.

Key idea 1. When *there is a* vertex (v, ℓ) in D_j then $\mathit{Copies}((v, \ell)) \subseteq D_j$, i.e., *all* its copies are in D_j.

On a very high level, the argument is that if there is a player-2 strategy to go from a vertex to S_j, then there exists a player-2 strategy from all copies of that vertex to S_j. While the idea is simple to state, a complicated machinery is needed to prove it formally. We prove the key idea in Claim 27 building on Definition 25 and Claim 26.

Now, if we follow the original running-time argument [11], then we can only claim that we remove 2^{i-1} vertices in *total* if the inner loop at Line 7 stops at iteration i, but the second key idea states something stronger.

Key idea 2. If the inner loop at Line 7 stops at iteration i^*, we remove 2^{i^*-1} *distinct* vertices.

Combining the key ideas, we remove from the game graph in iteration j all copies of those distinct vertices. The ith iteration of the loop at Lines 7–13 takes time $O(2^i nd)$ for constructing the auxiliary version of $(\Gamma^*)_i$ and performing the attractor computations. The iterations of the loop in Lines 7–13 before $i' < i$ amount to a total running time of $O(2^i nd)$. Thus, we charge the 2^{i-1} removed distinct vertices the cost of the iteration and the iterations before, i.e., each such removed orginal vertex is charged $O(nd)$. As we can remove only n distinct vertices since they correspond to the vertices in the game graph Γ, we have a total cost of $O(n^2 d)$.

For the second key idea to work, we must modify the original bounded Büchi instance (Γ, B, d) carefully. For every vertex in $v \in B$ we add a player-2 vertex v' which is not in B and an edge (v', v). Then we redirect all edges which go to v in the original instance and make them go to v' instead, i.e., for all $v \in B$ we have $V_2 \leftarrow V_2 \cup \{v'\}$ and $E \leftarrow (E \cup \{(v', v)\} \cup \{(u, v') \mid (u, v) \in E\}) \setminus \{(u, v) \in E\}$. Also, we increase d by one, as we increase the distance to all vertices in B by one. Note that this simple modification allows us to assume, without loss of generality, that all vertices in B have incoming edges from player-2 vertices only. Since we swap the player-1 vertices with player-2 vertices in Algorithm 4 we can assume that all incoming edges to a layer-0 vertex are from player-1 vertices. This adds at most n vertices and edges to Γ.

▶ **Observation 22.** *We can assume, without loss of generality, that all layer-*0 *vertices $v \in V^*$ of the auxiliary game graph Γ^* created at Line 4 in Algorithm 4 have no incoming edges from player-2 vertices, i.e., if $(v, 0) \in V^*$ then $\mathit{In}((v, 0)) \cap V_2^* = \emptyset$.*

With the above observation, we can prove the following proposition which is the crux of this section.

▶ **Proposition 23.** *Algorithm 4 runs in time $O(n^2d) = O(n^3)$.*

Proof. In this proof we denote by (Γ^*, B^*) the input of Algorithm 5 at Line 4 of Algorithm 4. The input to Algorithm 4 is (Γ, B, d). If we can show that the running time of the call to Algorithm 5 at Line 4 is in $O(n^2d) = O(n^3)$ we are done, as the rest of the operations of Algorithm 4 are in $O(md)$. This entails constructing (Γ^*, B^*) and going through W. We therefore prove the following lemma.

▶ **Lemma 24.** *The total time Algorithm 4 spends in Algorithm 5 is $O(n^2d) = O(n^3)$.*

Every vertex v in Γ^* has only $O(n)$ out-edges by the definition of the auxiliary game graph. Thus, when we consider the graphs $(\Gamma^*)_i$ of Definition 18 for $1 \leqslant i \leqslant \log n$, we have $(\Gamma^*)_{\log n} = \Gamma^*$. The construction of $(\Gamma^*)_i$ $(1 \leqslant i \leqslant \log n)$ takes time $O(nd \cdot 2^i)$.

We split the running time argument into two parts. In the first part, we bound the running time of all except the last iteration of the while loop at Line 4. In the second part of the analysis, we bound the running time of the last iteration of the same loop.

Running time bound for all iterations of the while loop except the last. Consider iteration j, and assume that Algorithm 5 stops the repeat-until loop at Line 13 with value i^* and it is not the last iteration of the while loop at Line 4. Thus, S_j is not empty. By Lemma 21, the set S_j is a separating cut in $(\Gamma^*)^j_{i^*}$. We make a detour to set up some claims.

We need the following definition because it helps us translate plays and strategies from a vertex to its copies.

▶ **Definition 25.** *If Γ^*_s is an induced subgraph of Γ^* such that for all (u, ℓ_s) in Γ^*_s we have that Copies$((u, \ell_s))$ are also in Γ^*_s, then we say that Γ^*_s has the* induced-symmetry *property or that it is* symmetrically induced*.*

The following claim is about the translation of a strategy from a vertex to its copy.

▷ Claim 26. Suppose Γ^*_s is symmetrically induced. Then, in Γ^*_s, if a player has a strategy to reach a copy of w from a copy of u, then from all copies of u, she has a strategy to reach some copy of w. More formally, in Γ^*_s, if player ρ has a strategy π to reach (w, ℓ_d) from (u, ℓ_s), then for all copies (u, ℓ'_s), she also has a strategy π' to reach (w, ℓ'_d) for some ℓ'_d.

Proof. We define π'. Consider a finite feasible play $\lambda^{(u,\ell'_s)}$ that ends in a player-ρ vertex (v, j). Let $\pi(\text{Shift}(\lambda^{(u,\ell'_s)}, \ell_s)) = (y, p)$. Define $\pi'(\lambda^{(v,\ell'_s)}) = (y, \text{NxtLyr}(v, y, j))$. Now, the play $\text{Shift}(\lambda^{(u,\ell'_s)}, \ell_s)$ is feasible and the strategy π' is well defined because Γ^*_s is symmetrically induced.

We argue why player ρ can reach a copy of w using π'. Let σ' be an arbitrary strategy for the other player, i.e., player $(3 - \rho)$. For any finite feasible play $\lambda^{(u,\ell_s)}$ that ends in a player-$(3 - \rho)$ vertex (v, j), let $\sigma'(\text{Shift}(\lambda^{(u,\ell_s)}, \ell'_s)) = (y, p)$. Define $\sigma(\lambda^{(u,\ell_s)}) = (y, \text{NxtLyr}(v, y, j))$. Again, $\text{Shift}(\lambda^{(u,\ell_s)}, \ell'_s)$ is feasible and σ is well defined because Γ^*_s is symmetrically induced.

Now, it is straightforward to show by induction on k that the first components of $\omega((u, \ell_s), \sigma, \pi)_k$ and $\omega((u, \ell'_s), \sigma', \pi')_k$ are the same. This means that if $\omega((u, \ell_s), \sigma, \pi)_k$ reaches (w, ℓ_d), then $\omega((u, \ell'_s), \sigma', \pi')_k$ reaches (w, ℓ'_d) for some ℓ'_d. ◁

The following claim is a formal version of the first key idea.

▷ Claim 27. If a vertex (v, ℓ) is in D_j, then $\mathit{Copies}((v, \ell)) \subseteq D_j$; and, $(\Gamma^*)^j$ has induced symmetry.

Proof. We prove the claim by induction on j.

Base case, $j = 0$. If $(v, \ell) \in D_0$, then there is a player-2 strategy π_1 to reach $(w, p) \in S_0$. The set $S_0 = V \setminus \mathit{attr}_1(B^*, \Gamma^*)$ is a player-1 closed set by Observation 3: This means that there is a player-2 strategy π_2 to stay inside S_0. By construction of Γ^*, any edge from a non-layer-d vertex goes to the next layer or to layer-0. Then, since $S_0 \cap B^* = \emptyset$, that is, since S_0 does not contain any layer-d vertices, any (infinite) play that stays inside S_0 must eventually return to layer-0. Thus, player 2 can first use π_1 to reach $(w, p) \in S_0$ from (v, ℓ), then use π_2 to reach $(x, 0) \in S_0$ from (w, p); effectively, this gives a player-2 strategy to go to $(x, 0) \in S_0$ from (v, ℓ). Then, by Claim 26, player 2 has a strategy to reach a copy of $(x, 0)$ from (v, ℓ') for any ℓ' because Γ^* itself has induced symmetry. Now, $(x, 0)$ does not have any other copy, this means player 2 has a strategy to reach $(x, 0) \in S_0$ from (v, ℓ'). By induced symmetry of Γ^* again, we have that all copies of (v, ℓ), i.e., $\mathit{Copies}((v, \ell))$ are in Γ^*; moreover, by the above argument, for each of these copies, there is a player-2 strategy to reach S_0, which implies that $\mathit{Copies}((v, \ell)) \subseteq D_0$. Noting that $(\Gamma^*)^0 = \Gamma^*$ has induced symmetry finishes the base case.

Induction step, $j \geqslant 1$. By induction hypothesis, $(\Gamma^*)^{j-1}$ has induced symmetry, and if a vertex (v, ℓ) is in D_{j-1}, then $\mathit{Copies}((v, \ell)) \subseteq D_{j-1}$. This implies that deleting D_{j-1} from $(\Gamma^*)^{j-1}$ to get $(\Gamma^*)^j$ means deleting all copies of a vertex being deleted. Therefore, since $(\Gamma^*)^{j-1}$ has induced symmetry, $(\Gamma^*)^j$ also has induced symmetry.

Since S_j is a separating cut (by Lemma 19), it is a player-1 closed set. Thus, by the same argument as in the base case that uses the induced symmetry of $(\Gamma^*)^j$, if (v, ℓ) is in D_j, then $\mathit{Copies}((v, \ell)) \subseteq D_j$. This completes the induction step and the proof. ◁

The following claim is the formal proof of the second key idea.

▷ Claim 28. The set S_j contains at least 2^{i^*-1} *distinct* vertices.

Proof. The proof is similar to the proof of [11, Lemma 2.13] except that we must now argue that all of the 2^{i^*-1} vertices are distinct. Consider the set S_j in the game graph of the iteration before, i.e., we argue about S_j in $(\Gamma^*)^j_{i^*-1}$. Note that we have the following two cases.

- In the first case, S_j contains a player-1 vertex (x, ℓ) for $1 \leqslant \ell \leqslant d$ that is blue in $(\Gamma^*)^j_{i^*-1}$. Thus, (x, ℓ) has outdegree at least 2^{i^*-1} in $(\Gamma^*)^j_{i^*}$ and none of these edges go to vertices in $V^j \setminus S_j$ in $(\Gamma^*)^j_{i^*}$. Thus, S_j contains at least 2^{i^*-1} vertices. Note that vertex (x, ℓ) can only have edges to vertices which are distinct to (x, ℓ), i.e., for all $((x, \ell), (y, \ell')) \in E^*$ we have $x \neq y$ because the game graph Γ does not have self loops.
- In the second case, all player-1 vertices in S_j are white in $(\Gamma^*)^j_{i^*-1}$. Thus, their outedges in $(\Gamma^*)^j_{i^*}$ and $(\Gamma^*)^j_{i^*-1}$ are identical. We now argue, why a player-2 vertex in S_j exists: Assume for contradiction that no player-2 vertex in S_j exists. Hence, S_j is a separating cut only consisting of player-1 vertices. As S_j is a separating cut in $(\Gamma^*)^j_{i^*}$ we have $S_j \cap B = \emptyset$. Thus, S_j is also a separating cut in $(\Gamma^*)^j_{i^*-1}$. But then, by Lemma 20, the algorithm would have terminated in iteration $i^* - 1$ which is a contradiction because it terminated in iteration i^*.
  Note that repeat-until loop at Lines 7–13 would have stopped in iteration $i^* - 1$ in $(\Gamma^*)^j_{i^*-1}$ as all player-1 vertices in S_j are white.

Consider a player-2 vertex u in S_j. Note that u must have an edge $(u,v) \in (E^*)^j_i$ with $v \in S_j$ because S_j is a separating cut in $(\Gamma^*)^j_{i^*}$ (Lemma 21). Again, there are two possibilities:

- For all player-2 vertices $u \in S_j$ there exists a vertex $v \in S_j$ with $(u,v) \in (E^*)^j_{i^*-1}$. But then S_j would be a separating cut in $(\Gamma^*)^j_{i^*-1}$ as the outedges of player 1 are identical in $(\Gamma^*)^j_{i^*}$ and $(\Gamma^*)^j_{i^*-1}$. By Lemma 20, the separating cut would have been found in iteration i^*-1 of the repeat-until loop at Line 7, which is a contradiction.
- Therefore, there exists a player-2 vertex $u \in S_j$ that has an edge $(u,v) \in (E^*)^j_{i^*}$ to a vertex $v \in S_j$ but this edge is not contained in $(E^*)^j_{i^*-1}$. This can only happen if v has at least 2^{i^*-1} other inedges in $(E^*)^j_{i^*-1}$. Note that u is a player-2 vertex not in $(B^*)^j$ (because all vertices of $(B^*)^j$ belong to Y^j), and hence the edge (u,v) has priority 1 and recall that by the fixed inorder of edges priority-1 edges come before all priority-0 edges. Thus, it follows that since the edge (u,v) is not in $(\Gamma^*)^j_{i^*-1}$, all inedges of v that are in $(\Gamma^*)^j_{i^*-1}$ must have priority 1 by the fixed order of inedges, that is, all the inedges of v in $(\Gamma^*)^j_{i^*-1}$ are from non-Büchi player-2 vertices. Note that $v \in S_j$ and since S_j is a separating cut and, thus, a closed set, all player-2 vertices which are not in B^* with an edge to v are also in S_j. Since v has at least 2^{i^*-1} inedges from player-2 vertices which are not in B^*, the set S_j must contain at least 2^{i^*-1} vertices.

  Furthermore, all incoming edges are from distinct vertices: Note that v cannot be a layer 0 vertex of Γ^*, because by Observation 22 all vertices in B of the given bounded Büchi objective have no incoming edges from a player-2 vertex. Also, layer-d vertices cannot be in S_j as they are in B^* and would be in the player-1 attractor $Y^j_{i^*}$ computed at Line 10. All other vertices in Γ^* have incoming edges only from distinct vertices. Thus, all 2^{i^*-1} such vertices are distinct. ◁

Due to Claim 28, S_j contains at least 2^{i^*-1} distinct vertices, and since $S_j \subseteq D_j$, the set D_j also contains all copies of all vertices in S_j due to Claim 27. All of D_j is deleted. We resume from the detour. The time spent in all graphs $(\Gamma^*)^j_1, \ldots, (\Gamma^*)^j_{i^*}$, i.e., the time spent in the repeat-until loop at Line 7 for the graph construction and the attractor computations, sums up to $O(2^{i^*} \cdot nd)$. We charge $O(nd)$ work to each distinct vertex. This accounts for all the running time except for the last iteration of the outer loop. Since we always remove all copies of a vertex $v \in S_j$, the algorithm deletes at most n distinct vertices throughout a run of the algorithm. Thus, the total time spent over the whole algorithm other than the last iteration is $O(n^2 d)$.

The last iteration of the outer loop. In the last iteration j^* of the outer loop, when no vertex is deleted, the algorithm works on all $\log n$ game graphs, spending time $O(n \cdot 2^i)$ on game graph $(\Gamma^*)^{j^*}_i$. Since each graph $(\Gamma^*)^{j^*}_i$ has at most $nd \cdot 2^{i+1}$ edges and there are $\log n$ graphs, the total number of edges worked in the last iteration is $\sum_{i=1}^{\log n} nd \cdot 2^{i+1} = 4nd \sum_{i=1}^{\log n} 2^{i-1} = 4nd(2^{\log n} - 1) = 4nd(n-1) = O(n^2 d)$. ◀

▶ **Theorem 29.** *The set of winning vertices for the bounded Büchi objective and bounded coBüchi objectives in game graphs can be computed in time $O(n^2 d) = O(n^3)$.*

References

1. Amir Abboud and Virginia Vassilevska Williams. Popular conjectures imply strong lower bounds for dynamic problems. In *FOCS*, pages 434–443. IEEE Computer Society, 2014. doi:10.1109/FOCS.2014.53.
2. Amir Abboud, Virginia Vassilevska Williams, and Huacheng Yu. Matching triangles and basing hardness on an extremely popular conjecture. *SIAM J. Comput.*, 47(3):1098–1122, 2018. doi:10.1137/15M1050987.
3. Bowen Alpern and Fred B. Schneider. Defining Liveness. *Information Processing Letters*, 21(4):181–185, 1985.
4. Rajeev Alur and Thomas A. Henzinger. Finitary Fairness. *ACM Transactions on Programming Languages and Systems*, 20(6):1171–1194, 1998.
5. C. Beeri. On the membership problem for functional and multivalued dependencies in relational databases. *ACM Trans. Datab. Sys.*, 5(3):241–259, 1980.
6. A. Biere, A. Cimatti, E. M. Clarke, O. Strichman, and Y. Zhu. Bounded model checking. *Advances in Computers*, 58:117–148, 2003.
7. Mikolaj Bojanczyk and Thomas Colcombet. Bounds in ω-Regularity. In *Proceedings of the 21st Annual IEEE Symposium on Logic in Computer Science, LICS'06*, pages 285–296. IEEE Computer Society, 2006.
8. Krishnendu Chatterjee, Wolfgang Dvorák, Monika Henzinger, and Veronika Loitzenbauer. Conditionally optimal algorithms for generalized büchi games. In *MFCS*, volume 58 of *LIPIcs*, pages 25:1–25:15. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2016. doi:10.4230/LIPIcs.MFCS.2016.25.
9. Krishnendu Chatterjee, Wolfgang Dvorák, Monika Henzinger, and Veronika Loitzenbauer. Model and objective separation with conditional lower bounds: Disjunction is harder than conjunction. In *LICS*, pages 197–206. ACM, 2016. doi:10.1145/2933575.2935304.
10. Krishnendu Chatterjee, Wolfgang Dvorák, Monika Henzinger, and Alexander Svozil. Algorithms and conditional lower bounds for planning problems. In Mathijs de Weerdt, Sven Koenig, Gabriele Röger, and Matthijs T. J. Spaan, editors, *Proceedings of the Twenty-Eighth International Conference on Automated Planning and Scheduling, ICAPS 2018, Delft, The Netherlands, June 24-29, 2018*, pages 56–64. AAAI Press, 2018. URL: https://aaai.org/ocs/index.php/ICAPS/ICAPS18/paper/view/17639.
11. Krishnendu Chatterjee and Monika Henzinger. Efficient and dynamic algorithms for alternating büchi games and maximal end-component decomposition. *J. ACM*, 2014.
12. Krishnendu Chatterjee, Thomas A. Henzinger, and Florian Horn. Finitary Winning in ω-regular Games. *ACM Transactions on Computational Logic*, 11(1), 2009.
13. Alonzo Church. Logic, arithmetic, and automata. In *Proceedings of the International Congress of Mathematicians*, pages 23–35, 1962.
14. Edmund M. Clarke, Thomas A. Henzinger, Helmut Veith, and Roderick Bloem, editors. *Handbook of Model Checking*. Springer, 2018.
15. E. A. Emerson, A. K. Mok, A. P. Sistla, and J. Srinivasan. Quantitative temporal reasoning. *Real-Time Systems*, 4(4):331–352, 1992.
16. N. Immerman. Number of quantifiers is better than number of tape cells. *Journal of Computer and System Sciences*, pages 384–406, 1981. doi:10.1016/0022-0000(81)90039-8.
17. Orna Kupferman, Nir Piterman, and Moshe Y. Vardi. From liveness to promptness. *Formal Methods Syst. Des.*, 34(2):83–103, 2009.
18. Zohar Manna and Amir Pnueli. *The Temporal Logic of Reactive and Concurrent Systems: Specification*. Springer-Verlag, 1992.
19. Donald A. Martin. Borel determinacy. *Annals of Mathematics*, 102(2):363–371, 1975.
20. Amir Pnueli and Roni Rosner. On the Synthesis of a Reactive Module. In *Proceedings of the 16th Annual ACM Symposium on Principles of Programming Languages, POPL'89*, pages 179–190, 1989.

21 Michael Oser Rabin. Automata on Infinite Objects and Church's Problem. *Transactions of the American Mathematical Society*, 141:1–35, 1969.

22 Robert E. Tarjan. Depth first search and linear graph algorithms. *SIAM J. Comput.*, 1(2):146–160, 1972.

23 Wolfgang Thomas. Languages, Automata, and Logic. In G. Rozenberg and A. Salomaa, editors, *Handbook of Formal Languages*, volume 3, Beyond Words. Springer, 1997.

24 Virginia Vassilevska Williams. On some fine-grained questions in algorithms and complexity. In *Proceedings of the ICM*, volume 3, pages 3431–3472, 2018.

25 W. Zielonka. Infinite games on finitely coloured graphs with applications to automata on infinite trees. *Theoretical Computer Science*, 200(1–2):135–183, 1998. doi:10.1016/S0304-3975(98)00009-7.

Inference Systems with Corules for Fair Subtyping and Liveness Properties of Binary Session Types

Luca Ciccone
University of Torino, Italy

Luca Padovani
University of Torino, Italy

Abstract

Many properties of communication protocols stem from the combination of *safety* and *liveness* properties. Characterizing such combined properties by means of a single inference system is difficult because of the fundamentally different techniques (coinduction and induction, respectively) usually involved in defining and proving them. In this paper we show that *Generalized Inference Systems* allow for simple and insightful characterizations of (at least some of) these combined inductive/coinductive properties for dependent session types. In particular, we illustrate the role of *corules* in characterizing weak termination (the property of protocols that can always eventually terminate), fair compliance (the property of interactions that can always be extended to reach client satisfaction) and also fair subtyping, a liveness-preserving refinement relation for session types.

2012 ACM Subject Classification Theory of computation → Axiomatic semantics; Theory of computation → Type structures; Theory of computation → Program specifications

Keywords and phrases Inference systems, session types, safety, liveness, induction, coinduction

Digital Object Identifier 10.4230/LIPIcs.ICALP.2021.125

Category Track B: Automata, Logic, Semantics, and Theory of Programming

Supplementary Material The Agda formalization of the notions and results presented in the paper is available at
Model (Agda Formalization): `https://github.com/boystrange/FairSubtypingAgda/tree/v1.0`
archived at `swh:1:rev:6d00f452a8380c2aacc659b2a69b9f4b3accfa27`

Acknowledgements We are grateful to Francesco Dagnino for his feedback on an early version of this paper. The anonymous ICALP reviewers provided useful comments and suggestions that helped us improving the paper.

1 Introduction

Session types [21, 22, 4, 23] describe communication protocols at the type level. By making sure that processes use *session channels* according to their session type, a session type system enables the modular enforcement of various desirable properties, including the absence of communication errors, protocol fidelity and in some cases deadlock freedom. These are all examples of *safety properties*, which are informally identified by the motto "nothing bad ever happens" [28]. Less frequent are (session) type systems also enforcing *liveness properties*, those identified by the motto "something good eventually happens". In a network of communicating processes, a typical example of liveness property is the fact that a protocol or a process can always eventually terminate. It is well known that characterizations and proofs of safety and liveness properties rely on fundamentally different (dual) techniques [24, 2, 3, 8]: safety properties are usually based on invariance (coinductive) arguments, whereas liveness properties are usually based on well foundedness (inductive) arguments. As a consequence, it is generally difficult to characterize and enforce complex properties that exhibit a mixture of safety and liveness by means of a single inference system (such as a session type system), in which the inference rules are interpreted either all inductively or all coinductively.

To tame such difficulty, in this work we advocate the use of *Generalized Inference Systems* (GISs) [5, 16]. A GIS allows for the definition of predicates that are a fixed point of the inference operator associated with the inference system, but not necessarily the least or the greatest one. This is made possible by *coaxioms and corules*, whose purpose is to provide an inductive definition of a space within which a coinductive definition is used. A recurring example in the literature of GISs is the predicate $\mathit{maxElem}(l, x)$, asserting that x is the maximum element of a possibly infinite list l. If we consider the inference system

$$\frac{}{\mathit{maxElem}(x :: \Lambda, x)} \qquad \frac{\mathit{maxElem}(l, y)}{\mathit{maxElem}(x :: l, \max\{x, y\})}$$

where Λ denotes the empty list and $::$ is the constructor, we can give two natural interpretations to these rules. The *inductive* one, obtained by taking the least fixed point of the inference operator, restricts the set of derivable judgments to those for which there is a *well-founded* derivation tree. In this case, the *maxElem* predicate is sound but not complete, since it does not hold for any infinite list, even those for which the maximum exists. The *coinductive* interpretation of these rules, obtained by taking the greatest fixed point of the inference operator, allows us to derive judgments by means of *non-well-founded* derivation trees. In this case, the *maxElem* predicate is complete but not sound. In particular, it becomes possible to derive any judgment $\mathit{maxElem}(l, x)$ where x is greater than the elements of the list, but is not an element of the list.

This is one situation in which the sought predicate is neither the least nor the greatest fixed point of a given inference operator, but is somewhere "in between" the two extremes. We can repair the above inference system by adding the following *coaxiom*:

$$\frac{}{\mathit{maxElem}(x :: l, x)}$$

Read naively, this coaxiom seems to assert that the first element of any list is also its maximum. In the context of a GIS, however, its effect is that of ruling out those judgments $\mathit{maxElem}(l, x)$ in which x is *not* an element of the list. In a sense, the coaxiom adds a *well-foundedness element* to the derivability of a judgment $\mathit{maxElem}(l, x)$, by requiring that x must be found in – at some finite distance from the head of – the list l.

The main contribution of this work is the realization that corules can be used to easily characterize properties of session types involving a mixture of coinduction/safety and induction/liveness. We consider three such properties: *weak termination* (the property of protocols that can always eventually terminate), *fair compliance* (the property of client/server interactions that can always be extended to reach client satisfaction) and *fair subtyping* (a liveness-preserving refinement relation for session types). We show how to provide sound and complete characterizations of these properties just by adding a few corules to the inference systems of their "unfair" counterparts, those focusing on safety but neglecting liveness. Not only the added corules shed light on the liveness(-preserving) property of interest, but we can conveniently appeal to the *bounded coinduction principle* of GISs [5] to prove the completeness of the provided characterizations, thus factoring out a significant amount of work. We also make two side contributions. First, the aforementioned characterizations are given for a family of *dependent session types* [33, 34, 32, 14] in which the length and structure of the protocol may depend in non-trivial ways on the *content* of exchanged messages. Thus, we extend previously given characterizations of fair subtyping [29, 30] to a much larger class of protocols. Second, we provide an Agda [27] formalization of all the notions and results stated in the paper. In particular, we give the first machine-checked formalization of a liveness-preserving refinement relation for (dependent) session types.

The rest of the paper is structured as follows. We quickly recall the key definitions of GISs in Section 2 and describe syntax and semantics of (dependent) session types in Section 3. We define and characterize weak termination, fair compliance and fair subtyping in Sections 4–6 and conclude in Section 7. The Agda formalization is accessible from a public repository [15].

2 Generalized Inference Systems

In this section we briefly recall the key notions of Generalized Inference Systems (GISs). In particular, we see how GISs enable the definition of predicates whose purely (co)inductive interpretation does not yield the intended meaning and we review the canonical technique to prove the completeness of a defined predicate with respect to a given specification. Further details on GISs may be found in the existing literature [5, 16].

An *inference system* [1] $\mathcal{I}$ over a *universe* $\mathcal{U}$ of *judgments* is a set of *rules*, which are pairs $\langle pr, j\rangle$ where $pr \subseteq \mathcal{U}$ is the set of *premises* of the rule and $j \in \mathcal{U}$ is the *conclusion* of the rule. A rule without premises is called *axiom*. Rules are typically presented using the syntax

$$\frac{pr}{j}$$

where the line separates the premises (above the line) from the conclusion (below the line).

▶ Remark 2.1. In many cases, and in this paper too, it is convenient to present inference systems using *meta-rules* instead of rules. A meta-rule stands for a possibly infinite set of rules, which are obtained by instantiating the *meta-variables* occurring in the meta-rule. For example, the rules for *maxElem* discussed in Section 1 are meta-rules referring to the meta-variables x, y and l. The actual rules of the discussed inference system result from all possible instantiations of such meta-variables with numbers and (possibly infinite) lists. In the rest of the paper we will not insist on this distinction and we will use "(co)rule" even when referring to meta-(co)rules. If necessary, we will use side conditions to constrain the valid instantiations of the meta-variables occurring in such meta-(co)rules. ⌟

An *interpretation* of an inference system $\mathcal{I}$ identifies a subset of $\mathcal{U}$ whose elements are called *derivable judgments*. To define the interpretation of an inference system $\mathcal{I}$, consider the *inference operator* associated with $\mathcal{I}$, which is the function $F_{\mathcal{I}} : \wp(\mathcal{U}) \to \wp(\mathcal{U})$ such that

$$F_{\mathcal{I}}(X) = \{j \in \mathcal{U} \mid \exists pr \subseteq X : \langle pr, j\rangle \in \mathcal{I}\}$$

for every $X \subseteq \mathcal{U}$. Intuitively, $F_{\mathcal{I}}(X)$ is the set of judgments that can be derived in one step from those in X by applying a rule of $\mathcal{I}$. Note that $F_{\mathcal{I}}$ is a monotone endofunction on the complete lattice $\wp(\mathcal{U})$, hence it has least and greatest fixed points.

▶ **Definition 2.2.** *The* inductive interpretation $\mathsf{Ind}[\![\mathcal{I}]\!]$ *of an inference system* $\mathcal{I}$ *is the least fixed point of* $F_{\mathcal{I}}$ *and the* coinductive interpretation $\mathsf{CoInd}[\![\mathcal{I}]\!]$ *is the greatest one.*

From a proof theoretical point of view, $\mathsf{Ind}[\![\mathcal{I}]\!]$ and $\mathsf{CoInd}[\![\mathcal{I}]\!]$ are the sets of judgments derivable with well-founded and non-well-founded proof trees, respectively.

Generalized Inference Systems enable the definition of (some) predicates for which neither the inductive interpretation nor the coinductive one give the expected meaning.

▶ **Definition 2.3** (generalized inference system)**.** *A* generalized inference system *is a pair* $\langle \mathcal{I}, \mathcal{I}_{\mathsf{co}}\rangle$ *where* $\mathcal{I}$ *and* $\mathcal{I}_{\mathsf{co}}$ *are inference systems whose elements are called* rules *and* corules*, respectively. The interpretation of a generalized inference system* $\langle \mathcal{I}, \mathcal{I}_{\mathsf{co}}\rangle$*, denoted by* $\mathsf{Gen}[\![\mathcal{I}, \mathcal{I}_{\mathsf{co}}]\!]$*, is the greatest post-fixed point of* $F_{\mathcal{I}}$ *that is included in* $\mathsf{Ind}[\![\mathcal{I} \cup \mathcal{I}_{\mathsf{co}}]\!]$*.*

From a proof theoretical point of view, a GIS $\langle \mathcal{I}, \mathcal{I}_{\mathsf{co}} \rangle$ identifies the set of judgments that are derivable with an arbitrary (not necessarily well-founded) proof tree in $\mathcal{I}$ and whose nodes (the judgments occurring in the proof tree) are all derivable with a well-founded proof tree in $\mathcal{I} \cup \mathcal{I}_{\mathsf{co}}$. Recalling *maxElem* from Section 1, the judgments referring to a "maximum" which does not belong to the list are ruled out since they cannot be derived using a well-founded proof tree in the inference system with the coaxiom.

Consider now a *specification* $\mathcal{S} \subseteq \mathcal{U}$ that contains the *valid judgments*. We can relate $\mathcal{S}$ to the interpretation of a (generalized) inference system using one of the following proof principles. The *induction principle* allows us to prove the *soundness* of an inductively defined predicate by showing that $\mathcal{S}$ is *closed* with respect to $\mathcal{I}$. That is, whenever the premises of a rule of $\mathcal{I}$ are all in $\mathcal{S}$, then the conclusion of the rule is also in $\mathcal{S}$.

▶ **Proposition 2.4.** *If* $F_{\mathcal{I}}(\mathcal{S}) \subseteq \mathcal{S}$*, then* $\mathsf{Ind}[\![\mathcal{I}]\!] \subseteq \mathcal{S}$*.*

The *coinduction principle* allows us to prove the *completeness* of a coinductively defined predicate by showing that $\mathcal{S}$ is *consistent* with respect to $\mathcal{I}$. That is, every judgment of $\mathcal{S}$ is the conclusion of a rule whose premises are also in $\mathcal{S}$.

▶ **Proposition 2.5.** *If* $\mathcal{S} \subseteq F_{\mathcal{I}}(\mathcal{S})$*, then* $\mathcal{S} \subseteq \mathsf{Coind}[\![\mathcal{I}]\!]$*.*

The *bounded coinduction principle* allows us to prove the *completeness* of a predicate defined by a generalized inference system $\langle \mathcal{I}, \mathcal{I}_{\mathsf{co}} \rangle$. In this case, one needs to show not only that $\mathcal{S}$ is consistent with respect to $\mathcal{I}$, but also that $\mathcal{S}$ is *bounded* by the inductive interpretation of the inference system $\mathcal{I} \cup \mathcal{I}_{\mathsf{co}}$. Formally:

▶ **Proposition 2.6.** *If* $\mathcal{S} \subseteq \mathsf{Ind}[\![\mathcal{I} \cup \mathcal{I}_{\mathsf{co}}]\!]$ *and* $\mathcal{S} \subseteq F_{\mathcal{I}}(\mathcal{S})$*, then* $\mathcal{S} \subseteq \mathsf{Gen}[\![\mathcal{I}, \mathcal{I}_{\mathsf{co}}]\!]$*.*

Proving the boundedness of $\mathcal{S}$ amounts to proving the completeness of $\mathcal{I} \cup \mathcal{I}_{\mathsf{co}}$ (inductively interpreted) with respect to $\mathcal{S}$. All of the GISs that we are going to discuss in Sections 4–6 are proven complete using the bounded coinduction principle.

3 Syntax and Semantics of Dependent Session Types

We assume a set $\mathbb{V}$ of *values* that can be exchanged in communications. This set may include booleans, natural numbers, strings, and so forth. Hereafter, we assume that $\mathbb{V}$ contains at least *two* elements, otherwise branching protocols cannot be described and the theoretical development that follows becomes trivial. We use x, y, z to range over the elements of $\mathbb{V}$.

We define the set $\mathbb{S}$ of *dependent session types* over $\mathbb{V}$ using coinduction, to account for the possibility that session types (and the protocols they describe) may be infinite.

▶ **Definition 3.1.** *Let* $\mathbb{S}$ *be the largest set such that* $T \in \mathbb{S}$ *implies either* $T = \mathsf{nil}$ *or* $T = ?f$ *or* $T = !f$ *where* $f \in \mathbb{V} \to \mathbb{S}$ *is a total function from* $\mathbb{V}$ *to* $\mathbb{S}$*. We use* T*,* S *and* R *to range over elements of* $\mathbb{S}$ *and* f*,* g *to range over elements of* $\mathbb{V} \to \mathbb{S}$*. We say that* $T \in \mathbb{S}$ *is a* (dependent) session type *over* $\mathbb{V}$ *and that* $f \in \mathbb{V} \to \mathbb{S}$ *is a* continuation *over* $\mathbb{V}$*.*

A session type can be of three forms. An *input session type* $?f$ describes a channel used first for receiving a message $x \in \mathbb{V}$ and then according to $f(x)$. An *output session type* $!f$ describes a channel used first for sending a message $x \in \mathbb{V}$ and then according to $f(x)$. Finally, the session type nil describes an *unusable* session channel. As we will see shortly, we use nil in combination with input and output session types to describe unexpected inputs and impossible outputs. We call the functions f *continuations* since they take as input a message (the one being exchanged on the session channel) and *compute* the session type that describes the usage of the channel after the exchange. Continuations allow us to describe

protocols whose structure *depends* on previously exchanged messages. We do not detail the concrete language in which continuations are specified, but in practice they will be a small subset of the computable functions. For example, in our Agda formalization [15, file `SessionType.agda`] continuations are well-typed Agda functions. This way, we can leverage on Agda and its library [27] for constructing dependent session types.

▶ Remark 3.2. Session types as defined in Definition 3.1 are isomorphic to the possibly infinite trees coinductively generated by the productions

$$S, T \ ::= \ \mathsf{nil} \mid ?\{x : T_x\}_{x \in \mathbb{V}} \mid !\{x : T_x\}_{x \in \mathbb{V}}$$

where we represent continuations f with their graph $\{x : f(x)\}_{x \in \mathbb{V}}$. The structure of session types we have chosen in Definition 3.1, besides suggesting an effective representation of *dependent* session types, is also aimed at streamlining as much as possible not only the syntax but also the semantics of session types. In the end, this choice allowed us to reduce the inevitable complexity bloat that we had to face when formalizing these notions and the related proofs in Agda [15]. ⌟

It is convenient to introduce some notation for presenting session types in a more readable and familiar form. Given a set $X \subseteq \mathbb{V}$ of values, we write $X.T$ for the continuation

$$(X.T)(x) \stackrel{\text{def}}{=} \begin{cases} T & \text{if } x \in X \\ \mathsf{nil} & \text{otherwise} \end{cases}$$

so that we can write session types like $!\mathbb{B}.T$ (send a boolean and continue as T) and $?\mathbb{N}.S$ (receive a natural number and continue as S). We abbreviate $\{x\}$ with x when no confusion may arise. So we write $!\mathsf{true}.T$ instead of $!\{\mathsf{true}\}.T$. We let $?\mathsf{end} \stackrel{\text{def}}{=} ?\emptyset.\mathsf{nil}$ and $!\mathsf{end} \stackrel{\text{def}}{=} !\emptyset.\mathsf{nil}$. Both $?\mathsf{end}$ and $!\mathsf{end}$ describe session channels on which no further communications may occur, although they differ slightly with respect to the session types they can be safely combined with (more on this later). We also define a partial binary operation $\sqcup$ on session types such that $\mathsf{nil} \sqcup T = T \sqcup \mathsf{nil} = T$ and that is undefined otherwise. We extend this operation pointwise to continuations, so that $(f \sqcup g)(x) = f(x) \sqcup g(x)$. It is intended that $f \sqcup g$ is undefined if so is $f(x) \sqcup g(x)$ for some $x \in \mathbb{V}$. We can now express the familiar external and internal choice of session types as the (partial) operations $+$ and $\oplus$ defined by

$$?f + ?g \stackrel{\text{def}}{=} ?(f \sqcup g) \qquad !f \oplus !g \stackrel{\text{def}}{=} !(f \sqcup g)$$

It it easy to see that $+$ and $\oplus$ are commutative and associative and respectively have $?\mathsf{end}$ and $!\mathsf{end}$ as units. We assume that they bind less tightly than the "." in continuations. Finally, we let $\mathsf{dom}(f) \stackrel{\text{def}}{=} \{x \in \mathbb{V} \mid f(x) \neq \mathsf{nil}\}$ be the *proper domain* of the continuation f, namely the set of messages for which f yields a session type other than nil.

▶ **Example 3.3.** The session types T_1 and S_1 that satisfy the equations

$$T_1 = !\mathsf{true}.!\mathbb{N}.T_1 \oplus !\mathsf{false}.?\mathsf{end} \qquad S_1 = !\mathsf{true}.!\mathbb{N}^+.S_1 \oplus !\mathsf{false}.?\mathsf{end}$$

both describe a channel used for sending a boolean. If the boolean is false, the communication stops immediately ($?\mathsf{end}$). If it is true, the channel is used for sending a natural number (a strictly positive one in S_1) and then according to T_1 or S_1 again. Notice how the structure of the protocol after the output of the boolean depends on the *value* of the boolean.

The session types T_2 and S_2 that satisfy the equations

$$T_2 = ?\mathsf{true}.!\mathbb{N}.T_2 + ?\mathsf{false}.?\mathsf{end} \qquad S_2 = ?\mathsf{true}.!\mathbb{N}^+.S_2 + ?\mathsf{false}.?\mathsf{end}$$

differ from T_1 and S_1 in that the channel they describe is used initially for *receiving* a boolean.

As a final example, the session type $T_3 = !f$ where

$$f(0) = T_3 \qquad f(n+1) = !\mathbb{B}.f(n)$$

describes an channel used for sending streams of sequences beginning with a natural number n followed by n boolean messages. ⌟

We define the operational semantics of session types by means of a *labeled transition system*. *Labels*, ranged over by α, β, γ, have either the form $?x$ (input of message x) or the form $!x$ (output of message x). Transitions $T \xrightarrow{\alpha} S$ are defined by the following axioms:

[T-INPUT] [T-OUTPUT]

$$\frac{}{?f \xrightarrow{?x} f(x)} \qquad \frac{}{!f \xrightarrow{!x} f(x)}\ x \in \mathsf{dom}(f)$$

There is a fundamental asymmetry between send and receive operations: the act of sending a message is *active* – the sender may choose the message to send – while the act of receiving a message is *passive* – the receiver cannot cherry-pick the message being received. We model this asymmetry with the side condition $x \in \mathsf{dom}(f)$ in [T-OUTPUT] and the lack thereof in [T-INPUT]: a process that uses a session channel according to $!f$ refrains from sending a message x if $x \notin \mathsf{dom}(f)$, whereas a process that uses a session channel according to $?f$ cannot decide which message x it will receive, but the session channel becomes unusable if an unexpected message arrives. These transition rules allow us to appreciate a little more the difference between !end and ?end. While both describe a session endpoint on which no further communications may occur, !end is "more robust" than ?end since it has no transitions, whereas ?end is "more fragile" than !end since it performs transitions, all of which lead to nil. For this reason, we use !end to flag successful session termination (Section 5), whereas ?end only means that the protocol has ended.

To describe *sequences* of consecutive transitions performed by a session type we use another relation $\stackrel{\varphi}{\Longrightarrow}$ where φ and ψ range over strings of labels. As usual, ε denotes the empty string and juxtaposition denotes string concatenation. The relation $\stackrel{\varphi}{\Longrightarrow}$ is the least one such that $T \stackrel{\varepsilon}{\Longrightarrow} T$ and if $T \xrightarrow{\alpha} S$ and $S \stackrel{\varphi}{\Longrightarrow} R$, then $T \stackrel{\alpha\varphi}{\Longrightarrow} R$.

4 Weak Termination

A session type is weakly terminating if it preserves the possibility of reaching !end or ?end along all of its transitions that do not lead to nil. Weak termination of T does not necessarily imply that there exists an upper bound to the length of communications that follow the protocol T, but it guarantees the absence of "infinite loops" whereby the communication is forced to continue forever.

To formalize weak termination we need the notion of *trace*, which is a finite sequence of actions performed on a session channel while preserving usability of the channel.

▶ **Definition 4.1** (traces and maximal traces). *The* traces *of* T *are defined as* $\mathtt{tr}(T) \stackrel{\text{def}}{=} \{\varphi \mid \exists S : T \stackrel{\varphi}{\Longrightarrow} S \neq \mathsf{nil}\}$. *We say that* $\varphi \in \mathtt{tr}(T)$ *is* maximal *if* $\varphi\psi \in \mathtt{tr}(T)$ *implies* $\psi = \varepsilon$.

For example, we have $\mathtt{tr}(\mathsf{nil}) = \emptyset$ and $\mathtt{tr}(!\mathsf{end}) = \mathtt{tr}(?\mathsf{end}) = \{\varepsilon\}$. Note that !end and ?end have the same traces but different transitions (hence different behaviors). A *maximal trace* is a trace that cannot be extended any further. For example ε is a maximal trace of both !end and ?end but not of $!\mathbb{B}.?\mathsf{end}$ whereas !true and !false are maximal traces of $!\mathbb{B}.?\mathsf{end}$.

$$[\text{NIL}]\ \frac{}{\mathsf{nil}\Downarrow} \qquad [\text{IN}]\ \frac{f(x)\Downarrow \ (\forall x \in \mathbb{V})}{?f\Downarrow} \qquad [\text{OUT}]\ \frac{f(x)\Downarrow \ (\forall x \in \mathbb{V})}{!f\Downarrow}$$

$$[\text{CO-IN}]\ \frac{f(x)\Downarrow}{?f\Downarrow}\ x \in \mathsf{dom}(f) \qquad [\text{CO-OUT}]\ \frac{f(x)\Downarrow}{!f\Downarrow}\ x \in \mathsf{dom}(f)$$

Figure 1 Generalized inference system $\langle \mathcal{T}, \mathcal{T}_{\mathsf{co}} \rangle$ for weak termination.

▶ **Definition 4.2** (weak termination). *We say that T is* weakly terminating *if, for every $\varphi \in \mathtt{tr}(T)$, there exists ψ such that $\varphi\psi \in \mathtt{tr}(T)$ and $\varphi\psi$ is maximal.*

▶ **Example 4.3.** All of the session types presented in Example 3.3 except T_3 are weakly terminating. The session type T_3 is not weakly terminating because no trace of T_3 can be extended to a maximal one. Note that also $S_3 \stackrel{\text{def}}{=} !\mathsf{true}.T_3 \oplus !\mathsf{false}.?\mathsf{end}$ is not weakly terminating, even though there is a path leading to $?\mathsf{end}$, because weak termination must be *preserved* along all possible transitions of the session type, whereas $S_3 \xrightarrow{!\mathsf{true}} T_3$ and T_3 is not weakly terminating. Finally, nil is trivially weakly terminating since it has no trace. ⌟

To find an inference system for weak termination observe that the set $\mathbb{W}$ of weakly terminating session types is the largest one that satifies the following two properties: (1) it must be possible to reach either $!\mathsf{end}$ or $?\mathsf{end}$ from every $T \in \mathbb{W} \setminus \{\mathsf{nil}\}$; (2) the set $\mathbb{W}$ must be closed by transitions, namely if $T \in \mathbb{W}$ and $T \xrightarrow{\alpha} S$ then $S \in \mathbb{W}$. Neither of these two properties, taken in isolation, suffices to define $\mathbb{W}$: the session type S_3 from Example 4.3 enjoys property (1) but is not weakly terminating; the set $\mathbb{S}$ is obviously the largest one with property (2), but not every session type is weakly terminating. This suggests the definition of $\mathbb{W}$ as the largest subset of $\mathbb{S}$ satisfying (2) and whose elements are *bounded* by property (1), which is precisely what corules allow us to specify.

Figure 1 shows a GIS $\langle \mathcal{T}, \mathcal{T}_{\mathsf{co}} \rangle$ for weak termination, where $\mathcal{T}$ consists of all the (singly-lined) rules whereas $\mathcal{T}_{\mathsf{co}}$ consists of all the (doubly-lined) corules (we will follow these naming and syntactic conventions also in the subsequent GISs). The axiom [NIL] indicates that nil is weakly terminating in a trivial way (it has no trace), while rules [IN] and [OUT] indicate that weak termination is closed by transitions. Note that these three rules, interpreted coinductively, are satisfied by all session types, hence $\{T \mid T\Downarrow \in \mathsf{CoInd}[\![\mathcal{T}]\!]\} = \mathbb{S}$.

▶ **Theorem 4.4.** *T is weakly terminating if and only if $T\Downarrow \in \mathsf{Gen}[\![\mathcal{T}, \mathcal{T}_{\mathsf{co}}]\!]$.*

The proof of the "if" part of Theorem 4.4 crucially relies on the corules to extend each trace of T to a maximal one. Indeed, suppose $T\Downarrow \in \mathsf{Gen}[\![\mathcal{T}, \mathcal{T}_{\mathsf{co}}]\!]$ and consider a trace $\varphi \in \mathtt{tr}(T)$. That is, $T \stackrel{\varphi}{\Longrightarrow} S$ for some $S \neq \mathsf{nil}$. Using [IN] and [OUT] we deduce $S\Downarrow \in \mathsf{Gen}[\![\mathcal{T}, \mathcal{T}_{\mathsf{co}}]\!]$ by means of a simple induction on φ. Now $S\Downarrow \in \mathsf{Gen}[\![\mathcal{T}, \mathcal{T}_{\mathsf{co}}]\!]$ implies $S\Downarrow \in \mathsf{Ind}[\![\mathcal{T} \cup \mathcal{T}_{\mathsf{co}}]\!]$ by Definition 2.3. Another induction on the (well-founded) derivation of this judgment, along with the witness messages of [CO-IN] and [CO-OUT], allows us to find ψ such that $\varphi\psi$ is a maximal trace of T.

5 Compliance

In this section we define and characterize two *compliance* relations for session types, which formalize the "successful" interaction between a client and a server connected by a session. The notion of "successful interaction" that we consider is biased towards client satisfaction, but see Remark 6.7 below for a discussion about alternative notions. To formalize compliance we need to model the evolution of a session as client and server interact. To this aim, we represent a session as a term $R \parallel T$ where R describes the behavior of the client and T that of the server. Sessions reduce according to the rule

$$\frac{}{R \parallel T \to R' \parallel S'} \text{ if } R \xrightarrow{\overline{\alpha}} R' \text{ and } T \xrightarrow{\alpha} T'$$

where $\overline{\alpha}$ is the *complementary action* of α defined by $\overline{?x} = !x$ and $\overline{!x} = ?x$. We extend $\overline{\cdot}$ to traces in the obvious way and we write $\Rightarrow$ for the reflexive, transitive closure of $\to$. We write $R \parallel T \to$ if $R \parallel T \to R' \parallel T'$ for some R' and T' and $R \parallel T \not\to$ if not $R \parallel T \to$.

The first compliance relation that we consider requires that, if the interaction in a session stops, it is because the client "is satisfied" and the server "has not failed" (recall that a session type can turn into nil only if an unexpected message is received). Formally:

▶ **Definition 5.1** (compliance). *We say that R is* compliant *with T if $R \parallel T \Rightarrow R' \parallel T' \not\to$ implies $R' = !\mathsf{end}$ and $T' \neq \mathsf{nil}$.*

This notion of compliance is an instance of *safety property* in which the invariant being preserved at any stage of the interaction is that either client and server are able to synchronize further, or the client is satisfied and the server has not failed.

The second compliance relation that we consider adds a *liveness* requirement namely that, no matter how long client and server have been interacting with each other, it is always possible to reach a configuration in which the client is satisfied and the server has not failed.

▶ **Definition 5.2** (fair compliance). *We say that R is* fair compliant *with T if $R \parallel T \Rightarrow R' \parallel T'$ implies $R' \parallel T' \Rightarrow !\mathsf{end} \parallel T''$ with $T'' \neq \mathsf{nil}$.*

It is easy to show that fair compliance implies compliance, but there exist compliant session types that are not fair compliant, as illustrated in the following example.

▶ **Example 5.3.** Recall Example 3.3 and consider the session types R_1 and R_2 such that

$$R_1 = ?\mathsf{true}.?\mathbb{N}.R_1 + ?\mathsf{false}.!\mathsf{end} \qquad R_2 = !\mathsf{true}.(?0.!\mathsf{end} + ?\mathbb{N}^+.R_2)$$

Then R_1 is fair compliant with both T_1 and S_1 and R_2 is compliant with both T_2 and S_2. Even if S_1 exhibits fewer behaviors compared to T_1 (it never sends 0 to the client), at the beginning of a new iteration it can always send false and steer the interaction along a path that leads R_1 to success. On the other hand, R_2 is fair compliant with T_2 but not with S_2. In this case, the client insists on sending true to the server in hope to receive 0, but while this is possible with the server T_2, the server S_2 only sends strictly positive numbers.

This example also shows that weak termination of both client and server is not sufficient, in general, to guarantee fair compliance. Indeed, both R_2 and S_2 are weakly terminating, but they are not fair compliant. The reason is that the sequences of actions leading to !end on the client side are not necessarily the same (complemented) traces that lead to ?end on the server side. Fair compliance takes into account the synchronizations that can actually occur between client and server. ⌟

Figure 2 presents the GIS $\langle \mathcal{C}, \mathcal{C}_{\mathsf{co}} \rangle$ for fair compliance. Rule [WIN] relates a satisfied client with a non-failed server. Rules [IN-OUT] and [OUT-IN] require that, no matter which message is exchanged between client and server, the respective continuations are still fair compliant. The side conditions $\mathsf{dom}(f) \neq \emptyset$ and $\mathsf{dom}(g) \neq \emptyset$ guarantee progress by making sure that the sender is capable of sending at least one message. As we will see, the coinductive interpretation of $\mathcal{C}$, which consists of these three rules, completely characterizes compliance (Definition 5.1). However, these rules do not ensure that the interaction between client and server can always reach a successful configuration as required by Definition 5.2. For this, the corules [CO-IN-OUT] and [CO-OUT-IN] are essential.

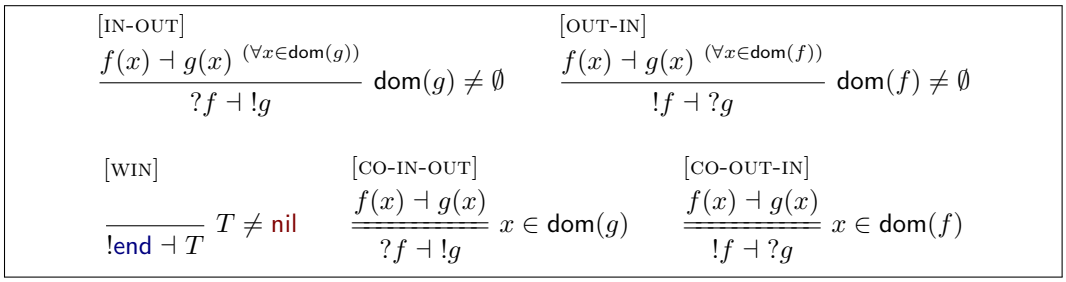


Figure 2 Generalized inference system $\langle \mathcal{C}, \mathcal{C}_{\mathsf{co}} \rangle$ for fair compliance.

▶ **Theorem 5.4** (compliance). *For every $R, T \in \mathbb{S}$, the following properties hold:*

1. *R is compliant with T if and only if $R \dashv T \in \mathsf{CoInd}[\![\mathcal{C}]\!]$;*
2. *R is fair compliant with T if and only if $R \dashv T \in \mathsf{Gen}[\![\mathcal{C}, \mathcal{C}_{\mathsf{co}}]\!]$.*

To illustrate the role of the corules, let us sketch the proof that the GIS in Figure 2 is sound with respect to fair compliance. Suppose that $R \dashv T \in \mathsf{Gen}[\![\mathcal{C}, \mathcal{C}_{\mathsf{co}}]\!]$ and consider a reduction $R \parallel T \Rightarrow R' \parallel T'$. An induction on the length of this reduction, along with [IN-OUT] and [OUT-IN], allows us to deduce $R' \dashv T' \in \mathsf{Gen}[\![\mathcal{C}, \mathcal{C}_{\mathsf{co}}]\!]$. Then we have $R' \dashv T' \in \mathsf{Ind}[\![\mathcal{C} \cup \mathcal{C}_{\mathsf{co}}]\!]$ by Definition 2.3. An induction on this (well-founded) derivation allows us to find a reduction $R' \parallel T' \Rightarrow !\mathsf{end} \parallel T''$ such that $T'' \neq \mathsf{nil}$.

Observe that the corules are at once essential and unsound. For example, without them we would be able to derive the judgment $R_2 \dashv S_2$ despite the fact that R_2 is not fair compliant with S_2 (Example 5.3). At the same time, if we treated corules as plain rules, we would be able to derive the judgment $!\mathbb{N}.!\mathsf{end} \dashv ?0.?\mathsf{end}$ despite the reduction $!\mathbb{N}.!\mathsf{end} \parallel ?0.?\mathsf{end} \to !\mathsf{end} \parallel \mathsf{nil}$ since *there exists* an interaction that leads to the successful configuration $!\mathsf{end} \parallel ?\mathsf{end}$ (if the client sends 0) but none of the others does.

6 Subtyping

The notions of compliance given in Section 5 induce corresponding semantic notions of subtyping, which embed a substitution principle for session types. The key idea is the same used for defining testing equivalences for processes [26, 20, 31], except that we use the term "client" instead of the term "test". Therefore, T is a subtype of S if any client that successfully interacts with T does so with S as well.

▶ **Definition 6.1** (subtyping). *We say that T is a* subtype *of S if R compliant with T implies R compliant with S for every R.*

▶ **Definition 6.2** (fair subtyping). *We say that T is a* fair subtype *of S if R fair compliant with T implies R fair compliant with S for every R.*

According to these definitions, when T is a (fair) subtype of S, a process that behaves according to T can be replaced by a process that behaves according to S without compromising (fair) compliance with the clients of T. At first sight this substitution principle appears to be just the opposite of the expected/intended one, whereby it is safe to use a session channel of type T where a session channel of type S is expected if T is a subtype of S. The mismatch is only apparent, however, and can be explained by looking carefully at the entities being replaced in the substitution principles recalled above (processes in one case, session channels in the other). The interested reader may refer to Gay [18] for a nice study of these two

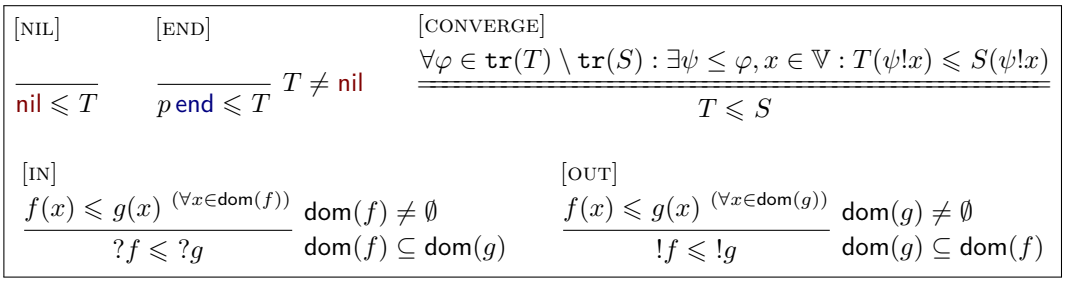


Figure 3 Generalized inference system $\langle \mathcal{F}, \mathcal{F}_{\mathsf{co}} \rangle$ for fair subtyping.

different, yet related viewpoints. What matters here is that the above notions of subtyping are "correct by definition" but do not provide any hint as to the shape of two session types T and S that are related by (fair) subtyping. This problem is well known in the semantic approaches for defining subtyping relations [17, 7] as well as in the aforementioned testing theories for processes [26, 20, 31], which the two definitions above are directly inspired from. Therefore, it is of paramount importance to provide equivalent characterizations of these relations, particularly in the form of inference systems.

The GIS $\langle \mathcal{F}, \mathcal{F}_{\mathsf{co}} \rangle$ for (fair) subtyping is shown in Figure 3 and described hereafter. Rule [NIL] states that nil is the least element of the subtyping preorder, which is justified by the fact that no client successfully interacts with nil. Rule [END] establishes that ?end and !end are the least elements among all session types different from nil. In our theory, this relation arises from the asymmetric form of compliance we have considered: a server $p\,\mathsf{end}$ satisfies only !end, which successfully interacts with any server different from nil. Rules [IN] and [OUT] indicate that inputs are covariant and outputs are contravariant. That is, it is safe to replace a server with another one that receives a superset of messages ($\mathsf{dom}(f) \subseteq \mathsf{dom}(g)$ in [IN]) and, dually, it is safe to replace a server with another one that sends a subset of messages ($\mathsf{dom}(g) \subseteq \mathsf{dom}(f)$ in [OUT]). The side condition $\mathsf{dom}(g) \neq \emptyset$ in [OUT] is important to preserve progress: if the server that behaves according to the larger session type is unable to send any message, the client may get stuck waiting for a message that is never sent. On the other hand, the side condition $\mathsf{dom}(f) \neq \emptyset$ is unnecessary from a purely technical view point, since the rule [IN] without this side condition is subsumed by [END]. We have included the side condition to minimize the overlap between different rules and for symmetry with respect to [OUT]. Overall, these rules are aligned with those of the subtyping relation for session types given by Gay and Hole [19] (see also Remark 6.7).

To discuss the corule [CONVERGE] that characterizes fair subtyping we need to introduce one last piece of notation concerning session types.

▶ **Definition 6.3** (residual of a session type). *Given a session type T and a trace φ of T we write $T(\varphi)$ for the* residual *of T after φ, namely for the session type S such that $T \stackrel{\varphi}{\Longrightarrow} S$.*

The notion of residual is well defined since session types are *deterministic*: if $T \stackrel{\varphi}{\Longrightarrow} S_1$ and $T \stackrel{\varphi}{\Longrightarrow} S_2$, then $S_1 = S_2$. It is implied that $T(\varphi)$ is undefined if $\varphi \notin \mathtt{tr}(T)$.

For the sake of presentation we describe the corule [CONVERGE] incrementally, showing how it contributes to the soundness proof of the GIS in Figure 3. In doing so, it helps bearing in mind that the relation $T \leqslant S$ is meant to preserve fair compliance (Definition 5.2), namely the possibility that any client of T can terminate successfully when interacting with S. As a first approximation observe that, when the traces of T are included in the traces of S, the corule [CONVERGE] boils down to the following coaxiom:

$$\frac{}{T \leqslant S}\ \mathtt{tr}(T) \subseteq \mathtt{tr}(S)$$

Now consider a client R that is fair compliant with T. It must be the case that $R \parallel T \Rightarrow !\mathsf{end} \parallel T'$ for some $T' \neq \mathsf{nil}$, namely that $R \stackrel{\overline{\varphi}}{\Longrightarrow} !\mathsf{end}$ and $T \stackrel{\varphi}{\Longrightarrow} T'$ for some sequence φ of actions. The side condition $\mathtt{tr}(T) \subseteq \mathtt{tr}(S)$ ensures that φ is also a trace of S, therefore $R \parallel S \Rightarrow !\mathsf{end} \parallel S'$ for the $S' \neq \mathsf{nil}$ such that $S \stackrel{\varphi}{\Longrightarrow} S'$. In general, we know from rule [OUT] that S may perform *fewer* outputs than T, hence not every trace of T is necessarily a trace of S. Writing $\leq$ for the prefix order relation on traces, the premises

$$\forall \varphi \in \mathtt{tr}(T) \setminus \mathtt{tr}(S) : \exists \psi \leq \varphi, x \in \mathbb{V} : T(\psi!x) \leqslant S(\psi!x)$$

of [CONVERGE] make sure that, for every trace φ of T that is not a trace of S, there exists a common prefix ψ of T and S and an output action $!x$ shared by both $T(\psi)$ and $S(\psi)$ such that the residuals of T and S after $\psi!x$ are *one level closer*, in the proof tree for $T \leqslant S$, to the residuals of T and S for which trace inclusion holds. The fact that ψ must be followed by an *output* $!x$ is fundamental, since the client R *must* be able to accept all the outputs of T.

Note that the corule is unsound in general. For instance, $!0.?\mathsf{end} \leqslant !\mathbb{N}.?\mathsf{end}$ is derivable by [CONVERGE] since $\mathtt{tr}(!0.?\mathsf{end}) \subseteq \mathtt{tr}(!\mathbb{N}.?\mathsf{end})$, but $!0.?\mathsf{end}$ is *not* a subtype of $!\mathbb{N}.?\mathsf{end}$.

▶ **Example 6.4.** Consider once again the session types T_i and S_i of Example 3.3. It is easy to see that $T_i \leqslant S_i \in \mathsf{CoInd}[\![\mathcal{F}]\!]$ for $i = 1, 2$. In order to derive $T_i \leqslant S_i$ in the GIS $\langle \mathcal{F}, \mathcal{F}_{\mathsf{co}} \rangle$ we must find a well-founded proof tree of $T \leqslant S$ in $\mathcal{F} \cup \mathcal{F}_{\mathsf{co}}$ and the only hope to do so is by means of [CONVERGE], since T_i and S_i share traces of arbitrary length. Observe that every trace φ of T_1 that is not a trace of S_1 has the form $(!\mathsf{true}!p_k)^k !\mathsf{true}!0 \dots$ where $p_k \in \mathbb{N}^+$. Thus, it suffices to take $\psi = \varepsilon$ and $x = 0$, noted that $T_1(!0) = S_1(!0) = ?\mathsf{end}$, to derive

$$\cfrac{\cfrac{}{?\mathsf{end} \leqslant ?\mathsf{end}}}{T_1 \leqslant S_1}$$

with two applications of [CONVERGE]. On the other hand, every trace $\varphi \in \mathtt{tr}(T_2) \setminus \mathtt{tr}(S_2)$ has the form $(?\mathsf{true}!p_k)^k ?\mathsf{true}!0 \dots$ where $p_k \in \mathbb{N}^+$. All the prefixes of such traces that are followed by an output and are shared by both T_2 and S_2 have the form $(?\mathsf{true}!p_k)^k ?\mathsf{true}$ where $p_k \in \mathbb{N}^+$, and $T_2(\psi!p) = T_2$ and $S_2(\psi!p) = S_2$ for all such prefixes and $p \in \mathbb{N}^+$. It follows that we are unable to derive $T_2 \leqslant S_2$ with a well-founded proof tree in $\mathcal{F} \cup \mathcal{F}_{\mathsf{co}}$. This is consistent with the fact that, in Example 5.3, we have found a client R_2 that is fair compliant with T_2 but not with S_2. Intuitively, R_2 insists on poking the server waiting to receive 0. This can eventually happen with T_2, but not with S_2. In the case of T_1 and S_1 no such client can exist, since the server may decide to interrupt the interaction at any time by sending a false message to the client. ⌟

Example 6.4 also shows that fair subtyping is a *context-sensitive* relation in that the applicability of a rule for deriving $T \leqslant S$ may depend on the context in which T and S occur. For instance, in the non-well-founded derivation

$$\cfrac{\cfrac{\cfrac{\vdots}{T_1 \leqslant S_1}}{!\mathbb{N}.T_1 \leqslant !\mathbb{N}^+.S_1}\,[\text{OUT}] \qquad \cfrac{}{?\mathsf{end} \leqslant ?\mathsf{end}}\,[\text{END}]}{T_1 \leqslant S_1}\,[\text{OUT}] \tag{1}$$

the rule [OUT] is used infinitely many times to relate the output session types $!\mathbb{N}.T_1$ and $!\mathbb{N}^+.S_2$. In this context, rule [OUT] can be applied harmlessly. On the contrary, if we attempt to find a derivation for $T_2 \leqslant S_2$ we obtain the non-well-founded tree

$$\dfrac{\dfrac{\dfrac{\vdots}{T_2 \leqslant S_2}}{!\mathbb{N}.T_2 \leqslant !\mathbb{N}^+.S_2}\,[\text{OUT}] \qquad \dfrac{}{?\mathsf{end} \leqslant ?\mathsf{end}}\,[\text{END}]}{T_2 \leqslant S_2}\,[\text{IN}] \tag{2}$$

which is isomorphic to the one shown in Equation (1) with the difference that some applications of [OUT] have been replaced by applications of [IN]. Here too [OUT] is used infinitely many times, but this time to relate the output session types $!\mathbb{N}.T_2$ and $!\mathbb{N}^+.S_2$. This derivation allows us to prove $T_2 \leqslant S_2 \in \mathsf{CoInd}[\![\mathcal{F}]\!]$, but not $T_2 \leqslant S_2 \in \mathsf{Gen}[\![\mathcal{F}, \mathcal{F}_{\mathsf{co}}]\!]$, because [OUT] removes the 0 output from S_2 that a client of T_2 may depend upon.

▶ Remark 6.5. As observed by one reviewer, rule [CONVERGE] is hard not only to understand, but also to formalize in Agda. We have been unable to conceive a sound and complete GIS for fair subtyping that is based on simpler corules, but it should be noted that the property enforced by [CONVERGE] is fundamentally *non-local* and therefore difficult to express in terms of immediate subtrees of a session type. To illustrate the point, consider the following alternative set of corules meant to replace [CONVERGE] in Figure 3:

$$[\text{CO-INC}]\ \dfrac{}{T \leqslant S}\ \mathtt{tr}(T) \subseteq \mathtt{tr}(S) \qquad [\text{CO-IN}]\ \dfrac{f(x) \leqslant g(x)\ {}^{(\forall x \in \mathsf{dom}(f))}}{?f \leqslant ?g} \qquad [\text{CO-OUT}]\ \dfrac{f(x) \leqslant g(x)}{!f \leqslant !g}\ x \in \mathsf{dom}(f) \cap \mathsf{dom}(g)$$

It is easy to see that these rules provide a sound approximation of [CONVERGE], but they are not complete. Indeed, consider the session types $T = ?\mathsf{true}.T + ?\mathsf{false}.(!\mathsf{true}.?\mathsf{end} \oplus !\mathsf{false}.?\mathsf{end})$ and $S = ?\mathsf{true}.S + ?\mathsf{false}.!\mathsf{true}.?\mathsf{end}$. We have $T \leqslant S$ and yet $T \leqslant_{\mathsf{Ind}} S$ cannot be proved with the above corules: it is not possible to prove $T \leqslant_{\mathsf{Ind}} S$ using [CO-INC] because $\mathtt{tr}(T) \not\subseteq \mathtt{tr}(S)$. If, on the other hand, we insist on visiting both branches of the topmost input as required by [CO-IN], we end up requiring a proof of $T \leqslant_{\mathsf{Ind}} S$ in order to derive $T \leqslant_{\mathsf{Ind}} S$. ⌟

▶ **Theorem 6.6.** *For every $T, S \in \mathbb{S}$ the following properties hold:*

1. *T is a subtype of S if and only if $T \leqslant S \in \mathsf{CoInd}[\![\mathcal{F}]\!]$;*
2. *T is a fair subtype of S if and only if $T \leqslant S \in \mathsf{Gen}[\![\mathcal{F}, \mathcal{F}_{\mathsf{co}}]\!]$.*

▶ Remark 6.7. Most session type theories adopt a symmetric form of session type compatibility whereby client and server are required to terminate the interaction at the same time. It is easy to define a notion of *symmetric compliance* (also known as *peer compliance* [7]) by turning $T' \neq \mathsf{nil}$ into $T' = ?\mathsf{end}$ in Definition 5.1. The subtyping relation induced by symmetric compliance has essentially the same characterization of Definition 6.1, except that the axiom [END] is replaced by the more familiar $p\,\mathsf{end} \leqslant q\,\mathsf{end}$ [19]. On the other hand, the analogous change in Definition 5.2 has much deeper consequences: the requirement that client and server must end the interaction at the same time creates a large family of session types that are syntactically very different, but semantically equivalent. For example, the session types T and S such that $T = ?\mathbb{N}.T$ and $S = !\mathbb{B}.S$, which describe completely unrelated protocols, would be equivalent for the simple reason that no client successfully interacts with them (they are not weakly terminating, since they do not contain any occurrence of end). We have not investigated the existence of a GIS for fair subtyping induced by symmetric fair compliance. A partial characterization (which however requires various auxiliary relations) is given by Padovani [30]. ⌟

7 Concluding Remarks

We have shown that generalized inference systems are an effective framework for defining sound and complete proof systems of (some) combined safety and liveness properties of (dependent) session types (Definitions 4.2 and 5.2), as well as of a liveness-preserving subtyping relation (Definition 6.2). We think that this achievement is more than a coincidence. One of the fundamental results in model checking states that every property can be expressed as the conjunction of a safety property and a liveness property [2, 3, 6]. The connections between safety and liveness on one side and coinduction and induction on the other make GISs appropriate for characterizing combined safety and liveness properties.

Murgia [25] studies a wide range of compliance relations for processes and session types, showing that many of them are fixed points of a functional operator, but not necessarily the least or the greatest ones. In particular, he shows that *progress compliance*, which is akin to our compliance (Definition 5.1), is a greatest fixed point and that *should-testing compliance*, which is akin to our fair compliance (Definition 5.2), is an intermediate fixed point. These results are consistent with Theorem 5.4. We have extended these results to subtyping (Definition 6.1) and fair subtyping (Definition 6.2). Previous alternative characterizations of fair subtyping and the related *should-testing preorder* either require several different relations [29, 30] or are denotational in nature [31] and therefore not as insightful as desirable. Using GISs, we have obtained complete characterizations of fair compliance and fair subtyping by simply *adding a few corules* to the proof systems of their "unfair" counterparts.

We have coded all the notions and results discussed in the paper in Agda [27], thus providing the first machine-checked formalization of liveness properties and liveness-preserving subtyping relations for dependent session types. Theorem 4.4 and item (2) of Theorems 5.4 and 6.6 are proved considering $\mathbb{V} = \mathbb{B}$ instead of an arbitrary set of values. This is because the version of the Agda library for GISs [11, 13] used for the formalization does not support (co)rules with infinitely many premises, which are necessary if $\mathbb{V}$ is infinite. However, all of the key aspects of the characterizations of weak termination, fair compliance and fair subtyping already emerge in this simplified setting. The Agda formalization is not entirely constructive since it makes use of three postulates: the *law of excluded middle*, the *extensionality axiom* and the duality between a universally quantified, inductive characterization of *convergence* (see [CONVERGE] in Figure 3) and its negation, which is characterized using an existentially quantified, coinductive definition. Note that the Agda library for GISs is a standalone development [11, 13, 12], on top of which we have built our own [15]. This makes it easy to extend our results to other families of processes or to different properties.

In this paper we have focused on properties of session types alone. The most important piece of future work that we plan to carry out next is the development of a session type system making use of fair subtyping for the enforcement of liveness properties of processes. This problem has remained open for a long time [29, 30] because the integration of fair subtyping into a coinductively-interpreted session type system is (unsurprisingly) challenging. By contrast, session type systems making use of safety-preserving subtyping relations are quite widespread [19, 9, 23, 10]. The achievements described in this paper suggest that GISs could provide just the right framework for defining such type system. Somewhat connected with this future development is also the handling of delegation and therefore of higher-order session types. Previous developments [30] have shown that delegation is orthogonal to the characterizing features of fair subtyping, since the communication of session channels does not (usually) affect the branching structure of session types (but there are a few exceptions [9, 10]). For this reason, we think that this extension can be accounted for without substantial issues.

References

1 Peter Aczel. An introduction to inductive definitions. In Jon Barwise, editor, *Handbook of Mathematical Logic*, volume 90 of *Studies in Logic and the Foundations of Mathematics*, pages 739–782. Elsevier, 1977. doi:10.1016/S0049-237X(08)71120-0.

2 Bowen Alpern and Fred B. Schneider. Defining liveness. *Inf. Process. Lett.*, 21(4):181–185, 1985. doi:10.1016/0020-0190(85)90056-0.

3 Bowen Alpern and Fred B. Schneider. Recognizing safety and liveness. *Distributed Comput.*, 2(3):117–126, 1987. doi:10.1007/BF01782772.

4 Davide Ancona, Viviana Bono, Mario Bravetti, Joana Campos, Giuseppe Castagna, Pierre-Malo Deniélou, Simon J. Gay, Nils Gesbert, Elena Giachino, Raymond Hu, Einar Broch Johnsen, Francisco Martins, Viviana Mascardi, Fabrizio Montesi, Rumyana Neykova, Nicholas Ng, Luca Padovani, Vasco T. Vasconcelos, and Nobuko Yoshida. Behavioral types in programming languages. *Found. Trends Program. Lang.*, 3(2-3):95–230, 2016. doi:10.1561/2500000031.

5 Davide Ancona, Francesco Dagnino, and Elena Zucca. Generalizing inference systems by coaxioms. In Hongseok Yang, editor, *Programming Languages and Systems - 26th European Symposium on Programming, ESOP 2017, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2017, Uppsala, Sweden, April 22-29, 2017, Proceedings*, volume 10201 of *Lecture Notes in Computer Science*, pages 29–55. Springer, 2017. doi:10.1007/978-3-662-54434-1_2.

6 Christel Baier and Joost-Pieter Katoen. *Principles of model checking*. MIT Press, 2008.

7 Giovanni Bernardi and Matthew Hennessy. Using higher-order contracts to model session types. *Log. Methods Comput. Sci.*, 12(2), 2016. doi:10.2168/LMCS-12(2:10)2016.

8 Julian C. Bradfield and Colin Stirling. Modal mu-calculi. In Patrick Blackburn, J. F. A. K. van Benthem, and Frank Wolter, editors, *Handbook of Modal Logic*, volume 3 of *Studies in logic and practical reasoning*, pages 721–756. North-Holland, 2007. doi:10.1016/s1570-2464(07)80015-2.

9 Giuseppe Castagna, Mariangiola Dezani-Ciancaglini, Elena Giachino, and Luca Padovani. Foundations of session types. In António Porto and Francisco Javier López-Fraguas, editors, *Proceedings of the 11th International ACM SIGPLAN Conference on Principles and Practice of Declarative Programming, September 7-9, 2009, Coimbra, Portugal*, pages 219–230. ACM, 2009. doi:10.1145/1599410.1599437.

10 Giuseppe Castagna, Mariangiola Dezani-Ciancaglini, Elena Giachino, and Luca Padovani. Foundations of session types: 10 years later. In Ekaterina Komendantskaya, editor, *Proceedings of the 21st International Symposium on Principles and Practice of Programming Languages, PPDP 2019, Porto, Portugal, October 7-9, 2019*, pages 1:1–1:3. ACM, 2019. doi:10.1145/3354166.3356340.

11 Luca Ciccone. Flexible coinduction in agda. Master's thesis, DIBRIS, Università di Genova, Italy, 2020. arXiv:2002.06047.

12 Luca Ciccone, Francesco Dagnino, and Elena Zucca. Flexible coinduction in Agda. In Schloss Dagstuhl Leibniz-Zentrum für Informatik, editor, *Proceedings of the 12th conference on Interactive Theorem Proving, ITP 2021*, 2021. to appear.

13 Luca Ciccone, Francesco Dagnino, and Elena Zucca. Inference Systems in Agda, 2021. URL: https://github.com/LcicC/inference-systems-agda [cited Feb 1, 2021].

14 Luca Ciccone and Luca Padovani. A Dependently Typed Linear π-Calculus in Agda. In *PPDP'20: 22nd International Symposium on Principles and Practice of Declarative Programming, Bologna, Italy, 9-10 September, 2020*, pages 8:1–8:14. ACM, 2020. doi:10.1145/3414080.3414109.

15 Luca Ciccone and Luca Padovani. Fair Subtyping in Agda, 2021. URL: https://github.com/boystrange/FairSubtypingAgda/tree/v1.0 [cited May 1, 2021].

16 Francesco Dagnino. Coaxioms: flexible coinductive definitions by inference systems. *Log. Methods Comput. Sci.*, 15(1), 2019. doi:10.23638/LMCS-15(1:26)2019.

17 Alain Frisch, Giuseppe Castagna, and Véronique Benzaken. Semantic subtyping: Dealing set-theoretically with function, union, intersection, and negation types. *J. ACM*, 55(4):19:1–19:64, 2008. doi:10.1145/1391289.1391293.

18 Simon J. Gay. Subtyping supports safe session substitution. In Sam Lindley, Conor McBride, Philip W. Trinder, and Donald Sannella, editors, *A List of Successes That Can Change the World - Essays Dedicated to Philip Wadler on the Occasion of His 60th Birthday*, volume 9600 of *Lecture Notes in Computer Science*, pages 95–108. Springer, 2016. doi:10.1007/978-3-319-30936-1_5.

19 Simon J. Gay and Malcolm Hole. Subtyping for session types in the pi calculus. *Acta Informatica*, 42(2-3):191–225, 2005. doi:10.1007/s00236-005-0177-z.

20 Matthew Hennessy. *Algebraic theory of processes*. MIT Press series in the foundations of computing. MIT Press, 1988.

21 Kohei Honda. Types for dyadic interaction. In Eike Best, editor, *CONCUR '93, 4th International Conference on Concurrency Theory, Hildesheim, Germany, August 23-26, 1993, Proceedings*, volume 715 of *Lecture Notes in Computer Science*, pages 509–523. Springer, 1993. doi:10.1007/3-540-57208-2_35.

22 Kohei Honda, Vasco Thudichum Vasconcelos, and Makoto Kubo. Language primitives and type discipline for structured communication-based programming. In Chris Hankin, editor, *Programming Languages and Systems - ESOP'98, 7th European Symposium on Programming, Held as Part of the European Joint Conferences on the Theory and Practice of Software, ETAPS'98, Lisbon, Portugal, March 28 - April 4, 1998, Proceedings*, volume 1381 of *Lecture Notes in Computer Science*, pages 122–138. Springer, 1998. doi:10.1007/BFb0053567.

23 Hans Hüttel, Ivan Lanese, Vasco T. Vasconcelos, Luís Caires, Marco Carbone, Pierre-Malo Deniélou, Dimitris Mostrous, Luca Padovani, António Ravara, Emilio Tuosto, Hugo Torres Vieira, and Gianluigi Zavattaro. Foundations of session types and behavioural contracts. *ACM Comput. Surv.*, 49(1):3:1–3:36, 2016. doi:10.1145/2873052.

24 Dexter Kozen. Results on the propositional mu-calculus. *Theor. Comput. Sci.*, 27:333–354, 1983. doi:10.1016/0304-3975(82)90125-6.

25 Maurizio Murgia. A note on compliance relations and fixed points. In Massimo Bartoletti, Ludovic Henrio, Anastasia Mavridou, and Alceste Scalas, editors, *Proceedings 12th Interaction and Concurrency Experience, ICE 2019, Copenhagen, Denmark, 20-21 June 2019*, volume 304 of *EPTCS*, pages 38–47, 2019. doi:10.4204/EPTCS.304.3.

26 Rocco De Nicola and Matthew Hennessy. Testing equivalences for processes. *Theor. Comput. Sci.*, 34:83–133, 1984. doi:10.1016/0304-3975(84)90113-0.

27 Ulf Norell. *Towards a Practical Programming Language Based on Dependent Type Theory*. PhD thesis, Department of Computer Science and Engineering, Chalmers University of Technology, Göteborg, Sweden., 2007. URL: http://www.cse.chalmers.se/~ulfn/papers/thesis.pdf.

28 Susan S. Owicki and Leslie Lamport. Proving liveness properties of concurrent programs. *ACM Trans. Program. Lang. Syst.*, 4(3):455–495, 1982. doi:10.1145/357172.357178.

29 Luca Padovani. Fair subtyping for open session types. In Fedor V. Fomin, Rusins Freivalds, Marta Z. Kwiatkowska, and David Peleg, editors, *Automata, Languages, and Programming - 40th International Colloquium, ICALP 2013, Riga, Latvia, July 8-12, 2013, Proceedings, Part II*, volume 7966 of *Lecture Notes in Computer Science*, pages 373–384. Springer, 2013. doi:10.1007/978-3-642-39212-2_34.

30 Luca Padovani. Fair subtyping for multi-party session types. *Math. Struct. Comput. Sci.*, 26(3):424–464, 2016. doi:10.1017/S096012951400022X.

31 Arend Rensink and Walter Vogler. Fair testing. *Inf. Comput.*, 205(2):125–198, 2007. doi:10.1016/j.ic.2006.06.002.

32 Peter Thiemann and Vasco T. Vasconcelos. Label-dependent session types. *Proc. ACM Program. Lang.*, 4(POPL):67:1–67:29, 2020. doi:10.1145/3371135.

33 Bernardo Toninho, Luís Caires, and Frank Pfenning. Dependent session types via intuitionistic linear type theory. In Peter Schneider-Kamp and Michael Hanus, editors, *Proceedings of the 13th International ACM SIGPLAN Conference on Principles and Practice of Declarative Programming, July 20-22, 2011, Odense, Denmark*, pages 161–172. ACM, 2011. doi:10.1145/2003476.2003499.

34 Bernardo Toninho and Nobuko Yoshida. Depending on session-typed processes. In Christel Baier and Ugo Dal Lago, editors, *Foundations of Software Science and Computation Structures - 21st International Conference, FOSSACS 2018, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2018, Thessaloniki, Greece, April 14-20, 2018, Proceedings*, volume 10803 of *Lecture Notes in Computer Science*, pages 128–145. Springer, 2018. doi:10.1007/978-3-319-89366-2_7.

Deterministic and Game Separability for Regular Languages of Infinite Trees

Lorenzo Clemente
University of Warsaw, Poland

Michał Skrzypczak
University of Warsaw, Poland

Abstract

We show that it is decidable whether two regular languages of infinite trees are separable by a deterministic language, resp., a game language. We consider two variants of separability, depending on whether the set of priorities of the separator is fixed, or not. In each case, we show that separability can be decided in EXPTIME, and that separating automata of exponential size suffice. We obtain our results by reducing to infinite duration games with ω-regular winning conditions and applying the finite-memory determinacy theorem of Büchi and Landweber.

2012 ACM Subject Classification Theory of computation → Automata over infinite objects; Theory of computation → Tree languages

Keywords and phrases separation, infinite trees, regular languages, deterministic automata, game automata

Digital Object Identifier 10.4230/LIPIcs.ICALP.2021.126

Category Track B: Automata, Logic, Semantics, and Theory of Programming

Related Version *Full Version*: `https://arxiv.org/abs/2105.01137` [15]

Funding *Lorenzo Clemente*: Partially supported by the Polish NCN grant 2017/26/D/ST6/00201.
Michał Skrzypczak: Partially supported by the Polish NCN grant 2017/26/D/ST6/00201.

1 Introduction

One of the most intriguing and motivating problems in the field of automata theory is the *membership problem*. For two fixed classes of languages $\mathcal{C}$ (*input class*) and $\mathcal{D}$ (*output class*), the $(\mathcal{C}, \mathcal{D})$-membership problem asks, given a representation of a language in $\mathcal{C}$, whether this language belongs to $\mathcal{D}$. Among the first results of this type is the famous theorem by Schutzenberger [40] and McNaughton-Papert [30], characterising, among all regular languages of finite words, the subclass of languages that can be defined in first-order logic.

In this paper we consider the class $\mathcal{C}$ of regular languages of infinite trees. While there are many semantically equivalent automata models for this class – e.g., Muller, Rabin, and Street automata [27] – *parity automata* are without doubt the most established such model [25]. The most important descriptional complexity measure of a parity automaton is the set of priorities $C \subseteq \mathbb{N}$ it is allowed to use, which is called its *index*. Not only a larger index allows the automaton to recognise more languages [32], but the computational complexity of known procedures for the emptiness problem crucially depends on the index (the current best bound is quasi-polynomial [8]). The most famous open problem in the area of regular languages of infinite trees is the *nondeterministic index membership problem*, which is the $(\mathcal{C}, \mathcal{D})$-membership problem for $\mathcal{D}$ the class of languages recognised by some nondeterministic parity automaton of a fixed index C (cf. [18]). In many cases, the solution of the membership problem relies either on algebraic representations or determinisation, however algebraic structures for regular languages of infinite trees are of limited availability (cf. [2])

and deterministic automata do not capture all regular languages. While on infinite words this problem was essentially solved by Wagner already at the end of the '70s [43], its solution for infinite trees seems still far away.

Known decidability results abound if we restrict either the input class $\mathcal{C}$ or the output class $\mathcal{D}$. Results of the first kind are known for $\mathcal{C}$ being the class of deterministic [35] and, more generally, game automata [26, Theorem 1.2]. Results of the second kind (i.e., when the input class $\mathcal{C}$ is the full class of regular languages) exist for the output class $\mathcal{D}$ being the lower levels of the index hierarchy [29, 44] and of the Borel hierarchy [4], the class of deterministic languages [33], and Boolean combinations of open sets [6]. Other variants of the index membership problem are known to be decidable, including the early result of Urbański showing that it is decidable whether a given deterministic parity tree automaton is equivalent to some nondeterministic Büchi one [42], the weak alternating index problems for the class of deterministic automata [31] and Büchi automata [17, 41], and deciding whether a given parity automaton is equivalent to some nondeterministic co-Büchi automaton [17].

Another problem closely related to membership is separability. The $(\mathcal{C}, \mathcal{D})$-*separability* problem asks, given a pair of languages L, M in $\mathcal{C}$, whether there exists a language S in $\mathcal{D}$ (called a *separator*) s.t. $L \subseteq S$ and[1] $S \perp M$. Intuitively, a separator S provides a certificate of disjointness, yielding information on the structure of L, M up to some chosen granularity. The separability problem is a generalisation of the membership problem if the class $\mathcal{C}$ is closed under complement, since we can always take M to be the complement of L, in which case the only candidate for the separator is L itself. There are many elegant results in computer science, formal logic, and mathematics showing that separators always exist. Instances include Lusin's separation theorem in topology (two disjoint analytic sets are always separable by a Borel set; cf. [28, Theorem 14.7]), a folklore result in computability theory (two disjoint co-recursively enumerable sets are separable by a recursive set), Craig's theorems in logic (jointly contradictory first-order formulas can be separated by a formula containing only symbols in the shared vocabulary [19]) and model theory (two disjoint projective classes are separable by an elementary class [19]); in formal language theory, a generalisation of a theorem suggested by Tarski and proved by Rabin [38, Theorem 29] states that two disjoint Büchi languages of infinite trees are separable by a weak language (cf. [39]).

In this work we study the $(\mathcal{C}, \mathcal{D})$-separability problems where $\mathcal{C}$ is the full class of regular languages of infinite trees, and $\mathcal{D}$ is one of four kinds of sub-classes thereof, depending on whether the automaton is deterministic or game, and depending on whether we fix a finite index $C \subseteq \mathbb{N}$ or we leave it unrestricted $C = \mathbb{N}$. Our main result is that all four kinds of the separability problems above are decidable and in EXPTIME. Moreover, we show that if a separator exists, then there is one of exponential size.

▶ **Theorem 1.** *The deterministic and game separability problems can be solved in EXPTIME, both for a fixed finite index $C \subseteq \mathbb{N}$, and an unrestricted one $C = \mathbb{N}$. Moreover, separators with exponentially many states and polynomially many priorities suffice.*

Our work is permeated by the observation that the separability problem for two languages L, M can be phrased in terms of a game of infinite duration with an ω-regular winning condition. In such a *separability game* there are two players, Separator trying to prove that L, M are separable, and Input with the opposite objective. In the simple case of $(\mathcal{C}, \mathcal{D})$-separability where $\mathcal{C}$ is the class of regular languages of ω-words and $\mathcal{D}$ the subclass induced by deterministic parity automata of finite index C, the i-th round of the game is as follows:

[1] We write $S \perp M$ for $S \cap M = \varnothing$.

- Separator plays a priority $c_i \in C$.
- Input plays a letter a_i from the finite alphabet Σ.

The resulting infinite play $(c_0, a_0)(c_1, a_1) \cdots$ is won by Separator if 1) $a_0 a_1 \cdots \in L$ implies $c_0 c_1 \cdots$ is accepting and 2) $a_0 a_1 \cdots \notin L$ implies $c_0 c_1 \cdots$ is rejecting. Since the winning condition is ω-regular, by the result of Büchi and Landweber [7] we can decide who wins the game and moreover finite-memory strategies for Separator suffice. Thanks to a correspondence between such strategies and deterministic separators, Separator wins such a game iff there exists a deterministic automaton with priorities in C separating L, M. This provides both decidability of the separability problem and an upper-bound on the size of separators. We design analogous games with ω-regular winning conditions for the more involved case of infinite trees for the separability problems mentioned above and apply [7].

The separability problems we consider have been open so far and generalise the corresponding membership problems. A solution for deterministic separability can easily be derived from [34], however our techniques based on games are novel and provide a unified view on all problems. When instantiated to the specific case of membership, our decidability results generalise the deterministic case (for both fixed and unconstrained index) [34, 33] and the game membership case for unconstrained index [26, Theorem 7.12]. We believe the game approach is much more direct than the combinatorial and pattern-based techniques used in the previous solutions, cf. [26, Section 7, pp. 29–37]. The game membership problem for a fixed index C has been open so far.

We are not aware of computation complexity results for separability problems over regular languages of infinite trees, neither of an analysis of the size of separators. Regarding deterministic membership, EXPTIME-completeness is known [34, Corollary 11], as well as EXPTIME upper [33, end of page 12] and lower bounds [44, Theorem 4.1] (cf., also [29]) for computing the optimal deterministic index. Devising non-trivial complexity lower bounds for the separability problem is left for future work, as well as extending our approach to other classes of separators.

Related works. Over finite words, variants of the $(\mathcal{C}, \mathcal{D})$-separability problem have been studied for classes $\mathcal{C}$ both more general than the regular languages, such as the context free languages [23, 45] and higher-order languages [14] (later extended to safe schemes over finite trees [1]), and for classes $\mathcal{D}$ more restrictive than the regular languages, such as in [36, 37]. The separability and membership problems have also been studied for several classes of infinite-state systems, such as vector addition systems [11, 10, 24], well-structured transition systems [22], one-counter automata [21], and timed automata [13, 12]. Recent developments on efficient algorithms solving parity games are based on the ability to find a simple separator, yielding both upper bounds on the problem, and lower bounds for a wide family of algorithms [5, 20, Chapter 3]. Finally, it is worth mentioning that games have already been successfully used to provide several characterisation results, such as in [18, 17, 16, 3, 41, 9].

Outline. In Section 2 we introduce automata and other mathematical preliminaries. In Sections 3–6 we present the game-theoretic characterisations of the separability problems we consider. We believe this is the most interesting aspect of this work. A technical report is available [15] where a detailed complexity analysis is performed and full proofs are provided.

2 Preliminaries

A nonempty finite set Σ of *letters* $a \in \Sigma$ is called an *alphabet*. A (Σ-*labelled*) *tree* is a function $t\colon \{\mathsf{L},\mathsf{R}\}^* \to \Sigma$ assigning to each *node* $u \in \{\mathsf{L},\mathsf{R}\}^*$ a *label* $t(a) \in \Sigma$. The *root* of a tree is denoted ϵ. The set of all Σ-labelled trees is denoted Tr_Σ. The symbols L, R are called *directions* and a *branch* is an infinite sequence thereof $d_0 d_1 \cdots \in \{\mathsf{L},\mathsf{R}\}^\omega$. A tree t is uniquely defined by the set of its *paths* $\mathsf{Path}(t) = \{(a_0,d_0)(a_1,d_1)\cdots \in (\Sigma\times\{\mathsf{L},\mathsf{R}\})^\omega \mid \forall i.\ a_i = t(d_0 d_1 \cdots d_{i-1})\}$, which is extended to languages pointwise as $\mathsf{Path}(L) = \{\mathsf{Path}(t) \mid t \in L\}$.

Automata. Fix a nonempty finite set of *priorities* $C \subseteq \mathbb{N}$. A *(top-down, nondeterministic, parity, tree) automaton* is a tuple $\mathcal{A} = (\Sigma, Q, q_0, \Omega, \Delta)$, where Σ is a finite alphabet, Q is a finite set of *states*, amongst which $q_0 \in Q$ is the *initial state*, $\Omega\colon Q \to C$ assigns a priority to every state, and $\Delta \subseteq Q\times\Sigma\times Q\times Q$ is a set of *transitions*. The priority function Ω is extended to a transition $\delta = (q,\_,\_,\_)$ as $\Omega(\delta) := \Omega(q)$, pointwise to an infinite sequence of states $\Omega(q_0 q_1\cdots) := \Omega(q_0)\Omega(q_1)\cdots \in C^\omega$ and transitions $\Omega(\delta_0\delta_1\cdots) = \Omega(\delta_0)\Omega(\delta_1)\cdots \in C^\omega$. An infinite sequence of priorities $c_0 c_1 \cdots \in C^\omega$ is *accepting* if the maximal priority occurring infinitely often is even. Similarly, an infinite sequence of states $\rho = q_0 q_1 \cdots \in Q^\omega$ or of transitions $\rho = \delta_0\delta_1\cdots \in \Delta^\omega$ is accepting whenever $\Omega(\rho)$ is accepting. We write $\Delta(q,a) = \{(q,a,q_\mathsf{L},q_\mathsf{R}) \in \Delta\}$ for the set of transitions from a state $q \in Q$ over a letter $a \in \Sigma$, and $\Delta(a) = \bigcup\{\Delta(q,a) \mid q \in Q\}$ for all transitions over a. We extend the notation above to an infinite path $b = (a_0,d_0)(a_1,d_1)\cdots \in (\Sigma\times\{\mathsf{L},\mathsf{R}\})^\omega$ by writing $\Delta(b)$ for the set of infinite sequences of transitions $\vec{\delta} = \delta_0\delta_1\cdots \in \Delta^\omega$ of the form $\delta_i = (q_i, a_i, q_{\mathsf{L},i}, q_{\mathsf{R},i})$ for every i, which are *conform to* b in the sense that q_0 is the initial state of the automaton and $q_{i+1} = q_{d_i,i}$.

A *run* of an automaton $\mathcal{A}$ as above over a tree $t \in \mathrm{Tr}_\Sigma$ is a Q-labelled tree $\rho \in \mathrm{Tr}_Q$ s.t. $\rho(\epsilon) = q_0$ is the initial state and for every node in the tree $u \in \{\mathsf{L},\mathsf{R}\}^*$ the quadruple $\big(\rho(u), t(u), \rho(u\mathsf{L}), \rho(u\mathsf{R})\big)$ belongs to Δ. Such a run is *accepting* if for every branch $d_0 d_1 \cdots \in \{\mathsf{L},\mathsf{R}\}^\omega$ the sequence of states $\big(\rho(d_0\cdots d_{i-1})\big)_{i\in\omega}$ is accepting. The set of all trees $t \in \mathrm{Tr}_\Sigma$ s.t. $\mathcal{A}$ has an accepting run over t is denoted $\mathrm{L}(\mathcal{A})$ and is called the *language* recognised by $\mathcal{A}$. The corresponding *path language* is $\mathrm{L}^{\mathsf{path}}(\mathcal{A}) := \mathsf{Path}(\mathrm{L}(\mathcal{A})) \subseteq (\Sigma\times\{\mathsf{L},\mathsf{R}\})^\omega$. If $q \in Q$ is a state of an automaton $\mathcal{A}$ then by $\mathcal{A}_q$ we denote the same automaton as $\mathcal{A}$ but with the initial state q_0 changed to q. Thus, $\mathrm{L}(\mathcal{A}_q)$ is the set of trees over which $\mathcal{A}$ has an accepting run ρ starting at $\rho(\epsilon) = q$. In the rest of the paper we assume that all states q in an automaton are *productive* in the sense that $\mathrm{L}(\mathcal{A}_q) \neq \varnothing$.

Deterministic and game automata. We say that $\mathcal{A}$ is a *game automaton* if, for every $q \in Q$ and $a \in \Sigma$, either we have a *conjunctive transition* $\Delta(q,a) = \{(q,a,q_\mathsf{L},q_\mathsf{R})\}$ or two *disjunctive transitions* $\Delta(q,a) = \{(q,a,q_\mathsf{L},\top),(q,a,\top,q_\mathsf{R})\}$ (cf. [26, Definition 3.2]), where $\top \neq q_0$ represents a distinguished state in Q accepting every tree (i.e., $\mathrm{L}(\mathcal{A}_\top) = \mathrm{Tr}_\Sigma$) and $q_\mathsf{L}, q_\mathsf{R} \neq \top$. An automaton $\mathcal{A}$ is *deterministic* if it is a game automaton with only conjunctive transitions and in this case for every tree $t \in \mathrm{Tr}_\Sigma$ there exists a unique run ρ of $\mathcal{A}$ over t. A tree language L is *deterministic*, resp., *game*, if it can be recognised by some deterministic, resp., game automaton. Game automata can be complemented with very low complexity by just increasing every priority by one and by swapping conjunctive and disjunctive transitions.

▶ **Lemma 2.** *If $\mathcal{A}$ is a game parity tree automaton, then $\mathrm{Tr}_\Sigma\backslash\mathrm{L}(\mathcal{A})$ can be recognised by a game parity tree automaton with the same number of states and priorities.*

Acceptance games. We present a game-theoretic view on accepting runs. This will serve both as an example of the kind of games that we consider throughout paper, and as a technical tool in the proofs from Sections 5 and 6. Let $t \in \mathrm{Tr}_\Sigma$ be a tree. The *acceptance game* $G^{\mathrm{acc}}(\mathcal{A}, t)$ is played in rounds by two players, Automaton and Pathfinder. The goal of Automaton is to show that $t \in \mathrm{L}(\mathcal{A})$; Pathfinder has the complementary objective $t \notin \mathrm{L}(\mathcal{A})$.

Acceptance game $G^{\mathrm{acc}}(\mathcal{A}, t)$

At the i-th round starting at a *position* $v_i = (u_i, q_i) \in V := \{\mathsf{L}, \mathsf{R}\}^* \times Q$:
- [A: δ] Automaton plays a transition $\delta_i = (q_i, t(u_i), q_{\mathsf{L},i}, q_{\mathsf{R},i}) \in \Delta(q_i, t(u_i))$.
- [P: d] Pathfinder plays a direction $d_i \in \{\mathsf{L}, \mathsf{R}\}$.

The next position is $v_{i+1} := (u_i d_i, q_{d_i,i})$.

The initial position is $v_0 := (\epsilon, q_0)$. Automaton *wins* the resulting infinite play $\pi = (\delta_0, d_0)(\delta_1, d_1) \cdots$ if the sequence of transitions $\delta_0 \delta_1 \cdots$ is accepting. Automaton's moves in the acceptance game $G^{\mathrm{acc}}(\mathcal{A}, t)$ are performed according to a *strategy* for Automaton. This is a tuple $\mathcal{M} = (M, \ell_0, \overline{\delta}, \tau)$, where M is a set of *memory states*, of which $\ell_0 \in M$ is the initial memory state, $\overline{\delta} \colon V \times M \to \Delta$ is an output function which in a position (u, q) and a memory state ℓ selects a transition $\overline{\delta}\big((u, q), \ell\big) \in \Delta(q, t(u))$ of $\mathcal{A}$, and $\tau \colon V \times M \times \{\mathsf{L}, R\} \to M$ is a memory update function which, in a given position v, memory state ℓ, and direction d selects the next memory state $\tau(v, \ell, d) \in M$. An infinite play π as above is *conform* to a strategy $\mathcal{M}$ if during the play π Automaton keeps track of the current position v_i and memory state ℓ_i, updating them after each round (i.e., $\ell_{i+1} := \tau(v_i, \ell_i, d_i)$) and her consecutive choices of transitions δ_i are done according to $\overline{\delta}(v_i, \ell_i)$. A strategy $\mathcal{M}$ is *winning* if every play conform to it is winning for Automaton. Automaton wins the acceptance game if she has a winning strategy. The following proposition is folklore.

▶ **Proposition 3.** *Let $t \in \mathrm{Tr}_\Sigma$ and $\mathcal{A}$ be an automaton over the alphabet Σ. Automaton wins the acceptance game $G^{\mathrm{acc}}(\mathcal{A}, t)$ if, and only if, $t \in \mathrm{L}(\mathcal{A})$.*

Disjointness games. Let $\mathcal{A}$ and $\mathcal{B}$ be two nondeterministic automata. We recall a standard game used to characterise whether $\mathrm{L}(\mathcal{A}) \perp \mathrm{L}(\mathcal{B})$. This will be crucial in the correctness proofs throughout Sections 3–6. The *disjointness game* $G^{\mathrm{dis}}(\mathcal{A}, \mathcal{B})$ is played by two players, Automaton and Pathfinder. Automaton's aim is to incrementally build a tree accepted by both $\mathcal{A}$ and $\mathcal{B}$, witnessing $\mathrm{L}(\mathcal{A}) \cap \mathrm{L}(\mathcal{B}) \neq \varnothing$, while Pathfinder has the opposite objective.[2] The set of positions of the game is $Q^{\mathcal{A}} \times Q^{\mathcal{B}}$, and the initial position is $(q_0^{\mathcal{A}}, q_0^{\mathcal{B}})$.

Disjointness game $G^{\mathrm{dis}}(\mathcal{A}, \mathcal{B})$

At the i-th round starting at a position $(q_i^{\mathcal{A}}, q_i^{\mathcal{B}})$:
- [A: a] Automaton plays a letter $a_i \in \Sigma$.
- [A: $\delta^{\mathcal{A}}$] Automaton plays a transition $\delta_i^{\mathcal{A}} = (q_i^{\mathcal{A}}, a_i, q_{\mathsf{L},i}^{\mathcal{A}}, q_{\mathsf{R},i}^{\mathcal{A}}) \in \Delta^{\mathcal{A}}(q_i^{\mathcal{A}}, a_i)$.
- [A: $\delta^{\mathcal{B}}$] Automaton plays a transition $\delta_i^{\mathcal{B}} = (q_i^{\mathcal{B}}, a_i, q_{\mathsf{L},i}^{\mathcal{B}}, q_{\mathsf{R},i}^{\mathcal{B}}) \in \Delta^{\mathcal{B}}(q_i^{\mathcal{B}}, a_i)$.
- [P: d] Pathfinder plays a direction $d_i \in \{\mathsf{L}, \mathsf{R}\}$.

The next position is $(q_{d_i,i}^{\mathcal{A}}, q_{d_i,i}^{\mathcal{B}})$.

[2] The disjointness game could equivalently be phrased as a nonemptiness game for the product automaton $\mathcal{A} \times \mathcal{B}$ recognising $\mathrm{L}(\mathcal{A}) \cap \mathrm{L}(\mathcal{B})$. However, in our technical development it will be more direct to use the disjointness game.

Let the resulting infinite play be $\pi = (a_0, \delta_0^{\mathcal{A}}, \delta_0^{\mathcal{B}}, d_0)(a_1, \delta_1^{\mathcal{A}}, \delta_1^{\mathcal{B}}, d_1)\cdots$. Such a play induces an infinite path $b = (a_0, d_0)(a_1, d_1)\cdots$ and two sequences of transitions $\vec{\delta}^{\mathcal{A}} := \delta_0^{\mathcal{A}}\delta_1^{\mathcal{A}}\cdots$ and $\vec{\delta}^{\mathcal{B}} := \delta_0^{\mathcal{B}}\delta_1^{\mathcal{B}}\cdots$. The rules of the game guarantee that $\vec{\delta}^{\mathcal{A}} \in \Delta^{\mathcal{A}}(b)$ and $\vec{\delta}^{\mathcal{B}} \in \Delta^{\mathcal{B}}(b)$. Automaton wins the play π if both sequences $\vec{\delta}^{\mathcal{A}}$ and $\vec{\delta}^{\mathcal{B}}$ are accepting.

In the rest of the paper it will be more useful to consider Pathfinder's point of view. Since her winning condition can be presented as Rabin condition, whenever she wins, she has a *memoryless* (i.e., $M = \{\ell_0\}$) winning strategy. Such a memoryless strategy for Pathfinder in the disjointness game can be represented by a function $\mathcal{P}\colon \left(\bigcup_{a\in\Sigma} \Delta^{\mathcal{A}}(a) \times \Delta^{\mathcal{B}}(a)\right) \to \{\mathsf{L}, \mathsf{R}\}$, which we call a *pathfinder*.

▶ **Lemma 4.** *If* $\mathrm{L}(\mathcal{A}) \perp \mathrm{L}(\mathcal{B})$ *then there is a pathfinder* $\mathcal{P}$ *which is winning for* Pathfinder *in the disjointness game* $G^{\mathrm{dis}}(\mathcal{A}, \mathcal{B})$.

▶ **Corollary 5.** *Assume that* $\mathrm{L}(\mathcal{A}) \perp \mathrm{L}(\mathcal{B})$ *and let* $\mathcal{P}$ *be a pathfinder as above. Let* $b = (a_0, d_0)(a_1, d_1)\cdots \in (\Sigma \times \{\mathsf{L}, \mathsf{R}\})^\omega$ *be a path and* $\vec{\delta}^{\mathcal{A}} = \delta_0^{\mathcal{A}}\delta_1^{\mathcal{A}}\cdots \in \Delta^{\mathcal{A}}(b)$, $\vec{\delta}^{\mathcal{B}} = \delta_0^{\mathcal{B}}\delta_1^{\mathcal{B}}\cdots \in \Delta^{\mathcal{B}}(b)$ *be two sequences of transitions of these automata that are conform to* b. *If for every* $i \in \omega$ *we have* $\mathcal{P}(\delta_i^{\mathcal{A}}, \delta_i^{B}) = d_i$ *then at least one of the sequences* $\vec{\delta}^{\mathcal{A}}$ *and* $\vec{\delta}^{\mathcal{B}}$ *is rejecting.*

The construction above has a specific property if one of the involved automata (e.g., $\mathcal{A}$) is a game automaton. Since we assume that every state is productive, positions of the form $(\top, q^{\mathcal{B}})$ are losing for Pathfinder in $G^{\mathrm{dis}}(\mathcal{A}, \mathcal{B})$. Therefore, without loss of generality we can assume that the pathfinder $\mathcal{P}$ satisfies the following observation.

▶ Remark 6. Consider a transition $\delta^{\mathcal{A}} = (q^{\mathcal{A}}, a, q_{\mathsf{L}}^{\mathcal{A}}, \top)$ (resp., $\delta^{\mathcal{A}} = (q^{\mathcal{A}}, a, \top, q_{\mathsf{R}}^{\mathcal{A}})$) in a game automaton $\mathcal{A}$. Then, $\mathcal{P}(\delta^{\mathcal{A}}, \_)$ is constantly equal to L (resp., R).

3 Separability by deterministic automata with priorities in C

In this section we present a game-theoretic characterisation of separability by deterministic automata over a fixed finite set of priorities $C \subseteq \mathbb{N}$. Let $\mathcal{A}$, $\mathcal{B}$ be two nondeterministic automata over infinite trees. We extend the game from the introduction over ω-words with two additional actions, a *selector* for Separator and a direction for Input.

C-deterministic-separability game $G^{\mathrm{sep}}_{\mathrm{det}}(\mathcal{A}, \mathcal{B}, C)$

At the i-th round:
- [S: c] Separator plays a priority $c_i \in C$.
- [I: a] Input plays a letter $a_i \in \Sigma$.
- [S: f] Separator plays a *selector* $f_i \in \{\mathsf{L}, \mathsf{R}\}^{\Delta^{\mathcal{B}}(a_i)}$.
- [I: d] Input plays a direction $d_i \in \{\mathsf{L}, \mathsf{R}\}$.

Intuitively, a selector encodes a direction for each (relevant) transition of $\mathcal{B}$ and this is used for the correctness of the separator. Assume that the resulting infinite play is $\pi = (c_0, a_0, f_0, d_0)(c_1, a_1, f_1, d_1)\cdots$, with the induced infinite path $b := (a_0, d_0)(a_1, d_1)\cdots$. Separator wins the play π if the following two conditions are satisfied:

1. $\pi \in \mathbf{W}_{\mathcal{A}}$: If there exists an accepting sequence of transitions $\vec{\delta}^{\mathcal{A}} = \delta_0^{\mathcal{A}}\delta_1^{\mathcal{A}}\cdots \in \Delta^{\mathcal{A}}(b)$, then $c_0c_1\cdots$ is accepting.
2. $\pi \in \mathbf{W}_{\mathcal{B}}$: If there exists an accepting sequence of transitions $\vec{\delta}^{\mathcal{B}} = \delta_0^{\mathcal{B}}\delta_1^{\mathcal{B}}\cdots \in \Delta^{\mathcal{B}}(b)$ s.t. for every $i \in \omega$ we have $f_i(\delta_i^{\mathcal{B}}) = d_i$, then $c_0c_1\cdots$ is rejecting.

The following lemma states that the separability game correctly characterises the deterministic separability problem.

▶ **Lemma 7.** *Separator wins $G^{\text{sep}}_{\text{det}}(\mathcal{A}, \mathcal{B}, C)$ if, and only if, $\mathrm{L}(\mathcal{A})$, $\mathrm{L}(\mathcal{B})$ can be separated by a deterministic parity tree automaton with priorities in C.*

We present a full proof in order to show the rôle of Separator's selectors.

Soundness. Assume that Separator wins the separability game $G := G^{\text{sep}}_{\text{det}}(\mathcal{A}, \mathcal{B}, C)$ by a finite-memory winning strategy $\mathcal{M} = (M, \ell_0, (\overline{c}, \overline{f}), \tau)$. Strategy $\mathcal{M}$ has two decision functions: $\overline{c}$ assigns to each $\ell \in M$ a priority $\overline{c}(\ell) \in C$, and $\overline{f}$ assigns to each $\ell \in M$ and $a \in \Sigma$ a selector $\overline{f}(\ell, a) \in \{\mathsf{L}, \mathsf{R}\}^{\Delta^{\mathcal{B}}(a)}$. Moreover, the type of the memory update function is $\tau \colon M \times \Sigma \times \{\mathsf{L}, \mathsf{R}\} \to M$. Consider a deterministic parity tree automaton $\mathcal{S} := (\Sigma, M, \ell_0, \Omega^{\mathcal{S}}, \Delta^{\mathcal{S}})$ which has the same set of states M and initial state ℓ_0 as $\mathcal{M}$, priorities are induced by the decision function $\overline{c}$ of $\mathcal{M}$ as $\Omega^{\mathcal{S}}(\ell) := \overline{c}(\ell)$, and transitions are of the form $\Delta^{\mathcal{S}} = \{(\ell, a, \tau(\ell, a, \mathsf{L}), \tau(\ell, a, \mathsf{R})) \mid \ell \in M, a \in \Sigma\}$.

We show that $\mathcal{S}$ separates $\mathrm{L}(\mathcal{A})$, $\mathrm{L}(\mathcal{B})$. We first show $\mathrm{L}(\mathcal{A}) \subseteq \mathrm{L}(\mathcal{S})$. Let $t \in \mathrm{L}(\mathcal{A})$ be a tree that is accepted by the automaton $\mathcal{A}$, as witnessed by an accepting run $\rho^{\mathcal{A}}$. Let $\rho^{\mathcal{S}}$ be the unique run of $\mathcal{S}$ over t. Consider any branch $d_0 d_1 \cdots \in \{\mathsf{L}, \mathsf{R}\}^{\omega}$. We need to show that the sequence of priorities $\big(\Omega^{\mathcal{S}}(\rho^{\mathcal{S}}(d_0 \cdots d_{i-1}))\big)_{i \in \omega}$ is accepting. Consider a play π of G where at the i-th round Separator plays according to the strategy $\mathcal{M}$ with current memory state $\ell_i \in M$ and Input plays according to the letters from t and directions $d_0 d_1 \cdots$ fixed above:

- [S: c] Separator plays the priority $c_i := \overline{c}(\ell_i) \in C$.
- [I: a] Input plays the letter $a_i := t(u_i) \in \Sigma$, where $u_i := d_0 \cdots d_{i-1}$.
- [S: f] Separator plays the selector $f_i := \overline{f}(\ell_i, a_i) \in \{\mathsf{L}, \mathsf{R}\}^{\Delta^{\mathcal{B}}(a_i)}$ (the selector is irrelevant in this part of the proof).
- [I: d] Input plays the direction $d_i \in \{\mathsf{L}, \mathsf{R}\}$ as fixed above.

The next memory state is $\ell_{i+1} := \tau(\ell_i, a_i, d_i)$. Let the resulting infinite play be $\pi = (c_0, a_0, f_0, d_0)(c_1, a_1, f_1, d_1) \cdots$. By the construction of $\mathcal{S}$ we know that $\ell_i = \rho^{\mathcal{S}}(u_i)$ and therefore $c_i = \Omega^{\mathcal{S}}(\rho^{\mathcal{S}}(u_i))$. Since $t \in \mathrm{L}(\mathcal{A})$, there exists an accepting sequence of transitions $\vec{\delta}^{\mathcal{A}} = \delta^{\mathcal{A}}_0 \delta^{\mathcal{A}}_1 \cdots \in \Delta^{\mathcal{A}}(b)$ along the path $b = (a_0, d_0)(a_1, d_1) \cdots$. Since Separator is winning, $\pi \in \mathbf{W}_{\mathcal{A}}$ and thus the sequence $c_0 c_1 \cdots$ is accepting, as required.

We now argue that $\mathrm{L}(\mathcal{S})$ and $\mathrm{L}(\mathcal{B})$ are disjoint. Towards reaching a contradiction, assume that $t \in \mathrm{L}(\mathcal{S}) \cap \mathrm{L}(\mathcal{B})$ belongs to their intersection. Let $\rho^{\mathcal{S}}$ be the unique run of $\mathcal{S}$ over t, and let $\rho^{\mathcal{B}}$ be an accepting run of $\mathcal{B}$ over t. Consider a play $\pi = (c_0, a_0, f_0, d_0)(c_1, a_1, f_1, d_1) \cdots$ of G where the i-th round is played as above except that Input plays the direction $d_i := f_i(\delta^{\mathcal{B}}_i)$, obtained by applying the selector f_i to the transition $\delta^{\mathcal{B}}_i := \big(\rho^{\mathcal{B}}(u_i), t(u_i), \rho^{\mathcal{B}}(u_i\mathsf{L}), \rho^{\mathcal{B}}(u_i\mathsf{R})\big)$ determined according to the run $\rho^{\mathcal{B}}$. By the choice of directions d_i's, the sequence of transitions $\vec{\delta}^{\mathcal{B}} = \delta^{\mathcal{B}}_0 \delta^{\mathcal{B}}_1 \cdots \in (\Delta^{\mathcal{B}})^{\omega}$ satisfies $f_i(\delta^{\mathcal{B}}_i) = d_i$ for every $i \in \omega$. Since the run $\rho^{\mathcal{B}}$ is accepting, $\vec{\delta}^{\mathcal{B}}$ is accepting. Since Separator is winning, $\pi \in \mathbf{W}_{\mathcal{B}}$ and thus the sequence of priorities $c_0 c_1 \cdots$ is rejecting. However, this is a contradiction, because for each $i \in \omega$ we have $\ell_i = \rho^{\mathcal{S}}(u_i)$ and $c_i = \Omega^{\mathcal{S}}(\ell_i)$ and we assumed that the run $\rho^{\mathcal{S}}$ is accepting. ◀

Completeness. Assume that $\mathcal{S} = (\Sigma, Q^{\mathcal{S}}, q^{\mathcal{S}}_0, \Delta^{\mathcal{S}}, \Omega^{\mathcal{S}})$ is a deterministic automaton with priorities in C separating $\mathrm{L}(\mathcal{A})$, $\mathrm{L}(\mathcal{B})$, and we show that Separator wins the separability game G. Since $\mathcal{S}$ is a separator, we have that $\mathrm{L}(\mathcal{S}) \perp \mathrm{L}(\mathcal{B})$, and by Lemma 4 there exists a pathfinder $\mathcal{P}$. Consider the following strategy of Separator, with memory structure $Q^{\mathcal{S}}$ and initial memory state $q^{\mathcal{S}}_0$. At the i-th round of G, starting with a memory state $q^{\mathcal{S}}_i$,

- [S: c] Separator plays the priority $c_i := \Omega^{\mathcal{S}}(q^{\mathcal{S}}_i) \in C$.
- [I: a] Input plays an arbitrary letter $a_i \in \Sigma$.
- [S: f] Separator plays the selector $f_i := \mathcal{P}(\delta^{\mathcal{S}}_i, \_) \in \{\mathsf{L}, \mathsf{R}\}^{\Delta^{\mathcal{B}}(a_i)}$, where $\Delta^{\mathcal{S}}(q^{\mathcal{S}}_i, a_i) = \{\delta^{\mathcal{S}}_i\}$.
- [I: d] Input plays an arbitrary direction $d_i \in \{\mathsf{L}, \mathsf{R}\}$.

The next memory state is $q^{\mathcal{S}}_{i+1} := q^{\mathcal{S}}_{d_i,i}$, where $\delta^{\mathcal{S}}_i = (q^{\mathcal{S}}_i, a_i, q^{\mathcal{S}}_{\mathsf{L},i}, q^{\mathcal{S}}_{\mathsf{R},i})$. This concludes the description of the i-th round of G. Let the resulting infinite play be $\pi = (c_0, a_0, f_0, d_0)(c_1, a_1, f_1, d_1)\cdots$, with induced infinite path $b := (a_0, d_0)(a_1, d_1)\cdots$. Let $\vec{\delta}^{\mathcal{S}} := \delta^{\mathcal{S}}_0\delta^{\mathcal{S}}_1\cdots$ be the sequence of transitions used to define the selectors f_i. Clearly $\vec{\delta}^{\mathcal{S}} \in \Delta^{\mathcal{S}}(b)$.

First, we argue that $\pi \in \mathbf{W}_{\mathcal{A}}$ holds. Let $\vec{\delta}^{\mathcal{A}} = \delta^{\mathcal{A}}_0\delta^{\mathcal{A}}_1\cdots \in \Delta^{\mathcal{A}}(b)$ be an accepting sequence of transitions of the automaton $\mathcal{A}$. Since each state of $\mathcal{A}$ is productive, one can construct a tree $t \in \mathrm{L}(\mathcal{A})$ s.t. $b \in \mathsf{Path}(t)$. Since $\mathrm{L}(\mathcal{A}) \subseteq \mathrm{L}(\mathcal{S})$ by the assumption, $t \in \mathrm{L}(\mathcal{S})$ as well, and since $\mathcal{S}$ is deterministic, the unique run of $\mathcal{S}$ over t is accepting. By the definition of Separator's strategy, the sequence of priorities along the branch $d_0d_1\cdots$ of this accepting run is precisely $c_0c_1\cdots$, which thus must be accepting, as required.

Regarding $\mathbf{W}_{\mathcal{B}}$, let $\vec{\delta}^{\mathcal{B}} := \delta^{\mathcal{B}}_0\delta^{\mathcal{B}}_1\cdots \in \Delta^{\mathcal{B}}(b)$ be an accepting sequence of transitions over the path b conform to the selectors f_i, i.e., for every $i \in \omega$ we have $f_i(\delta^{\mathcal{B}}_i) = d_i$. By the definition of f_i, for every $i \in \omega$ we have $d_i = \mathcal{P}(\delta^{\mathcal{S}}_i, \delta^{\mathcal{B}}_i)$. Thus, the assumptions of Corollary 5 are satisfied and at least one of the sequences $\vec{\delta}^{\mathcal{S}}$, $\vec{\delta}^{\mathcal{B}}$ must be rejecting. Since we assumed that $\vec{\delta}^{\mathcal{B}}$ is accepting, it means that $\vec{\delta}^{\mathcal{S}}$ is rejecting, and so is $c_0c_1\cdots$ since $c_i = \Omega^{\mathcal{S}}(\delta^{S}_i)$. ◀

4 Separability by deterministic automata

In this section we present a game-theoretic characterisation of the deterministic separability problem. Notice that here we do not fix in advance a finite set of priorities C. The deterministic-separability game $G^{\mathrm{sep}}_{\mathrm{det}}(\mathcal{A}, \mathcal{B})$ below is a variant of the game with fixed priorities C from Section 3.

Deterministic-separability game $G^{\mathrm{sep}}_{\mathrm{det}}(\mathcal{A}, \mathcal{B})$

At the i-th round:

[I: a] Input plays a letter $a_i \in \Sigma$.

[S: f] Separator plays a selector $f_i \in \{\mathsf{L}, \mathsf{R}\}^{\Delta^{\mathcal{B}}(a_i)}$.

[I: d] Input plays a direction $d_i \in \{\mathsf{L}, \mathsf{R}\}$.

Separator wins the resulting infinite play $\pi = (a_0, f_0, d_0)(a_1, f_1, d_1)\cdots$, with induced infinite path $b := (a_0, d_0)(a_1, d_1)\cdots$, if at least one of the two conditions below fails:

1. $\pi \in \mathbf{W}_{\mathcal{A}}$: There exists an accepting sequence of transitions $\vec{\delta}^{\mathcal{A}} = \delta^{\mathcal{A}}_0\delta^{\mathcal{A}}_1\cdots \in \Delta^{\mathcal{A}}(b)$.
2. $\pi \in \mathbf{W}_{\mathcal{B}}$: There exists an accepting sequence of transitions $\vec{\delta}^{\mathcal{B}} = \delta^{\mathcal{B}}_0\delta^{\mathcal{B}}_1\cdots \in \Delta^{\mathcal{B}}(b)$ s.t. for every $i \in \omega$ we have $f_i(\delta^{\mathcal{B}}_i) = d_i$.

Before we prove the equivalence between the game and the existence of a separator, we define a separator candidate, namely the path-closure of $\mathrm{L}(\mathcal{A})$. This is important since it will turn out that if a separator exists, then the path-closure is itself a separator. Given a language of trees L, its *path-closure*, denoted $\forall\mathsf{Path}(L)$, is the set of all trees t s.t. for every path $b \in \mathsf{Path}(t)$ there exists some tree $t' \in L$ s.t. $b \in \mathsf{Path}(t')$ as well. The path-closure operator is directly connected with deterministic automata.

▶ **Lemma 8** (cf. [34, Proposition 1]). *Given a nondeterministic automaton $\mathcal{A}$ one can construct a deterministic automaton $\mathcal{A}^{\mathrm{path}}$ recognising the path closure of $\mathrm{L}(\mathcal{A})$, i.e., $\mathrm{L}(\mathcal{A}^{\mathrm{path}}) = \forall\mathsf{Path}(\mathrm{L}(\mathcal{A}))$. Moreover, $\mathrm{L}(\mathcal{A}^{\mathrm{path}})$ is the smallest deterministic language containing $\mathrm{L}(\mathcal{A})$.*

The following lemma binds together the game $G^{\mathrm{sep}}_{\mathrm{det}}(\mathcal{A}, \mathcal{B})$, separability, and path-closures.

▶ **Lemma 9.** *The following three conditions are equivalent:*

1. *Separator wins the deterministic-separability game $G^{\mathrm{sep}}_{\mathrm{det}}(\mathcal{A}, \mathcal{B})$.*
2. *The automaton $\mathcal{A}^{\mathrm{path}}$ is a deterministic separator for $\mathrm{L}(\mathcal{A})$, $\mathrm{L}(\mathcal{B})$.*
3. *There exists a deterministic separator for $\mathrm{L}(\mathcal{A})$, $\mathrm{L}(\mathcal{B})$.*

Proof sketch. Consider the implication "1 ⇒ 2". Firstly, $\mathrm{L}(\mathcal{A}) \subseteq \mathrm{L}(\mathcal{A}^{\mathrm{path}})$ because the operator $\forall\mathsf{Path}(\_)$ is non-decreasing. Moreover, the fact that $\mathrm{L}(\mathcal{A}^{\mathrm{path}}) \perp \mathrm{L}(\mathcal{B})$ is witnessed by the choices of selectors f_i by a winning strategy of Separator in $G^{\mathrm{sep}}_{\mathrm{det}}(\mathcal{A}, \mathcal{B})$. The implication "2 ⇒ 3" is trivial. The proof of the implication "3 ⇒ 1" is similar to the proof of completeness in Lemma 7 – a separating automaton $\mathcal{S}$ can be used to construct a pathfinder $\mathcal{P}$, witnessing that $\mathrm{L}(\mathcal{S})$ and $\mathrm{L}(\mathcal{B})$ are disjoint. Now, one can construct a strategy of Separator in $G^{\mathrm{sep}}_{\mathrm{det}}(\mathcal{A}, \mathcal{B})$ by simulating $\mathcal{S}$ and using $\mathcal{P}$ to choose the selectors f_i. ◀

5 Separability by game automata

In this section we provide a game-theoretic characterisation for the game automata separability problem. Fix two automata $\mathcal{A}$ and $\mathcal{B}$ and consider the following separability game $G^{\mathrm{sep}}_{\mathrm{game}}(\mathcal{A}, \mathcal{B})$. The new ingredient is that Separator can choose a *mode* – a symbol from the set $\{\vee, \wedge\}$. It has two uses. First, in the construction of the separating game automaton, the mode dictates whether there will be a conjunctive or a disjunctive transition. Second, depending on the chosen mode, Separator will have to play a selector for the automaton $\mathcal{A}$ or $\mathcal{B}$, which will guarantee that the constructed automaton is a separator.

Game-separability game $G^{\mathrm{sep}}_{\mathrm{game}}(\mathcal{A}, \mathcal{B})$

At the i-th round:

- [I: a] Input plays a letter $a_i \in \Sigma$.
- [S: m] Separator plays a mode $m_i \in \{\vee, \wedge\}$.
- [S: f] Separator plays either
  - **a.** a selector $f_i \in \{\mathsf{L}, \mathsf{R}\}^{\Delta^{\mathcal{A}}(a_i)}$ for $\mathcal{A}$ if $m_i = \vee$ or
  - **b.** a selector $f_i \in \{\mathsf{L}, \mathsf{R}\}^{\Delta^{\mathcal{B}}(a_i)}$ for $\mathcal{B}$ if $m_i = \wedge$.
- [I: d] Input plays a direction $d_i \in \{\mathsf{L}, \mathsf{R}\}$.

Separator wins an infinite play $\pi = (a_0, m_0, f_0, d_0)(a_1, m_1, f_1, d_1)\cdots$ inducing a path $b = (a_0, d_0)(a_1, d_1)\cdots$ whenever at least one of the two conditions below fail:

1. $\pi \in \mathbf{W}_{\mathcal{A}}$: There exists an accepting sequence of transitions $\vec{\delta}^{\mathcal{A}} = \delta^{\mathcal{A}}_0\delta^{\mathcal{A}}_1\cdots \in \Delta^{\mathcal{A}}(b)$ s.t. for all $i \in \mathbb{N}$ we have $(m_i = \vee) \Rightarrow f_i(\delta^{\mathcal{A}}_i) = d_i$.
2. $\pi \in \mathbf{W}_{\mathcal{B}}$: There exists an accepting sequence of transitions $\vec{\delta}^{\mathcal{B}} = \delta^{\mathcal{B}}_0\delta^{\mathcal{B}}_1\cdots \in \Delta^{\mathcal{B}}(b)$ s.t. for all $i \in \mathbb{N}$ we have $(m_i = \wedge) \Rightarrow f_i(\delta^{\mathcal{B}}_i) = d_i$.

▶ **Lemma 10.** *Separator wins the separability game $G^{\mathrm{sep}}_{\mathrm{game}}(\mathcal{A}, \mathcal{B})$ if, and only if, there exists a game automaton $\mathcal{S}$ separating $\mathrm{L}(\mathcal{A})$, $\mathrm{L}(\mathcal{B})$.*

In the proof of this lemma we will build separating automata with a more general acceptance condition than the parity condition, which will simplify the technical details. A *generalised game automaton* $\mathcal{A} = (\Sigma, Q, q_0, \Delta, \mathcal{D})$ is just like a game automaton except that the priority mapping Ω is replaced by a deterministic ω-word parity automaton $\mathcal{D}$ over alphabet $\Sigma \times \{\mathsf{L}, \mathsf{R}\}$. A run $\rho \in \mathrm{Tr}_Q$ of such an automaton over a tree $t \in \mathrm{Tr}_\Sigma$ is *accepting* if for every path $b = (a_0, d_0)(a_1, d_1)\cdots \in \mathsf{Path}(t)$ either $\rho(d_0\cdots d_{i-1}) = \top$ for some $i \in \omega$, or $b \in \mathrm{L}(\mathcal{D})$. The acceptance game $G^{\mathrm{acc}}(\mathcal{A}, t)$ can easily be adapted to the case of a generalised game automaton $\mathcal{A}$ by only modifying the winning condition.

▶ **Lemma 11.** *A generalised game automaton $\mathcal{A}$ with a generalised acceptance condition recognised by a deterministic parity automaton $\mathcal{D}$ can be transformed into an equivalent (ordinary) game automaton $\mathcal{B}$ of size polynomial in $\mathcal{A}$ and $\mathcal{D}$.*

We now prove Lemma 10. Its proof is given in full details because, unlike in Sections 3 and 4, it is not obvious how to construct a separator from a winning strategy for Separator.

Soundness. Assume that Separator wins the game-separability game $G := G^{\text{sep}}_{\text{game}}(\mathcal{A}, \mathcal{B})$ and we show that there exists a game automaton $\mathcal{S}$ separating $\mathrm{L}(\mathcal{A})$ from $\mathrm{L}(\mathcal{B})$. Let $\mathcal{M} = (M, \ell_0, (\overline{m}, \overline{f}), \tau)$ be a finite-memory winning strategy of Separator in G.

Before we move to the construction of the separating automaton, we first define its generalised acceptance condition. Let $L_{\mathcal{A}}$ (resp., $L_{\mathcal{B}}$) be the set of those paths $b = (a_0, d_0)(a_1, d_1)\cdots \in (\Sigma \times \{\mathsf{L}, \mathsf{R}\})^\omega$ s.t. the unique play π of G in which Input plays consecutive letters and directions from b and Separator uses her winning strategy $\mathcal{M}$ satisfies the condition $\mathbf{W}_{\mathcal{A}}$ (resp., $\mathbf{W}_{\mathcal{B}}$). Since the strategy $\mathcal{M}$ is winning for Separator, the languages $L_{\mathcal{A}}$ and $L_{\mathcal{B}}$ are disjoint. Moreover, since the strategy $\mathcal{M}$ is finite memory and both $\mathbf{W}_{\mathcal{A}}$, $\mathbf{W}_{\mathcal{B}}$ are ω-regular, so are the languages $L_{\mathcal{A}}$ and $L_{\mathcal{B}}$. Let $\mathcal{D}$ be any deterministic automaton over ω-words that separates $L_{\mathcal{A}}$ from $L_{\mathcal{B}}$ (the simplest case is to take $\mathcal{D}$ recognising the language $L_{\mathcal{A}}$). We build a separating automaton as a generalised game automaton

$$\mathcal{S} := \alpha(\mathcal{M}, \mathcal{D}) := (\Sigma, M \cup \{\top\}, \ell_0, \Delta^{\mathcal{S}}, \mathcal{D}), \text{ where}$$

$$\Delta^{\mathcal{S}}\big(\ell, a\big) := \begin{cases} \{(\ell, a, \ell_{\mathsf{L}}, \top), (\ell, a, \top, \ell_{\mathsf{R}})\} & \text{if } \overline{m}(\ell, a) = \vee, \\ \{(\ell, a, \ell_{\mathsf{L}}, \ell_{\mathsf{R}})\} & \text{if } \overline{m}(\ell, a) = \wedge, \end{cases}$$

for every $\ell \in M$ and $a \in \Sigma$, where for $d \in \{\mathsf{L}, \mathsf{R}\}$ we have $\ell_d := \tau(\ell, a, d)$. We now show that $\mathcal{S}$ separates $\mathrm{L}(\mathcal{A})$ from $\mathrm{L}(\mathcal{B})$. In order to show $\mathrm{L}(\mathcal{A}) \subseteq \mathrm{L}(\mathcal{S})$, let $t \in \mathrm{L}(\mathcal{A})$ as witnessed by an accepting run $\rho^{\mathcal{A}}$. We show that Automaton wins the acceptance game $G_{\mathcal{S}} := G^{\text{acc}}(\mathcal{S}, t)$. To show this we play in parallel the separability game G and the acceptance game $G_{\mathcal{S}}$, maintaining the following invariant: At the i-th round, the current finite path of the input tree t is $(a_0, d_0)\cdots(a_{i-1}, d_{i-1})$, Separator's winning strategy $\mathcal{M}$ in the separability game G is in memory state ℓ_i, the current state of the separating automaton $\mathcal{S}$ in the acceptance game $G_{\mathcal{S}}$ is also ℓ_i, and $\rho^{\mathcal{A}}(d_0 \cdots d_{i-1}) = q_i^{\mathcal{A}}$. The i-th round is then played as follows:

$G.[\mathsf{I}\colon a]$ Input plays the letter $a_i := t(u_i)$ for $u_i := d_0 \cdots d_{i-1}$.

$G.[\mathsf{S}\colon m]$ Separator plays the mode $m_i := \overline{m}(\ell_i, a_i) \in \{\vee, \wedge\}$.

$G.[\mathsf{S}\colon f]$ Separator plays either

- **a.** a selector $f_i := \overline{f}(\ell_i, a_i) \in \{\mathsf{L}, \mathsf{R}\}^{\Delta^{\mathcal{A}}(a_i)}$ for $\mathcal{A}$ if $m_i = \vee$ or
- **b.** a selector $f_i := \overline{f}(\ell_i, a_i) \in \{\mathsf{L}, \mathsf{R}\}^{\Delta^{\mathcal{B}}(a_i)}$ for $\mathcal{B}$ if $m_i = \wedge$.

$G_{\mathcal{S}}.[\mathsf{A}\colon \delta]$ Automaton plays the transition $\delta_i^{\mathcal{S}} \in \Delta^{\mathcal{S}}(\ell_i, a_i)$, defined as follows. Let $\delta_i^{\mathcal{A}} := \big(\rho^{\mathcal{A}}(u_i), t(u_i), \rho^{\mathcal{A}}(u_i\mathsf{L}), \rho^{\mathcal{A}}(u_i\mathsf{R})\big)$ be the $\mathcal{A}$-transition used in u_i by the run $\rho^{\mathcal{A}}$. We distinguish two cases.

- **a.** In the first case, assume that Separator played $m_i = \vee$ and $f_i \in \{\mathsf{L}, \mathsf{R}\}^{\Delta^{\mathcal{A}}(a_i)}$. It means that $\Delta^{\mathcal{S}}\big((\ell_i, q_i), a_i\big)$ contains two disjunctive transitions, $\delta^{\mathcal{S}}_{\mathsf{L},i} := (\ell_i, a_i, \ell_{\mathsf{L},i}, \top)$ and $\delta^{\mathcal{S}}_{\mathsf{R},i} := (\ell_i, a_i, \top, \ell_{\mathsf{R},i})$. Let us put $\delta_i^{\mathcal{S}} := \delta^{\mathcal{S}}_{f_i(\delta_i^{\mathcal{A}}),i}$, i.e., the transition that sends a non-$\top$ state in the direction given by $f_i(\delta_i^{\mathcal{A}})$.
- **b.** In the second case, Separator played $m_i = \wedge$ and $f_i \in \{\mathsf{L}, \mathsf{R}\}^{\Delta^{\mathcal{B}}(a_i)}$. It means that $\Delta^{\mathcal{S}}\big(\ell_i, a_i\big)$ contains one conjunctive transition $\delta_i^{\mathcal{S}} := (\ell_i, a_i, \ell_{\mathsf{L},i}, \ell_{\mathsf{R},i})$.

$G_{\mathcal{S}}.[\mathsf{I}\colon d]$ Input plays an arbitrary direction $d_i \in \{\mathsf{L}, \mathsf{R}\}$.

$G.[\mathsf{I}\colon d]$ Input plays the direction $d_i \in \{\mathsf{L}, \mathsf{R}\}$.

If $m_i = \vee$ and $d_i \neq f_i(\delta_i^{\mathcal{A}})$ then the next position of the acceptance game $G_{\mathcal{S}}$ is $(u_i d_i, \top)$, which is a winning position for Automaton. Therefore, w.l.o.g. we assume that:

$$\forall i \in \omega.\ (m_i = \vee) \Rightarrow f_i(\delta_i^{\mathcal{A}}) = d_i. \tag{1}$$

Moreover, the new state of $\mathcal{S}$ in $G_{\mathcal{S}}$ is $\ell_{i+1} := \tau(\ell_i, a_i, d_i)$. Similarly, the new memory state of $\mathcal{M}$ in G is ℓ_{i+1}. This concludes the description of the i-th round of both games. Clearly the invariant is preserved. We argue that Automaton wins the resulting infinite play $(\delta_0^{\mathcal{S}}, d_0)(\delta_1^{\mathcal{S}}, d_1)\cdots$ of the acceptance game $G_{\mathcal{S}}$. Consider the infinite play $\pi = (a_0, m_0, f_0, d_0)(a_1, m_1, f_1, d_1)\cdots$ of the separability game G. Since the run $\rho^{\mathcal{A}}$ is accepting, the infinite sequence of $\mathcal{A}$-transitions $\delta_0^{\mathcal{A}}\delta_1^{\mathcal{A}}\cdots$ is accepting. Thus, (1) implies that $\pi \in \mathbf{W}_{\mathcal{A}}$. Therefore, the infinite path $b := (a_0, d_0)(a_1, d_1)\cdots$ belongs to $L_{\mathcal{A}} \subseteq \mathrm{L}(\mathcal{D})$ and thus the corresponding infinite play $(\delta_0^{\mathcal{S}}, d_0)(\delta_1^{\mathcal{S}}, d_1)\cdots$ of the acceptance game $G_{\mathcal{S}}$ is winning for Automaton, as required. This concludes the argument establishing $\mathrm{L}(\mathcal{A}) \subseteq \mathrm{L}(\mathcal{S})$.

It remains to show that $\mathrm{L}(\mathcal{S}) \perp \mathrm{L}(\mathcal{B})$, which is the same as $\mathrm{L}(\mathcal{B}) \subseteq \mathrm{L}(\mathcal{S}^{\mathrm{c}})$ for the complement game automaton. This follows directly from the construction above via the duality of the game G. ◀

Completeness. Assume that there exists a game automaton $\mathcal{S}$ that separates $\mathrm{L}(\mathcal{A})$ from $\mathrm{L}(\mathcal{B})$. We need to show that Separator wins the separability game $G := G^{\mathrm{sep}}_{\mathrm{game}}(\mathcal{A}, \mathcal{B})$. Let $\mathcal{R} := \mathcal{S}^{\mathrm{c}}$ be the syntactic dual of the game automaton $\mathcal{S}$ as in Lemma 2. Thus, the automata $\mathcal{S}$ and $\mathcal{R}$ share the same set of states. Also, their transitions are related: the conjunctive transitions of $\mathcal{S}$ correspond to disjunctive transitions of $\mathcal{R}$ and vice versa. By slightly rephrasing the separation condition, we have $\mathrm{L}(\mathcal{A}) \perp \mathrm{L}(\mathcal{R})$ and $\mathrm{L}(\mathcal{B}) \perp \mathrm{L}(\mathcal{S})$. This means that Pathfinder wins both disjointness games $G^{\mathrm{dis}}(\mathcal{R}, \mathcal{A})$ and $G^{\mathrm{dis}}(\mathcal{S}, \mathcal{B})$. Thus, we can apply Lemma 4 to obtain pathfinders $\mathcal{P}_{\mathcal{A}}\colon \big(\bigcup_{a\in\Sigma} \Delta^{\mathcal{R}}(a) \times \Delta^{\mathcal{A}}(a)\big) \to \{\mathsf{L}, \mathsf{R}\}$ and $\mathcal{P}_{\mathcal{B}}\colon \big(\bigcup_{a\in\Sigma} \Delta^{\mathcal{S}}(a) \times \Delta^{\mathcal{B}}(a)\big) \to \{\mathsf{L}, \mathsf{R}\}$.

We will now provide a strategy of Separator in G. The constructed strategy uses as its memory states the set of states of $\mathcal{S}$ that are distinct than $\top$. Let the initial memory state be q_0. Assume that the current memory state is q_i and consider the i-th round of the game.

[I: a] Input plays an arbitrary letter $a_i \in \Sigma$.

[S: m] Separator plays the mode $m_i \in \{\vee, \wedge\}$ defined as follows. We consider the following two cases for the mode of the transitions $\Delta^{\mathcal{S}}(q_i, a_i)$.

- **a.** If $\Delta^{\mathcal{S}}(q_i, a_i) = \{\delta_i^{\mathcal{S}}\}$ is a single conjunctive transition $\delta_i^{\mathcal{S}} = (q_i, a_i, q_{\mathsf{L},i}, q_{\mathsf{R},i})$ then we put $m_i := \wedge$ and $f_i := \mathcal{P}_{\mathcal{B}}(\delta_i^{\mathcal{S}}, \_)$ is a selector for $\mathcal{B}$.
- **b.** Otherwise, $\Delta^{\mathcal{S}}(q_i, a_i)$ is a pair of disjunctive transitions which means that $\Delta^{\mathcal{R}}(q_i, a_i)$ is a single conjunctive transition $\delta_i^{\mathcal{R}} = (q_i, a_i, q_{\mathsf{L},i}, q_{\mathsf{R},i})$. In this case we put $m_i := \vee$ and $f_i := \mathcal{P}_{\mathcal{A}}(\delta_i^{\mathcal{R}}, \_)$ is a selector for $\mathcal{A}$.

[S: f] Separator plays the selector f_i defined above (notice that f_i is either a selector for $\mathcal{A}$ or for $\mathcal{B}$, according to m_i).

[I: d] Input plays an arbitrary direction $d_i \in \{\mathsf{L}, \mathsf{R}\}$.

The next memory state of our strategy is the state $q_{d_i,i}$ taken from one of the transitions $\delta_i^{\mathcal{S}}$ or $\delta_i^{\mathcal{R}}$, see above. We now argue that Separator wins the corresponding infinite play $\pi = (a_0, m_0, f_0, d_0)(a_1, m_1, f_1, d_1)\cdots$. Let $b = (a_0, d_0)(a_1, d_1)\cdots$ be the corresponding path. Consider a number $i \in \omega$. By the construction of the strategy above, we have two cases:

1. If $m_i = \wedge$, then a conjunctive transition $\delta_i^{\mathcal{S}} = (q_i, a_i, q_{\mathsf{L},i}, q_{\mathsf{R},i})$ of $\mathcal{S}$ was used to determine f_i. In this case, define $\delta_i^{\mathcal{R}}$ as the following disjunctive transition of $\mathcal{R}$: if $d_i = \mathsf{L}$ then $\delta_i^{\mathcal{R}} := (q_i, a_i, q_{\mathsf{L},i}, \top)$, otherwise $d_i = \mathsf{R}$ and $\delta_i^{\mathcal{R}} := (q_i, a_i, \top, q_{\mathsf{R},i})$.
2. If $m_i = \vee$, then a conjunctive transition $\delta_i^{\mathcal{R}} = (q_i, a_i, q_{\mathsf{L},i}, q_{\mathsf{R},i})$ of $\mathcal{R}$ was used to determine f_i. In this case, define $\delta_i^{\mathcal{S}}$ as the following disjunctive transition of $\mathcal{S}$: if $d_i = \mathsf{L}$ then $\delta_i^{\mathcal{S}} := (q_i, a_i, q_{\mathsf{L},i}, \top)$, otherwise $d_i = \mathsf{R}$ and $\delta_i^{\mathcal{S}} := (q_i, a_i, \top, q_{\mathsf{R},i})$.

The definitions above provide two sequences of transitions $\vec{\delta}^{\mathcal{S}} := \delta_0^{\mathcal{S}}\delta_1^{\mathcal{S}}\cdots \in \Delta^{\mathcal{S}}(b)$, $\vec{\delta}^{\mathcal{R}} := \delta_0^{\mathcal{R}}\delta_1^{\mathcal{R}}\cdots \in \Delta^{\mathcal{R}}(b)$. Since for every $i \in \omega$ the transitions $\delta_i^{\mathcal{S}}$ and $\delta_i^{\mathcal{R}}$ are from the same state $q_i \neq \top$, $\vec{\delta}^{\mathcal{S}}$ is accepting in $\mathcal{S}$ if, and only if, $\vec{\delta}^{\mathcal{R}}$ is rejecting in $\mathcal{R}$. Assume that $\vec{\delta}^{\mathcal{S}}$ is accepting (the other case is analogous). We will show that $\mathbf{W}_{\mathcal{B}}$ is violated (if $\vec{\delta}^{\mathcal{R}}$ is accepting then $\mathbf{W}_{\mathcal{A}}$ is violated). Assume for the sake of contradiction that $\mathbf{W}_{\mathcal{B}}$ holds, as witnessed by a sequence of $\mathcal{B}$-transitions $\vec{\delta}^{\mathcal{B}} = \delta_0^{\mathcal{B}}\delta_1^{\mathcal{B}}\cdots \in \Delta^{\mathcal{B}}(b)$. By Remark 6 we obtain that whenever $m_i = \vee$ and $\delta_i^{\mathcal{S}}$ is a disjunctive transition of $\mathcal{S}$ then $\mathcal{P}_{\mathcal{B}}(\delta_i^{\mathcal{S}}, \_)$ is constantly equal to d_i. By the assumption on $\vec{\delta}^{\mathcal{B}}$ from $\mathbf{W}_{\mathcal{B}}$ we know that whenever $m_i = \wedge$ then $f_i(\delta_i^{\mathcal{B}}) = d_i$. However, if $m_i = \wedge$ then $f_i(\delta_i^{\mathcal{B}}) = \mathcal{P}_{\mathcal{B}}(\delta_i^{\mathcal{S}}, \delta_i^{\mathcal{B}})$. Therefore, in both cases we know that $\mathcal{P}_{\mathcal{B}}(\delta_i^{\mathcal{S}}, \delta_i^{\mathcal{B}}) = d_i$. This means that the assumptions of Corollary 5 are met and at least one of the sequences $\vec{\delta}^{\mathcal{S}}$, $\vec{\delta}^{\mathcal{B}}$ is rejecting – a contradiction, since we assumed both these sequences to be accepting. ◀

6 Separability by game automata with priorities in C

In this section we present our last game-theoretic characterisation, namely game automata separability for a fixed finite set $C \subseteq \mathbb{N}$ of priorities. Fix two automata $\mathcal{A} = (\Sigma, Q^{\mathcal{A}}, q_0^{\mathcal{A}}, \Omega^{\mathcal{A}}, \Delta^{\mathcal{A}})$ and $\mathcal{B} = (\Sigma, Q^{\mathcal{B}}, q_0^{\mathcal{B}}, \Omega^{\mathcal{B}}, \Delta^{\mathcal{B}})$ over the same alphabet Σ. The game is a variation of $G^{\text{sep}}_{\text{game}}(\mathcal{A}, \mathcal{B})$ from Section 5 where Separator additionally plays priorities from C.

C-game-automata separability game $G^{\text{sep}}_{\text{game}}(\mathcal{A}, \mathcal{B}, C)$

At the i-th round:

- [S: c] Separator plays a priority $c_i \in C$.
- [I: a] Input plays a letter $a_i \in \Sigma$.
- [S: m] Separator plays a mode $m_i \in \{\vee, \wedge\}$.
- [S: f] Separator plays either
  - **a.** a selector $f_i \in \{\mathsf{L}, \mathsf{R}\}^{\Delta^{\mathcal{A}}(a_i)}$ for $\mathcal{A}$ if $m_i = \vee$, or
  - **b.** a selector $f_i \in \{\mathsf{L}, \mathsf{R}\}^{\Delta^{\mathcal{B}}(a_i)}$ for $\mathcal{B}$ if $m_i = \wedge$.
- [I: d] Input plays a direction $d_i \in \{\mathsf{L}, \mathsf{R}\}$.

Separator wins an infinite play $\pi = (c_0, a_0, m_0, f_0, d_0)(c_1, a_1, m_1, f_1, d_1)\cdots$ inducing a path $b = (a_0, d_0)(a_1, d_1)\cdots$ whenever both conditions below hold:

1. $\pi \in \mathbf{W}_{\mathcal{A}}$: If there exists an accepting sequence of transitions $\vec{\delta}^{\mathcal{A}} = \delta_0^{\mathcal{A}}\delta_1^{\mathcal{A}}\cdots \in \Delta^{\mathcal{A}}(b)$ s.t. for all $i \in \omega$ we have $(m_i = \vee) \Rightarrow f_i(\delta_i^{\mathcal{A}}) = d_i$, then $c_0c_1\cdots$ is accepting.
2. $\pi \in \mathbf{W}_{\mathcal{B}}$: If there exists an accepting sequence of transitions $\vec{\delta}^{\mathcal{B}} = \delta_0^{\mathcal{B}}\delta_1^{\mathcal{B}}\cdots \in \Delta^{\mathcal{B}}(b)$ s.t. for all $i \in \omega$ we have $(m_i = \wedge) \Rightarrow f_i(\delta_i^{\mathcal{B}}) = d_i$, then $c_0c_1\cdots$ is rejecting.

▶ **Lemma 12.** *Separator wins $G^{\text{sep}}_{\text{game}}(\mathcal{A}, \mathcal{B}, C)$ if, and only if, there exists a game automaton $\mathcal{S}$ with priorities in C separating* $\mathrm{L}(\mathcal{A})$, $\mathrm{L}(\mathcal{B})$.

This lemma can be proved similarly as Lemma 10 except for the acceptance condition of the separator which is given by the priorities c_i's as in the proof of Lemma 7.

References

1 David Barozzini, Lorenzo Clemente, Thomas Colcombet, and Paweł Parys. Cost Automata, Safe Schemes, and Downward Closures. In *Proc. of ICALP'20*, LIPIcs, pages 109:1–109:18, 2020. doi:10.4230/LIPIcs.ICALP.2020.109.

2 Achim Blumensath. Recognisability for algebras of infinite trees. *Theoretical Computer Science*, 412(29):3463–3486, 2011.

3 Mikołaj Bojańczyk. Star height via games. In *LICS*, pages 214–219, 2015.

4 Mikołaj Bojańczyk, Filippo Cavallari, Thomas Place, and Michał Skrzypczak. Regular tree languages in low levels of the Wadge Hierarchy. *Log. Meth. Comput. Sci.*, Volume 15, Issue 3, 2019. URL: `https://lmcs.episciences.org/5743`.

5 Mikołaj Bojańczyk and Wojciech Czerwiński. An automata toolbox, February 2018. URL: `https://www.mimuw.edu.pl/~bojan/paper/automata-toolbox-book`.

6 Mikołaj Bojańczyk and Thomas Place. Regular languages of infinite trees that are boolean combinations of open sets. In *Proc. of ICALP'12*, ICALP'12, pages 104—-115, Berlin, Heidelberg, 2012. Springer-Verlag. `doi:10.1007/978-3-642-31585-5_13`.

7 J. Richard Büchi and Lawrence H. Landweber. Solving sequential conditions by finite-state strategies. *Transactions of the American Mathematical Society*, 138:295–311, 1969. URL: `http://www.jstor.org/stable/1994916`.

8 Cristian S. Calude, Sanjay Jain, Bakhadyr Khoussainov, Wei Li, and Frank Stephan. Deciding parity games in quasipolynomial time. In *Proc. of STOC'17*, 2017.

9 Filippo Cavallari, Henryk Michalewski, and Michał Skrzypczak. A characterisation of Π^0_2 regular tree languages. In *Proc. of MFCS'17*, 2017.

10 Lorenzo Clemente, Wojciech Czerwiński, Sławomir Lasota, and Charles Paperman. Regular separability of parikh automata. In Ioannis Chatzigiannakis, Piotr Indyk, Fabian Kuhn, and Anca Muscholl, editors, *Proc. of ICALP'17*, volume 80, pages 117:1–117:13, 2017. `doi:10.4230/LIPIcs.ICALP.2017.117`.

11 Lorenzo Clemente, Wojciech Czerwiński, Sławomir Lasota, and Charles Paperman. Separability of Reachability Sets of Vector Addition Systems. In *Proc. of STACS'17*, volume 66 of *LIPICs*, pages 24:1–24:14, 2017. `doi:10.4230/LIPIcs.STACS.2017.24`.

12 Lorenzo Clemente, Sławomir Lasota, and Radosław Piórkowski. Determinisability of One-Clock Timed Automata. In *Proc. of CONCUR'20*, volume 171 of *LIPIcs*, pages 42:1–42:17, 2020. `doi:10.4230/LIPIcs.CONCUR.2020.42`.

13 Lorenzo Clemente, Sławomir Lasota, and Radosław Piórkowski. Timed Games and Deterministic Separability. In *Proc. of ICALP'20*, volume 168 of *LIPIcs*, pages 121:1–121:16, 2020. `doi:10.4230/LIPIcs.ICALP.2020.121`.

14 Lorenzo Clemente, Paweł Parys, Sylvain Salvati, and Igor Walukiewicz. The diagonal problem for higher-order recursion schemes is decidable. In *Proc. of LICS'16*, 2016. `doi:10.1145/2933575.2934527`.

15 Lorenzo Clemente and Michał Skrzypczak. Deterministic and game separability for regular languages of infinite trees, May 2021. `arXiv:2105.01137`.

16 Thomas Colcombet. Fonctions régulières de coût. Habilitation thesis, Université Paris Diderot—Paris 7, 2013.

17 Thomas Colcombet, Denis Kuperberg, Christof Löding, and Michael Vanden Boom. Deciding the weak definability of Büchi definable tree languages. In *Proc. of CSL'13*, LIPIcs, pages 215–230, 2013. `doi:10.4230/LIPIcs.CSL.2013.215`.

18 Thomas Colcombet and Christof Löding. The Non-deterministic Mostowski Hierarchy and Distance-Parity Automata. In *Proc. of ICALP'08*, pages 398–409, 2008. `doi:10.1007/978-3-540-70583-3_33`.

19 William Craig. Three uses of the herbrand-gentzen theorem in relating model theory and proof theory. *The Journal of Symbolic Logic*, 22(3):269–285, 1957. URL: `http://www.jstor.org/stable/2963594`.

20 Wojciech Czerwiński, Laure Daviaud, Nathanaël Fijalkow, Marcin Jurdziński, Ranko Lazić, and Paweł Parys. Universal trees grow inside separating automata: Quasi-polynomial lower bounds for parity games. In *Proc. of SODA'19*, pages 2333–2349, USA, 2019. Society for Industrial and Applied Mathematics.

21 Wojciech Czerwiński and Sławomir Lasota. Regular Separability of One Counter Automata. *Logical Methods in Computer Science*, Volume 15, Issue 2, June 2019. URL: `https://lmcs.episciences.org/5563`.

22 Wojciech Czerwiński, Sławomir Lasota, Roland Meyer, Sebastian Muskalla, K. Narayan Kumar, and Prakash Saivasan. Regular Separability of Well-Structured Transition Systems. In *Proc. of CONCUR'18*, LIPIcs, pages 35:1–35:18, 2018. doi:10.4230/LIPIcs.CONCUR.2018.35.

23 Wojciech Czerwiński, Wim Martens, Lorijn van Rooijen, and Marc Zeitoun. A note on decidable separability by piecewise testable languages. In *Proc. of FCT'15*, 2015. doi:10.1007/978-3-319-22177-9_14.

24 Wojciech Czerwiński and Georg Zetzsche. An approach to regular separability in vector addition systems. In Holger Hermanns, Lijun Zhang, Naoki Kobayashi, and Dale Miller, editors, *LICS '20: 35th Annual ACM/IEEE Symposium on Logic in Computer Science, Saarbrücken, Germany, July 8-11, 2020*, pages 341–354. ACM, 2020. doi:10.1145/3373718.3394776.

25 E. Allen Emerson and Charanjit S. Jutla. Tree automata, mu-calculus and determinacy. In *Proc. of SFCS'91*, pages 368–377. IEEE Computer Society, 1991. doi:10.1109/SFCS.1991.185392.

26 Alessandro Facchini, Filip Murlak, and Michał Skrzypczak. Index problems for game automata. *ACM Trans. Comput. Logic*, 17(4):24:1–24:38, November 2016. doi:10.1145/2946800.

27 Erich Grädel, Wolfgang Thomas, and Thomas Wilke, editors. *Automata Logics, and Infinite Games - A Guide to Current Research*. Springer, 2002.

28 Alexander Kechris. *Classical Descriptive Set Theory*. Springer-Verlag, 1995.

29 Ralf Küsters and Thomas Wilke. Deciding the first level of the μ-calculus alternation hierarchy. In *Proc. of FSTTCS'02*, pages 241–252, Berlin, 2002.

30 Robert McNaughton and Seymour Papert. *Counter-free automata*. M.I.T. Press research monographs. M.I.T. Press, 1971.

31 Filip Murlak. Weak index versus Borel rank. In *Proc. of STACS'08*, volume 1 of *LIPIcs*, pages 573–584, 2008. doi:10.4230/LIPIcs.STACS.2008.1318.

32 Damian Niwiński. On fixed-point clones (extended abstract). In *Proc. of ICALP'86*, pages 464–473, 1986. URL: http://dl.acm.org/citation.cfm?id=646240.683678.

33 Damian Niwiński and Igor Walukiewicz. Relating hierarchies of word and tree automata. In *Proc. of STACS'98*, pages 320–331, 1998.

34 Damian Niwiński and Igor Walukiewicz. A gap property of deterministic tree languages. *Theoretical Computer Science*, 303(1):215–231, 2003. doi:10.1016/S0304-3975(02)00452-8.

35 Damian Niwiński and Igor Walukiewicz. Deciding nondeterministic hierarchy of deterministic tree automata. *Electronic Notes in Theoretical Computer Science*, 123:195–208, 2005. Proceedings of the 11th Workshop on Logic, Language, Information and Computation (WoLLIC 2004). doi:10.1016/j.entcs.2004.05.015.

36 Thomas Place and Marc Zeitoun. Separating regular languages with first-order logic. *Log. Methods Comput. Sci.*, 12(1), 2016. doi:10.2168/LMCS-12(1:5)2016.

37 Thomas Place and Marc Zeitoun. Adding successor: A transfer theorem for separation and covering. *ACM Trans. Comput. Logic*, 21(2), 2019. doi:10.1145/3356339.

38 Michael O. Rabin. Weakly definable relations and special automata. In *Mathematical Logic and Foundations of Set Theory*, volume 59 of *Studies in Logic and the Foundations of Mathematics*, pages 1–23. Elsevier, 1970. doi:10.1016/S0049-237X(08)71929-3.

39 Luigi Santocanale and André Arnold. Ambiguous classes in mu-calculi hierarchies. *Theoretical Computer Science*, 333(1):265–296, 2005. Foundations of Software Science and Computation Structures. doi:10.1016/j.tcs.2004.10.024.

40 Marcel Paul Schützenberger. On finite monoids having only trivial subgroups. *Information and Control*, 8(2):190–194, 1965.

41 Michał Skrzypczak and Igor Walukiewicz. Deciding the Topological Complexity of Büchi Languages. In *Proc. of ICALP'16*, volume 55 of *LIPIcs*, pages 99:1–99:13, 2016. doi:10.4230/LIPIcs.ICALP.2016.99.

42 Tomasz Fryderyk Urbanski. On Deciding if Deterministic Rabin Language Is in Büchi Class. In *Proc. of ICALP'00*, volume 1853 of *LNCS*, pages 663–674. Springer, 2000. doi:10.1007/3-540-45022-X_56.

43 Klaus Wagner. On omega-regular sets. *Information and Control*, 43(2):123–177, 1979. doi:10.1016/S0019-9958(79)90653-3.

44 Igor Walukiewicz. Deciding low levels of tree-automata hierarchy. *ENTCS*, 67:61–75, 2002. doi:10.1016/S1571-0661(04)80541-3.

45 Georg Zetzsche. An approach to computing downward closures. In *Proc. of ICALP'15*, volume 9135 of *LNCS*, pages 440–451, 2015. doi:10.1007/978-3-662-47666-6_35.

A Complexity Approach to Tree Algebras: the Bounded Case

Thomas Colcombet
Université de Paris, CNRS, IRIF, F-75006, Paris, France

Arthur Jaquard
Université de Paris, CNRS, IRIF, F-75006, Paris, France

Abstract

In this paper, we initiate a study of the expressive power of tree algebras, and more generally infinitely sorted algebras, based on their asymptotic complexity. We provide a characterization of the expressiveness of tree algebras of bounded complexity.

Tree algebras in many of their forms, such as clones, hyperclones, operads, etc, as well as other kind of algebras, are infinitely sorted: the carrier is a multi sorted set indexed by a parameter that can be interpreted as the number of variables or hole types. Finite such algebras – meaning when all sorts are finite – can be classified depending on the asymptotic size of the carrier sets as a function of the parameter, that we call the complexity of the algebra. This naturally defines the notions of algebras of bounded, linear, polynomial, exponential or doubly exponential complexity...

We initiate in this work a program of analysis of the complexity of infinitely sorted algebras. Our main result precisely characterizes the tree algebras of bounded complexity based on the languages that they recognize as Boolean closures of simple languages. Along the way, we prove that such algebras that are syntactic (minimal for a language) are exactly those in which, as soon as there are sufficiently many variables, the elements are invariant under permutation of the variables.

2012 ACM Subject Classification Theory of computation → Tree languages; Theory of computation → Regular languages

Keywords and phrases Tree algebra, infinite tree, language theory

Digital Object Identifier 10.4230/LIPIcs.ICALP.2021.127

Category Track B: Automata, Logic, Semantics, and Theory of Programming

Funding *Thomas Colcombet*: Supported by the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreement No.670624), and the DeLTA ANR project (ANR-16-CE40-0007).

1 Introduction

Infinitely sorted algebras occur naturally in many contexts of language theory, graph theory or logic. A typical example is the case of tree algebras (such as clones, hyperclones, operads): plugging a subtree into another one requires a mechanism for identifying the leaf/leaves in which the substitution has to be performed. Notions such as variables, hole types, or colors are used for that. Another example is the one of graphs (HR- and VR-algebras [8]) in which basic operations (a) glue graphs together using a set of colors (sometimes called ports) for identifying the glue-points, or (b) add all possible edges between vertices of fixed given colors. In these examples, the algebras are naturally sliced into infinitely many sorts based on the number of variables/hole types/colors that are used simultaneously.

However, a technical difficulty arises immediately when using such algebras. Even when all sorts are finite (what we call a finite algebra), these algebras are not really finite due to the infinite number of sorts. This forbids, for instance, to entirely and explicitly describe the whole algebra in a finite way. And this is of course a problem for describing and using these algebras in an algorithm. Indeed, a concrete algorithm can only maintain a subset of the

algebra in its memory, say up to some given sort, or resort to other forms of representations, which often means not really working with the algebra. This is particularly annoying since often these objects are used for analyzing properties that admit finite descriptions, such as tree automata or logical formulae (eg in monadic second-order logic for describing properties of graphs).

This hurdle to handle infinitely sorted algebras can, arguably, be seen as one of the causes of the many years that it took before having a good definition of an algebra for infinite trees [2], or the time that it took before it was possible to characterize logically the expressiveness of recognizable properties of graphs under bounded tree-width hypothesis [4]. Also, the long history of results characterizing language families by decidable algebraic properties (initiated by the famous Schützenberger result [11]) has proven hard to extend to these more complex objects, such as trees.

Classifying algebras based on their complexity. In this paper, we initiate a new approach in the study of such algebras, which is to try to understand infinitely sorted algebras in simpler cases. And we define these simpler cases using complexity considerations. Indeed, in each of the above cases, the sorts are naturally indexed by a natural number parameter: the number of variables, or hole types, or colors. Hence, an algebra $\mathcal{A}$ would have a carrier of the form

$$(A_n)_{n\in\mathbb{N}},$$

together with suitable operations that depend on the particular algebra type. This algebra will be called *finite* if all the A_n sets are finite, and, in this case, we naturally define the *complexity map* of the algebra $c_{\mathcal{A}} \colon \mathbb{N} \to \mathbb{N}$ as follows:

$$c_{\mathcal{A}}(n) = |A_n| \ , \qquad \text{for all } n \in \mathbb{N}.$$

Finite algebras can naturally be classified using this map c. Simple classes are then *algebras of bounded complexity* if $c_{\mathcal{A}}$ is bounded, of *polynomial complexity* if $c_{\mathcal{A}}$ is bounded from above by some polynomial in n, etc.

Other interesting complexity classes can be defined using orbits. Indeed, in all of the mentioned examples, there is a natural operation that performs a renaming of the variables/hole types/colors. This renaming is parameterized by a bijection over variables/hole types/colors, and this permutation acts on the corresponding sort. Said differently, in all examples, there is an action of the symmetric group over n elements, $\mathbf{Sym}(n)$, over A_n. It is thus natural to consider the *orbit complexity map* $c^{\circ}_{\mathcal{A}}$ as follows

$$c^{\circ}_{\mathcal{A}}(n) = |A_n/\mathbf{Sym}(n)| \ , \qquad \text{for all } n \in \mathbb{N},$$

and define accordingly what are finite algebras of *bounded orbit complexity*, or *polynomial orbit complexity*, etc.

Related works. As mentioned above, there is a long history of understanding the expressive power of regular languages of words based on algebraic properties. The first work in this direction [11], characterizing star-free languages, intiated a long list of deep results. It was natural to extend this approach to trees. Here, the notion of algebra was less obvious, and several definitions have been used. Some algebras for trees are one sorted, such as deterministic bottom up automata (that can be seen as algebras). Some are two sorted, such as forest algebras [7]. Some others, such as preclones [9], have infinitely many sorts.

Characterizations of classes have been obtained using these approaches [5, 10, 6], but remain very limited due to difficulties inherent to the tree case. The study of algebras for infinite trees renewed the interest in these questions [1, 2, 3]. This line of works also highlights the difficulty to work with tree algebras, and the poor understanding we have so far of the mechanism of recognition for infinite objects.

Contributions of the paper. In this paper we establish some first results in this complexity analysis of infinitely sorted algebras, for the simplest complexity class, bounded complexity. Our results are of two kinds: a characterization of algebras of bounded complexity; and a characterization of the languages that they recognize, meaning we give a syntactic description of the properties that can be recognized by algebras in this class. We more particularly prove:

- A characterization of syntactic finite tree algebras of bounded complexity as those syntactic algebras in which, as soon as there are sufficiently many variables, the elements are invariant under permutation of variables. See Theorem 5.
- A characterization of languages recognized by finite tree algebras of bounded complexity as Boolean closures of simple languages. See Theorem 14.

The second result actually uses the first as a building block in its proof.

Structure of the paper. In Section 2, we recall some classical definitions, and introduce our notions of algebras. In Section 3, we look at the permutations of variables in finite tree algebras, and prove Theorem 5. In Section 4, we study in more depth the bounded complexity case for finite tree algebras, and establish our main result, Theorem 14. Section 5 is our conclusion.

2 Definitions

We denote by $\mathbb{N}$ the set of all non-negative integers. Given $n \in \mathbb{N}$, we write $[n] = \{0, 1, ..., n-1\}$. The symmetric group (resp. alternating group) over $[n]$ is denoted $\mathbf{Sym}(n)$ (resp. $\mathbf{Alt}(n)$), the symmetric group of any set X is denoted $\mathbf{Sym}(X)$. We denote by A^c the complement of a set A.

We fix a finite *ranked alphabet* Σ; the *arity* of a *symbol* $a \in \Sigma$ is denoted $\mathrm{ar}(a)$. It is a *constant* if $\mathrm{ar}(a) = 0$, and is *unary* if $\mathrm{ar}(a) = 1$. For $k \in \mathbb{N}$, we set $\Sigma_k = \{a \in \Sigma \mid \mathrm{ar}(a) = k\}$. A^* is the set of finite words over A, and $A^+ = A^* \setminus \{\varepsilon\}$.

2.1 Trees

In this section, we introduce notions and notations for trees.

We fix a countable set of *variables*. Given a finite set of variables X, a Σ, X-tree is, informally, a tree in which nodes are labelled by elements of Σ and leaves also possibly by variables. All variables have to appear at least once. Formally, a Σ, X*-tree* is a partial map $t \colon \mathbb{N}^* \to \Sigma \uplus X$ such that $\mathrm{dom}(t)$ is non-empty and prefix-closed, and furthermore:

- For all $u \in \mathrm{dom}(t)$ there exists $n \in \mathbb{N}$ such that $\{i \mid ui \in \mathrm{dom}(t)\} = [n]$, and
  - either $t(u) \in \Sigma_n$ (*symbol node*), or
  - $t(u) \in X$ and $n = 0$ (*variable node*). Note that a variable node is always a leaf.
- All variables from X appear in t, i.e. for all $x \in X$, $t(u) = x$ for some $u \in \mathrm{dom}(t)$.
- The root is not a variable, i.e. $t(\varepsilon) \notin X$.

$\Sigma, \emptyset$-trees are simply called Σ-*trees*. The elements in $\mathrm{dom}(t)$ are called *nodes*. The prefix relation over nodes is called the *ancestor relation*. The node ε is called the *root* of the tree. The tree t is *finite* if it has finitely many nodes. A *branch* of a tree t is a maximal set of nodes ordered under the ancestor relation. Let $\mathrm{FiniteTrees}(\Sigma, X)$ bet the set of finite Σ, X-trees, for all finite set of variables X.

Building trees. We introduce now some operations on trees. See Fig. 1.

- $a(x_0, \dots, x_{n-1})$, for $x_0, \dots, x_{n-1}$ variables and $a \in \Sigma_n$, denotes the $\Sigma, \{x_0, \dots, x_{n-1}\}$-tree consisting of a root labelled a, and children $0, \dots, n-1$ labelled with variables $x_0, \dots, x_{n-1}$ respectively.
- $s \cdot_x t$, for two trees $s \in \mathrm{FiniteTrees}(\Sigma, X)$, $t \in \mathrm{FiniteTrees}(\Sigma, Y)$ and a variable $x \in X$, is the $\Sigma, (X \setminus \{x\}) \cup Y$-tree s in which t is substituted for every occurrences of the variable x.
- $\widetilde{\sigma}(t)$, for a tree $t \in \mathrm{FiniteTrees}(\Sigma, X)$ and $\sigma \colon X \to Y$ a surjective map, is the Σ, Y-tree obtained as t in which variable $\sigma(x)$ has been substituted to x for all $x \in X$. Note that $\widetilde{\sigma} \circ \widetilde{\tau} = \widetilde{\sigma \circ \tau}$.
- $t[x_0 \leftarrow t_0, \dots, x_{n-1} \leftarrow t_{n-1}]$ denotes the tree of sort $X \setminus \{x_0, \dots, x_{n-1}\} \cup \bigcup_i Y_i$ obtained from t by simultaneously substituting the tree t_i for the variable x_i for all $i \in [n]$, where t is a tree of sort X, $x_0, \dots, x_{n-1} \in X$, and $t_0, \dots, t_{n-1}$ are trees of sort Y_i for all $i \in [n]$. Note that this operation is equivalent to a combination of the previous ones.
- $a(t_0, ..., t_{n-1})$, for $a \in \Sigma_n$, denotes the tree of root a and children $t_0, \dots, t_{n-1}$ at respective positions $0, \dots, n-1$. Again, this operation is equivalent to a combination of the previous ones.

▶ **Lemma 1.** *All finite trees can be obtained from the $a(x_0, \dots, x_{n-1})$'s using the operation $\cdot$.*

Expressions denoting finite trees. For X a finite set of variables, an FT_Σ-*expression of sort X (over the alphabet Σ)* is an expression built inductively as follows:

- $a(x_0, \dots, x_{n-1})$ is an FT_Σ-expression of sort $\{x_0, \dots, x_{n-1}\}$ for every symbol $a \in \Sigma_n$,
- $S \cdot_x T$ is an FT_Σ-expression of sort $X \setminus \{x\} \cup Y$ for all FT_Σ-expressions S of sort X, all FT_Σ-expressions T of sort Y, and all variables $x \in X$ (*substitution*),
- $\tilde{\sigma}(T)$ is an FT_Σ-expression of sort Y for all FT_Σ-expressions T of sort X, and surjective map $\sigma \colon X \to Y$ (*renaming*).

For an FT_Σ-expression T of sort X, $[\![T]\!]$ denotes its evaluation into a finite Σ, X-tree using the operations of substitution and renaming.

Contexts. We define now contexts, which are terms with a specific leaf called the hole. Since we work in a multi-sorted algebra, the hole itself has a sort. Essentially, to a hole of sort X will be substituted a term of sort X. Formally, for fixed finite set of variables Y, an *context of sort X with hole of sort Y* (or simply a FT_Σ-*context*) is defined inductively as an expression of sort X, using the extra construction $\Box_Y$ (the *hole of sort Y*) which is a context of sort Y with hole of sort Y. This new construction may appear multiple times in a context but has to appear at least once.

For C a context of sort X with hole of sort Y, $[\![C]\!] \colon \mathrm{FiniteTrees}(\Sigma, Y) \to \mathrm{FiniteTrees}(\Sigma, X)$ is the function which to a tree of sort Y t associates the tree of sort X obtained by evaluating the operations as above, interpreting $\Box_Y$ as t.

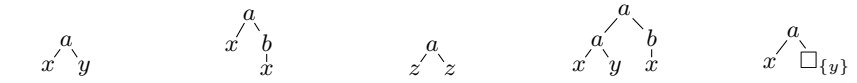


Figure 1 Trees and contexts with their notations. Here $\sigma(x) = \sigma(y) = z$.

2.2 Finite tree algebras

Our notion of tree algebra is the natural notion associated to finite trees equipped with the above operations. We give here a more formal definition, though the detail of identities is more for reference. What matters is that it is defined such that the free algebra coincides with finite trees.

An *FT_Σ-algebra* $\mathcal{A}$ consists of an infinite collection of carrier sets A_X indexed by finite sets of variables X, together with operations:

- $\sigma^{\mathcal{A}} \colon A_X \to A_Y$ for all surjective maps $\sigma \colon X \to Y$,
- $a(x_0, \dots, x_{n-1})^{\mathcal{A}} \in A_{\{x_0,\dots,x_{n-1}\}}$ for all $a \in \Sigma_n$ and variables $x_0, \dots, x_{n-1}$,
- $\cdot_x^{\mathcal{A}} \colon A_X \times A_Y \to A_{X \setminus \{x\} \cup Y}$ for all finite sets of variables X, Y and $x \in X$,

that satisfy the expected identities, i.e. the ones guaranteeing that several ways to describe the same tree yield the same evaluation in the algebra. Formally, for all s, t, u that belong to A_X, A_Y, A_Z respectively,

- $(s \cdot_x^{\mathcal{A}} t) \cdot_y^{\mathcal{A}} u = x \cdot_x^{\mathcal{A}} (t \cdot_y^{\mathcal{A}} u)$ for all $x \in X \cap Z$ and $y \in X \cap Y$ (horizontal associativity), and
- $(s \cdot_x^{\mathcal{A}} t) \cdot_y^{\mathcal{A}} u = x \cdot_x^{\mathcal{A}} (t \cdot_y^{\mathcal{A}} u)$ for all $x \in X$ and $y \in Y \setminus X \cup \{x\}$ (vertical associativity),

for all s, t that belong to A_X, A_Y, $x \in X$ and surjection $\sigma \colon X \to Y$,

- $\sigma^{\mathcal{A}}(s \cdot_x^{\mathcal{A}} t) = \sigma^{\mathcal{A}}(s) \cdot_x^{\mathcal{A}} t$ if $\sigma^{-1}(\sigma(x)) = \{x\}$ and $\sigma(y) = y$ for all $y \in X \cap Y \setminus \{x\}$,
- $\sigma^{\mathcal{A}}(s \cdot_x^{\mathcal{A}} t) = s \cdot_x \sigma^{\mathcal{A}}(t)$ if $\sigma(y) = y$ for all $y \in X \cap Y \setminus \{x\}$,

for all surjective maps $\sigma \colon X \to Y$ and $\tau \colon Y \to Z$, $(\tau \circ \sigma)^{\mathcal{A}} = \tau^{\mathcal{A}} \circ \sigma^{\mathcal{A}}$, and for all surjections $\sigma \colon \{x_0, \dots x_{n-1}\} \to Y$ and $a \in \Sigma_n$, $\sigma^{\mathcal{A}}(a(x_0, \dots, x_{n-1})^{\mathcal{A}}) = a(\sigma(x_0), \dots, \sigma(x_{n-1}))^{\mathcal{A}}$. In practice, we shall not explicitly use these identities, and simply write two elements of the algebra equal as soon as they obviously come from expressions denoting the same trees.

A *morphism of FT_Σ-algebras* from $\mathcal{A}$ to $\mathcal{B}$ is a family of maps $\alpha_X \colon A_X \to B_X$ for all finite sets of variables X which preserves all operations, i.e. $\alpha_Y(\sigma^{\mathcal{A}}(s)) = \sigma^{\mathcal{B}}(\alpha_X(s))$ for all surjective map $\sigma \colon X \to Y$, $\alpha(a(x_0, \dots, x_{n-1})^{\mathcal{A}}) = a(x_0, \dots, x_{n-1})^{\mathcal{B}}$, and $\alpha_{X \setminus \{x\} \cup Y}(s \cdot_x^{\mathcal{A}} t) = \alpha_X(s) \cdot_x^{\mathcal{B}} \alpha_Y(t)$ for all $s \in A_X$, $t \in A_Y$ and $x \in X$.

The FiniteTrees(Σ, X) sets equipped with the operations of substitution and renaming form an FT_Σ-algebra (it is the free FT_Σ-algebra generated by $\emptyset$). For $\mathcal{A}$ a FT_Σ-algebra, its associated *evaluation morphism* is the unique morphism from FiniteTrees(Σ) to $\mathcal{A}$.

A *congruence* $\sim$ over a FT_Σ-algebra $\mathcal{A}$ is a family $\sim$ of equivalence relations over the A_X's (each denoted $\sim$) such that, for any $a \sim b \in A_X$, $c \sim d \in A_Y$, $y \in Y$ and surjective $\sigma \colon X \to Y$: $c \cdot_y a \sim c \cdot_y b$; $c \cdot_y a \sim d \cdot_y a$ and $\widetilde{\sigma}(a) \sim \widetilde{\sigma}(b)$. From such a congruence, one can define the *quotient algebra* $\mathcal{A}/\sim$ in the natural way.

Compact presentation, and complexity. Our definition of algebras does not so far match the one used in the introduction. In the introduction, algebras were considered as using natural numbers as sorts, while here, our sorts are indexed by finite sets. What would be these algebras should be pretty clear. The presentation used here is simpler to present and to use. It is an exercise to show the equivalence.

What matters is that in what follows, a FT_Σ-algebra is of *bounded complexity* if there exists a bound K such that $|A_X| \leqslant K$ for all finite sets of variables X.

2.3 Languages and syntactic algebras

A *language of finite* Σ*-trees* L is a set of Σ-trees. It is *recognized by* an FT_Σ-algebra $\mathcal{A}$ if there is a set $P \subseteq A_\emptyset$ such that $L = \alpha^{-1}(P)$ in which α is the evaluation morphism of $\mathcal{A}$.

The *syntactic congruence* $\sim_L$ *of a language* L of finite Σ-trees is defined in the following way $s \sim_L t$ for s, t finite Σ-trees if, for all FT_Σ-contexts C, $[\![C]\!](s) \in L$ if and only if $[\![C]\!](t) \in L$. It is easy to prove that $\sim_L$ is indeed a congruence. The quotient algebra $\text{FiniteTrees}(\Sigma)/_{\sim_L}$ is called the *syntactic algebra* of L.

▶ **Example 2.** The language of all finite trees in which the symbol a appears has for syntactic FT_Σ-algebra the algebra with sorts $A_X = \{0, 1\}$, for all finite set of variables X, and whose evaluation morphism is such that $\alpha(t) = 1$ if and only if t is a tree in which a appears.

2.4 Tree automata

A *tree automaton* $\mathcal{B} = (Q, I, (\delta_a)_{a\in\Sigma})$ over Σ has a finite set Q of *states*, a set of *accepting states* $I \subseteq Q$ and a *transition relation* $\delta_a \subseteq Q \times Q^{\text{ar}(a)}$ for every symbol $a \in \Sigma$. A *run of* $\mathcal{B}$ over a finite tree t is a mapping $\rho\colon \text{dom}(t) \to Q$ such that, for any vertex $u \in \text{dom}(t)$ with $t(u) = a \in \Sigma$, $(\rho(u), (\rho(u0), ..., \rho(u(\text{ar}(a) - 1)))) \in \delta_a$. A run is *accepting* if $\rho(\varepsilon) \in I$. A language L of finite trees is called *regular* if it is recognized by a tree automaton $\mathcal{B}$, meaning the trees in L are exactly those for which there is an accepting run in $\mathcal{B}$.

Ex. 3 shows the translation from tree automata to tree algebra.

▶ **Example 3** (Automaton algebra)**.** Consider a regular language L of finite trees recognized by the tree automaton $\mathcal{B} = (Q, q_0, (\delta_a)_{a\in\Sigma})$.

Consider some finite set of variables X. An X*-run profile* is a tuple $\tau \in Q \times \mathcal{P}(Q)^X$. For a Σ, X-tree t, $\tau = (p, (U_x)_{x\in X})$ is a *run profile over* t if there exists a run ρ of the automaton over Q such that $\rho(\varepsilon) = p$ and for all variables $x \in X$, U_x is the set of states assumed by ρ at leaves labelled x. We define a tree algebra $\mathcal{A}$ that has as elements of sort X sets of X-run profiles. The definition of the operations is natural, and is such that the image of a Σ, X-tree t under the evaluation morphism yields the set of run profiles over t. It naturally recognizes the language L.

Note that this definition yields an algebra of doubly exponential complexity (and hence, this is an upper bound for regular languages). Of course, in practice, one can restrict the algebra to the reachable elements, and this may dramatically reduce the complexity.

The converse translation is also true, yielding the following classical result (it is for instance proved for preclones in [9]).

▶ **Proposition 4.** *A finite tree language is regular if and only if it is recognized by a finite* FT_Σ*-algebra.*

3 Fundamental results on permutations in tree algebras

This section studies some fundamental phenomena concerning the effect of variable permutations on tree algebras. Its main objective is to prove Theorem 5. It is a characterization of syntactic FT_Σ-algebras of bounded complexity which turns out to be key in the subsequent developments. Beyond that, several intermediate results in this section may also be relevant in the analysis of algebras of unbounded complexity.

In this section, given an FT_Σ-algebra $\mathcal{A}$, $\varphi_X^{\mathcal{A}}\colon \mathbf{Sym}(X) \to \mathbf{Sym}(A_X)$ (or simply φ_X when there is no ambiguity) denotes the group morphism $\sigma \mapsto \sigma^{\mathcal{A}}$, for all finite sets of variables X.

▶ **Theorem 5.** *A finite syntactic FT_Σ-algebra $\mathcal{A}$ is of bounded complexity if and only for all sufficiently large finite set of variables X, $\ker(\varphi_X^{\mathcal{A}}) = \boldsymbol{Sym}(X)$.*

The meaning of $\ker(\varphi_X) = \mathbf{Sym}(X)$ is that permuting the variables has no effect on A_X (i.e. $\sigma^{\mathcal{A}} = \mathrm{Id}_{A_X}$ for every $\sigma \in \mathbf{Sym}(X)$). We fix from now the FT_Σ-algebra $\mathcal{A}$.

Our first step is to define a relation $\equiv_{\mathcal{A}}$ which we show to be a congruence equivalent to the syntactic one (Proposition 6). We set for all $a \in A_X$:

$$\begin{aligned} \langle a \rangle : \quad (A_\emptyset)^X &\to A_\emptyset \\ b &\mapsto a[x_0 \leftarrow b(x_0), ..., x_{n-1} \leftarrow b(x_{n-1})] \ , \end{aligned}$$

in which $X = \{x_0, ..., x_{n-1}\}$. Define now $a \equiv_{\mathcal{A}} a'$, for $a, a' \in A_X$ to hold if $\langle a \rangle = \langle a' \rangle$ (note that it does not depend on the numbering of variables).

▶ **Proposition 6.** *$\equiv_{\mathcal{A}}$ is a congruence. If $\mathcal{A}$ is a syntactic FT_Σ-algebra, then $a = b$ if and only if $a \equiv_{\mathcal{A}} b$, for all a, b in $\mathcal{A}$.*

By a simple counting argument, we obtain the following corollary.

▶ **Corollary 7.** *Let $\mathcal{A}$ be a finite syntactic FT_Σ-algebra, then*

$$|A_X| \leq |A_\emptyset|^{|A_\emptyset|^{|X|}} \ .$$

Recall the following result from finite group theory:

▶ **Proposition 8.** *Let $\varphi \colon \boldsymbol{Sym}(X) \to G$ be a group morphism. If $|X| \geq 5$, then $\ker(\varphi)$ equals either $\boldsymbol{Sym}(X)$, $\boldsymbol{Alt}(X)$ or $\{\mathrm{Id}_X\}$.*

Using Propositions 6 and 8, we prove that, whenever X is large enough, $\ker(\varphi_X)$ may only be $\mathbf{Sym}(X)$ or $\{\mathrm{Id}_X\}$.

▶ **Proposition 9.** *Let $\mathcal{A}$ be a finite syntactic FT_Σ-algebra. There is an integer M such that, for all X of cardinal at least M, either $\ker(\varphi_X) = \boldsymbol{Sym}(X)$ or $\ker(\varphi_X) = \{\mathrm{Id}_X\}$.*

Proof. Let $M = \max(5, |A_\emptyset| + 1)$. Let X be a finite set of variables such that $|X| \geq M$. By Prop. 8, $\ker(\varphi_X)$ is either $\mathbf{Sym}(X)$, $\mathbf{Alt}(X)$ or $\{\mathrm{Id}_X\}$. Assume, for the sake of contradiction, that $\ker(\varphi_X) = \mathbf{Alt}(X)$. This implies that the image of φ_X has exactly 2 elements: permutations of signature $+1$ are sent to Id_{A_X}, and those of signature -1 are sent to another (distinct) element. Let us call τ this permutation of A_X.

Let $t \in \mathbf{Sym}(X)$ be a transposition, let us show that $t^{\mathcal{A}} = \mathrm{Id}_{A_X}$. According to Proposition 6, we only need to prove that $\langle t^{\mathcal{A}}(a) \rangle = \langle a \rangle$ for all $a \in A_X$. Let $b \in (A_\emptyset)^X$, since $n \geq |A_\emptyset| + 1$, there must exists $x \neq y$ in X such that $b(x) = b(y)$, thus: $\langle t^{\mathcal{A}}(a) \rangle(b) = \langle \tau(a) \rangle(b) = \langle (x\ y)^{\mathcal{A}}(a) \rangle(b) = \langle a \rangle(b \circ (x\ y)) = \langle a \rangle(b)$. Since this holds for all $a \in A_X$, $t^{\mathcal{A}} = \mathrm{Id}_{A_X}$. This is a contradiction. ◀

According to Proposition 9, for X large enough, $\ker(\varphi_X)$ may only be $\mathbf{Sym}(X)$ or $\{\mathrm{Id}_X\}$. The next result shows that one of these two cases in fact holds for all sufficiently large X.

▶ **Proposition 10.** *Let $\mathcal{A}$ be a finite syntactic FT_Σ-algebra. There is an integer M such that, either $\ker(\varphi_X) = \boldsymbol{Sym}(X)$ for all X with $|X| \geq M$, or $\ker(\varphi_X) = \{\mathrm{Id}_X\}$ for all X with $|X| \geq M$.*

By simple counting, if $\ker(\varphi_X) = \{\mathrm{Id}_X\}$, then $|\mathbf{Sym}(A_X)| \geq |\mathbf{Sym}(X)|$ and hence $|A_X| \geq |X|$. This yields the following corollary, which is one direction of Theorem 5.

▶ **Corollary 11.** *Let $\mathcal{A}$ be a syntactic FT_Σ-algebra of bounded complexity. There is an integer M such that, for every X with $|X| \geq M$,* $\ker(\varphi_X) = \boldsymbol{Sym}(X)$.

Heading toward the converse implication, we now show that when $\ker(\varphi_X) = \mathbf{Sym}(X)$, then $\langle a \rangle$ can only take very simple forms.

▶ **Lemma 12.** *Let $\mathcal{A}$ be a finite syntactic FT_Σ-algebra, and n be such that* $\ker(\varphi_X) = \boldsymbol{Sym}(X)$ *whenever $|X| \in \{n-1, n\}$. Then, for all $a \in A_X$ with $|X| = n$,*

$$\langle a \rangle(b) = \langle a \rangle(b')$$

for all $b, b' \in A_\emptyset^X$ such that $\mathrm{Im}(b) = \mathrm{Im}(b')$.

Proof. Note first that for $|X| \in \{n-1, n\}$ we have $\ker(\varphi_X) = \mathbf{Sym}(X)$. Hence for for all permutations $\sigma \in \mathbf{Sym}(X)$,

$$\langle a \rangle(b \circ \sigma) = \langle \sigma^{\mathcal{A}}(a) \rangle(b) = \langle a \rangle(b) \ . \tag{$\star$}$$

As a consequence, what matters is to prove that we can "duplicate" some $b(x)$'s. For $Y \subseteq X$ where $Y = \{y_0, ..., y_{k-1}\}$ as well as $h \in (A_\emptyset)^Y$ and $a \in A_X$, we simplify the notation with

$$a[h] = a[y_0 \leftarrow h(y_0), \ldots, y_{k-1} \leftarrow h(y_{k-1})] \ .$$

Let $x, y, z \in X$ be distinct variables and $b, b' \in (A_\emptyset)^X$ be such that b and b' coincide everywhere but for $b(z) = b(y)$ and $b'(z) = b(x)$, we claim that

$$\langle a \rangle(b) = \langle a \rangle(b') \ . \tag{$\star\star$}$$

Indeed, for σ that maps y to z and leaves all other variables unchanged, we have

$$\begin{aligned}
a[x \leftarrow d, y \leftarrow d', z \leftarrow d'][h] &= \sigma^{\mathcal{A}}(a)[x \leftarrow d, z \leftarrow d'][h] \\
&= \sigma^{\mathcal{A}}(a)[x \leftarrow d', z \leftarrow d][h] && \text{(by } (\star) \text{ applied to } X \setminus \{y\}) \\
&= a[x \leftarrow d', y \leftarrow d, z \leftarrow d][h] && \text{(by } (\star) \text{ applied to } X) \\
&= a[x \leftarrow d, y \leftarrow d', z \leftarrow d][h] \ .
\end{aligned}$$

in which $d, d' \in A_\emptyset$ and $h \in (A_\emptyset)^{X \setminus \{x,y,z\}}$, from which the claim $(\star\star)$ follows.

At this point, $(\star\star)$ allows to change the value $b(z)$ to another providing that its values before and after the change appear elsewhere in b, and $(\star)$ allows to permute all the $b(x)$'s. Hence, by iterative applications of them, we obtain of $(\star)$ and $(\star\star)$, $\langle a \rangle(b) = \langle a \rangle(b')$ providing that b and b' have same image. ◀

As a consequence of the above lemma, assuming that $\ker(\varphi_X) = \mathbf{Sym}(X)$ for all sufficiently large X, we can bound the number of possible elements in A_X. This yields Corollary 13 below, which is the second direction of Theorem 5.

▶ **Corollary 13.** *A finite syntactic FT_Σ-algebra such that* $\ker(\varphi_X) = \boldsymbol{Sym}(X)$ *for all sufficiently large set of variables X, has bounded complexity.*

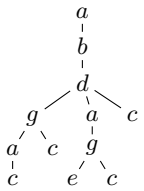
Proof. By Lemma 12, we know that, for $a \in A_X$, $\langle a \rangle$ must be chosen in a set of at most $|A_\emptyset|^{2^{|A_\emptyset|}}$ functions, this is an upper bound on the number of elements of A_X that does not depend on $|X|$ for X sufficiently large. ◀

4 Finite tree algebras of bounded complexity

The main theorem of this section, Theorem 14, provides a characterization of the languages recognized by FT_Σ-algebras of bounded complexity as Boolean combinations of simple languages. We proceed as follows. We state Theorem 14 and establish its easier parts in Section 4.1. In Section 4.2, we establish Lemma 22 which essentially amounts to the result for trees with "sufficiently many branches", which is the hardest part of the proof of Theorem 14.

4.1 Statement of the result

The main theorem of this section, Theorem 14, requires some preliminary definitions.



$\mathrm{upref}(t) = ab$ (unary prefix)

$\mathrm{fnu}(t) = d$ (first non-unary symbol)

$\mathrm{symb}(t) = \{a, b, c, d, e, g\}$ (symbols in t)

$\mathrm{pbsymb}(t) = \{a, c, e, g\}$ (post-branching symbols)

Figure 2 A finite tree t and its associated data.

For a given finite tree t, we associate to it some data (see Figure 2 for an illustration). Let n, be the depth of the first node labelled with a non unary symbol; formally, n is the least natural number such that $\mathrm{ar}(t(0^n)) \neq 1$. The *unary prefix* of t, denoted $\mathrm{upref}(t)$ is the word $t(0^1)...t(0^{n-1}) \in \Sigma_1^*$. The *first non-unary symbol of* t is $t(0^n)$, which we denote by $\mathrm{fnu}(t)$. The *set of symbols in* t is $\mathrm{symb}(t) = \{t(u) \mid u \in \mathrm{dom}(t)\}$ and its *set of post-branching symbols* is, if it exists, $\mathrm{pbsymb}(t) = \{t(0^n v) \mid 0^n v \in \mathrm{dom}(t), v \neq \varepsilon\}$.

▶ **Theorem 14.** *For a language of finite trees, the following properties are equivalent:*

1. *being recognized by a FT_Σ-algebra of bounded complexity,*
2. *having its syntactic FT_Σ-algebra of bounded complexity,*
3. *being equal to a Boolean combination of languages of the following kinds:*
   - **a.** *The language of finite trees with unary prefix in a given regular language of words $L \subseteq \Sigma_1^*$.*
   - **b.** *The language of finite trees with first non-unary symbol a for a fixed non-unary symbol a.*
   - **c.** *The language of finite trees with post-branching symbols B, for $B \subseteq \Sigma$.*
   - **d.** *A regular language K of* bounded branching*, meaning that there exists a natural number k such that all trees $t \in K$ have at most k branches.*

Let us establish the easiest parts of this statement. To start with, it is a generic fact– generic meaning independent of the type of algebras under consideration–that the syntactic FT_Σ-algebra of a language divides[1] any FT_Σ-algebra that recognizes the same language. Hence, each sort of the syntactic FT_Σ-algebra has a lesser size than the same sort in any other FT_Σ-algebra recognizing the same language. Thus 1 implies 2.

We now prove the second easiest implication, 3 implies 1. For this, it is sufficient to prove that the languages of kind 3a to 3d are recognized by FT_Σ-algebras of bounded complexity, and that FT_Σ-algebras of bounded complexity are closed under all the Boolean connectives. This is stated in Lemmas 15 to 17 below. All are straightforward.

[1] Divides in the morphism sense: being a quotient of a sub-algebra.

Given a regular language of words $L \subseteq \Sigma_1^*$, a non unary symbol a and a set $B \subseteq \Sigma$ of symbols, let us denote by $\mathrm{UPref}(L)$, $\mathrm{FNU}(a)$ and $\mathrm{PBSymb}(B)$, respectively the language of trees with unary prefix in L, the language of trees with first non-unary symbol c, and the language of trees t with $\mathrm{pbsymb}(t) = B$. Lemma 15 shows that these languages are recognized by algebras of bounded complexity, i.e. it treats Cases 3a to 3c.

▶ **Lemma 15.** *Given a regular language of words L, a non unary symbol $a \in \Sigma_{\neq 1}$ and a set of symbols $B \subseteq \Sigma$, the languages* $\mathrm{UPref}(L)$, $\mathrm{FNU}(a)$ *and* $\mathrm{PBSymb}(B)$ *are recognized by FT_Σ-algebras of bounded complexity.*

Proof. For space considerations, we only detail the case of $\mathrm{UPref}(L)$, which is arguably the hardest one. Let $\varphi\colon \Sigma_1^* \to M$ be a monoid morphism that recognizes L, meaning there exists $P \subseteq M$ such that $\varphi^{-1}(P) = L$. Define on $\mathrm{FiniteTrees}(\Sigma)$ the relation $\sim$ by $s \sim t$ if $\varphi(\mathrm{upref}(s)) = \varphi(\mathrm{upref}(t))$, this relation is easily seen to be a congruence, and $\mathrm{UPref}(L)$ is obviously recognized by the quotient algebra $\mathrm{FiniteTrees}(\Sigma)/\sim$. Because $\sim$ only has $|M|$ equivalence classes in every sort, we just described a FT_Σ-algebra recognizing $\mathrm{UPref}(L)$ of bounded complexity. ◀

Next we deal with the languages that have bounded branching, i.e. Case 3d. It is done with a modification of the automaton algebra of Example 3 so that it is also able to count the number of branches of a tree up to the bound k. The key observation is that a tree with k different variables must have at least k branches. This means that the sort A_X where X is a finite set of variables such that $|X| > k$ can be collapsed to one element only. Note that this is tightly related to our assumption that every variable from X must occur in all Σ, X-trees. We do not give any further detail here.

▶ **Lemma 16.** *The regular languages of finite trees of bounded branching are recognized by FT_Σ-algebras of bounded complexity.*

Finally, using standard constructions, we can provide the last ingredient of the proof that 3 implies 1 in Theorem 14:

▶ **Lemma 17.** *The languages recognized by FT_Σ-algebras of bounded complexity are closed under Boolean operations.*

4.2 Trees with many branches

In this section, we fill the missing gap in the proof of Theorem 14, namely the implication from 2 to 3.

Given two finite trees s and t and a FT_Σ-algebra $\mathcal{A}$ with evaluation morphism α, let $s \simeq_{\mathcal{A}} t$ hold if $\alpha(s) = \alpha(t)$. We omit the sub- and superscript $\mathcal{A}$ when it is clear from the context. Our goal is to prove Lemma 22 which states that if $\mathcal{A}$ is of bounded complexity, then for all trees s and t with sufficiently many branches, if $\mathrm{upref}(s) = \mathrm{upref}(t)$, $\mathrm{fnu}(s) = \mathrm{fnu}(t)$ and $\mathrm{pbsymb}(s) = \mathrm{pbsymb}(t)$, then $s \simeq_{\mathcal{A}} t$

Given an $\Sigma, X \uplus \{\gamma_1, \ldots, \gamma_n\}$-tree t in which the γ_i's only appear once, and given trees $t_1, \ldots, t_n$, we denote $t(t_1, \ldots, t_n)$ the tree $t[\gamma_1 \leftarrow t_1, \ldots, \gamma_n \leftarrow t_n]$. We will use these distinguished variables γ_i's in this section.

The results we prove throughout this section are consequences of the properties of permutations in FT_Σ-algebras of bounded complexity and, more particularly, consequences of Corollary 11. Our first objective is to show that if there are sufficiently many branches in a tree, it is possible to exchange any two subtrees which are not related by the ancestor

relationwithout changing evaluation in the algebra (see Lemma 20). Lemma 18 is a first step towards this goal: it says that, in a syntactic algebra, whenever a tree has many branches, it is possible to exchange *some* of its subtrees. More precisely:

▶ **Lemma 18.** *For every syntactic FT_Σ-algebras $\mathcal{A}$ of bounded complexity, there exists an integer N_0 such that, for all finite trees $t(t_1, t_2)$ such that t has at least N_0 branches,*

$$t(t_1, t_2) \simeq_{\mathcal{A}} t(t_2, t_1) \ .$$

Proof. Let N_0 be the integer introduced in Corollary 11. Assume that t is a $\Sigma, X \uplus \{\gamma_1, \gamma_2\}$-tree that has at least N_0 branches. Let s be a $\Sigma, X \uplus \{\gamma_1, \gamma_2\} \uplus \{x_1, ..., x_{N_0-2}\}$-tree and let s_0, ..., s_{N_0-2} be trees such that $s[x_1 \leftarrow s_1, ..., x_{N_0-2} \leftarrow s_{N_0-2}] = t$. As such, s has at least N_0 variables. Hence by Corollary 11, $t' \simeq_{\mathcal{A}} \sigma^{\mathcal{A}}(t')$ in which σ is the transposition of γ_1 and γ_2. We now compute:

$$\begin{aligned} t(t_1, t_2) &\simeq_{\mathcal{A}} s[\gamma_1 \leftarrow t_1, \gamma_2 \leftarrow t_2][x_1 \leftarrow s_1, ..., x_{N_0-2}] \\ &\simeq_{\mathcal{A}} \sigma^{\mathcal{A}}(s)[\gamma_1 \leftarrow t_1, \gamma_2 \leftarrow t_2][x_1 \leftarrow s_1, ..., x_{N_0-2}] \\ &= s[\gamma_1 \leftarrow t_2, \gamma_2 \leftarrow t_1][x_1 \leftarrow s_1, ..., x_{N_0-2}] = t(t_2, t_1) \ . \end{aligned}$$

◀

Here, notice the similarity between this proof and the observations we made in the proof of Lemma 12: this is the exact same argument, we just used the fact that a tree with many branches can always be obtained from some tree with many variables. Taking this similarity further, we may apply the other arguments we used in the proof of Lemma 12 to prove Lemma 19, and change the number of occurrences of plugged subtrees.

▶ **Lemma 19.** *For every syntactic FT_Σ-algebras $\mathcal{A}$ of bounded complexity, there exists an integer N_1 such that, for any finite tree $t(t_1, t_2, t_2)$ such that t has at least N_1 branches,*

$$t(t_1, t_2, t_2) \simeq_{\mathcal{A}} t(t_1, t_1, t_2) \ .$$

The next step is to establish Lemma 20, which is very similar to the previous Lemma 18 but for the fact that it is sufficient to have many branches in $t(t_1, t_2)$ instead of many branches in t to be allowed to swap t_1 and t_2. It is obtained by repeated and careful applications of Lemma 18.

▶ **Lemma 20.** *For all syntactic FT_Σ-algebras $\mathcal{A}$ of bounded complexity, there is an integer N_3 such that, for any finite tree $t(t_1, t_2)$ where $t(t_1, t_2)$ has at least N_3 branches,*

$$t(t_1, t_2) \simeq_{\mathcal{A}} t(t_2, t_1) \ .$$

As such, we may exchange two subtrees of a tree with many branches without changing evaluation in the algebra. We will use this result to prove that two trees with the same unary prefix, first non-unary symbol and set of post-branching symbols are not distinguished by the algebra (Lemma 22).

Using Lemmas 19 and 20, we first establish Lemma 21 that allows in some situation to make a symbol "appear" or "disappear".

▶ **Lemma 21.** *For all syntactic FT_Σ-algebras $\mathcal{A}$ of bounded complexity, there exists an integer N_4 that has the following property. For all finite trees $s(\gamma_1, \gamma_2)$, all finite trees t with at least N_4 branches, and all symbols $c, d \in \mathrm{symb}(t)$, c constant,*

$$s(t, c) \simeq_{\mathcal{A}} s(t, d(c, \dots, c)) \ .$$

In combination with Lemmas 19 and 20, it means that under the same assumptions, $s(t, c) \simeq_{\mathcal{A}} s(t, t')$ for all finite trees t' that use only symbols appearing in t. This building block, used iteratively, allows to shuffle and change the number of occurrences of all symbols that appear below a first non-unary symbol as soon as there are sufficiently many branches. This is our key Lemma 22.

▶ **Lemma 22.** *Let $\mathcal{A}$ be a syntactic FT_Σ-algebra of bounded complexity. There is an integer N, such that, for all finite trees s and t, both of which have at least N branches, if* $\mathrm{upref}(s) = \mathrm{upref}(t)$, $\mathrm{fnu}(s) = \mathrm{fnu}(t)$ *and* $\mathrm{pbsymb}(t) = \mathrm{pbsymb}(t)$, *then $t \simeq_{\mathcal{A}} s$.*

Putting everything together we establish the last implication in Theorem 14:

▶ **Lemma 23.** *A language of finite trees L recognized by a FT_Σ-algebra of bounded complexity can be written as a Boolean combination of languages of kinds 3a-3d in Theorem 14.*

The idea behind this last proof is as follows. Given a regular word language $K \subseteq \Sigma_1^*$, a non-unary symbol a, and a set of symbols B, let $L_{K,a,B}$ be the set of trees t such that $\mathrm{upref}(t) \in K$, $\mathrm{fnu}(t) = a$ and $\mathrm{pbsymb}(t) = B$. Such languages can be written as the intersection of $\mathrm{UPref}(K)$, $\mathrm{FNU}(a)$ and $\mathrm{PBSymb}(B)$ which are of the kinds 3a-3d in Theorem 14. Let N be the value from Lemma 22.

In a first step, we construct finitely many tuples (K_i, a_i, B_i), such that L and $\bigcup_i L_{K_i,a_i,B_i}$ coincide over all trees with sufficiently many branches. For this, for all $t \in L$ with at least N branches, consider the least language $K_t \subseteq \Sigma_1^*$ recognized by $A_{\{x\}}$ such that $\mathrm{upref}(t) \in K_t$ (in which $\mathrm{upref}(t)$ is recognized by $A_{\{x\}}$ when seen as a tree made of unary symbols and the variable x). The language K_t is regular and has the property that exchanging the unary prefix of t for any other word in K_t leaves the tree in L. We use as the (K_i, a_i, B_i)'s all the tuples $(K_t, \mathrm{fnu}(t), \mathrm{pbsymb}(t))$ for t ranging over the trees in L with at least N branches (note that since all the K_t's are recognized by $A_{\{x\}}$, there are finitely many of them). One can show using Lemma 22 that, as claimed, L and $\bigcup_i L_{K_i,a_i,B_i}$ coincide over all trees with at least N branches. In a second step one defines L_i to be the set of trees in L_{K_i,a_i,B_i} that have at least N branches. Let also L' be L restricted to trees with less than N branches. One gets

$$L = L' \cup \bigcup_i L_i \ ,$$

and this is by construction a Boolean combination of languages of the kinds 3a-3d.

This concludes our proof of Lemma 23, and hence Theorem 14.

5 Conclusion

In this paper, we initiated a complexity analysis of the expressiveness of infinitely sorted algebras. Our main result gives a descriptive characterization of the languages of finite trees recognized by algebras of bounded complexity, Theorem 14. In this work, we made a design choice in the definition of tree algebras. Indeed, we require that in a tree of sort X, every variable occurs at least once. Removing this assumption would change our bounded complexity characterization result, yielding only Boolean combinations of languages of the form "the root symbol is a". Another possible variant is to allow trees restricted to a single variable: in this case our results remain unchanged.

Extensions to infinite trees. We also obtained a similar characterization for algebras for infinite trees. We did not include it in this short abstract for space considerations (this was in fact our original question). In this case, the algebras have to include an extra iterating construct that allows to build all infinite regular trees (i.e. unfolding of finite graphs). By Rabin's lemma, regular languages of infinite trees are entirely characterized by the regular trees they contain, and as a consequence such algebras describe regular languages of infinite trees. Our result characterizes such algebras of bounded complexity along the same line as Theorem 14. We show that over infinite trees, such algebras can express two extra things (1) the existence of subtrees of unary shape that belong to a prescribed prefix-invariant regular language of infinite words, and (2) the existence of subtrees in which a set of letters C appears densely, i.e. every letter in C appears in every subtree.

Future work. The next simplest cases seem to be the algebras of polynomial complexity and of bounded orbit complexity. An example in this case is the language of Σ-trees such that the leftmost branch does not contain the symbol a (we assume the existence of other symbols of arity 0, and at least 2). It is recognized by an algebra which is both of polynomial complexity and bounded orbit complexity. So far, we do not even know whether these two classes differ.

References

1. Achim Blumensath. Recognisability for algebras of infinite trees. *Theor. Comput. Sci.*, 412(29):3463–3486, 2011. doi:10.1016/j.tcs.2011.02.037.
2. Achim Blumensath. Regular tree algebras. *Logical Methods in Computer Science*, 16, 2020.
3. Mikolaj Bojanczyk and Bartek Klin. A non-regular language of infinite trees that is recognizable by a sort-wise finite algebra. *Log. Methods Comput. Sci.*, 15(4), 2019. URL: https://lmcs.episciences.org/5927, doi:10.23638/LMCS-15(4:11)2019.
4. Mikolaj Bojanczyk and Michal Pilipczuk. Definability equals recognizability for graphs of bounded treewidth. In Martin Grohe, Eric Koskinen, and Natarajan Shankar, editors, *Proceedings of the 31st Annual ACM/IEEE Symposium on Logic in Computer Science, LICS'16, New York, NY, USA, July 5-8, 2016*, pages 407–416. ACM, 2016. doi:10.1145/2933575.2934508.
5. Mikolaj Bojanczyk and Luc Segoufin. Tree languages defined in first-order logic with one quantifier alternation. *Log. Methods Comput. Sci.*, 6(4), 2010. doi:10.2168/LMCS-6(4:1)2010.
6. Mikolaj Bojanczyk, Luc Segoufin, and Howard Straubing. Piecewise testable tree languages. *Log. Methods Comput. Sci.*, 8(3), 2012. doi:10.2168/LMCS-8(3:26)2012.
7. Mikolaj Bojanczyk and Igor Walukiewicz. Forest algebras. In Jörg Flum, Erich Grädel, and Thomas Wilke, editors, *Logic and Automata: History and Perspectives [in Honor of Wolfgang Thomas]*, volume 2 of *Texts in Logic and Games*, pages 107–132. Amsterdam University Press, 2008.
8. Bruno Courcelle and Joost Engelfriet. *Graph structure and monadic second-order logic: a language-theoretic approach*, volume 138. Cambridge University Press, 2012.
9. Zoltán Ésik and Pascal Weil. Algebraic recognizability of regular tree languages. *Theor. Comput. Sci.*, 340(1):291–321, 2005. doi:10.1016/j.tcs.2005.03.038.
10. Zoltán Ésik and Pascal Weil. Algebraic characterization of logically defined tree languages. *Int. J. Algebra Comput.*, 20(2):195–239, 2010. doi:10.1142/S0218196710005595.
11. Marcel-Paul Schützenberger. On finite monoids having only trivial subgroups. *Information and Control*, 8:190–194, 1965.

Improved Lower Bounds for Reachability in Vector Addition Systems

Wojciech Czerwiński
University of Warsaw, Poland

Sławomir Lasota
University of Warsaw, Poland

Łukasz Orlikowski
University of Warsaw, Poland

Abstract

We investigate computational complexity of the reachability problem for vector addition systems (or, equivalently, Petri nets), the central algorithmic problem in verification of concurrent systems. Concerning its complexity, after 40 years of stagnation, a non-elementary lower bound has been shown recently: the problem needs a tower of exponentials of time or space, where the height of tower is linear in the input size. We improve on this lower bound, by increasing the height of tower from linear to exponential. As a side-effect, we obtain better lower bounds for vector addition systems of fixed dimension.

2012 ACM Subject Classification Theory of computation → Parallel computing models

Keywords and phrases Petri nets, vector addition systems, reachability problem

Digital Object Identifier 10.4230/LIPIcs.ICALP.2021.128

Category Track B: Automata, Logic, Semantics, and Theory of Programming

Funding *Wojciech Czerwiński*: Supported by the ERC grant LIPA, agreement no. 683080.
Sławomir Lasota: Supported by the NCN grant 2017/27/B/ST6/02093.

1 Introduction

Petri nets [38] are a long established model of concurrency, with extensive applications in modelling and analysis of hardware [6, 25], software [16, 5, 20] and database [3, 4] systems, as well as chemical [1], biological [37, 2] and business [43, 31] processes (the references on applications are illustrative). The model admits various alternative but essentially equivalent presentations, most notably *vector addition systems* (VAS) [22], and *vector addition systems with states* (VASS) [17, Sec.5], [19]. The central algorithmic problem for this model is reachability: whether from the given initial configuration there exists a sequence of valid execution steps that reaches the given final configuration. Each of the alternative presentations admits its own formulation of the reachability problem, all of them being equivalent due to straightforward polynomial-time translations that preserve reachability; for further details, see e.g. Schmitz's survey [42, Section 2.1]. For instance, in terms of VAS, the problem is stated as follows: given a finite set T of integer vectors in d-dimensional space and two d-dimensional vectors $\mathbf{v}$ and $\mathbf{w}$ of nonnegative integers, does there exist a walk from $\mathbf{v}$ to $\mathbf{w}$ such that it stays within the nonnegative orthant, and its every step modifies the current position by adding some vector from T? Emphasizing VASS, a natural extension of VAS with finite control, in the sequel we use the name 'VASS reachability problem'.

Importance of the problem is widespread, as a plethora of problems from formal languages [8], logic [21, 11, 10, 7], concurrent systems [15, 13], process calculi [35], linear algebra [18] and other areas (the references are again illustrative) are known to admit reductions from the VASS reachability problem; for more such problems and a wider discussion, we refer to [42].

State of the art. The complexity of the VASS reachability problem was remaining unsettled over the past half century. The late 1970s and the early 1980s saw the initial burst of activity. After an incomplete proof by Sacerdote and Tenney [40], decidability of the problem was established by Mayr [33, 34], whose proof was then simplified by Kosaraju [23]. Building on the further refinements made by Lambert in the 1990s [24], there has been substantial progress over the past ten years [26, 27, 28], culminating in the first upper bound on the complexity [29], recently improved to Ackermannian [30].

In contrast to the progress on refining the proof of decidability and obtaining upper bounds on the complexity, Lipton's landmark result that the VASS reachability problem requires exponential space [32] has remained the state of the art on lower bounds for over 40 years. Moreover, in conjunction with an apparent tightness of Lipton's construction, this has led to the conjecture that the problem is ExpSpace-complete becoming common in the community. The conjecture has been falsified by a recent breakthrough construction of [9] that shows the VASS reachability problem is hard for the class Tower of all decision problems that are solvable in time or space bounded by a tower of exponentials whose height is an elementary (bounded by a tower of exponentials of fixed height) function of the input size.

Contribution. This paper provides a further improvement on the lower bound. The construction of [9] proves hardness of the VASS reachability problem for the class Tower, with respect to elementary reductions. However, in terms of the more fine-grained hierarchy defined with respect to polynomial-time reductions only, the result states that the problem needs a tower of exponentials of time or space, where the height of tower is linear in input size. As our primary result we increase the height of the tower from linear to exponential, namely we prove that the problem actually needs a tower of exponentials of time or space, where the height of the tower is itself exponential in input size:

$$\left.2^{2^{2^{\cdots^{n}}}}\right\} \text{ height } \mathcal{O}(n) \qquad \rightsquigarrow \qquad \left.2^{2^{2^{\cdots^{n}}}}\right\} \text{ height } 2^{\mathcal{O}(n)}.$$

In addition, as a side effect of our improved construction we obtain better lower bounds for VASS of fixed dimension. It has been known, as shown in [9], that the reachability problem for VASS in dimension d is $\mathcal{O}(d)$-ExpSpace-hard (specifically: d-ExpSpace-hardness for dimension $d + 13$). We show that the reachability problem for VASS in dimension d is $2^{\mathcal{O}(d)}$-ExpSpace-hard (specifically: 2^d-ExpSpace-hardness for dimension $2d + 13$).

Clearly, these new lower bounds do not formally exclude the VASS reachability problem from still being in Tower (and hence being Tower-complete with respect to elementary reductions). However, we believe that the result makes this less likely. Intuitively speaking, we rule out a natural algorithmic scheme of solving the reachability problem, where each additional control state or dimension leads to an additional exponential blowup of time or space. Presently, the most optimistic scheme must suffer, for each additional control state or dimension, from at least *doubling* of the number of exponentials.

Organisation of the paper. We start by defining the model, in Section 2, and then the reachability problem, in Section 3. In the latter section we also recall the lower bounds of [9] and formally formulate our improved ones. The following two sections are devoted to proofs. First, in Section 4 we show 2^d-ExpSpace-hardness of the reachability problem for VASS of dimension $2d + 13$, as a preparation before Section 5, in which we prove our primary result: without restriction on dimension, the problem needs a tower of exponentials of time or space, of height exponential in input size.

2 Counter programs

As mentioned in the introduction, Petri nets [38], VAS [22], and VASS [17, 19] are alternative presentations of the same model of concurrent processes, in the sense that between each pair there exist straightforward polynomial-time translations that preserve reachability [42, Sec. 2.1]. Following [12] and [9], instead of working directly with VASS, as our primary language we choose imperative nondeterministic programs operating on variables called counters, that range over nonnegative integers. These programs may be seen as user-friendly presentations of VASS.

Counter programs. A counter program is a sequence of commands, each of which is of one of the following three types:

$\mathsf{x} \mathrel{+}= 1$	(increment counter x)
$\mathsf{x} \mathrel{-}= 1$	(decrement counter x)
goto L **or** L'	(jump to either line L or line L')

except that the last command is of the form:

halt if $\mathsf{x}_1, \ldots, \mathsf{x}_l = 0$ (terminate provided all the listed counters are zero).

Counters are only allowed to have nonnegative values, they may be incremented or decremented but, notably, counters may not be zero-tested. As an illustration, consider the following program:

1: **goto** 7 **or** 2
2: $\mathsf{x} \mathrel{+}= 1$
3: $\mathsf{x}' \mathrel{-}= 1$
4: $\mathsf{y} \mathrel{+}= 1$
5: $\mathsf{y} \mathrel{+}= 1$
6: **goto** 1
7: **halt if** $\mathsf{x}' = 0$.

It repeats the block of three commands in lines 2–5 some number of times chosen nondeterministically (possibly zero, although not infinite because x' is decreasing and hence its initial value bounds the number of iterations) and then halts provided counter x' is zero.

We emphasise that counters are not permitted to have negative values. In the example above, that is why the decrement in line 3 works also as a non-zero test.

We assume that initially all counters are set to 0. A run of a counter program is a sequence of commands, as expected. We say that such a run is *halting* if, and only if it has successfully executed its **halt** command (which is necessarily the program's last); otherwise, the run is either *partial* or *infinite*. Observe that, due to a decrement that would cause a counter to become negative, or due to an unsuccessful terminal check for zero, a partial run may fail to continue because it is blocked from further execution. Moreover, due to nondeterministic jumps, the same program may have various runs in each of the three categories: halting runs, maximal partial runs, and infinite runs.

By the *size* of a program we mean the number of commands in it, and by its *dimension* we mean the number of counters. We sometimes speak also of *program fragments*, which are not ended with a **halt** command.

3 The reachability problem

Counter programs can be seen as presentations of VASS, where the latter are required to start with all vector components zero and to finish with vector components zero as specified by the **halt** command, and hence the VASS reachability problem can be stated as:

VASS REACHABILITY PROBLEM
Input: A counter program.
Question: Does it have a halting run?

We remark that restricting further to programs where no counter is required to be zero finally (i.e., where the last command is just **halt**) turns this problem into the VASS *coverability* problem, which is concerned with reachability of just a control location, with no requirement on the final values of counters. Lipton's EXPSPACE lower bound [32] holds already for the coverability problem, which is in fact EXPSPACE-complete [39].

For stating our results we recall the standard complexity classes bases on exponential hierarchy of fast-growing functions. We write $\mathcal{T}(n)$ for a tower of exponentials:

$$\mathcal{T}_2(k,n) \;=\; 2^{2^{2^{\cdots^{n}}}} \Big\} \text{ height } k \qquad\qquad \mathcal{T}(n) \;=\; \mathcal{T}_2(n,n) \;=\; 2^{2^{2^{\cdots^{n}}}} \Big\} \text{ height } n.$$

Formally, let $\mathcal{T}_2(0,n) = n$ and $\mathcal{T}_2(k+1,n) = 2^{\mathcal{T}_2(k,n)}$. For a fixed positive integer k, the class k-EXPSPACE contains all decision problems solvable in space $\mathcal{T}_2(k, \text{poly}(n))$. The class TOWER contains all decision problems solvable in time or space bounded by a tower of exponentials whose height is an elementary function of the input size, namely

$$\text{TOWER} \;=\; \bigcup_{f \text{ elementary}} \text{SPACE}(\mathcal{T}(f(n)))$$

with $f(\_)$ ranging over all *elementary* functions, i.e., functions bounded by $\mathcal{T}_2(k,\_)$ for some k. TOWER is thus closed under elementary reductions.

For the results of this paper we need a fine-grained hierarchy inside TOWER, with respect to dependency of the height of exponentials on the input size. For a fixed elementary function f, let $f(n)$-TOWER denote the class of all decision problems solvable in (time or) space $\mathcal{T}(f(\text{poly}(n)))$, i.e., in (time or) space bounded by a tower of exponentials of height $f(n^k)$, for some k, where n is the input size:

$$f(n)\text{-TOWER} \;=\; \bigcup_{k\in\mathbb{N}} \text{SPACE}(\mathcal{T}(f(n^k))).$$

Each class $f(n)$-TOWER is a strict subclass of TOWER as it is closed under polynomial-time reductions but, contrarily to TOWER, not under arbitrary elementary reductions.

We recall the results of [9]. First, once one fixes the dimension of VASS, the reachability problem requires a tower of exponentials of space, where the height of the tower is linear in the dimension:

▶ **Theorem 1** ([9], Cor. 5). *For any positive integer d, the* VASS *reachability problem in dimension $d+13$ is d-*EXPSPACE*-hard, with respect to polynomial-time*[1] *reductions.*

[1] All results mentioned in this section are actually shown using linear-time reductions.

The main result of [9] is a non-elementary lower bound of the VASS reachability problem: TOWER-hardness with respect to elementary reductions. The construction of [9] shows that the problem requires a tower of exponentials of space, where the height of tower is linear in the input size, and hence the result can be stated in terms of the fine-grained hierarchy as follows:

▶ **Theorem 2** ([9], Thm. 4). *The* VASS *reachability problem is hard for* n-TOWER*, with respect to polynomial-time reductions.*

As our first main result, we improve the lower bound of Theorem 1, namely we prove that solving the VASS reachability problem in dimension $2d + 13$ requires at least $\mathcal{T}_2(2^d, n)$ space:

▶ **Theorem 3.** *For any positive integer* d*, the* VASS *reachability problem in dimension* $2d + 13$ *is* 2^d*-*EXPSPACE*-hard.*

Clearly, without restriction of dimension, Theorem 3 does not exclude membership of the problem in TOWER, neither it excludes its membership in n-TOWER. Our primary result excludes the latter possibility: it states that the VASS reachability problem requires a tower of exponentials of space, where the height of tower is exponential in the input size, thus improving the bound of Theorem 2:

▶ **Theorem 4.** *The* VASS *reachability problem is hard for* 2^n*-*TOWER*, with respect to polynomial-time reductions.*

Again, Theorem 4 does not formally exclude membership of the VASS reachability problem in TOWER, it makes it however less imaginable as mentioned in the introduction.

Theorems 3–4 are proved in the following two sections. The proofs are a refinement and an optimisation of the construction of [9], involving certain amount of programming effort in the setting of counter programs.

4 Proof of Theorem 3

In this and in the next section we proceed by reductions from space-bounded variants of the halting problem for the standard model of (deterministic) Minsky machines [36]. The reader is referred to [14, Theorem 3.1] for translations between space-bounded 3-counter Minsky machines and Turing machines.

Following [9], for technical reasons we prefer to work with factorials instead of exponentials. We write $\mathcal{F}(k, \_)$ for the tower of factorials of height k: $\mathcal{F}(0, n) = n$ and $\mathcal{F}(k+1, n) = \mathcal{F}(k, n)!$. We note that all the complexity classes from the previous section are robust with respect to the choice of the fast-growing function hierarchy: exponential function can be equivalently replaced by factorial [41, Section 4.1].

Our proof is a refinement of the reduction of [9], from the following bounded version of the halting problem for Minsky machines with 3 counters, which is $(k-1)$-EXPSPACE-complete for any fixed positive integer k:

k-EXP-BOUNDED HALTING PROBLEM

Input: A 3-counter Minsky machine $\mathcal{M}$ of size n.

Question: Does it have a halting run where all counters are bounded by $\mathcal{F}(k, n)$?

Let h be a fixed positive integer. Given a 3-counter Minsky machine $\mathcal{M}$ of size n, the reduction of [9, Lem. 6] transforms $\mathcal{M}$, in time $\mathcal{O}(n+h)$, into a counter program $\mathcal{P}$ with $h+13$ counters, including $h+2$ counters $\mathsf{x}_{-1}, \mathsf{x}_0, \mathsf{x}_1, \ldots, \mathsf{x}_h$ which are required to be zero by the final **halt** command, plus 11 other counters, of the form[2]

$$\underbrace{\mathcal{H}_{-1}}_{\text{size } \mathcal{O}(n)} \quad \underbrace{\mathcal{H}_0 \quad \mathcal{H}_1 \quad \cdots \quad \mathcal{H}_h}_{\text{constant size each}} \quad \underbrace{\mathcal{H}_{h+1}}_{\text{size } \mathcal{O}(n)} \quad \textbf{halt if } \mathsf{x}_{-1}, \mathsf{x}_0, \ldots, \mathsf{x}_h = 0,$$

for some program fragments $\mathcal{H}_{-1}, \mathcal{H}_0, \ldots, \mathcal{H}_{h+1}$ (for technical convenience, the indexing starts at -1). We refer to the program fragments $\mathcal{H}_{-1}, \mathcal{H}_0, \ldots, \mathcal{H}_{h+1}$ as *segments*. Furthermore, the first segment $\mathcal{H}_{-1}$ and the last one $\mathcal{H}_{h+1}$ are of size $\mathcal{O}(n)$, the remaining ones are of constant size, and the reduction is correct due to:

▷ Claim 5 (Correctness). $\mathcal{M}$ has a halting run with all counters bounded by $\mathcal{F}(h+1, n)$ if, and only if, $\mathcal{P}$ has a halting run.[3]

Moreover, the counter program $\mathcal{P}$ has the following two crucial properties:

(i) Each counter x_j appearing in the final **halt** command appears in segments $\mathcal{H}_j$ and $\mathcal{H}_{j+1}$ only, for $j = -1, 0, \ldots, h$:

$$\underbrace{\mathcal{H}_{-1}}_{\mathsf{x}_{-1} \text{ only here}} \quad \overbrace{\underbrace{\mathcal{H}_0}_{\mathsf{x}_1 \text{ only here}} \quad \mathcal{H}_1}^{\mathsf{x}_0 \text{ only here}} \quad \overbrace{\mathcal{H}_2 \quad \mathcal{H}_3}^{\mathsf{x}_2 \text{ only here}} \quad \ldots \quad \underbrace{\mathcal{H}_{h-1}}_{\mathsf{x}_{h-1} \text{ only here}} \quad \overbrace{\mathcal{H}_h \quad \mathcal{H}_{h+1}}^{\mathsf{x}_h \text{ only here}}.$$

We say, to some extent informally, that $\mathcal{P}$ is a *relay-race* with respect to counters $\mathsf{x}_{-1}, \mathsf{x}_0, \ldots, \mathsf{x}_h$, by which we mean the last appearance of every counter x_j, for $j < h$, is in the same segment as the first appearance of the next counter x_{j+1}, and counters x_j and x_{j+2}, for $j < h-1$, never appear in the same segment.

(ii) Each segment $\mathcal{H}_j$, for $j = 0, 1, \ldots, h$, is obtained from the same[4] program fragment $\mathcal{H}$, not using counters $\mathsf{x}_{-1}, \mathsf{x}_0, \ldots, \mathsf{x}_h$, by replacing a counter x appearing in $\mathcal{H}$ by x_{j-1}, and another counter x' appearing in $\mathcal{H}$ by x_j. We denote the replacement as:

$$\mathcal{H}_j \quad = \quad \mathcal{H}[\mathsf{x} \mapsto \mathsf{x}_{j-1},\ \mathsf{x}' \mapsto \mathsf{x}_j]. \tag{1}$$

Let $h = 2^d$ for some positive integer d. We prove Theorem 3 by transforming $\mathcal{P}$ into an equivalent counter program $\widetilde{\mathcal{P}}$ of (slightly larger) size $\mathcal{O}(n + d \cdot 2^d)$, but of exponentially smaller dimension $2d+13$. We will rely on the relay-race property with respect to h counters $\mathsf{x}_0, \mathsf{x}_1, \ldots, \mathsf{x}_{h-1}$ (call these counters *relay-race counters*), and thus the transformation will not affect counters x_{-1} and x_h. The intuitive idea is to re-cycle counters $\mathsf{x}_0, \mathsf{x}_1, \ldots, \mathsf{x}_{h-1}$ appearing in segments $\mathcal{H}_0, \mathcal{H}_1, \ldots, \mathcal{H}_h$. Our transformation proceeds in $(d-1)$ steps, in each step reducing by half the number of relay-race counters and adding two additional fresh counters.

---

[2] Whenever convenient we compose counter program fragments horizontally, for the ease of presentation.

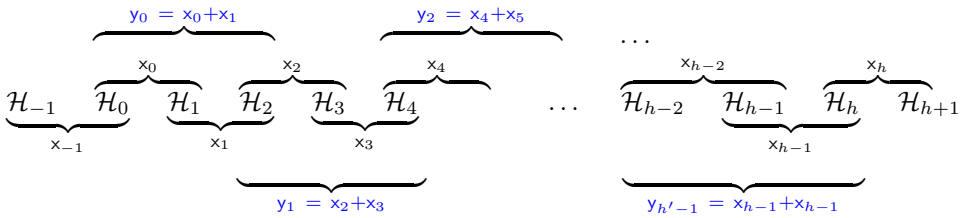
[3] Roughly speaking, $\mathcal{H}_{-1}$ simply computes n, then the sequence of segments $\mathcal{H}_0, \mathcal{H}_1, \ldots, \mathcal{H}_h$ is responsible for computation of $(h+1)$th iterate of factorial of n, which is then used in $\mathcal{H}_{h+1}$ for simulating $\mathcal{M}$.

[4] For the ease of presentation we slightly simplify the structure of $\mathcal{P}$; in fact, in order to rigorously express the construction of [9] one needs to use two different program fragments instead of just one $\mathcal{H}$, one of them for even j and the other one for odd j. As will be clear later, this simplification is inessential.

The first step. We split the first step of the transformation into two sub-steps. As the first sub-step, consider the following transformation of $\mathcal{P}$. Let $h' = \frac{1}{2}h = 2^{d-1}$. We introduce h' additional fresh counters $\mathsf{y}_0, \ldots, \mathsf{y}_{h'-1}$, initially set to 0, with the idea that counter y_k invariantly equals to the sum of counters $\mathsf{x}_{2k} + \mathsf{x}_{2k+1}$. This is achieved by adding, immediately after each command anywhere in fragments $\mathcal{H}_0, \mathcal{H}_1, \ldots, \mathcal{H}_{h-1}$ that operates on counter x_j, for $j = 0, \ldots, h-1$, an additional command operating in the same way on $\mathsf{y}_{j \text{ div } 2}$:

$$\mathsf{x}_j \mathrel{*}= 1 \qquad \mapsto \qquad \mathsf{x}_j \mathrel{*}= 1 \quad \mathsf{y}_{j \text{ div } 2} \mathrel{*}= 1,$$

where j div 2 is the largest nonnegative integer k such that $2k \leq j$, and $* \in \{+, -\}$. This addition yields a counter program $\mathcal{P}'$ (recall that both h and h' are even):



(2)

Furthermore, we remove counters $\mathsf{x}_0, \mathsf{x}_1, \ldots, \mathsf{x}_{h-1}$ from the **halt** command, and put counters $\mathsf{y}_0, \mathsf{y}_1, \ldots, \mathsf{y}_{h'-1}$ instead:

$$\textbf{halt if } \mathsf{x}_{-1}, \mathsf{x}_0, \ldots, \mathsf{x}_h = 0 \qquad \mapsto \qquad \textbf{halt if } \mathsf{x}_{-1}, \mathsf{x}_h, \mathsf{y}_0, \mathsf{y}_1, \ldots, \mathsf{y}_{h'-1} = 0.$$

▷ Claim 6. $\mathcal{P}'$ satisfies invariantly $\mathsf{y}_k = \mathsf{x}_{2k} + \mathsf{x}_{2k+1}$, for $k = 0, 1, \ldots, h'-1$.

We use that claim for entailing:

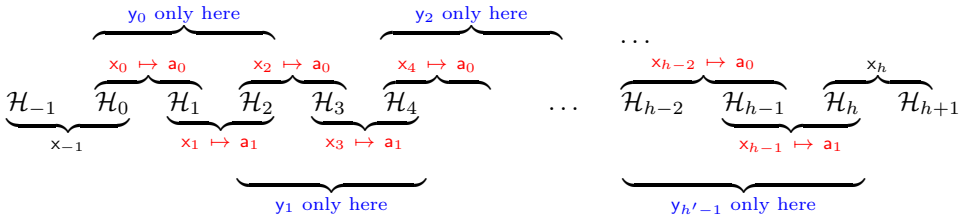
▷ Claim 7 (Correctness). $\mathcal{P}$ has a halting run if, and only if, $\mathcal{P}'$ has one.

Proof. We show that halting runs of $\mathcal{P}$ are in one-to-one correspondence with halting runs of $\mathcal{P}'$, using Claim 6 in both directions. In one direction, every halting run of $\mathcal{P}$, ending with all counters $\mathsf{x}_{-1}, \mathsf{x}_0, \ldots, \mathsf{x}_h$ equal 0, has a corresponding halting run of $\mathcal{P}'$ ending with $\mathsf{x}_{-1}, \mathsf{x}_h$ and all added counters $\mathsf{y}_0, \mathsf{y}_1, \ldots, \mathsf{y}_{h'-1}$ equal 0 as well. For the opposite direction consider any halting run of $\mathcal{P}'$. As every counter y_k is 0 at the end of the run, the sum of every two consecutive counters $\mathsf{x}_{2k} + \mathsf{x}_{2k+1}$ is 0 at the end, and hence also each individual counter x_j is *forced* to be 0 as well (without even checking that it is so, in the final **halt** command). Therefore, dropping operations on the added counters yields a halting run of $\mathcal{P}$. ◁

As the second sub-step, we introduce two fresh counters a_0, a_1, and replace each operation on every counter x_j, for $j = 0, 1, \ldots, h-1$, anywhere in segments $\mathcal{H}_0, \mathcal{H}_1, \ldots, \mathcal{H}_{h-1}$, by the analogous operation on $\mathsf{a}_{j \text{ mod } 2}$.

$$\mathsf{x}_j \mathrel{*}= 1 \qquad \mapsto \qquad \mathsf{a}_{j \text{ mod } 2} \mathrel{*}= 1. \tag{3}$$

The replacement removes h counters $\mathsf{x}_0, \mathsf{x}_1, \ldots, \mathsf{x}_{h-1}$ definitely from $\mathcal{P}'$, yielding a counter program $\mathcal{P}''$:



(4)

with the same **halt** command as $\mathcal{P}'$. Note that the size of $\mathcal{P}''$ is larger by $\mathcal{O}(2^d)$ than the size of $\mathcal{P}$, but its dimension decreased from $2^d + 13$ to $2^{d-1} + 15$, as $h = 2^d$ counters have been removed and another $h' + 2 = 2^{d-1} + 2$ counters have been introduced instead.

We have thus completed the description of a transformation $\mathcal{P} \mapsto \mathcal{P}''$. Formally, by composing the two sub-steps, we see that $\mathcal{P}''$ has the form:

$$\underbrace{\mathcal{H}_{-1}}_{\text{size } \mathcal{O}(n)} \quad \underbrace{\mathcal{H}''_0 \quad \mathcal{H}''_1 \quad \cdots \quad \mathcal{H}''_h}_{\text{constant size each}} \quad \underbrace{\mathcal{H}_{h+1}}_{\text{size } \mathcal{O}(n)} \quad \textbf{halt if } \mathsf{x}_{-1}, \mathsf{x}_h, \mathsf{y}_0, \mathsf{y}_1, \ldots, \mathsf{y}_{h'-1} = 0,$$

where each segment $\mathcal{H}''_i$ is obtained from $\mathcal{H}_i$ by the simultaneous substitution, for all $j = 0, 1, \ldots, h-1$:

$$\mathsf{x}_j \mathrel{*}= 1 \qquad \mapsto \qquad \mathsf{a}_{j \bmod 2} \mathrel{*}= 1 \quad \mathsf{y}_{j \operatorname{div} 2} \mathrel{*}= 1. \tag{5}$$

The following fact is crucial for correctness, i.e., for proving that the second sub-step, and hence also the whole step, preserves reachability:

▷ Claim 8. In every halting run of $\mathcal{P}''$, $\mathsf{a}_{j \bmod 2} = 0$ at the end of $\mathcal{H}''_{j+1}$ for every $j \in \{0, \ldots, h-1\}$.

Proof. Consider any halting run of $\mathcal{P}''$. By induction on j we prove that for every *even* $j \in \{0, 1, \ldots, h-1\}$, $\mathsf{a}_0 = 0$ at the end of $\mathcal{H}''_{j+1}$, and $\mathsf{a}_1 = 0$ at the end of $\mathcal{H}''_{j+2}$.

Let $j \geq 0$ and asssume the claim holds for smaller values of j. Let $v_0 \in \mathbb{N}$ (resp. $v_1 \in \mathbb{N}$) denote the value of counter a_0 (resp. counter a_1), at the beginning of fragment $\mathcal{H}''_j$ (resp. $\mathcal{H}''_{j+1}$). By induction assumption (or due to initial 0 values of counters) we have $v_0 = v_1 = 0$. According to the substitution (5), every operation on a_0 in $\mathcal{H}''_j$ and $\mathcal{H}''_{j+1}$ (recall that these are the only fragments where x_j appears) is also performed on $\mathsf{y}_{j \operatorname{div} 2}$. Likewise, every operation on a_1 in $\mathcal{H}''_{j+1}$ and $\mathcal{H}''_{j+2}$ (these are the only fragments where x_{j+1} appears) is also performed on $\mathsf{y}_{j \operatorname{div} 2}$. Also according to (5), these are the only operations on $\mathsf{y}_{j \operatorname{div} 2}$ performed along the run. Therefore, denoting by v'_0 (resp. v'_1) the value of counter a_0 (resp. counter a_1), at the end of fragment $\mathcal{H}''_{j+1}$ (resp. $\mathcal{H}''_{j+2}$), we know that the value of counter $\mathsf{y}_{j \operatorname{div} 2}$ at the end of the run is $(v'_0 - v_0) + (v'_1 - v_1) = v'_0 + v'_1$. As the counter $\mathsf{y}_{j \operatorname{div} 2}$ is checked to be 0 by the final **halt** command, we deduce $v'_0 + v'_1 = 0$. Finally, since both values are necessarily nonnegative, we deduce $v'_0 = v'_1 = 0$, as required. ◁

▷ Claim 9 (Correctness). $\mathcal{P}'$ has a halting run if, and only if, $\mathcal{P}''$ has one.

Proof. We show that halting runs of $\mathcal{P}'$ are in one-to-one correspondence with halting runs of $\mathcal{P}''$. As $\mathcal{P}''$ is obtained from $\mathcal{P}'$ by merging all counters x_j, for even j, to one counter a_0, and all counters x_j, for odd j, to one counter a_1, every halting run of $\mathcal{P}'$ has a corresponding halting run of $\mathcal{P}''$ such that a_0 is the sum of values of all counters x_j for even j and a_1 is the sum of values of all counters x_j for odd j. For the opposite direction we rely on Claim 8. According to the claim, replacing in a halting run of $\mathcal{P}''$ each operation on $\mathsf{a}_{j \bmod 2}$ in $\mathcal{H}''_j$ and $\mathcal{H}''_{j+1}$ by the same operation on x_j is safe, namely it is guaranteed that counters x_j stay nonnegative. Therefore we get a halting run of $\mathcal{P}''$. ◁

Iterating steps. A crucial but simple observation, cf. (4), is that the so described step of the transformation preserves the relay-race property, and hence can be iterated:

▷ Claim 10 (Relay-race preservation). The counter program $\mathcal{P}''$ is a relay-race with respect to counters $\mathsf{y}_0, \mathsf{y}_1, \ldots, \mathsf{y}_{h'-1}$.

Proof. According to the substitution (5), each counter y_k only appears in three consecutive segments $\mathcal{H}''_{2k}, \mathcal{H}''_{2k+1}$ and $\mathcal{H}''_{2k+2}$. Therefore the relay-race condition is satisfied: the last appearance of every counter y_k, for $k+1 < h'$, is in the same segment $\mathcal{H}''_{2k+2}$ as the first appearance of the next counter y_{k+1}, and counters y_k and y_{k+2}, for $k+2 < h'$, never appear in the same segment. ◁

We apply the transformation step $d-1$ times. Initially, $\mathcal{P}$ has 2^d relay-race counters, and 13 other counters including x_{-1} and x_h. Each ith iteration, for $i = 0, 1, \ldots, d-2$, decreases the number of relay-race counters twice, from 2^{d-i} to 2^{d-i-1}, and adds two additional counters a^i_0 and a^i_1. Therefore, after $d-1$ iterations[5] we arrive at a counter program $\widetilde{\mathcal{P}}$ with just two relay-race counters, say $\mathsf{z}_0, \mathsf{z}_1$, plus $2(d-1)$ added counters, say a^i_0, a^i_1 for $i = 0, 1, \ldots, d-2$, and 13 remaining counters, of the form:

$$\underbrace{\mathcal{H}_{-1}}_{\text{size } \mathcal{O}(n)} \quad \underbrace{\widetilde{\mathcal{H}}_0 \quad \widetilde{\mathcal{H}}_1 \quad \cdots \quad \widetilde{\mathcal{H}}_h}_{\text{size } \mathcal{O}(d) \text{ each}} \quad \underbrace{\mathcal{H}_{h+1}}_{\text{size } \mathcal{O}(n)} \qquad \textbf{halt if } \mathsf{x}_{-1}, \mathsf{x}_h, \mathsf{z}_0, \mathsf{z}_1 = 0. \tag{6}$$

By iterating the substitution (5), one sees that $\widetilde{\mathcal{H}}_i$ is actually obtained from $\mathcal{H}_i$ by the following simultaneous substitution, for $* \in \{+, -\}$ and $j = 0, 1, \ldots, h-1$

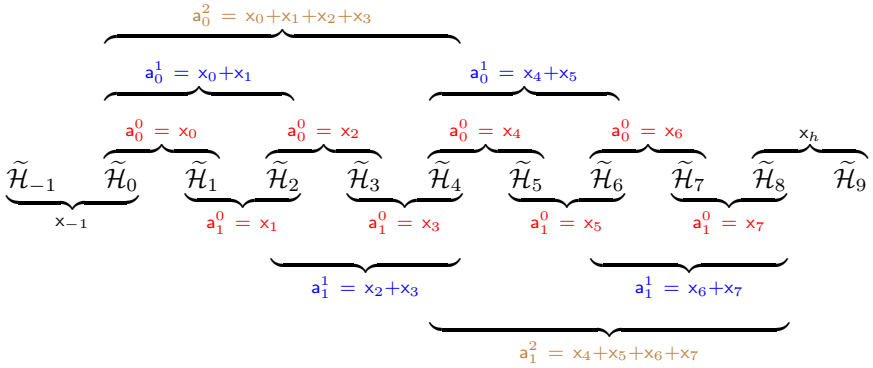
$$\mathsf{x}_j \mathrel{*}= 1 \qquad \mapsto \qquad \mathsf{a}^0_{j_0} \mathrel{*}= 1 \quad \mathsf{a}^1_{j_1} \mathrel{*}= 1 \quad \ldots \quad \mathsf{a}^{d-2}_{j_{d-2}} \mathrel{*}= 1 \quad \mathsf{z}_{j_{d-1}} \mathrel{*}= 1, \tag{7}$$

where $j_{d-1} \ldots j_1 j_0 = \text{BIN}(j)$ is the binary representation of j using d bits, in the order of decreasing significance of bits. Indeed, observe that $j_0 = j \bmod 2$, $j_1 = (j \operatorname{div} 2) \bmod 2$, $j_2 = ((j \operatorname{div} 2) \operatorname{div} 2) \bmod 2$, and so on. Therefore, by iterating the substitution (5), we first replace x_j by $\mathsf{a}^0_{j_0}$ and a relay-race counter $\mathsf{y}_{j \operatorname{div} 2}$, then replace $\mathsf{y}_{j \operatorname{div} 2}$ by $\mathsf{a}^1_{j_1}$ and some further relay-race counter $\mathsf{z}_{(j \operatorname{div} 2) \operatorname{div} 2}$, and so on. In the sequel we uniformly write a^{d-1}_b instead of z_b appearing in (6) and (7), for $b = 0, 1$. Let $\overline{b} = 1 - b$ denote bit negation.

▷ Claim 11. In every halting run of $\widetilde{\mathcal{P}}$, $\mathsf{a}^i_b = 0$ at the end of $\widetilde{\mathcal{H}}_j$ for every $j \in \{1, \ldots, h\}$ such that $j_i = \overline{b}$ and $j_{i-1} = \ldots = j_0 = 0$.

Proof. Follows by iterating the substitution (5) and Claim 8. ◁

▶ **Example 12.** As an illustration consider $d = 3$, and hence $h = 8$. The construction yields a counter program $\widetilde{\mathcal{P}}$ with 6 added counters $\mathsf{a}^0_0, \mathsf{a}^0_1, \mathsf{a}^1_0, \mathsf{a}^1_1, \mathsf{a}^2_0, \mathsf{a}^2_1$, and hence of dimension 19. Halting runs of $\mathcal{P}$ and $\widetilde{\mathcal{P}}$ are in one-to-one correspondence, and the values of counters $\mathsf{x}_0, \mathsf{x}_1, \ldots, \mathsf{x}_7$ in a halting run of $\mathcal{P}$ are related, via the equalities depicted below, to the values of counters $\mathsf{a}^0_0, \mathsf{a}^0_1, \mathsf{a}^1_0, \mathsf{a}^1_1, \mathsf{a}^2_0, \mathsf{a}^2_1$ in the corresponding halting run of $\widetilde{\mathcal{P}}$.



[5] One additional dth iteration could be also performed, which would however result in replacing 2 counters by 3 other ones, and hence the dimension would increase.

Indeed, iterating the proof of Claims 7 and 9 three times, we learn that a_b^i is the sum of values of all counters x_j where $j_i = b$, for instance $\mathsf{a}_0^1 = \mathsf{x}_0 + \mathsf{x}_1 + \mathsf{x}_4 + \mathsf{x}_5$. However, according to Claim 11, we have $\mathsf{a}_0^1 = 0$ at the end of $\widetilde{\mathcal{H}}_2$, and hence $\mathsf{a}_0^1 = \mathsf{x}_4 + \mathsf{x}_5$ in $\widetilde{\mathcal{H}}_4 \dots \widetilde{\mathcal{H}}_6$.

In particular, we obtain (cf. Claim 14 below) 8-ExpSpace-hardness for VASS in dimension 19 which, according to Theorem 1, has been known before to be 6-ExpSpace-hard. ⌟

▷ Claim 13 (Correctness). $\mathcal{P}$ has a halting run if, and only if, $\widetilde{\mathcal{P}}$ has one.

Proof. Iterating Claims 7 and 9 we deduce that halting runs of $\mathcal{P}$ are in one-to-one correspondence with halting runs of $\widetilde{\mathcal{P}}$. In one direction, every halting run of $\mathcal{P}$ has a corresponding halting run of $\widetilde{\mathcal{P}}$ where each counter a_b^i, for $i = 0, 1, \dots, d-2$ and $b = 0, 1$, invariantly equals to the sum of all counters x_j with $j_i = b$: $\mathsf{a}_b^i = \sum_{j \,:\, j_i = b} \mathsf{x}_j$. For the opposite direction we deduce, using Claim 11, that dropping operations on the added counters in every halting run of $\widetilde{\mathcal{P}}$, and replacing each operation on $\mathsf{a}_{j \bmod 2}$ in $\widetilde{\mathcal{H}}_j$ and $\widetilde{\mathcal{H}}_{j+1}$ by the same operation on x_j, yields a halting run of $\mathcal{P}$. ◁

The dimension of $\widetilde{\mathcal{P}}$ is $2d + 13$. The size of every segment $\widetilde{\mathcal{H}}_j$, for $j = 0, 1, \dots, h$, is $\mathcal{O}(d)$ times the constant size of $\mathcal{H}_j$, therefore the size of $\widetilde{\mathcal{P}}$ is $\mathcal{O}(n + d \cdot 2^d)$.

Due to Claims 5 and 13, the reduction $\mathcal{M} \mapsto \mathcal{P}$ of [9], composed with the transformation $\mathcal{P} \mapsto \widetilde{\mathcal{P}}$ just described, yield the required reduction $\mathcal{M} \mapsto \widetilde{\mathcal{P}}$:

▷ Claim 14. Let d be a positive integer. Given a 3-counter Minsky machine $\mathcal{M}$ of size n, one can compute in time $\mathcal{O}(n + d \cdot 2^d)$ a counter program $\widetilde{\mathcal{P}}$ of dimension $2d + 13$ such that $\mathcal{M}$ has a halting run with counters bounded by $\mathcal{F}(2^d + 1, n)$ if, and only if, $\widetilde{\mathcal{P}}$ has a halting run.

For every fixed d the reduction is computable in linear time with respect to the input size n, and hence the VASS reachability problem in dimension $2d + 13$ is 2^d-ExpSpace-hard.

5 Proof of Theorem 4

In order to prove Theorem 4 we refine further our reduction from Section 4. The refinement involves certain amount of low-level programming with counter programs.

Formally, we will set up a linear-time reduction from the following problem:

EXP-TOWER HALTING PROBLEM
Input: A 3-counter Minsky machine $\mathcal{M}$ of size n.
Question: Does it have a halting run where all counters are bounded by $\mathcal{F}(2^n, n)$?

We argue, similarly as before, that the problem is complete for the class 2^n-Tower, with respect to polynomial-time reductions, cf. [14, Theorem 3.1] and [41, Section 4.1].

In this section we strongly rely on condition (ii) from Section 4: each segment $\mathcal{H}_j$ in (6), for $j = 0, 1, \dots, h$, is obtained by the substitution (1) applied to the same program fragment $\mathcal{H}$ of constant size. Combining this substitution with the substitution (7) we deduce that each $\widetilde{\mathcal{H}}_j$, for $j = 0, 1, \dots, h$, is obtained from $\mathcal{H}$ by applying the following two substitutions:

$$\begin{array}{lcl} \mathsf{x} \mathrel{*}= 1 & \mapsto & \mathcal{C}^*_{j-1} \\ \mathsf{x}' \mathrel{*}= 1 & \mapsto & \mathcal{C}^*_{j} \end{array} \tag{8}$$

where program fragments $\mathcal{C}^*_j$ are defined, for $* \in \{+, -\}$ and $j = 0, 1, \dots, h-1$, as the sequence of commands:

$$\mathcal{C}^*_j \quad = \quad \mathsf{a}^0_{j_0} \mathrel{*}= 1 \quad \mathsf{a}^1_{j_1} \mathrel{*}= 1 \quad \dots \quad \mathsf{a}^{d-2}_{j_{d-2}} \mathrel{*}= 1 \quad \mathsf{a}^{d-1}_{j_{d-1}} \mathrel{*}= 1.$$

We apply here the proviso that the former substitution, referring to x, is not applied when $j = 0$, and the latter one, referring to x', is not applied when $j = h$. Indeed, recalling (1) and (4), x in $\mathcal{H}_0$ refers actually to x_0, and likewise x' in $\mathcal{H}_j$ refers actually to x_{h+1}.

Relying on the substitutions (8) we are going to optimise the counter program $\widetilde{\mathcal{P}}$, as defined in (6), by replacing all segments $\widetilde{\mathcal{H}}_1, \ldots, \widetilde{\mathcal{H}}_{h-1}$ by a *single* program fragment $\overline{\mathcal{G}}$ of size $\mathcal{O}(d)$, and thus obtaining the final counter program $\overline{\mathcal{P}}$ (see (11) below).

We start by defining a single program fragment $\mathcal{C}^*$ of size $\mathcal{O}(d)$ that achieves the effect of any $\mathcal{C}^*_j$ in a way parametric with respect to j. The program fragment $\mathcal{C}^*$ uses a bit-wise representation of the value of j, ranging over $0, \ldots, h-1$. To this aim we introduce d new counters $\mathsf{b}_0, \mathsf{b}_1, \ldots, \mathsf{b}_{d-1}$ to represent the binary representation $\textsc{bin}(j) = j_{d-1} \ldots j_0$ of current value j; thus these counters will only take values 0 or 1. In addition, we introduce another d counters $\overline{\mathsf{b}}_0, \overline{\mathsf{b}}_1, \ldots, \overline{\mathsf{b}}_{d-1}$ to represent negations of bits $\mathsf{b}_0, \mathsf{b}_1, \ldots, \mathsf{b}_{d-1}$. We are going to enforce the following equalities to hold invariantly:

$$\mathsf{b}_i + \overline{\mathsf{b}}_i = 1 \qquad (\text{for } i = 0, \ldots, d-1). \tag{9}$$

In particular, as expected, counters b_i are initialised to 0, while counters $\overline{\mathsf{b}}_i$ are incremented to 1 at the start of $\overline{\mathcal{P}}$. Having (9), one easily implements a conditional construct

$$\textbf{if } \mathsf{b}_i = 1 \textbf{ then } \mathcal{G} \textbf{ else } \mathcal{G}' \tag{10}$$

that executes one of program fragments $\mathcal{G}$, $\mathcal{G}'$ depending on the value of the ith bit b_i:

1: **goto** 2 **or** 5
2: b_i $-{=}$ 1 $\quad$ b_i $+{=}$ 1
3: $\mathcal{G}$
4: **goto** 7
5: $\overline{\mathsf{b}}_i$ $-{=}$ 1 $\quad$ $\overline{\mathsf{b}}_i$ $+{=}$ 1
6: $\mathcal{G}'$
7: ...

Note that exactly one of the two branches fails (because of $\mathsf{b}_i = 0$ or $\overline{\mathsf{b}}_i = 0$) and the other one succeeds. Line 2 is a non-zero test on b_i, while line 5 is, due to the equality (9), a *zero-test* on b_i. The original values of counters $\mathsf{b}_i, \overline{\mathsf{b}}_i$ are retrieved by increments in lines 2 and 5. The conditional construct allows us to write the code for $\mathcal{C}^*$:

if $\mathsf{b}_0 = 1$ **then** $\mathsf{a}_1^0 *{=} 1$ **else** $\mathsf{a}_0^0 *{=} 1$
if $\mathsf{b}_1 = 1$ **then** $\mathsf{a}_1^1 *{=} 1$ **else** $\mathsf{a}_0^1 *{=} 1$
...
if $\mathsf{b}_{d-1} = 1$ **then** $\mathsf{a}_1^{d-1} *{=} 1$ **else** $\mathsf{a}_0^{d-1} *{=} 1$

Also, using the above conditional construct (10) as a building block, one writes down two further program fragments $\mathcal{I}nc$ and $\mathcal{D}ec$, both of size $\mathcal{O}(d)$. Fragment $\mathcal{I}nc$ (resp. $\mathcal{D}ec$) increments (resp. decrements) the current value j represented by counters $\mathsf{b}_0, \ldots, \mathsf{b}_{d-1}$, assuming that this value is below $h-1$ (resp. above 0), by means of bit operations. In order to keep the invariant (9), every increment b_i $+{=}$ 1 of ith bit is accompanied by the corresponding decrement $\overline{\mathsf{b}}_i$ $-{=}$ 1, and symmetrically every decrement b_i $-{=}$ 1 of ith bit is accompanied by the increment $\overline{\mathsf{b}}_i$ $+{=}$ 1.

Let $\overline{\mathcal{H}}$ denote the result of applying the following substitutions to $\mathcal{H}$, for $* \in \{+, -\}$:

$$\mathsf{x} *{=} 1 \quad \mapsto \quad \mathcal{D}ec \;\; \mathcal{C}^* \;\; \mathcal{I}nc$$
$$\mathsf{x}' *{=} 1 \quad \mapsto \quad \mathcal{C}^*$$

This exactly implements substitutions (8): assuming the current value of j is represented in counters b_i and $\overline{\mathsf{b}}_i$, as described above, execution of the program fragment $\mathcal{C}^*$ has the same effect as execution of $\mathcal{C}_j^*$; and execution of $\mathcal{D}ec\ \mathcal{C}^*\ \mathcal{I}nc$ has the same effect as decrementing j, executing $\mathcal{C}_j^*$, and incrementing j to retrieve its actual current value, which is equivalent to execution of $\mathcal{C}_{j-1}^*$.

The size of $\overline{\mathcal{H}}$ is $\mathcal{O}(d)$. It remains to wrap up this program fragment inside a loop that increments the value j, represented by counters $\mathsf{b}_0, \ldots, \mathsf{b}_{d-1}$, from 0 to $h-1$. Using an aggregated conditional construct **if** $\mathsf{b}_0 = \mathsf{b}_1 = \ldots = \mathsf{b}_{d-1} = 1$ **then goto** L of size $\mathcal{O}(d)$, defined as expected, we write down the following program fragment $\overline{\mathcal{G}}$, again of size $\mathcal{O}(d)$:[6]

1: $\mathcal{I}nc$
2: **if** $\mathsf{b}_0 = \mathsf{b}_1 = \ldots = \mathsf{b}_{d-1} = 1$ **then goto** 6
3: $\overline{\mathcal{H}}$
4: $\mathcal{I}nc$
5: **goto** 2
6: ...

Assuming the initial values $\mathsf{b}_i = 0$ and $\overline{\mathsf{b}}_i = 1$ for all $j = 0, \ldots, d-1$, $\overline{\mathcal{G}}$ sets values $\mathsf{b}_i = 1$ and $\overline{\mathsf{b}}_i = 0$ for all $j = 0, \ldots, d-1$, once line 6 is reached. We use $\overline{\mathcal{G}}$ as a part of the final counter program $\overline{\mathcal{P}}$. As the substitutions (8) apply fully only when $j = 1, \ldots, h-1$, we keep the first and the last segments $\widetilde{\mathcal{H}}_0$ and $\widetilde{\mathcal{H}}_h$ as defined in (6)–(7), and replace all others by $\overline{\mathcal{G}}$. The counter program $\overline{\mathcal{P}}$ has the following form:

$$\underbrace{\mathcal{H}_{-1}}_{\text{size } \mathcal{O}(n)} \quad \underbrace{\widetilde{\mathcal{H}}_0}_{\text{size } \mathcal{O}(d)} \quad \underbrace{\overline{\mathcal{G}}}_{\text{size } \mathcal{O}(d)} \quad \underbrace{\widetilde{\mathcal{H}}_h}_{\text{size } \mathcal{O}(d)} \quad \underbrace{\mathcal{H}_{h+1}}_{\text{size } \mathcal{O}(n)} \quad \textbf{halt if } \mathsf{x}_{-1}, \mathsf{x}_h, \mathsf{a}_0^{d-1}, \mathsf{a}_1^{d-1} = 0. \tag{11}$$

Its size is $\mathcal{O}(n+d)$ and its dimension is larger by $2d$ than the dimension of $\widetilde{\mathcal{P}}$.

▷ Claim 15 (Correctness). $\widetilde{\mathcal{P}}$ has a halting run if, and only if, $\overline{\mathcal{P}}$ has one.

Proof. Again, halting runs of $\widetilde{\mathcal{P}}$ are in one-to-one correspondence with halting runs of $\overline{\mathcal{P}}$. Indeed, a halting run of $\widetilde{\mathcal{P}}$ is faithfully simulated in $\overline{\mathcal{P}}$ by iterating through $j = 0, 1, \ldots, h-1$ in $\overline{\mathcal{G}}$. Conversely, by the very construction of $\overline{\mathcal{P}}$, its every halting run simulates in this way some halting run of $\widetilde{\mathcal{P}}$. Note that $\overline{\mathcal{P}}$ has also other runs which are necessarily partial (i.e., they fail to reach the **halt** command), because of choosing a failing branch in some of the conditional constructs (10). ◁

The optimisation $\mathcal{P} \mapsto \widetilde{\mathcal{P}}$ described above yields a reduction $\mathcal{M} \mapsto \overline{\mathcal{P}}$, whose correctness follows by Claims 5, 13 and 15:

▷ Claim 16. Given a 3-counter Minsky machine $\mathcal{M}$ of size n and a positive integer d, one can compute in time $\mathcal{O}(n+d)$ a counter program $\overline{\mathcal{P}}$ of dimension $4d+13$ such that $\mathcal{M}$ has a halting run with counters bounded by $\mathcal{F}(2^d+1, n)$ if, and only if, $\overline{\mathcal{P}}$ has a halting run.

Putting $d = n$, we obtain a linear-time reduction from the EXP-TOWER HALTING PROBLEM. In consequence, the VASS reachability problem is 2^n-TOWER-hard, with respect to polynomial-time reductions.

[6] As remarked in the footnote 4 in Section 4, $\overline{\mathcal{G}}$ adapts straightforwardly if, instead of just one $\mathcal{H}$, two different program fragments are used, say $\mathcal{H}_0$ and $\mathcal{H}_1$, one of them for even j and the other one for odd j. It is enough to replace line 3 by **if** $\mathsf{b}_0 = 1$ **then** $\overline{\mathcal{H}}_1$ **else** $\overline{\mathcal{H}}_0$

References

1 David Angeli, Patrick De Leenheer, and Eduardo D. Sontag. Persistence results for chemical reaction networks with time-dependent kinetics and no global conservation laws. *SIAM Journal of Applied Mathematics*, 71(1):128–146, 2011. doi:10.1137/090779401.

2 Paolo Baldan, Nicoletta Cocco, Andrea Marin, and Marta Simeoni. Petri nets for modelling metabolic pathways: a survey. *Natural Computing*, 9(4):955–989, 2010. doi:10.1007/s11047-010-9180-6.

3 Mikołaj Bojańczyk, Claire David, Anca Muscholl, Thomas Schwentick, and Luc Segoufin. Two-variable logic on data words. *ACM Trans. Comput. Log.*, 12(4):27:1–27:26, 2011. doi:10.1145/1970398.1970403.

4 Mikołaj Bojańczyk, Anca Muscholl, Thomas Schwentick, and Luc Segoufin. Two-variable logic on data trees and XML reasoning. *J. ACM*, 56(3):13:1–13:48, 2009. doi:10.1145/1516512.1516515.

5 Ahmed Bouajjani and Michael Emmi. Analysis of recursively parallel programs. *ACM Trans. Program. Lang. Syst.*, 35(3):10:1–10:49, 2013. doi:10.1145/2518188.

6 Frank P. Burns, Albert Koelmans, and Alexandre Yakovlev. WCET analysis of superscalar processors using simulation with coloured Petri nets. *Real-Time Systems*, 18(2/3):275–288, 2000. doi:10.1023/A:1008101416758.

7 Thomas Colcombet and Amaldev Manuel. Generalized data automata and fixpoint logic. In *FSTTCS*, volume 29 of *LIPIcs*, pages 267–278. Schloss Dagstuhl, 2014. doi:10.4230/LIPIcs.FSTTCS.2014.267.

8 Stefano Crespi-Reghizzi and Dino Mandrioli. Petri nets and Szilard languages. *Information and Control*, 33(2):177–192, 1977. doi:10.1016/S0019-9958(77)90558-7.

9 Wojciech Czerwinski, Slawomir Lasota, Ranko Lazic, Jérôme Leroux, and Filip Mazowiecki. The reachability problem for Petri nets is not elementary. In Moses Charikar and Edith Cohen, editors, *Proc. STOC 2019*, pages 24–33. ACM, 2019.

10 Normann Decker, Peter Habermehl, Martin Leucker, and Daniel Thoma. Ordered navigation on multi-attributed data words. In *CONCUR*, volume 8704 of *LNCS*, pages 497–511. Springer, 2014. doi:10.1007/978-3-662-44584-6_34.

11 Stéphane Demri, Diego Figueira, and M. Praveen. Reasoning about data repetitions with counter systems. *Logical Methods in Computer Science*, 12(3), 2016. doi:10.2168/LMCS-12(3:1)2016.

12 Javier Esparza. Decidability and complexity of Petri net problems — an introduction. In *Lectures on Petri Nets I*, volume 1491 of *LNCS*, pages 374–428. Springer, 1998. doi:10.1007/3-540-65306-6_20.

13 Javier Esparza, Pierre Ganty, Jérôme Leroux, and Rupak Majumdar. Verification of population protocols. *Acta Inf.*, 54(2):191–215, 2017. doi:10.1007/s00236-016-0272-3.

14 Patrick C. Fischer, Albert R. Meyer, and Arnold L. Rosenberg. Counter machines and counter languages. *Mathematical Systems Theory*, 2(3):265–283, 1968. doi:10.1007/BF01694011.

15 Pierre Ganty and Rupak Majumdar. Algorithmic verification of asynchronous programs. *ACM Trans. Program. Lang. Syst.*, 34(1):6:1–6:48, 2012. doi:10.1145/2160910.2160915.

16 Steven M. German and A. Prasad Sistla. Reasoning about systems with many processes. *J. ACM*, 39(3):675–735, 1992. doi:10.1145/146637.146681.

17 Sheila A. Greibach. Remarks on blind and partially blind one-way multicounter machines. *Theor. Comput. Sci.*, 7:311–324, 1978. doi:10.1016/0304-3975(78)90020-8.

18 Piotr Hofman and Sławomir Lasota. Linear equations with ordered data. In *CONCUR*, volume 118 of *LIPIcs*, pages 24:1–24:17. Schloss Dagstuhl, 2018. doi:10.4230/LIPIcs.CONCUR.2018.24.

19 John E. Hopcroft and Jean-Jacques Pansiot. On the reachability problem for 5-dimensional vector addition systems. *Theor. Comput. Sci.*, 8:135–159, 1979. doi:10.1016/0304-3975(79)90041-0.

20 Alexander Kaiser, Daniel Kroening, and Thomas Wahl. A widening approach to multithreaded program verification. *ACM Trans. Program. Lang. Syst.*, 36(4):14:1–14:29, 2014. doi:10.1145/2629608.

21 Max I. Kanovich. Petri nets, Horn programs, linear logic and vector games. *Ann. Pure Appl. Logic*, 75(1–2):107–135, 1995. doi:10.1016/0168-0072(94)00060-G.

22 Richard M. Karp and Raymond E. Miller. Parallel program schemata. *J. Comput. Syst. Sci.*, 3(2):147–195, 1969. doi:10.1016/S0022-0000(69)80011-5.

23 S. Rao Kosaraju. Decidability of reachability in vector addition systems (preliminary version). In *STOC*, pages 267–281. ACM, 1982. doi:10.1145/800070.802201.

24 Jean-Luc Lambert. A structure to decide reachability in Petri nets. *Theor. Comput. Sci.*, 99(1):79–104, 1992. doi:10.1016/0304-3975(92)90173-D.

25 Hélène Leroux, David Andreu, and Karen Godary-Dejean. Handling exceptions in Petri net-based digital architecture: From formalism to implementation on FPGAs. *IEEE Trans. Industrial Informatics*, 11(4):897–906, 2015. doi:10.1109/TII.2015.2435696.

26 Jérôme Leroux. The general vector addition system reachability problem by Presburger inductive invariants. *Logical Methods in Computer Science*, 6(3), 2010. doi:10.2168/LMCS-6(3:22)2010.

27 Jérôme Leroux. Vector addition system reachability problem: a short self-contained proof. In *POPL*, pages 307–316. ACM, 2011. doi:10.1145/1926385.1926421.

28 Jérôme Leroux. Vector addition systems reachability problem (A simpler solution). In *Turing-100*, volume 10 of *EPiC Series in Computing*, pages 214–228. EasyChair, 2012. URL: http://www.easychair.org/publications/paper/106497.

29 Jérôme Leroux and Sylvain Schmitz. Demystifying reachability in vector addition systems. In *LICS*, pages 56–67. IEEE Computer Society, 2015. doi:10.1109/LICS.2015.16.

30 Jérôme Leroux and Sylvain Schmitz. Reachability in vector addition systems is primitive-recursive in fixed dimension. In *Proc. LICS 2019*, pages 1–13. IEEE, 2019.

31 Yuliang Li, Alin Deutsch, and Victor Vianu. VERIFAS: A practical verifier for artifact systems. *PVLDB*, 11(3):283–296, 2017. URL: http://www.vldb.org/pvldb/vol11/p283-li.pdf, doi:10.14778/3157794.3157798.

32 Richard J. Lipton. The reachability problem requires exponential space. Technical Report 62, Yale University, 1976. URL: http://cpsc.yale.edu/sites/default/files/files/tr63.pdf.

33 Ernst W. Mayr. An algorithm for the general Petri net reachability problem. In *STOC*, pages 238–246. ACM, 1981. doi:10.1145/800076.802477.

34 Ernst W. Mayr. An algorithm for the general Petri net reachability problem. *SIAM J. Comput.*, 13(3):441–460, 1984. doi:10.1137/0213029.

35 Roland Meyer. A theory of structural stationarity in the *pi*-calculus. *Acta Inf.*, 46(2):87–137, 2009. doi:10.1007/s00236-009-0091-x.

36 Marvin L. Minsky. *Computation: finite and infinite machines*. Prentice-Hall, Inc., 1967. URL: https://dl.acm.org/citation.cfm?id=1095587.

37 Mor Peleg, Daniel L. Rubin, and Russ B. Altman. Research paper: Using Petri net tools to study properties and dynamics of biological systems. *JAMIA*, 12(2):181–199, 2005. doi:10.1197/jamia.M1637.

38 Carl Adam Petri. *Kommunikation mit Automaten*. PhD thesis, Universität Hamburg, 1962. URL: http://edoc.sub.uni-hamburg.de/informatik/volltexte/2011/160/.

39 Charles Rackoff. The covering and boundedness problems for vector addition systems. *Theor. Comput. Sci.*, 6:223–231, 1978. doi:10.1016/0304-3975(78)90036-1.

40 George S. Sacerdote and Richard L. Tenney. The decidability of the reachability problem for vector addition systems (preliminary version). In *STOC*, pages 61–76. ACM, 1977. doi:10.1145/800105.803396.

41 Sylvain Schmitz. Complexity hierarchies beyond elementary. *TOCT*, 8(1):3:1–3:36, 2016. doi:10.1145/2858784.

42 Sylvain Schmitz. The complexity of reachability in vector addition systems. *SIGLOG News*, 3(1):4–21, 2016. doi:10.1145/2893582.2893585.

43 Wil M. P. van der Aalst. Business process management as the "killer app" for Petri nets. *Software and System Modeling*, 14(2):685–691, 2015. doi:10.1007/s10270-014-0424-2.

New Techniques for Universality in Unambiguous Register Automata

Wojciech Czerwiński
University of Warsaw, Poland

Antoine Mottet
Department of Algebra, Faculty of Mathematics and Physics,
Charles University, Prague, Czech Republic

Karin Quaas
University of Leipzig, Germany

Abstract

Register automata are finite automata equipped with a finite set of registers ranging over the domain of some relational structure like $(\mathbb{N}; =)$ or $(\mathbb{Q}; <)$. Register automata process words over the domain, and along a run of the automaton, the registers can store data from the input word for later comparisons. It is long known that the universality problem, i.e., the problem to decide whether a given register automaton accepts all words over the domain, is undecidable. Recently, we proved the problem to be decidable in 2-ExpSpace if the register automaton under study is over $(\mathbb{N}; =)$ and unambiguous, i.e., every input word has at most one accepting run; this result was shortly after improved to 2-ExpTime by Barloy and Clemente. In this paper, we go one step further and prove that the problem is in ExpSpace, and in PSpace if the number of registers is fixed. Our proof is based on new techniques that additionally allow us to show that the problem is in PSpace for single-register automata over $(\mathbb{Q}; <)$. As a third technical contribution we prove that the problem is decidable (in ExpSpace) for a more expressive model of unambiguous register automata, where the registers can take values nondeterministically, if defined over $(\mathbb{N}; =)$ and only one register is used.

2012 ACM Subject Classification Theory of computation → Automata over infinite objects

Keywords and phrases Register Automata, Data Languages, Unambiguity, Unambiguous, Universality, Containment, Language Inclusion, Equivalence

Digital Object Identifier 10.4230/LIPIcs.ICALP.2021.129

Category Track B: Automata, Logic, Semantics, and Theory of Programming

Funding *Wojciech Czerwiński*: Supported by the European Research Council (ERC) grant LIPA, grant agreement No 683080.
Antoine Mottet: This author has received funding from the ERC under the European Union's Horizon 2020 research and innovation programme (grant agreement No 771005).
Karin Quaas: Funded by the Deutsche Forschungsgemeinschaft (DFG), project 406907430.

Acknowledgements We thank Lorenzo Clemente, Sławomir Lasota, and Radosław Piórkowski for inspiring discussions on URA.

1 Introduction

Certainly, determinism plays a central role in the research about computation models. Recently, a lot of active research work [1, 6, 3, 16, 14] is devoted to its weaker form: *unambiguity*. A system is *unambiguous* if for every input word there is at most one accepting run. Unambiguous systems exhibit elegant properties; in particular many natural computational problems turn out to be easier compared to the general case. A prominent example is the *universality problem* for finite automata, *i.e.*, the problem of deciding whether a given automaton accepts *every* input word. It is in PTime [18] and even in NC^2 [19] in the unambiguous case, as opposed to PSpace-complete in the general case.

In his seminal overview article about unambiguity, Colcombet [5] states some very natural conjectures about unambiguous systems that are so fundamental that one can be surprised that they are still open. An example conjecture, motivated by the fact that the universality problem for unambiguous finite automata is in PTime, was that for every unambiguous finite automaton the complement of its language can be accepted by another unambiguous finite automaton with at most polynomial size with respect to the size of the original automaton. This conjecture was surprisingly resolved negatively by Raskin [17], who provided a family of automata where a blowup $\Theta(n^{\log\log\log(n)})$ is unavoidable. Still, a lot of other natural questions remain unresolved. Some of them are not algorithmic (as the above one), while others ask for the existence of faster algorithms in the unambiguous case.

Usually one cannot hope for designing more efficient algorithms for the emptiness problem, as it is often easy to transform a nondeterministic system to a deterministic (and thus unambiguous) system which has empty language if and only if the accepted language of the original system is empty. Indeed, it is often sufficient to change the labelling of every transition of the system to its unique transition name. This transformation preserves the emptiness property, but not much more. Therefore there is a hope that the unambiguity assumption may result in faster solving of problems like universality, equivalence and language containment. Recently there was a substantial amount of research in this area [11, 4, 7, 14, 6, 1]. The considered problem is often the universality problem. Indeed, the universality problem is probably the easiest nontrivial problem for which there is a hope to obtain an improvement in the unambiguous case. Equivalence and containment are often not much harder, even though sometimes a bit more involved techniques are needed.

For *register automata*, this line of research was started in [14]. Register automata (RA, for short) extend finite automata with a finite set of registers that take values from an infinite data domain for later comparisons. More detailed, RA are defined over a relational structure, like $(\mathbb{N};=)$ or $(\mathbb{Q};<,=)$; they process finite words over the domain of the relational structure, and the registers can store values from the input word for comparing them using the relations provided by the relational structure. In the more expressive model of register automata *with guessing* (GRA) the registers can even take arbitrary values. In [14] it is shown that for unambiguous RA (URA) over $(\mathbb{N};=)$ the containment problem is in 2-ExpSpace and in ExpSpace for a fixed number of registers. Without the unambiguity assumption, this problem is known to be much harder. Concretely, the universality problem is undecidable as soon as the automaton uses two registers [12, 15, 9], and Ackermann-complete in the one-register case [10]. In the case of GRA even the one-register case is undecidable.[1] The result for URA in [14] was improved by Barloy and Clemente [1] who have shown that the problem is in 2-ExpTime and in ExpTime for a fixed number of registers, using very different tools such as linear recursive sequences in two dimensions.

Our contribution

Our result improves statements of Barloy and Clemente [1] even further. We provide three results shown by two different techniques. Our first technique is to show that in some cases one can assume that only a linear or exponential number of different configurations can be reached via an input word. This claim immediately provides an improved upper bound compared to [1].

[1] A proof for undecidability can be done using a reduction from the undecidable reachability problem for Minsky machines, following the lines of the proof of Theorem 5.2 in [8]. The nondeterministic guessing can be used to express that there exists some decrement for which there is no matching preceding increment.

▶ **Theorem 1.** *The containment problem $L(\mathcal{A}) \subseteq L(\mathcal{B})$ is in* ExpSpace*, if $\mathcal{A}$ is an* RA *and $\mathcal{B}$ is a* URA *over $(\mathbb{N}; =)$. The containment problem is in* PSpace *on inputs $\mathcal{A}, \mathcal{B}$ both having a bounded number of registers.*

This approach can also be applied to unambiguous one-register automata over $(\mathbb{Q}; <, =)$.

▶ **Theorem 2.** *The universality problem for one-register* URA *over $(\mathbb{Q}; <, =)$ is in* PSpace*.*

Our techniques used to show Theorems 1 and 2 differ from techniques used previously in [14]. In [14] a reached configuration was compressed if it was too big. More concretely, if two different data values were equivalent in a certain sense with respect to the current configuration, one of them was eliminated from the configuration, and the resulting configuration was equivalent. In that way, a compressed configuration had only non-equivalent data values and therefore its size was bounded. In this work, we analyse a reachable configuration much more locally, looking only at states with the same location. We show that if there are too many states with the same location in a certain configuration, then it is not possible that the considered unambiguous automaton is universal. In that quite different way we provide a bound on the size of a reachable configuration in an unambiguous universal automaton.

However, we will see that the techniques for URA do not work for unambiguous GRA (GURA), not even in the one-register case. In that case we solve the universality problem, and even the containment problem, with the use of more sophisticated analysis, closer related to the technique from [14]. In short, we show that we can modify the set of reachable configurations such that it becomes small and equivalent in some sense, which also allows us to obtain a more efficient algorithm.

▶ **Theorem 3.** *The containment problem $L(\mathcal{A}) \subseteq L(\mathcal{B})$ is in* ExpSpace*, if $\mathcal{A}$ is a* GRA *over $(\mathbb{N}; =)$ and $\mathcal{B}$ is a one-register* GURA *over $(\mathbb{N}; =)$.*

Independently from our work, Bojańczyk, Klin and Moerman [2] have recently published a result about orbit-finite vector spaces, which, for GURA with register values comparable wrt. a linear order, implies an algorithm that works in ExpTime if the number of registers is not fixed, and in PTime if the number of registers is fixed. However, we believe that our contribution does not only provide an improved complexity of the considered problem (compared with previous results), but also techniques that can be useful in future research on unambiguous systems.

2 Preliminaries

In this section, we define *register automata*, introduced by Kaminski et al. [12, 13]. We start with some basic notions used throughout the paper. We use Σ to denote a finite alphabet, and $\mathbb{N}$ and $\mathbb{Q}$ denote the set of non-negative integers and rational numbers, respectively. Given $a, b \in \mathbb{N}$ with $a \leq b$, we write $[a, b]$ to denote the set $\{a, a+1, \ldots, b\}$.

A *relational structure* is a tuple $\mathcal{D} = (\mathbb{D}; R_1, \ldots, R_k)$, where $\mathbb{D}$ is an infinite domain, and $R_1, \ldots, R_k$ are binary relations over $\mathbb{D}$, and we assume that R_k is the equality relation. In this paper, we are mainly interested in the relational structures $(\mathbb{N}; =)$ of the non-negative integers with equality, and $(\mathbb{Q}; <, =)$ of the rationals with the usual order and equality relations.

A *data word* is a finite sequence $(\sigma_1, d_1) \ldots (\sigma_k, d_k) \in (\Sigma \times \mathbb{D})^*$. If $\Sigma = \{\sigma\}$ is a singleton set, we may write $d_1 \cdot d_2 \cdot \ldots \cdot d_k$ shortly for $(\sigma, d_1)(\sigma, d_2) \ldots (\sigma, d_k)$. We use ε to denote the empty data word. A *data language* is a set of data words. We use $\mathrm{data}(w)$ to denote the set $\{d_1, \ldots, d_k\}$ of all data occurring in w.

Let $\mathbb{D}_\perp$ denote the set $\mathbb{D} \cup \{\perp\}$, where $\perp \notin \mathbb{D}$. We let $\perp \neq d$ for all $d \in \mathbb{D}$, and $\perp$ is incomparable with respect to $\leq$ to all $d \in \mathbb{D}$. We use boldface lower-case letters like $\boldsymbol{a}, \boldsymbol{b}, \ldots, \boldsymbol{u} \ldots$ to denote tuples in $\mathbb{D}_\perp^n$, where $n \in \mathbb{N}$. Given a tuple $\boldsymbol{a} \in \mathbb{D}_\perp^n$, we write a_i for its i-th component, and $\mathrm{data}(\boldsymbol{a})$ denotes the set $\{a_1, \ldots, a_n\} \subseteq \mathbb{D}_\perp$ of all data occurring in $\boldsymbol{a}$.

Let $\mathcal{R} = \{r_1, \ldots, r_n\}$ be a finite set of *registers*. A *register valuation* is a mapping $\boldsymbol{u} : \mathcal{R} \to \mathbb{D}_\perp$; we may write u_i as shorthand for $\boldsymbol{u}(r_i)$. Let $\mathbb{D}_\perp^{\mathcal{R}}$ denote the set of all register valuations. A *register constraint over $\mathcal{D}$ and $\mathcal{R}$* is defined by the grammar

$$\phi ::= \texttt{true} \mid R(t_1, t_2) \mid \neg\phi \mid \phi \wedge \phi$$

where R is a binary relation symbol from the relational structure $\mathcal{D}$, and $t_i \in \{\#\} \cup \{r, \dot{r} \mid r \in \mathcal{R}\}$. Here $\#$ is a symbol representing the current input datum, r refers to the current value of the register r, and $\dot{r}$ refers to the future value of the register r. We use $\Phi(\mathcal{D}, \mathcal{R})$ to denote the set of all register constraints over $\mathcal{D}$ and $\mathcal{R}$. The satisfaction relation $\models$ on $\mathbb{D}_\perp^{\mathcal{R}} \times \mathbb{D} \times \mathbb{D}_\perp^{\mathcal{R}}$ is defined by structural induction as follows. We only give some atomic cases; the other cases can be derived easily. We have $(\boldsymbol{u}, d, \boldsymbol{v}) \models \phi$ if

- ϕ is of the form $\texttt{true}$,
- ϕ is of the form $R(r_i, \#)$ and $\mathcal{D} \models R(u_i, d)$,
- ϕ is of the form $R(\dot{r}_i, r_i)$ and $\mathcal{D} \models R(v_i, u_i)$,
- ϕ is of the form $R(\dot{r}_i, \#)$ and $\mathcal{D} \models R(v_i, d)$.

For example, $\phi := \neg(r = \#) \wedge (\dot{r} = r)$ is a register constraint over $(\mathbb{N}; =)$ and $\mathcal{R} = \{r\}$, and we have $(1, 2, 1) \models \phi$, whereas $(1, 2, 3) \not\models \phi$.

It is important to note that only register constraints of the form $\dot{r} = r$ and $\dot{r} = \#$ uniquely determine the new value of r. In absence of such a register constraint, the register r can nondeterministically take any of infinitely many data values from $\mathbb{D}$, with the following restrictions: the register constraint $\neg(\dot{r} = \#)$ requires that the new value of r is different from the current input datum, so that r may take any datum in $\mathbb{D}$ except for the input datum. Likewise, the register constraint $\neg(\dot{r} = r)$ requires that r takes any datum in $\mathbb{D}$ except for the current value of r. Register automata that allow for such nondeterministic *guessing* of future register values are also called *register automata with guessing*. Formally, a register automaton with guessing (GRA) over $\mathcal{D}$ and Σ is a tuple $\mathcal{A} = (\mathcal{R}, \mathcal{L}, \ell_{\mathrm{init}}, \mathcal{L}_{\mathrm{acc}}, E)$, where

- $\mathcal{R}$ is a finite set of registers,
- $\mathcal{L}$ is a finite set of locations,
- $\ell_{\mathrm{init}} \in \mathcal{L}$ is the initial location,
- $\mathcal{L}_{\mathrm{acc}} \subseteq \mathcal{L}$ is the set of accepting locations,
- $E \subseteq \mathcal{L} \times \Sigma \times \Phi(\mathcal{D}, \mathcal{R}) \times \mathcal{L}$ is a finite set of edges.

If every edge of $\mathcal{A}$ contains some constraint of the form $\dot{r} = r$ or $\dot{r} = \#$, for every $r \in \mathcal{R}$, so that the future value of every register is uniquely determined, then we simply speak of *register automata* (RA, for short), *i.e.*, register automata without guessing. If the number of registers of a GRA (RA, respectively) is fixed to $k \in \mathbb{N}$, then we speak of k-GRA (k-RA, respectively).

A *state* of $\mathcal{A}$ is a pair $(\ell, \boldsymbol{u}) \in \mathcal{L} \times \mathbb{D}_\perp^{\mathcal{R}}$, where ℓ is the current location and $\boldsymbol{u}$ is the current register valuation. Abusing notation a bit, we usually write $\ell(\boldsymbol{u})$ instead of $(\ell, \boldsymbol{u})$. The state $\ell_{\mathrm{init}}(\boldsymbol{u}_{\mathrm{init}})$, where $\boldsymbol{u}_{\mathrm{init}}$ maps every register $r \in \mathcal{R}$ to $\perp$, is called the *initial state*, and a state $\ell(\boldsymbol{u})$ is called *accepting* if $\ell \in \mathcal{L}_{\mathrm{acc}}$. Given two states $\ell(\boldsymbol{u})$ and $\ell'(\boldsymbol{u'})$ and some input letter $(\sigma, d) \in \Sigma \times \mathbb{D}$, we postulate a transition $\ell(\boldsymbol{u}) \xrightarrow{\sigma,d}_{\mathcal{A}} \ell'(\boldsymbol{u'})$ if there exists some edge $(\ell, \sigma, \phi, \ell') \in E$ such that $(\boldsymbol{u}, d, \boldsymbol{u'}) \models \phi$. A *run* of $\mathcal{A}$ on the data word $(\sigma_1, d_1) \ldots (\sigma_k, d_k)$ is a sequence $\ell_0(\boldsymbol{u^0}) \xrightarrow{\sigma_1,d_1}_{\mathcal{A}} \ell_1(\boldsymbol{u^1}) \xrightarrow{\sigma_2,d_2}_{\mathcal{A}} \ldots \xrightarrow{\sigma_k,d_k}_{\mathcal{A}} \ell_k(\boldsymbol{u^k})$ of such transitions. We say that a run as above *starts in* $\ell_0(\boldsymbol{u^0})$; similarly, the run *ends in* $\ell_k(\boldsymbol{u^k})$. A state $\ell(\boldsymbol{u})$ is *reachable*

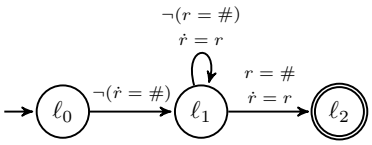


Figure 1 A 1-GURA.

Table 1 Universality over $(\mathbb{N}; =)$.

# registers	RA	URA
$*$	undecidable [8]	**in ExpSpace (Th. 7)**
1	Ackermann-cpl. [10]	**in PSpace (Th. 7)**
# registers	GRA	GURA
$*$	undecidable [13]	
1	undecidable	**in ExpSpace (Th. 3)**

in $\mathcal{A}$ if there exists a run that ends in $\ell(\boldsymbol{u})$. A run is *initialized* if it starts in the initial state, and a run is *accepting* if it ends in some accepting state. A data word w is *accepted from* $\ell(\boldsymbol{u})$ if there exists an accepting run on w that starts in $\ell(\boldsymbol{u})$. The data language *accepted* by $\mathcal{A}$, denoted by $L(\mathcal{A})$, is the set of data words that are accepted from the initial state.

A GRA is *unambiguous* if for every input data word w there is at most one initialized accepting run. Note that unambiguity is a semantic condition; it can be checked in polynomial time [5]. We write GURA and URA to denote unambiguous GRA and RA, respectively.

▶ **Example 4.** Let us study the behaviour of the 1-GRA depicted in Figure 1. The GRA is over $(\mathbb{N}; =)$ and the singleton alphabet $\Sigma = \{\sigma\}$ (we omit the letter σ from all transitions in the figure). Suppose the first input letter is d_1. In order to satisfy the constraint of the transition from ℓ_1 to ℓ_2, the automaton has to nondeterministically guess some datum $d' \neq d_1$ and store it into its register r. Being in the state $\ell_1(d')$, the automaton can only move to the accepting location ℓ_2 if the next input datum is equal to d' (indicated by the constraint $r = d$); for every other input letter, the automaton satisfies the constraint $\neg(r = d)$ and stays in ℓ_1, and it keeps the register value to satisfy the constraint $\dot{r} = r$. In this way, the automaton accepts the language $\{d_1 \cdot \ldots \cdot d_k \mid \exists k \geq 2\ \forall 1 \leq i < k.\ d_i \neq d_k\}$. Note that the automaton is unambiguous: for every input data word there is only one accepting run. We remark that the accepted data language cannot be accepted by any RA (without guessing) [13]. Hence, GRA are more expressive than RA.

In this paper, we study the *universality problem*: given a GRA $\mathcal{A}$, is $\mathcal{A}$ universal, *i.e.*, does $L(\mathcal{A}) = (\Sigma \times \mathbb{D})^*$ hold? In Table 1, we give an overview of the decidability status for register automata over $(\mathbb{N}; =)$, in bold the new results for unambiguous register automata that we present in this paper.

3 Basic Notions for Deciding Universality

For many computational models, a standard approach for solving the universality problem is to explore the (potentially infinite) state space of the automaton under study. Starting from the initial state, the basic idea is to input one letter after the other, and keep track of the *sets of states* that are reached, building a reachability graph whose nodes are the reached sets of states (per input letter). The key property of this state space is that it contains sufficient information to decide whether the automaton under study is universal: this is the case if, and only if, every node of the graph contains an accepting state. Let us formalize this intuition for register automata.

Fix a k-GRA $\mathcal{A} = (\mathcal{R}, \mathcal{L}, \ell_{\text{init}}, \mathcal{L}_{\text{acc}}, E)$ over $\mathcal{D}$ and Σ, for some $k \in \mathbb{N}$. A *configuration of* $\mathcal{A}$ is a subset of $\mathcal{L} \times \mathbb{D}_\perp^k$. The set C_{init}, denoting the singleton set containing the initial state of $\mathcal{A}$, is a configuration, henceforth called the *initial configuration*. Let C be a configuration, and let $(\sigma, d) \in (\Sigma \times \mathbb{D})$. We use $\text{Succ}_{\mathcal{A}}(C, (\sigma, d))$ to denote the *successor of* C *on the input* (σ, d), formally defined by

$$\mathrm{Succ}_{\mathcal{A}}(C,(\sigma,d)) := \{\ell(\boldsymbol{u}) \mid \exists\, \ell'(\boldsymbol{u'}) \in C\ \ell'(\boldsymbol{u'}) \xrightarrow{\sigma,d}_{\mathcal{A}} \ell(\boldsymbol{u})\}.$$

In order to extend this definition to data words, we define inductively $\mathrm{Succ}_{\mathcal{A}}(C,\varepsilon) := C$ and $\mathrm{Succ}_{\mathcal{A}}(C, w\cdot(\sigma,d)) := \mathrm{Succ}_{\mathcal{A}}(\mathrm{Succ}_{\mathcal{A}}(C,w),(\sigma,d))$. We say that a configuration C is *reachable in $\mathcal{A}$ by the data word w* if $C = \mathrm{Succ}_{\mathcal{A}}(C_{\mathrm{init}}, w)$; we say that C is *reachable in $\mathcal{A}$* if there exists some data word w such that C is reachable in $\mathcal{A}$ by w. We say that C is *coverable* if there exists some $C' \supseteq C$ such that C' is reachable in $\mathcal{A}$. Given a configuration C, we use $\mathrm{data}(C)$ to denote the set $\bigcup_{\ell(\boldsymbol{u})\in C}\mathrm{data}(\boldsymbol{u})$ of data occurring in C. Notice that every configuration reachable in an RA (without guessing) is necessarily finite. In contrast, the configuration $\{\ell_1(d') \mid d' \in \mathbb{N}, d' \neq d_1\}$ is reachable in the GRA in Figure 1 by the single-letter data word (σ, d_1). If $C = \{\ell(\boldsymbol{u})\}$ is a singleton set, then we may, in slight abuse of notation, omit the curly brackets and write $\ell(\boldsymbol{u})$.

We say that a configuration C is *accepting* if there exists $\ell(\boldsymbol{u}) \in C$ such that $\ell \in \mathcal{L}_{\mathrm{acc}}$; otherwise we say that C is *non-accepting*. Clearly, $\mathcal{A}$ is universal if, and only if, every configuration reachable in $\mathcal{A}$ is accepting. This suggests to reduce the universality problem to a reachability problem for the state space corresponding to the given input GRA. However, the state space of a GRA is infinite, in two different aspects.

First of all, the state space is *infinitely branching*, as each of the infinite data in $\mathbb{D}$ may give rise to a unique successor configuration. The standard approach for solving this complication is to abstract from concrete data, using the simple observation that, *e.g.*, the data word $3\cdot 4$ is accepted from the state $\ell(4)$ if, and only if, $5\cdot 2$ is accepted from the state $\ell(2)$. This is formalized in the following paragraph.

A *partial isomorphism of $\mathcal{D}_\perp$* is an injective mapping $\pi\colon D \to \mathbb{D}_\perp$ with domain $\mathrm{dom}(\pi) := D \subseteq \mathbb{D}$ such that:

- for every relation R of $\mathcal{D}$ and $a, b \in \mathrm{dom}(\pi)$, we have $(a,b) \in R \Leftrightarrow (\pi(a),\pi(b)) \in R$,
- if $\perp \in D$ then $\pi(\perp) = \perp$.

Let π be a partial isomorphism of $\mathcal{D}_\perp$ and let C be a configuration such that $\mathrm{data}(C) \subseteq \mathrm{dom}(\pi)$. We define the configuration $\pi(C) := \{\ell(\pi(d_1),\dots,\pi(d_k))) \mid \ell(d_1,\dots,d_k) \in C\}$; likewise, if $\{d_1,\dots,d_k\} \subseteq \mathrm{dom}(\pi)$, we define the data word $\pi(w) = (\sigma_1,\pi(d_1))\dots(\sigma_k,\pi(d_k))$. We write $\langle C, w\rangle \sim \langle C', w'\rangle$ if there exists a partial isomorphism of $\mathcal{D}_\perp$ such that $\pi(C) = C'$ and $\pi(w) = w'$.

▶ **Proposition 5.** *Let $\mathcal{A}$ be a* GRA*. If $\langle C, w\rangle \sim \langle C', w'\rangle$, then $\mathrm{Succ}_{\mathcal{A}}(C,w) \sim \mathrm{Succ}_{\mathcal{A}}(C',w')$.*

Secondly, there can be infinitely many reachable configurations even up to the equivalence relation $\sim$. As an example, consider the GURA in Figure 1. For every $n \geq 1$, the configuration $C_n := \{\ell_1(d') \mid d' \in \mathbb{N}\backslash\{d_1,\dots,d_n\}\} \cup \{\ell_2(d_n)\}$ with pairwise distinct data values $d_1,\dots,d_n$ is reachable by the data word $d_1\cdot d_2\cdot\ldots\cdot d_n$, and $C_n \not\sim C_{n'}$ for $n \neq n'$. There are similar examples also for URA, cf. [14].

In order to obtain our results, we will prove that one can solve the reachability problem for the state space of $\mathcal{A}$ by focussing on a subset of configurations reachable in the automaton under study. The concrete methods are different for URA and GURA, however, for both models we will take advantage of Proposition 5 and its simple consequence (cf. [14]).

▶ **Corollary 6.** *Let $\mathcal{A}$ be a* GRA*. If $\langle C, w\rangle \sim \langle C', w'\rangle$ and $\mathrm{Succ}_{\mathcal{A}}(C,w)$ is non-accepting (accepting, respectively), then $\mathrm{Succ}_{\mathcal{A}}(C',w')$ is non-accepting (accepting, respectively).*

4 The Universality Problem for URA over $(\mathbb{N}; =)$

In this section, we study the complexity of the universality problem for URA over the relational structure $(\mathbb{N}; =)$. We prove the following theorem.

▶ **Theorem 7.** *The universality problem is*
- *in* PSpace *for* k-URA *for any fixed* $k \in \mathbb{N}$,
- *in* ExpSpace *for* URA.

We start by showing that we can assume URA to have a specific form that simplifies the coming proofs. Given some k-URA $\mathcal{A}$, we say that $\mathcal{A}$ is *pruned* if for every state $\ell(\boldsymbol{u})$ that is reachable in $\mathcal{A}$ there exists a data word w that is accepted from $\ell(\boldsymbol{u})$, and $u_i \neq u_j$ for all $1 \leq i < j \leq k$, *i.e.*, no datum appears more than once in $\boldsymbol{u}$. The proof of the following proposition is simple and omitted.

▶ **Proposition 8.** *For every* k-URA *one can compute in polynomial time an equivalent pruned* k-URA.

In the following we always assume that a k-URA is pruned, even if we do not explicitly mention it. For simplicity, we also assume that the alphabet of the URA we consider are singletons. The techniques we develop can be easily lifted to the more general case where Σ is not a singleton.

We introduce some constants that bound from above the number of states with the same location occurring in a configuration reachable in a universal URA. Let $\mathcal{A}$ be a k-URA. For a configuration C of $\mathcal{A}$, define $M_C \in \mathbb{N}$ to be the maximal number M such that in C there are M different states with the same location. Define $M_{\mathcal{A}} \in \mathbb{N} \cup \{\infty\}$ to be the supremum of M_C, for C ranging over all the configurations C reachable in $\mathcal{A}$, if $\mathcal{A}$ is a *universal* k-URA, *i.e.*, $L(\mathcal{A}) = \mathbb{D}^*$. In the sequel, we show that $M_{\mathcal{A}} < \infty$. In order to do so, for $k \in \mathbb{N}$, define $\mathbf{M}_k \in \mathbb{N} \cup \{\infty\}$ to be the supremum of all the $M_{\mathcal{A}}$, for $\mathcal{A}$ ranging over pruned and universal k-URA. The main technical result of this section is showing that $\mathbf{M}_k$ is finite and moreover upper-bounded by an exponential function of k.

Let n be the number of locations of $\mathcal{A}$. First observe that showing $\mathbf{M}_k \in \mathbb{N}$ easily implies the existence of a NPSpace algorithm deciding whether $\mathcal{A}$ is universal. Indeed, if $\mathbf{M}_k < \infty$, then every configuration C reachable in $\mathcal{A}$ has size at most $n \cdot \mathbf{M}_k$, as otherwise C contains more than $\mathbf{M}_k$ states with the same location. Thus, in order to decide whether $\mathcal{A}$ is not universal, we can apply the following algorithm:
- By Corollary 6, $\mathcal{A}$ is not universal iff $\mathcal{A}$ does not accept some data word $(\sigma_1, d_1)(\sigma_2, d_2)\ldots$, where $d_i \in \{0, \ldots, i\}$ for all i.
- Guess, letter by letter, an input data word $(\sigma_1, d_1)(\sigma_2, d_2)\ldots$, where $d_i \in \{0, \ldots, i\}$.
- For each $i \geq 1$, define $C_i := \mathrm{Succ}_{\mathcal{A}}(C_{i-1}, (\sigma_i, d_i))$, where $C_0 = C_{\mathrm{init}}$.
- If for some $i \geq 1$, the configuration C_i is not accepting or its size exceeds $n \cdot \mathbf{M}_k$, we know that $\mathcal{A}$ is not universal.
- Otherwise we keep the configuration in the space linear with respect to n and count the length of the word. If the length exceeds the number of possible configurations, then this run is not accepting. The length counter can be also kept in linear space.

The above is hence a PSpace-algorithm for deciding non-universality for k-URA. By Savitch's theorem, there also exists one for deciding universality for k-URA. Moreover, if we show that $\mathbf{M}_k$ is exponential in k, then the above algorithm works in space exponential with respect to k, so is in ExpSpace even without fixing the number of registers k. Therefore, in order to show Theorem 7, it is enough to prove that $\mathbf{M}_k$ is bounded by some exponential function of k. The rest of this Section is devoted mainly to showing the following lemma.

▶ **Lemma 9.** $\mathbf{M}_k \leq (k \cdot 4^k \cdot k!)^k$.

We first give here an argument showing that $\mathbf{M}_k$ is bounded by some doubly-exponential function in k. We show this argument in order to illustrate the techniques, which needed to be refined in our proof of Lemma 9. First recall that the *Ramsey number* $R_m(n)$ is the smallest number of vertices k of the graphs such that any clique of k vertices with its edges coloured on m different colours contain a monochromatic subgraph G of n vertices, namely such that all the edges in G are of the same colour. It can be shown by induction that $R_m(n)$ is finite, and indeed its growth is bounded by $2^{n^{O(m)}}$.

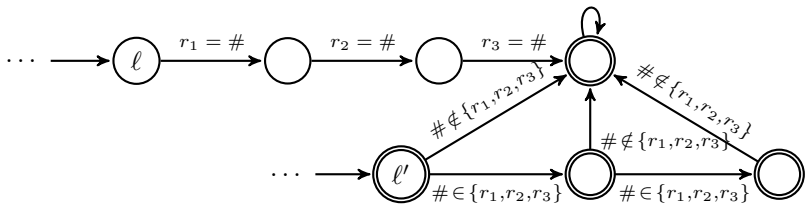
▶ **Proposition 10.** *Every set $T \subseteq \mathbb{D}_\perp^k$ of size at least $R_{k+1}(4^k(k+1)!+1)$ contains a k-full subset of size at least $k+1$.*

Proof. Construct a graph with vertices being tuples from T and edge between $\boldsymbol{t}$ and $\boldsymbol{t'}$ be coloured by the number of data values that $\boldsymbol{t}$ and $\boldsymbol{t'}$ have in common. Clearly the colour belongs to the set $\{0, \ldots, k\}$, so there are $k+1$ colours. Because $|S| \geq R_{k+1}(4^k(k+1)!+1)$ we know from Ramsey's theorem that there are at least $4^k(k+1)!+1$ tuples such that every intersection is of the same size - assume this size to be m. Let S be a set of $4^k(k+1)!+1$ such tuples and let $\boldsymbol{s} \in S$ be one of them. Let $\boldsymbol{s} = (d_1, \ldots, d_k)$. Divide all the other tuples $\boldsymbol{s'}$ into $\binom{k}{m}^2 \cdot m!$ sets depending on which m data values from $\{d_1, \ldots, d_k\}$ belong to $\boldsymbol{s'}$ (there are $\binom{k}{m}$ options), on which positions they are located in $\boldsymbol{s'}$ (also $\binom{k}{m}$ options) and in which order ($m!$ options). It is easy to see that $\binom{k}{m}^2 \cdot m! \leq 4^k k!$, as $\binom{k}{m} \leq 2^k$ and $m! \leq k!$. We divide $4^k(k+1)!$ tuples (we omit $\boldsymbol{s}$) into at most $4^k k!$ sets, so by the pigeonhole principle at least one of them contains at least $k+1$ elements: let these elements be $\boldsymbol{s^1}, \ldots, \boldsymbol{s^{k+1}}$. Notice now that the tuples $\boldsymbol{s^1}, \ldots, \boldsymbol{s^{k+1}}$ form a k-full set: indeed on positions on which they have the m shared data they are identical and on the other positions all the data values are totally different. Thus T contains a k-full set of size $k+1$, which finishes the proof. ◀

Before refining our argument to give an exponential bound, we remark that these techniques alone *cannot* be used to lower even more the complexity of the universality problem for k-URA or for URA. This is because $\mathbf{M}_k \geq k!$, which is the subject of the following lemma.

▶ **Lemma 11.** $\mathbf{M}_k \geq k!$.

Proof. We define a family of pruned universal k-URA $(\mathcal{A}_k)_{k\geq 1}$ over $\Sigma = \{\sigma\}$ such that $M_{\mathcal{A}_k} \geq k!$. Consider the following part of a pruned universal k-URA $\mathcal{A}_k$ (shown for the case $k = 3$):



The rest of the automaton makes sure that the configuration

$$\{\ell(\boldsymbol{u}) \mid \boldsymbol{u} \in \{1, \ldots, k\}^k \text{ is a permutation}\} \cup \{\ell'(1, \ldots, k)\}$$

is reachable in $\mathcal{A}_k$ by the k-letter data word $1 \cdot 2 \cdot \ldots \cdot k$ (*e.g.*, each $\ell(\boldsymbol{u})$ is reached by a path storing the input data in a different order). By taking the (disjoint) union with an unambiguous automaton accepting every data word of length $< k$ and every k letter word that has a repeated data value, we obtain a universal automaton. ◀

Our main tool to prove Lemma 9 is a structural observation, which delivers an understanding of how reachable configurations in universal k-URA can look like. Before diving into it we present an intuition by the following example.

▶ **Example 12.** Let C be a configuration reachable in some universal 2-URA $\mathcal{A}$ over $\Sigma = \{\sigma\}$ by some data word w, and assume that C contains three states $\ell(1,2)$, $\ell(3,4)$ and $\ell(5,6)$ sharing the same location ℓ. We will argue that this is impossible. Assume that from $\ell(1,2)$ the data word $1 \cdot 2 \cdot 7$ is accepted. Then clearly also $3 \cdot 4 \cdot 7$ is accepted from $\ell(3,4)$, and $5 \cdot 6 \cdot 7$ is accepted from $\ell(5,6)$. Let us now consider the data word $8 \cdot 9 \cdot 7$, where 8 and 9 are *fresh* data values, that is, they do not occur in w. Since A is universal, the data word $8 \cdot 9 \cdot 7$ must be accepted from some state $\ell'(d_1, d_2)$ in C. The set $\{d_1, d_2\}$ has only two elements, and so the intersection with at least one of the sets $\{1,2\}$, $\{3,4\}$ and $\{5,6\}$ must be empty. For instance, assume that $\{d_1, d_2\} \cap \{1,2\} = \emptyset$ and $(d_1, d_2) = (3,6)$. Note that $\langle \ell'(3,6), 8 \cdot 9 \cdot 7 \rangle \sim \langle \ell'(3,6), 1 \cdot 2 \cdot 7 \rangle$, so that by Corollary 6, the data word $1 \cdot 2 \cdot 7$ is accepted from state $\ell'(3,6)$, too. But then that there are two accepting runs for $w \cdot 1 \cdot 2 \cdot 7$, a contradiction to the unambiguity of $\mathcal{A}$. Below we generalise this reasoning, in particular to the case where some registers in the reached states keep the same value (*i.e.*, not all are different, as 1, 2, 3, 4, 5 and 6 in the above example). However the intuition stays the same.

We say that a set of tuples $T \subseteq \mathbb{D}_\perp^m$ is *m-full* (or simply *full* if m is clear from the context) if there exists a set of indices $I \subseteq [1,m]$ such that:

- all the tuples in T are identical in indices from I, namely for all $i \in I$ and all $\boldsymbol{t}, \boldsymbol{t'} \in T$ we have $t_i = t'_i$;
- all the data values occurring in tuples in T on indices outside I are different, namely for all $i \notin I$, all $j \in \{1, \ldots, m\}$, and all $\boldsymbol{t}, \boldsymbol{t'} \in T$ we have $t_i \neq t'_j$ unless both $\boldsymbol{t} = \boldsymbol{t'}$ and $i = j$. Note that in particular, this condition applies to the case $\boldsymbol{t'} = \boldsymbol{t}$, and thus $t_i \neq t_j$ whenever $i \notin I$ and $j \in \{1, \ldots, m\}$ are different.

For instance, the set containing the tuples $(1,3,2,4), (1,3,5,6), (1,3,7,8)$ is a 4-full set with $I = \{1,2\}$, and the set $\{(3,7,2,10,8)\}$ is a 5-full set, where $I \subseteq [1,5]$ can be chosen arbitrarily. In contrast, $\{(1,2),(3,1)\}$ is *not* a full set.

For a location ℓ and set of tuples $T \subseteq \mathbb{D}_\perp^k$ we write $\ell(T) = \{\ell(\boldsymbol{t}) \mid \boldsymbol{t} \in T\}$. The following lemma delivers the key observation, which uses the notion of k-full sets.

▶ **Lemma 13.** *If $\mathcal{A}$ is a pruned universal k-URA, then there exists no configuration C reachable in $\mathcal{A}$ such that $\ell(T) \subseteq C$ for some location $\ell \in \mathcal{L}$ and some k-full set of tuples $T \subseteq \mathbb{D}_\perp^k$ of size more than k.*

Proof. Let $\mathcal{A}$ be a pruned universal k-URA, and suppose towards contradiction that there exists a configuration C reachable in $\mathcal{A}$ such that $\ell(T) \subseteq C$ for some location ℓ and some k-full set $T \subseteq \mathbb{D}_\perp^k$ of size more than k. Let w be the data word such that $C = \mathrm{Succ}_{\mathcal{A}}(C_{\mathrm{init}}, w)$. Assume without loss of generality that the indices on which tuples from T are identical are $I = \{1, \ldots, n\}$ for some $n \leq k$. Let us choose some $k+1$ tuples from T, let the i-th of it be of the form $\boldsymbol{t}^i = (c_1, \ldots, c_n, o_1^i, \ldots, o_m^i)$, where $n + m = k$. We call the c_j the *common* data values and the o_j^i the *own* data values of $\boldsymbol{t}^i$. $\mathcal{A}$ is pruned and $\ell(\boldsymbol{t}^1)$ is reachable in $\mathcal{A}$, so there must exist a data word $w_1 \in (\Sigma \times \mathbb{D})^*$ that is accepted from $\ell(\boldsymbol{t}^1)$. Without loss of generality we can assume that w_1 does not contain the own data values of any of the other tuples $\boldsymbol{t}^2, \ldots, \boldsymbol{t}^{k+1}$. Indeed, if this is the case, we can replace synchronously all occurrences of such a data value by a *fresh* data value not occurring in $\mathrm{data}(w)$; the resulting data word is still accepted from $\ell(\boldsymbol{t}^1)$. For every $i \in [2, k+1]$, let w_i be the word w_1 in which for each $j \in [1,m]$ the own data value o_j^1 is replaced by the data value o_j^i. Clearly, for every $i \in [2, k+1]$ $\langle \ell(\boldsymbol{t}^1), w_1 \rangle \sim \langle \ell(\boldsymbol{t}^i), w_i \rangle$, so that by Corollary 6 the data word w_i is accepted from $\ell(\boldsymbol{t}^i)$.

Let us now consider the data word w_{fresh} that is obtained from w_1 by replacing synchronously every occurrence of every o^1_j by some fresh data value each. As $\mathcal{A}$ is universal, also the data word w_{fresh} needs to be accepted from some state in C. Let $q_{\text{fresh}} = \ell'(e_1, \ldots, e_k)$ be the state in C from which w_{fresh} is accepted. Notice that we do not enforce $\ell \neq \ell'$, similarly e_i may be equal to some of the c_i or o^j_i, but this does not have an effect on our reasoning. For each tuple $\boldsymbol{t}^i = (c_1, \ldots, c_n, o^i_1, \ldots, o^i_m)$, let the set of its own data values be $O_i = \{o^i_1, \ldots, o^i_m\}$. By assumption all the sets $O_1, \ldots, O_{k+1}$ are pairwise disjoint. As there are $k+1$ of them, we know that at least one of them is disjoint from the set of data values in the state q_{fresh}, namely with $E = \{e_1, \ldots, e_k\}$. So let $1 \leq i \leq k+1$ be such that $O_i \cap E = \emptyset$. Then we have $\langle q_{\text{fresh}}, w_i\rangle \sim \langle q_{\text{fresh}}, w_{\text{fresh}}\rangle$, so that by Corollary 6 w_i is also accepted from q_{fresh}. In consequence, there are at least two accepting runs over w_i from configuration C, one from $\ell(\boldsymbol{t}^1)$ and one from q_{fresh}. Hence there are at least two initialized accepting runs over $w \cdot w_i$. This is a contradiction to the unambiguity of $\mathcal{A}$. ◀

The following result directly implies Lemma 9, when setting $B = n = k$.

▶ **Lemma 14.** *Every set $T \subseteq \mathbb{D}^n_\perp$ of size at least $(B \cdot 4^n \cdot n!)^n + 1$ contains an n-full subset of size larger than B.*

Proof. Let us denote by $D_{B,n}$ the maximal size of the set of n-tuples such that any n-full subset has size at most B; in other words, $D_{B,n}$ is the least integer such that if X is a set of n-tuples of size $D_{B,n}+1$, then X contains an n-full subset of size $B+1$. Our aim is to show that $D_{B,n} \leq (B \cdot 4^n \cdot n!)^n$. We show it by induction on n.

For the induction base assume $n = 1$. Then any set of data values is a full set, so clearly $D_{B,1} \leq B \leq B \cdot 4^1 \cdot 1!$.

Assume now that $D_{B,m} \leq (B \cdot 4^m \cdot m!)^m$ for all $m < n$ and consider some set $T \subseteq \mathbb{D}^n_\perp$ of n-tuples. Assume that T contains no n-full subset of size larger than B. Pick some tuple $\boldsymbol{t} = (d_1, \ldots, d_n) \in T$. We first show that there can be at most $4^n \cdot n! \cdot D_{B,n-1}$ tuples in T whose data intersect data($\boldsymbol{t}$). Let us denote $N = 4^n \cdot n! \cdot D_{B,n-1}$. Let S be the set of those tuples, assume towards contradiction that the size of S exceeds the bound N. For each tuple $\boldsymbol{s} \in S$ there are at most $2^n - 1$ choices for data($\boldsymbol{s}$) $\cap$ data($\boldsymbol{t}$), so by the pigeonhole principle there are more than $2^n \cdot n! \cdot D_{B,n-1}$ tuples which have the same set data($\boldsymbol{s}$) $\cap$ data($\boldsymbol{t}$). Those data values can occur in tuples from S on at most 2^n different sets of indices, and in at most $n!$ different orders, so by the pigeonhole principle more than $D_{B,n-1}$ tuples from S have the same data values shared with $\boldsymbol{t}$ on the same indices. After ignoring the indices shared with $\boldsymbol{t}$ at most $n-1$ indices remain on these tuples. So by induction assumption there is some full set of size more than B among these tuples, which leads to a contradiction to the assumption that for more than N tuples from T their data intersects data($\boldsymbol{t}$).

Therefore we know that all the tuples but the mentioned N ones have data disjoint with data($\boldsymbol{t}$). Let use denote $\boldsymbol{t^1} = \boldsymbol{t}$ and T_1 to be the set of tuples with data disjoint from data($\boldsymbol{t^1}$). Let $\boldsymbol{t^2} \in T_1$. We now repeat the argument for $\boldsymbol{t^2}$ similarly as for $\boldsymbol{t^1}$ and get that there are at most N tuples with data intersecting data($\boldsymbol{t^2}$). Repeating this argument we get a sequence of tuples $\boldsymbol{t^1}, \boldsymbol{t^2}, \ldots, \boldsymbol{t^m}$ such that for each $i \neq j$ we have data($\boldsymbol{t^i}$) $\cap$ data($\boldsymbol{t^j}$) $= \emptyset$. After adding each tuple $\boldsymbol{t^j}$ to the sequence we define the set T_{j+1} of elements, which have disjoint data with all the tuples $\boldsymbol{t^1}, \ldots, \boldsymbol{t^j}$. As long as T_{j+1} is nonempty we can continue the process. It is easy to see that $|T_{j+1}| \geq |T_j| - N$. Assume now towards contradiction that $D_{B,n} > (B \cdot 4^n \cdot n!)^n$, which implies that $D_{B,n} > (B \cdot 4^n \cdot n!) \cdot D_{B,n-1} = B \cdot N$. We can see now that $|T_B| > 0$, which means that we can construct tuples $\boldsymbol{t^1}, \boldsymbol{t^2}, \ldots, \boldsymbol{t^B}, \boldsymbol{t^{B+1}}$ such that for each $i \neq j$ we have data($\boldsymbol{t^i}$) $\cap$ data($\boldsymbol{t^j}$) $= \emptyset$. This however means that $\{\boldsymbol{t^1}, \ldots, \boldsymbol{t^{B+1}}\}$ is a full set of size $B+1$, which is more than B. This contradicts the assumption, which shows that $D_{B,n} \leq (B \cdot 4^n \cdot n!)^n$ and finishes the proof. ◀

We can now apply a reduction from containment to universality provided by Barloy and Clemente (Lemma 8 in [1]) to obtain Theorem 1 from the introduction.

5 Universality for URA over $(\mathbb{Q}; <, =)$

In this section, we prove Theorem 2 by using the techniques developed in the preceding section. Let us define constants $\mathbf{M}_k^{\mathcal{O}}$ for k-URA with order similarly as $\mathbf{M}_k$ for k-URA. The main technical lemma is the following; Theorem 2 follows.

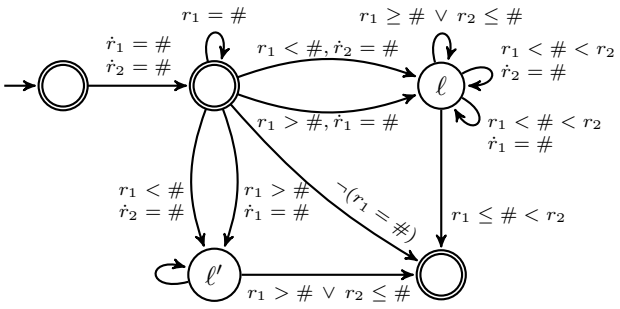
▶ **Lemma 15.** $\mathbf{M}_1^{\mathcal{O}} = 1$.

Proof. Towards contradiction suppose that for some pruned universal 1-URA $\mathcal{A}$ with order there is a configuration C reachable in $\mathcal{A}$ by a data word $w_{\text{pref}} \in (\Sigma \times \mathbb{Q})^*$, such that $\ell(d_1), \ell(d_2) \in C$ for some location ℓ and data values $d_1 < d_2$. Because $\mathcal{A}$ is pruned, there exists a data word $w_1 \in (\Sigma \times \mathbb{Q})^*$ that is accepted from $\ell(d_1)$. Without loss of generality we can assume that w_1 does not contain any data in $(d_1, d_2]$. Indeed, if w_1 contains some datum in $(d_1, d_2]$, then we can replace it synchronously by some datum greater than d_2, while taking care that that the relative order of all data in w_1 is preserved, so that, for the resulting data word w, we have $\langle \ell(d_1), w_1 \rangle \sim \langle \ell(d_1), w \rangle$. By Corollary 6, the resulting data word w is also accepted from $\ell(d_1)$. Notice that for similar reasons, also the data word w_2 obtained from w_1 by replacing every occurrence of d_1 by d_2 is accepted from $\ell(d_2)$. Now, if w_1 does not contain d_1, then $w_1 = w_2$. Hence w_1 is accepted from both $\ell(d_1)$ and $\ell(d_2)$, contradiction to unambiguity of $\mathcal{A}$. So let us assume w_1 contains d_1. Pick some data value d_{fresh} that is fresh, *i.e.*, it does not occur in w_{pref}, and additionally $d_1 < d_{\text{fresh}} < d_2$. We clearly can choose such a fresh data value, as there are infinitely many rational numbers between d_1 and d_2 and only finitely many of them occur in w_{pref}. Let w_{fresh} be the word obtained from w_1 by synchronously replacing every occurrence of d_1 by d_{fresh}. The word w_{fresh} is accepted from some configuration in C, let it be $\ell'(d')$. Notice now that if $d' < d_{\text{fresh}}$, then $\langle \ell'(d'), w_{\text{fresh}} \rangle \sim \langle \ell'(d'), w_2 \rangle$, so that $\ell'(d')$ accepts also w_2 by Corollary 6; in the other case, *i.e.*, if $d' > d_{\text{fresh}}$, then we have $\langle \ell'(d'), w_{\text{fresh}} \rangle \sim \langle \ell'(d'), w_1 \rangle$, so that $\ell'(d')$ also accepts w_1. Therefore in the first case automaton $\mathcal{A}$ has two accepting runs over $w_{\text{pref}} \cdot w_2$ and in the second case over $w_{\text{pref}} \cdot w_1$. This is a contradiction to the unambiguity of $\mathcal{A}$. ◀

The following lemma shows that our techniques by themselves are not sufficient to solve the case of 2-URA with order.

▶ **Lemma 16.** $\mathbf{M}_2^{\mathcal{O}} = \infty$.

Proof. For all $n \geq 1$, consider the configuration $C_n := \{\ell'(1, n)\} \cup \{\ell(1, 2), \ldots, \ell(n-1, n)\}$, which is for all $n \geq 1$ a subset of a configuration reachable in the following pruned universal 2-URA.



The state $\ell'(d_1, d_2)$ keeps track of the first two distinct data read, with $d_1 < d_2$. It is responsible for accepting any datum d outside the interval $[d_1, d_2)$. The state $\ell(x, y)$ is such that $d_1 \leq x < y \leq d_2$ and it is responsible for accepting every datum d' in the interval $[x, y)$. Moreover, if $d' \in (x, y)$, then $\ell(x, y)$ splits into $\ell(x, d')$ and $\ell(d', y)$. The automaton ensures that no two intervals $[x, y), [x', y')$ overlap, and that all the intervals $[x, y)$ present in a configuration cover the interval $[d_1, d_2)$, thus the automaton is unambiguous and universal. ◀

6 Containment for GURA over $(\mathbb{N}; =)$

In this section, we aim to prove the decidability of the universality problem for the more expressive model of GURA. Let us first argue that the techniques developed in Section 4 do not work for GURA.

▶ **Example 17.** One can easily construct a universal 1-GURA with reachable configuration C containing $\ell(0), \ell(1)$, and $\{\ell'\} \times \{n \in \mathbb{N} \mid n \neq 0, 1\}$. If from both ℓ and ℓ' there are outgoing edges with constraint $r = \#$ to some accepting state, then every data word n is accepted from C. In particular, the word 0 is accepted from $\ell(0)$, but we cannot replace 0 by some *fresh* datum to obtain a contradiction as in Example 12.

The example shows that we need more sophisticated methods to solve the universality problem. Moreover, and in contrast to the result for RA, we cannot rely on the reduction from containment to universality by Barloy and Clemente [1], as it holds for RA without guessing only. We hence present a direct proof for containment as stated in Theorem 3. The idea is based on exploring a sufficiently big part of the infinite *synchronized state space* of both automata $\mathcal{A}$ and $\mathcal{B}$, following the approach in [14]. The main difference with [14] lies in the complications that arise due to the fact that a configuration of a GURA may be *infinite*.

6.1 Synchronized Configurations and Bounded Supports

For the rest of this section, let $\mathcal{A} = (\mathcal{R}^{\mathcal{A}}, \mathcal{L}^{\mathcal{A}}, \ell^{\mathcal{A}}_{\text{init}}, \mathcal{L}^{\mathcal{A}}_{\text{acc}}, E^{\mathcal{A}})$ be a GRA with $\mathcal{R}^{A} = \{r_1, \ldots, r_m\}$, and let $\mathcal{B} = (\mathcal{R}^{\mathcal{B}}, \mathcal{L}^{\mathcal{B}}, \ell^{\mathcal{B}}_{\text{init}}, \mathcal{L}^{\mathcal{B}}_{\text{acc}}, E^{\mathcal{B}})$ be a GURA with a single register r.

We aim to reduce the containment problem $L(\mathcal{A}) \subseteq L(\mathcal{B})$ to a reachability problem in $(\mathbb{S}, \Rightarrow)$ where:

- $\mathbb{S}$ is the set of *synchronized configurations* $(\ell(\boldsymbol{d}), C)$, where $\ell(\boldsymbol{d}) \in (\mathcal{L}^{\mathcal{A}} \times \mathbb{N}_{\perp}^{\mathcal{R}^{\mathcal{A}}})$ is a single state of $\mathcal{A}$, and C is a configuration of $\mathcal{B}$,
- $(\ell(\boldsymbol{d}), C) \Rightarrow (\ell'(\boldsymbol{d'}), C')$ if there exists a letter $(\sigma, d) \in (\Sigma \times \mathbb{N})$ such that $\ell(\boldsymbol{d}) \xrightarrow{\sigma, d}_{\mathcal{A}} \ell'(\boldsymbol{d'})$, and $\text{Succ}_{\mathcal{B}}(C, (\sigma, d)) = C'$.

We define $S_{\text{init}} := (\ell^{\mathcal{A}}_{\text{init}}(\boldsymbol{v}_{\text{init}}), C_{\text{init}})$ to be the *initial synchronized configuration of* $\mathcal{A}$ *and* $\mathcal{B}$. We say that a synchronized configuration S' is *reachable* from S if there is a $\Rightarrow$-path from S to S'. S is *reachable* if it is reachable from S_{init}. Call a synchronized configuration $(\ell(\boldsymbol{d}), C)$ *bad* if $\ell \in \mathcal{L}^{\mathcal{A}}_{\text{acc}}$ is an accepting location and C is non-accepting, *i.e.*, $\ell' \notin \mathcal{L}^{\mathcal{B}}_{\text{acc}}$ for all $(\ell', u) \in C$. Thus, a bad synchronized configuration is reachable iff $L(\mathcal{A}) \not\subseteq L(\mathcal{B})$.

We extend the equivalence relation $\sim$ defined in Section 3 to synchronized configurations in a natural manner, *i.e.*, $(\ell(\boldsymbol{d}), C) \sim (\ell(\boldsymbol{d'}), C')$ if there exists a partial isomorphism π of $\mathbb{N}_{\perp}$ such that $\text{data}(\boldsymbol{d}) \cup \text{data}(C) \subseteq \text{dom}(\pi)$ and satisfying $\pi(\boldsymbol{d}) = \boldsymbol{d'}$ and $\pi(C) = C'$. Clearly, an analogue of Corollary 6 holds for this extended relation. In particular, we have the following:

▶ **Proposition 18.** *Let S, S' be two synchronized configurations of $(\mathbb{S}, \Rightarrow)$ such that $S \sim S'$. If S reaches a bad synchronized configuration, so does S'.*

The *support* of a configuration C of $\mathcal{B}$ is the set $\mathrm{supp}(C)$ of data d' such that at least one of the following two conditions holds:

- $\ell(d') \in C$ for some $\ell \in \mathcal{L}$ such that $(\{\ell\} \times \mathbb{D}) \cap C$ is finite,
- $\ell(d') \notin C$ for some $\ell \in \mathcal{L}$ such that $(\{\ell\} \times \mathbb{D}) \cap C$ is cofinite.

Note that $\mathrm{supp}(C) \subseteq \mathrm{data}(w)$ whenever $C = \mathrm{Succ}_{\mathcal{B}}(C_{\mathrm{init}}, w)$.

Let $S = (\ell(\boldsymbol{d}), C)$ be a synchronized configuration, and let $a, b \in \mathrm{supp}(C)$ be two data values in the support of C. We say that *a and b are indistinguishable in S*, written $a \equiv_S b$, if $a, b \notin \mathrm{data}(\boldsymbol{d})$ and $\{\ell \in \mathcal{L} \mid \ell(a) \in C\} = \{\ell \in \mathcal{L} \mid \ell(b) \in C\}$.

Given a configuration C of $\mathcal{B}$, we define for every datum $d \in \mathbb{N}$ the sets $C_d^+ := \{\ell(d) \in \mathcal{L} \times \{d\} \mid \ell(d) \in C \text{ and } \mathrm{data}(C \cap (\{\ell\} \times \mathbb{N})) \text{ is finite}\}$, and $C_d^- := \{\ell(d) \in \mathcal{L} \times \{d\} \mid \ell(d) \notin C \text{ and } \mathrm{data}(C \cap (\{\ell\} \times \mathbb{N})) \text{ is infinite}\}$.

We give here an example for the definition of C_d^+ and C_d^-.

▶ **Example 19.** Let $C = \{\ell_1(0), \ell_1(1)\} \cup \{\ell_2(d) \mid d \in \mathbb{N}\backslash\{1,2\}\} \cup \{\ell_3(d) \mid d \in \mathbb{N}\backslash\{0,1\}\}$. Then

$$\begin{array}{lll} C_0^+ = \{\ell_1(0)\} & C_1^+ = \{\ell_1(1)\} & C_2^+ = \emptyset \\ C_0^- = \{\ell_3(0)\} & C_1^- = \{\ell_2(1), \ell_3(1)\} & C_2^- = \{\ell_2(2)\} \end{array}$$

We say that a configuration C is *essentially coverable* if for every two $\ell(u), \ell'(u') \in C$, the set $\{\ell(u), \ell'(u')\}$ is coverable.

▶ **Proposition 20.** *Let C be an essentially coverable configuration, and let $b \in \mathrm{supp}(C)$. Then $(C\backslash C_b^+) \cup C_b^-$ is essentially coverable, too.*

Proof. Let $\ell(c), \ell'(c') \in ((C\backslash C_b^+) \cup C_b^-)$. If $\ell(c), \ell'(c') \in C\backslash C_b^+$, then $\{\ell(c), \ell'(c')\}$ is coverable by essential coverability of C. Suppose $\ell(c), \ell'(c') \in C_b^-$. By definition of C_b^-, $c = c' = b$. Pick some value $e \in \mathbb{N}\backslash\{b\}$ such that $\ell(e), \ell'(e) \in C$. Note that such a value e must exist, as by definition of C_b^-, the sets $\mathrm{data}((\{\ell\} \times \mathbb{N}) \cap C)$ and $\mathrm{data}((\{\ell'\} \times \mathbb{N}) \cap C)$ are cofinite, and hence their intersection is non-empty. By essential coverability of C, $\{\ell(e), \ell'(e)\}$ is coverable. There must thus exist some data word w such that $\{\ell(e), \ell'(e)\} \subseteq \mathrm{Succ}(\ell_{\mathrm{init}}(\bot), w)$. Let π be any partial isomorphism satisfying $\pi(e) = b$ and whose domain contains $\mathrm{data}(w)$. Clearly, $\{\ell(b), \ell'(b)\} \subseteq \mathrm{Succ}(\ell_{\mathrm{init}}(\bot), \pi(w))$, and hence $\{\ell(b), \ell'(b)\}$ is coverable. Finally, suppose $\ell(c) \in C \setminus C_b^+$ and $\ell'(c') \in C_b^-$. By definition, we have $c \neq b = c'$. Since $\mathrm{data}((\ell' \times \mathbb{N}) \cap C)$ is cofinite, there is $d \neq c$ such that $\ell'(d) \in C$. By essential coverability of C, there exists a data word w such that $\{\ell(c), \ell'(d)\} \subseteq \mathrm{Succ}(\ell_{\mathrm{init}}(\bot), w)$. By picking a partial isomorphism π such that $\pi(d) = c'$ and $\pi(c) = c$, we obtain that $\{\ell(c), \ell'(c')\} \subseteq \mathrm{Succ}(\ell_{\mathrm{init}}(\bot), \pi(w))$, which concludes the proof. ◀

The following is the main technical result of this section.

▶ **Proposition 21.** *Let $S = (\ell^{\mathcal{A}}(\boldsymbol{d}), C)$ be a synchronized configuration of $\mathcal{A}$ and $\mathcal{B}$ such that C is essentially coverable, and let $a \neq b$ be such that $a, b \in \mathrm{supp}(C)$ and $a \equiv_S b$. Then S reaches a bad configuration in $(\mathbb{S}, \Rightarrow)$ if, and only if, $S' := (\ell^{\mathcal{A}}(\boldsymbol{d}), (C \setminus C_b^+) \cup C_b^-)$ reaches a bad configuration in $(\mathbb{S}, \Rightarrow)$.*

Proof. ($\Leftarrow$) Suppose there exists some data word w such that there exists an accepting run of $\mathcal{A}$ on w that starts in $\ell^{\mathcal{A}}(\boldsymbol{d})$, and $\mathrm{Succ}_{\mathcal{B}}(C\backslash C_b^+ \cup C_b^-, w)$ is non-accepting. We assume in the following that $\mathrm{Succ}_{\mathcal{B}}(C_b^+, w)$ is accepting; otherwise we are done. Let $\ell^+(b) \in C_b^+$ be the unique state such that $\mathrm{Succ}_{\mathcal{B}}(\ell^+(b), w)$ is accepting. In the following, we prove that we can without loss of generality assume that w does not contain any a's. Pick some $a' \in \mathbb{N}$ such that $a' \notin \mathrm{data}(w) \cup \mathrm{supp}(C) \cup \mathrm{data}(\boldsymbol{d})$. Let π be the isomorphism defined by $\pi(a) = a'$,

$\pi(a') = a$, and $\pi(d) = d$ for all $d \in \mathbb{N}_\perp \backslash \{a, a'\}$. Then $\pi\langle \ell^{\mathcal{A}}(\boldsymbol{d}), w\rangle = \langle \ell^{\mathcal{A}}(\boldsymbol{d}), \pi(w)\rangle$ (as $a \notin \mathrm{data}(\boldsymbol{d})$ by $a \equiv_S b$), and $\pi\langle \ell^+(b), w\rangle = \langle \ell^+(b), \pi(w)\rangle$. By Corollary 6, there exists an accepting run of $\mathcal{A}$ on $\pi(w)$ that starts in $\ell^{\mathcal{A}}(\boldsymbol{d})$, and $\mathrm{Succ}_{\mathcal{B}}(\ell^+(b), \pi(w))$ is accepting. We prove that $\mathrm{Succ}_{\mathcal{B}}(\ell(c), \pi(w))$ is non-accepting, for every $\ell(c) \in C \setminus \{\ell^+(b)\} \cup C_b^-$: first, let $\ell(c) \in C \setminus \{\ell^+(b)\}$. By essential coverability of C, $\{\ell^+(b), \ell(c)\}$ is coverable. By unambiguity of $\mathcal{B}$, $\mathrm{Succ}_{\mathcal{B}}(\ell(c), \pi(w))$ must be non-accepting. Second, let $\ell(c) \in C_b^-$. But then $c = b$, and hence $\pi\langle \ell(c), w\rangle = \langle \ell(c), \pi(w)\rangle$. By assumption, $\mathrm{Succ}_{\mathcal{B}}(\ell(c), w)$ is non-accepting, so that by Corollary 6, $\mathrm{Succ}_{\mathcal{B}}(\ell(c), \pi(w))$ is non-accepting, too. Note that $\pi(w)$ indeed does not contain any a's. We can hence continue the proof assuming that w does not contain any a's.

Next, we prove that if we replace all b's occurring in w by some fresh datum not occurring in $\mathrm{supp}(C) \cup \mathrm{data}(w) \cup \mathrm{data}(\boldsymbol{d})$, we obtain a data word that guides S to a bad synchronized configuration. Formally, pick some datum $b' \notin \mathrm{data}(w) \cup \mathrm{supp}(C) \cup \mathrm{data}(\boldsymbol{d})$, and let π be the partial isomorphism defined by $\pi(b) = b'$, $\pi(b') = b$, and $\pi(d) = d$ for all $d \in \mathbb{N}_\perp \backslash \{b, b'\}$. Note that $\pi(w)$ does not contain any a's or b's. Clearly, $\pi\langle \ell^{\mathcal{A}}(\boldsymbol{d}), w\rangle = \langle \ell^{\mathcal{A}}(\boldsymbol{d}), \pi(w)\rangle$. By Corollary 6, there still exists an accepting run of $\mathcal{A}$ on $\pi(w)$ that starts in $\ell^{\mathcal{A}}(\boldsymbol{d})$. We prove that $\mathrm{Succ}_{\mathcal{B}}(C, \pi(w))$ is non-accepting. Let $\ell(c) \in C$. We distinguish three cases.

1. Let $c \notin \{b, b'\}$. Then $\pi\langle \ell(c), w\rangle = \langle \ell(c), \pi(w)\rangle$. Since $\mathrm{Succ}_{\mathcal{B}}(\ell(c), w)$ is non-accepting by assumption, so that by Corollary 6 also $\mathrm{Succ}_{\mathcal{B}}(\ell(c), \pi(w))$ is non-accepting.
2. Let $c = b$. By $a \equiv_C b$, the state $\ell(a)$ is in C and $\ell(a), \pi(w) \sim \ell(c), \pi(w)$ since a and c do not appear in w. By essential coverability of C, $\{\ell(a), \ell(c)\} \subseteq C$ is coverable. By unambiguity of $\mathcal{B}$, we obtain that $\mathrm{Succ}_{\mathcal{B}}(\ell(c), \pi(w))$ is non-accepting.
3. Let $c = b'$. Note that $\pi\langle \ell(b), w\rangle = \langle \ell(b'), \pi(w)\rangle$. Recall that $b' \notin \mathrm{supp}(C)$. This implies that $\mathrm{data}(C \cap (\{\ell\} \times \mathbb{N}_\perp))$ is cofinite. We distinguish two cases.
   - $b \in \mathrm{data}(C \cap (\{\ell\} \times \mathbb{N}_\perp))$, i.e., $\ell(b) \in C$. But note that $\ell(b) \notin C_b^+$ by cofiniteness of $\mathrm{data}(C \cap (\{\ell\} \times \mathbb{N}_\perp))$. Hence $\ell(b) \in C \backslash \{\ell^+(b)\}$.
   - $b \notin \mathrm{data}(C \cap (\{\ell\} \times \mathbb{N}_\perp))$, i.e., $\ell(b) \in C_b^-$.

   In both cases, we have proved above that $\mathrm{Succ}(\ell(b), w)$ is non-accepting. By $\pi\langle \ell(b), w\rangle = \langle \ell(b'), \pi(w)\rangle$ and Corollary 6, $\mathrm{Succ}_{\mathcal{B}}(\ell(b'), \pi(w))$ is non-accepting, too.

Altogether we have proved that $\mathrm{Succ}_{\mathcal{B}}(C, \pi(w))$ is non-accepting, while there exists some accepting run of $\mathcal{A}$ on $\pi(w)$ starting in $\ell^{\mathcal{A}}(\boldsymbol{d})$. This concludes the proof for the ($\Leftarrow$)-direction.

($\Rightarrow$) Suppose there exists some data word w such that there exists some accepting run of $\mathcal{A}$ on w starting in $\ell^{\mathcal{A}}(\boldsymbol{d})$, and $\mathrm{Succ}_{\mathcal{B}}(C, w)$ is non-accepting. We assume in the following that $\mathrm{Succ}_{\mathcal{B}}(C \setminus C_b^+ \cup C_b^-, w)$ is accepting; otherwise we are done. Let $\ell^-(b)$ be a state in C_b^- such that $\mathrm{Succ}_{\mathcal{B}}(\ell^-(b), w)$ is accepting. Pick some datum $a' \in \mathbb{N}_\perp$ such that $a' \notin \mathrm{data}(w) \cup \mathrm{supp}(C) \cup \mathrm{data}(\boldsymbol{d})$. Let π be the isomorphism defined by $\pi(b) = a$, $\pi(a) = a'$, $\pi(a') = b$, and $\pi(d) = d$ for all $d \in \mathbb{N} \backslash \{a, b, a'\}$. Clearly, $\pi\langle \ell^{\mathcal{A}}(\boldsymbol{d}), w\rangle = \langle \ell^{\mathcal{A}}(\boldsymbol{d}), \pi(w)\rangle$, so that by Corollary 6, there exists some accepting run of $\mathcal{A}$ on $\pi(w)$ starting in $\ell^{\mathcal{A}}(\boldsymbol{d})$. We prove that $\mathrm{Succ}_{\mathcal{B}}(C \backslash C_b^+ \cup C_b^-, \pi(w))$ is non-accepting. Let $\ell(c) \in C \backslash C_b^+ \cup C_b^-$. We distinguish the following cases:

1. Let $c = a$, i.e., $\ell(a) \in C$. By $a \equiv_S b$, we also have $\ell(b) \in C$. Note that $\pi\langle \ell(b), w\rangle = \langle \ell(a), \pi(w)\rangle$ and that $\ell(b) \neq \ell^-(b)$. By assumption, $\mathrm{Succ}_{\mathcal{B}}(\ell(b), w)$ is non-accepting. By Corollary 6, $\mathrm{Succ}_{\mathcal{B}}(\ell(a), \pi(w))$ is non-accepting, too.
2. Let $c \neq a$. Note that also $\pi\langle \ell^-(b), w\rangle = \langle \ell^-(a), \pi(w)\rangle$. Recall that $\mathrm{Succ}_{\mathcal{B}}(\ell^-(b), w)$ is accepting. By Corollary 6, $\mathrm{Succ}_{\mathcal{B}}(\ell^-(a), \pi(w))$ is accepting. We prove below that $\{\ell^-(a), \ell(c)\}$ is coverable. By unambiguity of $\mathcal{B}$, this directly implies that $\mathrm{Succ}_{\mathcal{B}}(\ell(c), \pi(w))$ is non-accepting.

   Recall that $\{d \in \mathbb{N} \mid \ell^-(d) \in C\}$ is cofinite. Pick some datum $d \in \mathbb{N} \backslash \{c\}$ such that $\ell^-(d) \in C$. We distinguish two cases.

- Assume $\ell(c) \in C \backslash C_b^+$. Since C is essentially coverable, the set $\{\ell^-(d), \ell(c)\}$ is coverable. Hence there must exist some data word u such that $\{\ell^-(d), \ell(c)\} \subseteq \mathrm{Succ}_{\mathcal{B}}(\ell_{\mathrm{init}}(\bot), u)$. Let π' be a partial isomorphism satisfying $\pi'(d) = a$, $\pi'(a) = d$, and $\pi'(e) = e$ for all $e \in \mathrm{data}(u) \cup \{c\}$. Then $\{\ell^-(a), \ell(c)\} \subseteq \mathrm{Succ}_{\mathcal{B}}(\ell_{\mathrm{init}}(\bot), \pi'(u))$, hence $\{\ell^-(a), \ell(c)\}$ is coverable.
- Second suppose $\ell(c) \in C_b^-$, i.e., $c = b$. This implies that $\{e \in \mathbb{N} \mid \ell(e) \in C\}$ is cofinite. Pick some datum $e \in \mathbb{N} \backslash \{d\}$ such that $\ell(e) \in C$. Since C is essentially coverable, the set $\{\ell^-(d), \ell(e)\}$ is coverable. Hence there must exist some data word u such that $\{\ell^-(d), \ell(e)\} \subseteq \mathrm{Succ}_{\mathcal{B}}(\ell_{\mathrm{init}}(\bot), u)$. Let π' be a partial isomorphism satisfying $\pi'(d) = a$, $\pi'(a) = d$, $\pi'(b) = e$, $\pi'(e) = b$, and $\pi'(f) = f$ for all $f \in \mathrm{data}(u)$. Then $\{\ell(b), \ell^-(a)\} \subseteq \mathrm{Succ}_{\mathcal{B}}(\ell_{\mathrm{init}}(\bot), \pi'(u))$, hence $\{\ell(c), \ell^-(a)\}$ is coverable.

Altogether we have proved that $\mathrm{Succ}_{\mathcal{B}}((C \backslash C_b^+) \cup C_b^-, \pi(w))$ is non-accepting, while there is an accepting run of $\mathcal{A}$ on $\pi(w)$ starting in $\ell^{\mathcal{A}}(\boldsymbol{d})$. This finishes the proof for the ($\Rightarrow$)-direction, and thus the proof of the Proposition. ◀

References

1. Corentin Barloy and Lorenzo Clemente. Bidimensional linear recursive sequences and universality of unambiguous register automata. In *Proceedings of STACS'21*, 2021. To appear.
2. Mikolaj Bojanczyk, Bartek Klin, and Joshua Moerman. Orbit-finite-dimensional vector spaces and weighted register automata. In *Proceedings of LICS'21*, 2021. To appear. `arXiv:2104.02438`.
3. Alin Bostan, Arnaud Carayol, Florent Koechlin, and Cyril Nicaud. Weakly-Unambiguous Parikh Automata and Their Link to Holonomic Series. In Artur Czumaj, Anuj Dawar, and Emanuela Merelli, editors, *47th International Colloquium on Automata, Languages, and Programming (ICALP 2020)*, volume 168 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 114:1–114:16, Dagstuhl, Germany, 2020. Schloss Dagstuhl–Leibniz-Zentrum für Informatik. `doi:10.4230/LIPIcs.ICALP.2020.114`.
4. Nicolas Bousquet and Christof Löding. Equivalence and inclusion problem for strongly unambiguous Büchi automata. In *Language and Automata Theory and Applications, 4th International Conference, LATA 2010, Trier, Germany, May 24-28, 2010. Proceedings*, pages 118–129, 2010. `doi:10.1007/978-3-642-13089-2_10`.
5. Thomas Colcombet. Unambiguity in automata theory. In *Proceedings of DCFS 2015*, pages 3–18, 2015.
6. Wojciech Czerwinski, Diego Figueira, and Piotr Hofman. Universality problem for unambiguous VASS. In *Proceedings of CONCUR 2020*, pages 36:1–36:15, 2020.
7. Laure Daviaud, Marcin Jurdzinski, Ranko Lazic, Filip Mazowiecki, Guillermo A. Pérez, and James Worrell. When is containment decidable for probabilistic automata? In *45th International Colloquium on Automata, Languages, and Programming, ICALP 2018, July 9-13, 2018, Prague, Czech Republic*, pages 121:1–121:14, 2018. `doi:10.4230/LIPIcs.ICALP.2018.121`.
8. Stéphane Demri and Ranko Lazic. LTL with the freeze quantifier and register automata. *ACM Trans. Comput. Log.*, 10(3):16:1–16:30, 2009. `doi:10.1145/1507244.1507246`.
9. Stéphane Demri, Ranko Lazic, and Arnaud Sangnier. Model checking freeze LTL over one-counter automata. In *Proceedings of FOSSACS 2008*, pages 490–504, 2008.
10. Diego Figueira, Santiago Figueira, Sylvain Schmitz, and Philippe Schnoebelen. Ackermannian and primitive-recursive bounds with dickson's lemma. In *Proceedings of LICS 2011*, pages 269–278, 2011.
11. Dimitri Isaak and Christof Löding. Efficient inclusion testing for simple classes of unambiguous ω-automata. *Inf. Process. Lett.*, 112(14-15):578–582, 2012. `doi:10.1016/j.ipl.2012.04.010`.

12 Michael Kaminski and Nissim Francez. Finite-memory automata. *Theor. Comput. Sci.*, 134(2):329–363, 1994.

13 Michael Kaminski and Daniel Zeitlin. Finite-memory automata with non-deterministic reassignment. *International Journal of Foundations of Computer Science*, Volume 21, Issue 05, 2010.

14 Antoine Mottet and Karin Quaas. The containment problem for unambiguous register automata. In *Proceedings of STACS 2019*, pages 53:1–53:15, 2019.

15 Frank Neven, Thomas Schwentick, and Victor Vianu. Finite state machines for strings over infinite alphabets. *ACM Trans. Comput. Log.*, 5(3):403–435, 2004.

16 Erik Paul. Finite Sequentiality of Finitely Ambiguous Max-Plus Tree Automata. In Artur Czumaj, Anuj Dawar, and Emanuela Merelli, editors, *47th International Colloquium on Automata, Languages, and Programming (ICALP 2020)*, volume 168 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 137:1–137:15, Dagstuhl, Germany, 2020. Schloss Dagstuhl–Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.ICALP.2020.137.

17 Mikhail Raskin. A superpolynomial lower bound for the size of non-deterministic complement of an unambiguous automaton. In *Proceedings of ICALP 2018*, pages 138:1–138:11, 2018.

18 Richard Edwin Stearns and Harry B. Hunt III. On the equivalence and containment problems for unambiguous regular expressions, regular grammars and finite automata. *SIAM J. Comput.*, 14(3):598–611, 1985.

19 Wen-Guey Tzeng. On path equivalence of nondeterministic finite automata. *Inf. Process. Lett.*, 58(1):43–46, 1996.

The Theory of Concatenation over Finite Models

Dominik D. Freydenberger
Loughborough University, UK

Liat Peterfreund
DI ENS, ENS Paris, CNRS, PSL University, INRIA, France

Abstract

We propose FC, a new logic on words that combines finite model theory with the theory of concatenation – a first-order logic that is based on word equations. Like the theory of concatenation, FC is built around word equations; in contrast to it, its semantics are defined to only allow finite models, by limiting the universe to a word and all its factors. As a consequence of this, FC has many of the desirable properties of FO on finite models, while being far more expressive than FO[<]. Most noteworthy among these desirable properties are sufficient criteria for efficient model checking, and capturing various complexity classes by adding operators for transitive closures or fixed points.

Not only does FC allow us to obtain new insights and techniques for expressive power and efficient evaluation of document spanners, but it also provides a general framework for logic on words that also has potential applications in other areas.

2012 ACM Subject Classification Theory of computation → Database query languages (principles); Theory of computation → Logic and databases; Theory of computation → Finite Model Theory

Keywords and phrases finite model theory, word equations, descriptive complexity, model checking, document spanners

Digital Object Identifier 10.4230/LIPIcs.ICALP.2021.130

Category Track B: Automata, Logic, Semantics, and Theory of Programming

Related Version *Full Version*: https://arxiv.org/abs/1912.06110

Funding *Dominik D. Freydenberger*: Supported by EPSRC grant EP/T033762/1.
Liat Peterfreund: Supported by Fondation des Sciences Mathématiques de Paris (FSMP). A part of this work was done while affiliated with IRIF, CNRS, Université de Paris and with Edinburgh University.

Acknowledgements The authors thank Joel D. Day, Manfred Kufleitner, Leonid Libkin, Sam M. Thompson, and the reviewers of the current and previous versions for their helpful comments.

1 Introduction

This paper proposes a finite model version of the theory of concatenation: FC, a new logic that is designed to describe properties of words and to query them. While the idea of using logic on words is by no means new, the advantage of FC is its combination of expressive power and tractable model checking and evaluation.

Logic on words. A common way of using logic on words is *monadic second-order logic* (MSO) over a linear order (e. g. [53]). That is, a word w is seen as a sequence of positions, and predicates $\mathsf{P}_a(x)$ express "letter a at position x of w". This approach comes with two disadvantages for querying. The first is that factors (continuous subwords) cannot be expressed directly. Consider the query "return all factors of w". As variables refer to positions, the query would not return a factor u directly, but represent it as a set (or tuple) of positions that describe a specific occurrence of u in w. If u occurs more than once, the query result would contain each occurrence – unless the logic is powerful enough to prevent this.

This leads us to the second disadvantage, namely that MSO cannot compare factors of unbounded length. That is, while MSO can express queries like "return the positions of factors of length k that occur twice in w" for a fixed length k, it is impossible to express "return the positions of factors that occur twice in w"; or non-regular languages, like that of all words ww with $w \in \{\mathtt{a}, \mathtt{b}\}^*$. As a result, many natural relations on factors of w are inexpressible in MSO, in particular the concatenation $x = yz$.

Another approach to logic on words is the *theory of concatenation* (short: C). First defined by Quine [48], this logic is built on *word equations*, that is, equations of the form $xx \doteq yyy$, where variables like x and y stand for words over a finite alphabet Σ. While less prominent than FO or MSO, the theory of concatenation has been studied extensively since the 1970s, with particular emphasis on word equations. A fairly recent survey on the satisfiability of word equations is [14]. More current research on word equations and the theory of concatenation can be found in e. g. [7, 10, 11, 12, 44, 50].

In contrast to MSO, the logic C allows us to treat words as words (instead of intervals of positions) and a position in w can be expressed as the corresponding prefix of w. More importantly, C can express properties like "u is a factor of v" or concatenation like $x = yz$. This expressive power comes at a price – even limited use of negation leads to an undecidable theory (i. e., satisfiability is undecidable, see [17, 48]). Contrast this to *first-order logic* (FO) over finite models: By Trakhtenbrot's theorem, satisfiability is undecidable; but the model checking problem is not just decidable, but can even be made tractable (see e. g. [18, 36]).

Another situation where using queries for words together with an open universe causes problems occurs in *string databases*, see [4, 5, 27, 28]. These query languages treat words as entries of the database instead of operating on a single word. Furthermore, they offer transformation operations that assume an infinite universe. As a result, these query language usually express Turing-complete functions from words to words.

Introducing FC. The new logic FC aims to bring the advantages of FO on finite models to the theory of concatenation. The universe for C is usually assumed to be Σ^*, which means that there is no meaningful distinction between satisfiability and model checking. The key idea of FC is changing universe from Σ^* to the set of all factors of a word w; comparable to how the universe for MSO consists of all positions of a word w.

As FC-formulas are based on word equations, concatenation is straightforward to use. For example, "return all factors that occur twice in w" can be expressed as

$$\varphi_1(x) := \exists p_1, p_2, s_1, s_2 \colon \big(\mathfrak{u} \doteq p_1\, x\, s_1 \wedge \mathfrak{u} \doteq p_2\, x\, s_2 \wedge \neg p_1 \doteq p_2\big),$$

where $\mathfrak{u}$ represents w. In detail, φ_1 expresses that there are two different ways of decomposing w into $w = p\, x\, s$. If we also wanted to know the positions of these occurrences, we could return p_1 and p_2 (by removing their quantifiers), as these encode the start of each occurrence in w. This formula does not rely on the requirement that variables can only be mapped to factors of w, as the $\mathfrak{u}$ on the left side of the equations ensures this already. Instead, consider

$$\varphi_2(x) := \exists y, z \colon y \doteq x\, z\, x,$$

which returns all factors x that have two non-overlapping occurrences in w. As we not need to know where in w the factor xzx occurs, $\mathfrak{u}$ is not needed in the formula.

The restriction to a finite universe allows us to translate various classical results from FO to FC. Most importantly, model checking becomes not only decidable, but upper bounds can be lowered in the same way as for FO (Section 4.1). In fact, FC can be extended with iteration operators to characterize complexity classes from L to PSPACE, analogously to FO

on ordered structures; which allows us to define a version of Datalog on words that includes concatenation (Section 4.3). Furthermore, Section 5 also describes how FC can be extended with *constraints* (aka *predicates*), while still keeping model checking tractable.

Spanners. An immediate application of FC is as a logic for *document spanners* (or just *spanners*). Spanners are a rule-based framework for information extraction that was proposed by Fagin, Kimelfeld, Reiss, and Vansummeren [19] to study the formal properties of the query language AQL of IBM's SystemT for information extraction. They can be understood as a combination of regular expressions and relational algebra.

In the last years, spanners have received considerable attention in the database theory community. The two main areas of interest are expressive power [19, 22, 23, 41, 43, 45, 47, 52] and efficient evaluation [3, 21, 24, 41, 42, 45, 46, 52]; further topics include updates [3, 25, 37], cleaning [20], distributed query planning [15], and a weighted variant [16].

But most of these articles do not focus on the full class of spanners that was introduced by Fagin et al. (called *core spanners*, as they describe the core of AQL), but a much smaller subclass, the *regular spanners*. The difference between these is that regular spanners cannot express equality of factors. Hence, techniques for finite automata and MSO often work on regular spanners; but they rarely work for core spanners. Furthermore, although spanners are conceptually similar to relational algebra, many canonical approaches for relational databases and the underlying FO are not viable in the spanner setting. In particular, while *acyclic conjuctive queries* are well-known to be tractable for FO (see e. g. [1]), this does not hold for the corresponding class of spanners (see [24]).

Although "pure" FC is not powerful enough to express core spanners, extending it with constraints that decide regular languages results in a logic that captures core spanners (Section 5.2). In addition to providing us with a rich and natural class of tractable spanners, this connection also allows us to develop a new inexpressibility method (Section 5.3).

2 Preliminaries

Let ε denote the *empty word*. We use $|x|$ for the length of a word, a formula, or a regular expression x, or the number of elements of a finite set x. A word v is a *factor* of a word w, written $v \sqsubseteq w$, if there exists (possibly empty) words p, s with $w = pvs$. For words x and y, let $x \sqsubseteq_{\mathsf{p}} y$ (x is a *prefix* of y) if $y = xs$ for some s, and $x \sqsubset_{\mathsf{p}} y$ if $x \sqsubseteq_{\mathsf{p}} y$ and $x \neq y$.

For alphabets A, B, a *morphism* is a function $h\colon A^* \to B^*$ with $h(u \cdot v) = h(u) \cdot h(v)$ for all $u, v \in A^*$. To define h, it suffices to define $h(a)$ for all $a \in A$. Let Σ be a finite *terminal alphabet*, and let Ξ be an infinite *variable alphabet* with $\Sigma \cap \Xi = \emptyset$. We assume Σ is fixed and $|\Sigma| \geq 2$, unless stated otherwise. As a convention, we use typewriter letters (like $\mathtt{a}$ and $\mathtt{b}$) for terminals.

Patterns and the theory of concatenation. A *pattern* is a word from $(\Sigma \cup \Xi)^*$. For every pattern $\eta \in (\Sigma \cup \Xi)^*$, let $\mathsf{Var}(\eta)$ denote the set of variables that occur in η. A *pattern substitution* (or just *substitution*) is a partial morphism $\sigma\colon (\Sigma \cup \Xi)^* \to \Sigma^*$ with $\sigma(\mathtt{a}) = \mathtt{a}$ for all $\mathtt{a} \in \Sigma$. When applying a substitution σ to a pattern η, we assume σ is defined on $\mathsf{Var}(\eta)$, that is, $\mathsf{Dom}(\sigma) \supseteq \mathsf{Var}(\eta)$. A *word equation* is a pair of patterns, that is, a pair (η_L, η_R) with $\eta_L, \eta_R \in (\Sigma \cup \Xi)^*$. We also write $\eta_L \doteq \eta_R$, and call η_L and η_R the *left side* and the *right side* of the equation. A *solution* of $\eta_L \doteq \eta_R$ is a substitution σ with $\sigma(\eta_L) = \sigma(\eta_R)$.

The theory of concatenation combines word equations with first-order logic. First the syntax: The set C of formulas of the *theory of concatenation* uses word equations $(\eta_L \doteq \eta_R)$ with $\eta_L, \eta_R \in (\Sigma \cup \Xi)^*$ as atoms. The connectives are conjunction, disjunction, negation, and quantifiers with variables from Ξ. For every $\varphi \in \mathsf{C}$, we define its set of *free variables* $\mathsf{free}(\varphi)$ by $\mathsf{free}(\eta_L \doteq \eta_R) := \mathsf{Var}(\eta_L) \cup \mathsf{Var}(\eta_R)$; extending this canonically.

The semantics build on solutions of word equations: For all $\varphi \in \mathsf{C}$ and all pattern substitutions σ with $\mathsf{Dom}(\sigma) \supseteq \mathsf{free}(\varphi)$, we define $\sigma \models \varphi$ as follows: Let $\sigma \models (\eta_L \doteq \eta_R)$ if $\sigma(\eta_L) = \sigma(\eta_R)$. For the quantifiers, we say $\sigma \models \exists x\colon \varphi$ (or $\sigma \models \forall x\colon \varphi$) if $\sigma_{x \mapsto w} \models \varphi$ holds for an (or all) $w \in \Sigma^*$, where $\sigma_{x \mapsto w}$ is defined by $\sigma_{x \mapsto w}(x) := w$ and $\sigma_{x \mapsto w}(y) := \sigma(y)$ for all $y \in (\Sigma \cup \Xi) - \{x\}$. The connectives' semantics are defined canonically.

▶ **Example 2.1.** Let $\varphi := x\mathtt{abc}y \doteq y\mathtt{bca}x \wedge \neg(x \doteq \varepsilon \vee y \doteq \varepsilon)$. Then $\sigma \models \varphi$ if and only if $\sigma(x\mathtt{abc}y) = \sigma(y\mathtt{bca}x)$, $\sigma(x) \neq \varepsilon$, and $\sigma(y) \neq \varepsilon$. For example, if $\sigma(x) = \mathtt{abca}$ and $\sigma(y) = \mathtt{a}$.

We freely add and omit parentheses as long as the meaning stays clear. E-C, the *existential fragment* of C, consists of those formulas that do not use universal quantifiers and that apply negation only to word equations. The *existential-positive fragment* EP-C allows neither universal quantifiers, nor negation. We also use this notation for other logics that we define.

3 Finite models in the theory of concatenation

The new logic *finite model version of the theory of concatenation*, namely FC, is built around word equations; similarly to the theory of concatenation C. The latter can be understood as first-order logic over the universe Σ^* with concatenation – see for example [29], which refers to C as $\mathsf{FO}(A^*, \cdot)$. In other words, for C, we can consider the universe to be fixed (for a given terminal alphabet Σ). The key idea of FC is to replace the universe Σ^* with a single word and all its factors. In the formulas, this word is represented by a distinguished variable:

▶ **Definition 3.1.** *We distinguish a variable* $\mathfrak{u} \in \Xi$ *and call it the* universe variable.

As the universe variable represents the universe (hence its name), it has a special role in both syntax and semantics of FC. The *syntax of FC* restricts the syntax of C in two ways:

▶ **Definition 3.2.** *The set* FC *of* FC-formulas *is defined recursively: The atoms are word equations* $(\eta_L \doteq \eta_R)$ *with* $\eta_L \in \Xi$ *and* $\eta_R \in (\Sigma \cup \Xi)^*$*. These can be combined using disjunction* $(\varphi \vee \psi)$*, conjunction* $(\varphi \wedge \psi)$*, negation* $\neg\varphi$*, and quantifiers* $\exists x\colon \varphi$ *and* $\forall x\colon \varphi$ *with* $x \in \Xi - \{\mathfrak{u}\}$.

In other words, firstly, every word equation has a single variable on its left side. Secondly, the universe variable $\mathfrak{u}$ may not be bound by quantifiers. The reason for the first restriction is a bit subtle; we shall discuss it after defining the semantics. But the other follows immediately from the intuition that $\mathfrak{u}$ shall represent the universe – hence, binding it would make no sense. For the same reason, we also exclude $\mathfrak{u}$ from the free variables of FC-formulas:

▶ **Definition 3.3.** *The set* $\mathsf{free}(\varphi)$ *of free variables of an* FC*-formula* φ *is defined as for* C*-formulas, with the exception that* $\mathfrak{u}$ *is not considered a free variable.*

The *semantics of FC* combine those of C with the additional condition that the universe consists only of factors of the content of the universe variable $\mathfrak{u}$:

▶ **Definition 3.4.** *For* $\varphi \in \mathsf{FC}$ *and a pattern substitution* σ *with* $\mathsf{Dom}(\sigma) \supseteq \mathsf{free}(\varphi) \cup \{\mathfrak{u}\}$*, we define* $\sigma \models \varphi$ *as for* C*, but with the additional condition that* $\sigma(x) \sqsubseteq \sigma(\mathfrak{u})$ *for all* $x \in \mathsf{Dom}(\sigma)$.

To highlight the special role of $\mathfrak{u}$, we also write $(w, \sigma) \models \varphi$ if $\sigma \models \varphi$ and $w = \sigma(\mathfrak{u})$. We may shorten this to $w \models \varphi$ if φ is a *sentence* – that is, if $\mathsf{free}(\varphi) = \emptyset$. We write $\varphi(\vec{x})$ to denote that $\vec{x}$ is a tuple of free variables of φ.

▶ **Example 3.5.** Define $\varphi_1(y) := \exists x\colon x \doteq \mathtt{papaya}\, y\, \mathtt{banana}$ and $\varphi_2 := \exists x\colon (x \doteq \mathtt{papaya} \vee x \doteq \mathtt{banana})$. Then $(w, \sigma) \models \varphi_1$ if and only if $\sigma(y)$ occurs in w between $\mathtt{papaya}$ and $\mathtt{banana}$, and $w \models \varphi_2$ if and only if w contains $\mathtt{papaya}$ or $\mathtt{banana}$ as factor. Finally, let $\varphi_3(x) := \exists p, s\colon \big(\mathfrak{u} \doteq p\, x\, s \wedge \neg \exists \hat{p}, \hat{s}\colon (\mathfrak{u} \doteq \hat{p}\, x\, \hat{s} \wedge \neg \hat{p} \doteq p)\big)$. Then $(w, \sigma) \models \varphi_3$ if and only if $\sigma(x)$ occurs exactly once in w.

When applying σ to an FC-formula, $\sigma(\mathfrak{u})$ always needs to be defined – otherwise, we would have no universe to work with. But FC-formulas do not need to contain $\mathfrak{u}$. As a rule of thumb, $\mathfrak{u}$ is only required when referring to some "global" property of w. If we describe properties that are more "local" (as in the next example), we usually do not need to use $\mathfrak{u}$.

▶ **Example 3.6.** Let $\varphi^{\sqsubseteq_\mathsf{p}}(x, y) := \exists z\colon y \doteq xz$. Then $\sigma \models \varphi$ if and only if $\sigma(x)$ and $\sigma(y)$ are factors of $\sigma(\mathfrak{u})$ with $\sigma(x) \sqsubseteq_\mathsf{p} \sigma(y)$. In other words, $\varphi^{\sqsubseteq_\mathsf{p}}$ expresses $x \sqsubseteq_\mathsf{p} y$. Consequently, we can express $x \sqsubset_\mathsf{p} y$ through $\varphi^{\sqsubset_\mathsf{p}}(x, y) := \varphi^{\sqsubseteq_\mathsf{p}}(x, y) \wedge \neg x \doteq y$.

In fact, $\sqsubset_\mathsf{p}$ and inequality can be expressed without negation (or universal quantifiers). First, define $\varphi(x)^{\neq\varepsilon} := \exists y\colon \bigvee_{\mathtt{a}\in\Sigma} x \doteq \mathtt{a}\, y$ to express $x \neq \varepsilon$ – that is, $\sigma \models \varphi^{\neq\varepsilon}$ if and only if $\sigma(x) \sqsubseteq \sigma(\mathfrak{u})$ and $\sigma(x) \neq \varepsilon$. We use this in $\psi^{\sqsubset_\mathsf{p}}(x, y) := \exists z\colon (y \doteq x\, z \wedge \varphi^{\neq\varepsilon}(z))$. Like $\varphi^{\sqsubset_\mathsf{p}}$, this expresses $x \sqsubset_\mathsf{p} y$; but without negation.

Finally, let $\varphi^{\neq}(x, y) := \psi^{\sqsubset_\mathsf{p}}(x, y) \vee \psi^{\sqsubset_\mathsf{p}}(y, x) \vee \bigvee_{\mathtt{a},\mathtt{b}\in\Sigma,\mathtt{a}\neq\mathtt{b}} \exists x_1, y_1, z\colon (x \doteq z\, \mathtt{a}\, x_1 \wedge y \doteq z\, \mathtt{b}\, y_1)$. This states that $x \sqsubset_\mathsf{p} y$, $y \sqsubset_\mathsf{p} x$, or x and y differ after a common prefix z – that is, $x \neq y$.

We say $\varphi, \psi \in \mathsf{FC}$ are *equivalent*, written $\varphi \equiv \psi$, if for all σ with $\mathsf{Dom}(\sigma) \supseteq \mathsf{free}(\varphi) \cup \mathsf{free}(\psi) \cup \{\mathfrak{u}\}$, we have that $\sigma \models \varphi$ holds if and only if $\sigma \models \psi$. Thus, in Example 3.6, we have $\varphi^{\sqsubset_\mathsf{p}} \equiv \psi^{\sqsubset_\mathsf{p}}$. If $\varphi \in \mathsf{FC}$ is a sentence, we define its language as $\mathcal{L}(\varphi) := \{w \mid w \models \varphi\}$.

▶ **Example 3.7.** A language is called *star-free* if it is defined by a regular expression α that is constructed from the empty set $\emptyset$, terminals $\mathtt{a} \in \Sigma$, concatenation $\cdot$, union $\cup$, and complement $\overline{\alpha}$. Given such an α, we define $\varphi^\alpha := \exists x\colon (\mathfrak{u} \doteq x \wedge \psi^\alpha(x))$, where $\psi^\alpha(x)$ is defined recursively by $\psi^\emptyset(x) := \neg(x \doteq x)$, $\psi^\mathtt{a}(x) := (x \doteq \mathtt{a})$, $\psi^{(\alpha_1 \cdot \alpha_2)}(x) := \exists x_1, x_2\colon \big(x \doteq x_1\, x_2 \wedge \psi^{\alpha_1}(x_1) \wedge \psi^{\alpha_2}(x_2)\big)$, $\psi^{(\alpha_1 \cup \alpha_2)}(x) := \psi^{\alpha_1}(x) \vee \psi^{\alpha_1}(x)$, and $\psi^{\overline{\alpha}}(x) := \neg\psi^\alpha(x)$. Then $\sigma \models \psi^\alpha$ if and only if $\sigma(x) \in \mathcal{L}(\alpha)$ and $\sigma(x) \sqsubseteq \sigma(\mathfrak{u})$. Thus, $\mathcal{L}(\varphi^\alpha) = \mathcal{L}(\alpha)$.

We are now ready to discuss why Definition 3.2 restricts the left sides of word equations to single variables. Assume we allowed, for instance, the word equation $xy \doteq yx$ in an FC-formula, and consider the case of $\sigma(\mathfrak{u}) = \mathtt{a}^3$ and $\sigma(x) = \sigma(y) = \mathtt{a}^2$. Then $\sigma(x) \sqsubseteq \sigma(\mathfrak{u})$, $\sigma(y) \sqsubseteq \sigma(\mathfrak{u})$, and $\sigma(xy) = \sigma(yx)$ hold, but $\sigma(xy) = \mathtt{a}^4$ is not a factor of $\sigma(\mathfrak{u})$, which means that it is not in the universe.

There are two straightforward ways of allowing arbitrary word equations $\eta_L \doteq \eta_R$ in FC without changing the underlying universe. The first is adding the additional requirements $\sigma(\eta_L) \sqsubseteq \sigma(\mathfrak{u})$ and $\sigma(\eta_R) \sqsubseteq \sigma(\mathfrak{u})$ to the definition of $\sigma \models (\eta_L \doteq \eta_R)$. This can also be understood as declaring the concatenation as undefined if its result is not a factor of $\sigma(\mathfrak{u})$. The second is interpreting $\eta_L \doteq \eta_R$ as syntactic sugar for $\exists x\colon (x \doteq \eta_L \wedge x \doteq \eta_R)$, where x is a new variable.

On the other hand, this re-interpretation of the solutions of word equations can be considered non-intuitive, which makes formulas that rely on these easy to misunderstand. To avoid these issues, this paper restricts every left side to a single variables, even though this is not strictly necessary.

4 Properties of FC

In this section we analyze FC dynamically by discussing its *evaluation problem* – given a formula $\varphi \in \mathsf{FC}$ and a pattern substitution σ, decide whether $\sigma \models \varphi$. We call the special case where φ is a sentence the *model checking problem*. We also consider the *satisfiability problem* – given $\varphi \in \mathsf{FC}$, decide whether there is a pattern substitution σ with $\sigma \models \varphi$. After that, we also consider aspects of optimization of formulas.

4.1 Model checking vs satisfiability

For C, the model checking problem is undecidable. This is due to two reasons. Firstly, the satisfiability problem for C is undecidable by Quine [48]. Secondly, for C, satisfiability reduces to model checking – from a given $\varphi \in \mathsf{C}$, we can construct a C-sentence φ' by binding all free variables of φ existentially. Then φ is satisfiable if and only if $\sigma \models \varphi'$, no matter which substitution σ we choose. In contrast to this, the finite universe of FC drastically reduces the complexity of model checking.

▶ **Theorem 4.1.** *Evaluation is* PSPACE*-complete for* FC *and* NP*-complete for* EP-FC*.*

In fact, the proof shows that the lower bounds hold even in the special case of model checking $\mathsf{a} \models \varphi$. Both the drop in complexity and the fact a very simple structure suffice are comparable to FO on finite relations (see e. g. [36]), and the proofs are equally straightforward. For both logics, the hardness of the problem comes from parameters of the formula and not of the word or the relational structure. FO provides us with another parameter to lower the complexity of model checking. We define the *width* $\mathsf{wd}(\varphi)$ of a formula φ as the maximum number of free variables in any of its subformulas.

▶ **Theorem 4.2.** *Model checking for* FC *can be solved in time* $O(k|\varphi|n^{2k})$*, for* $k := \mathsf{wd}(\varphi)$ *and* $n := |\sigma(\mathfrak{u})|$*.*

The proof also shows that this is only a rough upper bound; taking properties of variables into account lowers the exponent. In principle, we can apply various structure parameters for first-order formulas (see e. g. Adler and Weyer [2]) to FC. This assumes that we treat word equations as atomic formulas, which is certainly possible – but we can do better than that.

Decomposing patterns. Using a word equation $x \doteq \alpha$ as an atom results in a formula that has a width of at least $|\mathsf{Var}(\alpha)|$. Our goal is to lower that bound, by decomposing the pattern into a formula. Technically, a pattern $\alpha = \alpha_1 \cdots \alpha_n$ with $\alpha_i \in (\Xi \cup \Sigma)$ is a term $f(\alpha_1, \ldots, \alpha_n)$, where the function f is the n-ary concatenation. But there is a syntactic criterion that allows us to decompose α into a conjunction of binary concatenations. This builds on a result from combinatorics on words and formal languages, where a pattern α is also treated as generators of the *pattern languages* $\mathcal{L}(\alpha)$; the set of images of α under pattern substitutions. In this context, Reidenbach and Schmid [49] started a series of articles on classes of *pattern languages* with a polynomial time membership problem (surveyed in [40]), most of which rely on the following definition (see [8] for the definition of treewidth).

▶ **Definition 4.3.** *The* standard graph *of a pattern* $\alpha = \alpha_1 \cdots \alpha_n$ *with* $n \geq 1$ *and* $\alpha_i \in (\Sigma \cup \Xi)$ *is* $\mathcal{G}_\alpha := (V_\alpha, E_\alpha)$ *with* $V_\alpha := \{1, \ldots, n\}$ *and* $E_\alpha := E_\alpha^{<} \cup E_\alpha^{=}$*, where* $E_\alpha^{<}$ *is the set of all* $\{i, i+1\}$ *with* $1 \leq i < n$*, and* $E_\alpha^{=}$ *is the set of all* $\{i, j\}$ *such that* α_i *is some* $x \in \Xi$*, and* α_j *is the next occurrence of* x *in* α*. Then* $\mathsf{tw}(\alpha)$*, the* treewidth *of* α*, is the treewidth of* $\mathcal{G}_\alpha$*.*

As artificial (but simple) example, consider the sequence of patterns $\alpha_n := x_1x_1x_2x_2 \cdots x_nx_n$. Then $|\mathsf{Var}(\alpha_n)| = n$, affecting the width of formulas that use α_n accordingly, but $\mathsf{tw}(\alpha_n) = 1$. Using tree decompositions, we can rewrite patterns of bounded treewidth into formulas of bounded width (similar to the proof of Kolaitis and Vardi [34] for variable bounded FO).

▶ **Theorem 4.4.** *Let $\varphi := \exists x_1, \dots, x_m \colon y \doteq \alpha$. Then there exists $\psi \in$ EP-FC with $\psi \equiv \varphi$ and $\mathsf{wd}(\psi) \leq 2\mathsf{tw}(\alpha) + v$, where $v = 2 + |\mathsf{free}(y \doteq \alpha) - \{x_1, \dots, x_m\}|$.*

For every fixed k, given φ with $\mathsf{tw}(\alpha) \leq k$, we can compute ψ in polynomial time.

Combining Theorems 4.2 and 4.4 yields a (slightly) different proof of the polynomial time decidability of the membership problem for classes of patterns with bounded treewidth from [49]. As pointed out in [9], bounded treewidth does not cover all pattern languages with a polynomial time membership problem, like e. g. patterns α^k where $\mathsf{tw}(\alpha)$ is bounded. But these languages can be expressed by $\exists x \colon (\mathfrak{u} \doteq x^k \wedge \varphi_\alpha(x))$, where $\varphi_\alpha(x)$ is a formula that expresses $x \in \mathcal{L}(\alpha)$, thus increasing the width by one. We leave a systematic examination whether all criteria for patterns with a tractable membership problem map to FC-formulas of bounded width for future work.

Satisfiability. Another parallel to FO is that satisfiability is undecidable for FC, even if we use only few variables. Let FC^k denote the set of formulas with width at most k.

▶ **Proposition 4.5.** *Satisfiability for FC^3 is undecidable if $|\Sigma| \geq 2$.*

The problem is trivial for FC^0 (see the proof of Theorem 4.8) and open for FC^1 and FC^2.

4.2 Static optimization

Apart from the width, Theorem 4.2 highlights the length of a formula as another parameter that influences the complexity of model checking. While the length of the patterns in the word equations might not seem to be factor that is overly important, there are patterns where straightforward optimizations can lead to an exponential advantage.

▶ **Example 4.6.** For $k \geq 1$, let $\varphi_k(y) := \exists x \colon y \doteq x^{2^k}$. Then $\varphi_k \equiv \psi_k := \exists x_1, \dots, x_k \colon \big(y \doteq x_1x_1 \wedge \bigwedge_{i=1}^{k-1} x_i \doteq x_{i+1}x_{i+1}\big)$, and $|\varphi_k|$ is exponential in k, while $|\psi_k|$ is linear in k. We can also rewrite ψ_k into a formula of width 3 by pulling quantifiers inwards and reusing variables. More specifically, we first rewrite each ψ_k into the equivalent formula

$$\exists x_1 \colon (y \doteq x_1x_1 \wedge (\exists x_2 \colon x_1 \doteq x_2x_2 \wedge \cdots (\exists x_{k-1} \colon x_k \doteq x_{k-1}x_{k-1}) \cdots)).$$

Then we replace every variable x_i with x_1 if i is odd or x_2 if i is even. The resulting formula has width 3 (due to y), is equivalent to φ_k, and has the same length.

This raises the questions whether we can computably minimize formulas and whether some fragments are more succinct than others. We address these questions in order.

▶ **Theorem 4.7.** *There is no algorithm that, given $\varphi \in$ FC, computes an equivalent ψ such that $|\psi|$ is minimal. This holds even if we restrict this to minimization within $\mathsf{EP\text{-}FC}^4$.*

This leaves open the decidability of, given $\varphi \in$ FC (or $\varphi \in$ EP-FC) and $k > 0$, is there an equivalent $\psi \in \mathsf{FC}^k$. But without suitable inexpressibility methods (see Section 5.3), we cannot even show that a language is inexpressible in FC^k for some $k > 0$, which complicates tackling this problem. The proof of Theorem 4.7 is actually more general and also demonstrates the undecidability of other common problems, like containment and equivalence.

Via Hartmanis' [30] meta theorem, certain undecidability results provide insights into the relative succinctness of models (see [35] or e. g. [23] for details). For two logics $\mathcal{F}_1$ and $\mathcal{F}_2$, the *tradeoff* from $\mathcal{F}_1$ to $\mathcal{F}_2$ is *non-recursive* if, for every computable $f\colon \mathbb{N} \to \mathbb{N}$, there exists some $\varphi \in \mathcal{F}_1$ that is expressible in $\mathcal{F}_2$, but $|\psi| \geq f(|\varphi|)$ holds for every $\psi \in \mathcal{F}_2$ with $\psi \equiv \varphi$.

▶ **Theorem 4.8.** *There are non-recursive tradeoffs from* EP-FC^4 *to regular expressions and* FC^0*; and from* FC^4 *to* EP-FC*, patterns, and singleton sets* $\{w\}$*.*

Note in particular that patterns can be parts of word equations. Hence, where Example 4.6 showed an exponential advantage in the rewriting, Theorem 4.8 shows FC^4 can obtain far larger advantages on certain classes of patterns.

4.3 Iteration and recursion

Iteration and recursion have been extensively studied in finite model theory and database theory. In particular, FO[<] that is extended with operators for transitive closure or fixed points captures various complexity classes (see e. g. [18, 36]). This is also closely connected to the recursive query language Datalog (see e. g. [1]). In this section, we shall define FC-Datalog, an FC-analog of Datalog. On the way, we shall also see that using transitive closures or fixed points on FC instead of FO[<] characterizes the same complexity classes.

Iteration. The definitions of these operators and the resulting extensions of FC are straightforward adaptions of their FO[<]-versions (see e. g. [36] or [18]). But as they are also rather lengthy, we only give the intuitions behind them (detailed definitions can be found in the full version of this paper).

For $k \geq 1$, an FC-formula with $2k$ free variables can be viewed as generator of a relation over $R \subseteq S^k \times S^k$, where S is the universe (the set of factors of $\mathfrak{u}$). The operator tc computes the *transitive closure* $\mathsf{tc}(R)$ of R. If we view R as edge set of a directed graph over S^k, then tc computes the reachability relation in this graph. The *deterministic transitive closure* dtc is defined analogously, with the additional restriction that dtc stops at nodes with more than one outgoing edge. FC$^{\mathsf{tc}}$ and FC$^{\mathsf{dtc}}$ extend FC with tc and dtc, respectively.

For fixed points, we introduce special relation symbols as part of inductive definitions. Inside a fixed point operator, a formula φ may use a symbol $\dot{R}$ and at the same time also define the relation R inductively. We start with $R_0 := \emptyset$ and let R_1 be the relation that is defined by φ if $\dot{R}$ represents R_0. This is repeated, each R_i giving rise to R_{i+1}, until a fixed point is reached. For *least fixed points*, we ensure $R_i \subseteq R_{i+1}$ for all i. For *partial fixed points*, this is not required. We use FC$^{\mathsf{lfp}}$ and FC$^{\mathsf{pfp}}$ for the respective extensions of FC.

Complexity classes are commonly defined as classes of languages; and as we can treat FC and its extensions as language generators, connecting these two worlds is straightforward. We say that a logic $\mathcal{F}$ *captures* a complexity class $\mathbb{C}$ if $\mathbb{C}$ is the class of languages that are $\mathcal{F}$-definable – that is, $\mathbb{C} = \{\mathcal{L}(\varphi) \mid \varphi \in \mathcal{F}\}$.

The following result mirrors that for the respective extensions of FO[<]:

▶ **Theorem 4.9.** FC$^{\mathsf{dtc}}$*,* FC$^{\mathsf{tc}}$*,* FC$^{\mathsf{lfp}}$*,* FC$^{\mathsf{pfp}}$ *capture* L*,* NL*,* P*,* PSPACE*, respectively.*

The result holds even if the formulas are required to be existential-positive. Thus, FC and even EP-FC behave under fixed-points and transitive closures like FO[<].

Recursion. This connection immediately suggests another: Recall that FO with least-fixed point operators can be used to define Datalog (see e. g. Part D of [1]). Analogously, we define FC-Datalog, a version of Datalog that is based on word equations.

An FC-Datalog-program is a tuple $P := (\mathcal{R}, \Phi, \mathsf{Ans})$, where $\mathcal{R}$ is a set of relation symbols that contains a special output symbol Ans, each $R \in \mathcal{R}$ has an arity $\mathsf{ar}(R)$, and Φ is a finite set of rules $R(\vec{x}) \leftarrow \varphi_1(\vec{y}_1), \dots, \varphi_m(\vec{y}_m)$ with $R \in \mathcal{R}$, $m \geq 1$, each φ_i is an FC-word equation, and each x of $\vec{x}$ appears in some $\vec{y}_i$.

We define $[\![P]\!](w)$ incrementally, initializing the relations of all $R \in \mathcal{R}$ to $\emptyset$. For each rule $R(\vec{x}) \leftarrow \varphi_1(\vec{y}_1), \dots, \varphi_m(\vec{y}_m)$, we enumerate all σ with $\sigma(\mathfrak{u}) = w$ and check if $\sigma \models \exists \vec{y}\colon \bigwedge_{i=1}^{m} \varphi_i$, where $\vec{y} := (\bigcup_{i=1}^{m} \vec{y}_i) - \vec{x}$. If this holds, we add $\sigma(\vec{x})$ to R. This is repeated until all relations have stabilized. Then $[\![P]\!](w)$ is the content of the relation Ans.

▶ **Example 4.10.** Define an FC-Datalog-program $(\{\mathsf{Ans}, E\}, \Phi)$, where $\mathsf{ar}(\mathsf{Ans}) = 0$, $\mathsf{ar}(E) = 3$, and Φ consists of the rules $\mathsf{Ans}() \leftarrow \mathfrak{u} \doteq xyz, E(x, y, z)$, and $E(x, y, z) \leftarrow x \doteq \varepsilon, y \doteq \varepsilon, z \doteq \varepsilon$, and $E(x, y, z) \leftarrow x \doteq \hat{x}\mathtt{a}, y \doteq \hat{y}\mathtt{b}, z \doteq \hat{z}\mathtt{c}, E(\hat{x}, \hat{y}, \hat{z})$. This defines the language $\{\mathtt{a}^n\mathtt{b}^n\mathtt{c}^n \mid n \geq 0\}$.

▶ **Theorem 4.11.** FC-Datalog *captures* P.

This is unsurprising, considering Datalog on ordered structures captures P, see e. g. [36], and the analogous result for spanners with recursion [47]. But it allows us to use word equations as a basis for Datalog on words. This provides potential applications for future insights into acyclicity for patterns, which could be combined with existing techniques for Datalog.

FC-Datalog can also be seen as a generalization of *range concatenation grammars (RCGs)*, see [6, 31], to use outputs and relations. There has been some work on parsing of RCGs (see [32] and its references). In the future, these might help identify tractable fragments of FC-Datalog. Vice versa, insights into the latter might lead to new approaches to RCG-parsing.

5 FC as a logic for document spanners

Fagin et al. [19] introduced *document spanners* (or just *spanners*) as a formal model of information extraction that is based on relational algebra (see e. g. [1]). This section connects spanners to FC. After stating the necessary definitions (Section 5.1), we extend FC into a logic for spanners (Section 5.2) and then use this for an inexpressibility proof (Section 5.3).

5.1 Spans and document spanners

A *span* of $w := \mathtt{a}_1 \cdots \mathtt{a}_n$ with $n \geq 1$ is an interval $[i, j\rangle$ with $1 \leq i \leq j \leq n + 1$. It describes the factor $w_{[i,j\rangle} = \mathtt{a}_i \cdots \mathtt{a}_{j-1}$. For finite $V \subset \Xi$ and $w \in \Sigma^*$, a (V, w)-*tuple* is a function μ that maps each variable in V to a span of w. A *spanner* with variables V is a function P that maps every $w \in \Sigma^*$ to a set $P(w)$ of (V, w)-tuples. We use $\mathsf{Var}(P)$ for the variables of a spanner P. Accordingly, a spanner P is a function that takes an input word w and computes a relation $P(w)$ of $(\mathsf{Var}(P), w)$-tuples.

Like [19], we base spanners on *regex formulas*; regular expressions with variable bindings $x\{\alpha\}$. This matches the same words as the expression α and assigns the corresponding span of w to the variable x. For the purpose of this article, this informal definition shall suffice. Detailed definitions of the syntax and semantics of regex formulas can be found in [19] (the original definition that uses parse trees) and [22] (a more lightweight definition that uses the ref-words from Schmid [51]).

A regex formula is *functional* if on every word, every match has exactly one assignment for each variable. The set of functional regex formulas is RGX. For $\alpha \in \mathsf{RGX}$, we define the spanner $[\![\alpha]\!]$ as follows. Every match on $w \in \Sigma^*$ defines a $(\mathsf{Var}(\alpha), w)$-tuple μ, where each $\mu(x)$ is the span assigned to x; and $[\![\alpha]\!](w)$ is the set of all these μ.

▶ **Example 5.1.** We consider the regex formula $\alpha := \Sigma^*\big(x\{\texttt{banana}\} \cup x\{\texttt{papaya}\}\big)\Sigma^*$, which matches every word w that contains an occurrence of `banana` or `papaya`. The corresponding spanner $[\![\alpha]\!](w)$ contains all spans $[i, j\rangle$ with $w_{[i,j\rangle} \in \{\texttt{banana}, \texttt{papaya}\}$. Next, we define $\beta := \Sigma^* x\{\Sigma^*\}\Sigma^* y\{\Sigma^*\}\Sigma^*$. For every $w \in \Sigma^*$, we have that $[\![\beta]\!](w)$ contains those μ where $\mu(x)$ refers to a span to the left of $\mu(y)$.

We use the spanner operations union $\cup$, natural join $\bowtie$, projection π, set difference $-$, and equality selection $\zeta^=$. For spanners P_1 and P_2 with $\mathsf{Var}(P_1) = \mathsf{Var}(P_2)$, *union* and *set difference* are defined by $(P_1 \cup P_2)(w) := P_1(w) \cup P_2(w)$ and $(P_1 - P_2)(w) := P_1(w) - P_2(w)$ on all $w \in \Sigma^*$. Furthermore, the *projection* $\pi_V P$ for a spanner P and a set of variables $V \subset \mathsf{Var}(P)$ is obtained for every $w \in \Sigma^*$ by restricting the domain of every $\mu \in P(w)$ to V.

The *natural join* $P_1 \bowtie P_2$ combines spanner results by merging tuples that agree on the common variables. That is, $(P_1 \bowtie P_2)(w)$ contains those $(\mathsf{Var}(P_1) \cup \mathsf{Var}(P_2), w)$-tuples μ for which there exist $\mu_1 \in P_1(w)$ and $\mu_2 \in P_2(w)$ such that $\mu_1(x) = \mu_2(x)$ for all $x \in \mathsf{Var}(P_1) \cap \mathsf{Var}(P_2)$. An important consequence of this definition is that join is defined using spans (and thereby positions in the input word), not using the factors that occur in the spans. To compare factors, we use the *equality selection* $\zeta^=_{x,y} P$ with $x, y \in \mathsf{Var}(P)$. This is defined by $\zeta^=_{x,y} P(w) := \{\mu \in P(w) \mid w_{\mu(x)} = w_{\mu(y)}\}$ for $w \in \Sigma^*$.

By combining regex formulas with symbols for spanner operations, we obtain *spanner representations*; and their semantics are defined by applying the operations. The class of *generalized core spanner representations* $\mathsf{RGX}^{\mathsf{gcore}}$ consists of combinations of RGX and any of the five operators; the *core spanner representations* $\mathsf{RGX}^{\mathsf{core}}$ exclude set difference. According to Fagin et al. [19], "core spanners" capture the core functionality of IBM's SystemT.

▶ **Example 5.2.** Let α and β be the regex formulas from Example 5.1. We define the spanner representation $\varrho_1 := \alpha(x) \bowtie \alpha(y) \bowtie \beta(x, y)$. Then $\mathsf{Var}(\varrho_1) = \{x, y\}$, and $[\![\varrho_1]\!](w)$ contains those μ where $\mu(x)$ occurs before $\mu(y)$ in w and each of $w_{\mu(x)}$ and $w_{\mu(y)}$ is `banana` or `papaya`. Now let $\varrho_2 := \zeta^=_{x,y}\varrho_1$. Then $[\![\varrho_2]\!](w)$ is the subset of $[\![\varrho_1]\!](w)$ that also has $w_{\mu(x)} = w_{\mu(y)}$.

We identify spanners and their representations; e. g. by referring to a representation ϱ as a spanner (technically, $[\![\varrho]\!]$ is the spanner) or by calling the elements of $\mathsf{RGX}^{\mathsf{core}}$ *core spanners*.

5.2 Adding expressive power to FC

As core spanners are based on regular expressions, they can define all regular languages. This makes them more powerful than EP-FC. To prove this, we first connect FC to C.

▶ **Lemma 5.3.** *Given $\varphi \in \mathsf{FC}$, we can construct in polynomial time $\psi \in \mathsf{C}$ such that $\sigma \models \varphi$ if and only if $\sigma \models \psi$. This also preserves the properties existential and existential-positive.*

Hence, EP-FC is not more expressive than EP-C, which cannot express all regular languages – not even comparatively "harmless" languages like e. g. $\{\mathtt{a}, \mathtt{b}\}^*\mathtt{c}$ (see Karhumäki, Mignosi, Plandowski [33]). While we could define this specific language using negation, we shall address the issue in a way that generalizes far beyond regular languages and that does not require us to leave the existential-positive fragment (and its friendlier upper bounds). Complexity is also a reason why we do not use MSO or define a second-order version of FC.

Instead, take inspiration from C (see Diekert [13]). The *theory of concatenation with regular constraints*, C[REG], extends C by allowing *regular constraints* $x \mathrel{\dot\in} \alpha$ as atoms, where $x \in \Xi$, α is a regular expression, and $\sigma \models x \mathrel{\dot\in} \alpha$ if $\sigma(x) \in \mathcal{L}(\alpha)$. We define FC[REG] analogously, where $\sigma \models x \mathrel{\dot\in} \alpha$ has the additional condition that $\sigma(x) \sqsubseteq \sigma(\mathfrak{u})$ must hold.

▶ **Theorem 5.4.** *Theorem 4.1 and Theorem 4.2 also hold if we replace* FC *with* FC[REG] *and* EP-FC *with* EP-FC[REG].

In other words, evaluation is PSPACE-complete for FC[REG] and NP-complete for EP-FC[REG], and formula width can be used as parameter to bound model checking for FC[REG]. This generalizes to all constraints that can be decided in polynomial time, which allows us to adapt FC to other settings as well.

For example, string solvers often use *length constraints*. There are predicates that compare words by applying arithmetic to their lengths, like $|x| + |y| = |z|$. While the applications of EP-C in a string solver context usually rely on deciding satisfiability, cases where model checking suffices could benefit from using FC with appropriate constraints.

Regarding prior work, the C[REG]-fragments SpLog and $\mathsf{SpLog}^{\neg}$ were introduced in [22] as alternatives to $\mathsf{RGX}^{\mathsf{core}}$ and $\mathsf{RGX}^{\mathsf{gcore}}$, respectively. As these ensure the finite universe purely through syntax, they are more cumbersome than FC and do not generalize as nicely.

Before we connect FC[REG] to spanners, we take a brief look at restricted regular expressions that can be expressed in FC. We call a regular expression *simple* if the operator * is only applied to terminal words or to Σ (a shorthand for $\bigcup_{\mathtt{a}\in\Sigma}\mathtt{a}$). That is, if $\Sigma = \{\mathtt{a}, \mathtt{b}, \mathtt{c}\}$, then $(\mathtt{abc})^*\Sigma^*$ is simple, but $(\mathtt{a} \cup \mathtt{b})^*$ and $(\mathtt{a}(\mathtt{b})^*)^*$ are not.

▶ **Lemma 5.5.** *For every simple regular expression α, there is $\varphi^\alpha(x) \in$ EP-FC such that $(w, \sigma) \models \varphi^\alpha$ if and only if $\sigma(x) \in \mathcal{L}(\alpha)$ and $\sigma(x) \sqsubseteq w$.*

The proof uses the characterization of commuting words (see e. g. Lothaire [38]). We shall use this Lemma in the proof of Theorem 5.14, to replace regular constraints.

FC[REG] and Spanners. As we want to use FC[REG] for spanners, we still need to close a formal gap, namely that spanners reason over positions in a word, while FC[REG] reasons over words. We bridge this gap through the notion of one *realizing* the other, which [22] introduced for the logic SpLog. We begin with formulas that realize spanners.

▶ **Definition 5.6.** *A substitution σ* expresses *a (V, w)-tuple μ if $\mathsf{Dom}(\sigma) \supseteq \{x^P, x^C \mid x \in V\}$ and, for all $x \in V$, we have $\sigma(x^P) = w_{[1,i\rangle}$ and $\sigma(x^C) = w_{[i,j\rangle}$ for $[i, j\rangle = \mu(x)$.*

A formula $\varphi \in$ FC[REG] realizes *a spanner P if $\mathsf{free}(\varphi) = \{x^P, x^C \mid x \in \mathsf{Var}(P)\}$ and, for all $w \in \Sigma^*$, we have $(w, \sigma) \models \varphi$ if and only if σ expresses some $\mu \in P(w)$.*

In other words, x^C is $w_{\mu(x)}$ (the content of x), and x^P is the prefix of w before $w_{\mu(x)}$.

▶ **Example 5.7.** In Example 5.1, we defined $\alpha := \Sigma^*\big(x\{\mathtt{banana}\} \cup x\{\mathtt{papaya}\}\big)\Sigma^*$. Its spanner $[\![\alpha]\!]$ is realized by $\varphi(x^P, x^C) := \exists y\colon \mathfrak{u} \doteq x^P x^C y \wedge \big(x^C \doteq \mathtt{banana} \vee x^C \doteq \mathtt{papaya}\big)$.

Then $(w, \sigma) \models \varphi$ if σ expresses some $\mu \in [\![\alpha]\!](w)$. That is, $\sigma(x^C)$ contains $w_{\mu(x)}$ (i. e., banana or papaya), and $\sigma(x^P)$ contains the prefix in w before it.

To show that FC[REG] cannot express more than the classes of spanners that we consider, we also define the notion of spanners that realize formulas.

▶ **Definition 5.8.** *A spanner P* realizes *$\varphi \in$ FC[REG] if $\mathsf{Var}(P) = \mathsf{free}(\varphi)$ and, for all $w \in \Sigma^*$, we have $\mu \in P(w)$ if and only if $(w, \sigma) \models \varphi$ for the σ with $\sigma(x) := w_{\mu(x)}$ for all $x \in \mathsf{Var}(P)$.*

There are *polynomial-time conversions* from a class of formulas (or spanners) A to a class of spanners (or formulas) B if, given $x \in A$, we can compute in polynomial time $y \in B$ that realizes x. We write $A \equiv_{\mathsf{poly}} B$ if there are polynomial-time conversions from A to B and from B to A.

▶ **Theorem 5.9.** $\mathsf{FC[REG]} \equiv_{\mathsf{poly}} \mathsf{RGX}^{\mathsf{gcore}}$ *and* $\mathsf{EP\text{-}FC[REG]} \equiv_{\mathsf{poly}} \mathsf{RGX}^{\mathsf{core}}$.

5.3 Inexpressibility for FC, FC[REG], and spanners

There are currently only few inexpressibility methods for FC and FC[REG], as there are only few such methods for related models like spanners or the theory of concatenation. A detailed discussion from the point of view of $\mathsf{RGX}^{\mathsf{core}}$ and SpLog can be found in Section 6 of [22]. These techniques do not account for negation, which makes them inapplicable for FC or FC[REG]. A standard tool for FO-inexpressibility are Ehrenfeucht–Fraïssé games (e. g. [36]). But as concatenation acts as a generalized addition, using these for FC or FO[EQ] is far from straightforward. Another standard tool is the Feferman-Vaught theorem (see [39]). While this can be used for FC, the factor universe of FC makes decomposing the structure into disjoint sets inconvenient. Instead of following down this road, we introduce FO[EQ], an extension of FO[<] that has the same expressive power as FC.

Connecting FC to FO[<]. In this section, we establish connections between FC and "classical" relational first-order logic. It is probably safe to say that in finite model theory, the most common way of applying first-order logic to words is the logic FO[<] (and the more general MSO). This uses the equality $\doteq$ and a vocabulary that consists of a binary relation symbol $<$ and unary relation symbols $\mathsf{P}_\mathtt{a}$ for each $\mathtt{a} \in \Sigma$. Every word $w = a_1 \cdots a_n \in \Sigma^+$ with $n \geq 1$ is represented by a structure $\mathcal{A}_w$ with universe $\{1, \ldots, n\}$. For every $\mathtt{a} \in \Sigma$, the relation $\mathsf{P}_\mathtt{a}$ consists of those i that have $a_i = \mathtt{a}$. To simplify dealing with ε, we slightly deviate from this standard structure. For every $w \in \Sigma^*$, we extend $\mathcal{A}_w$ to $\mathcal{A}'_w$ by adding an additional "letter-less" node $|w| + 1$ that occurs in no $\mathsf{P}_\mathtt{a}$. Then we have a one-to-one correspondence between pairs (i, j) with $i \leq j$ from the universe of $\mathcal{A}'_w$ and the spans $[i, j\rangle$ of w (see Section 5.1), and $w = \varepsilon$ does not require a special case.

▶ **Definition 5.10.** FO[EQ] *extends* FO[<] *with constants* min *and* max*, the binary relation symbol* succ*, and the 4-ary relation symbol* Eq*. For every* $w \in \Sigma^*$ *and the corresponding structure* $\mathcal{A}'_w$*, these symbols express* $\mathsf{min} = 1$*,* $\mathsf{max} = |w| + 1$*,* $\mathsf{succ} = \{(i, i+1) \mid 1 \leq i \leq |w|\}$*, and* Eq *contains those* (i_1, j_1, i_2, j_2) *with* $i_1 \leq j_1$ *and* $i_2 \leq j_2$ *such that* $w_{[i_1,j_1\rangle} = w_{[i_2,j_2\rangle}$*. We write* $(w, \alpha) \models \varphi$ *to denote that* α *is a satisfying assignment for* φ *on* $\mathcal{A}'_w$*.*

▶ **Example 5.11.** The FO[EQ]-formula $\exists x \colon \mathsf{Eq}(\mathsf{min}, x, x, \mathsf{max})$ defines $\{ww \mid w \in \Sigma^*\}$.

Technically, we do not need the symbols min, max, or succ, as these can be directly expressed in FO[<]. But these constants allows us to better preserve the structural similarities when converting between various fragments of FC and FO[EQ].

When comparing FC to FO[EQ], we need to address that one operates on words and the other on positions. We can handle this in a way that is similar to the situation between FC and spanners; and this can be used to show that there are polynomial time conversions between FC and FO[EQ] that preserve the properties existential and existential-positive, and only marginally increase the width of the formulas.

In fact, these transformations show that one could choose FO[EQ] over FC as a logic for words (or for spanners, if one extends FO[EQ] with regular constraints or generalizes it to MSO with Eq). This is a valid choice, if one prefers writing $\exists x_1, \ldots, x_6 \colon \big(\mathsf{P}_\mathtt{p}(x_1) \wedge \mathsf{P}_\mathtt{a}(x_2) \wedge \mathsf{P}_\mathtt{p}(x_3) \wedge \mathsf{P}_\mathtt{a}(x_4) \wedge \mathsf{P}_\mathtt{y}(x_5) \wedge \mathsf{P}_\mathtt{a}(x_6) \wedge \bigwedge_{i=1}^{5} \mathsf{succ}(x_i, x_{i+1})\big)$ over $\exists x \colon x \doteq \mathtt{papaya}$ or if one wants to express $x \doteq yz$ as $\exists x^m \colon \big(\mathsf{Eq}(x^o, x^m, y^o, y^c) \wedge \mathsf{Eq}(x^m, x^c, z^o, z^c)\big)$ instead.

Details on these conversions (and the required definitions) can be found in the full version of this paper. For the sake of finding an inexpressibility result, we only require the following.

▶ **Lemma 5.12.** *A language is definable in* FC *if and only if it is definable in* FO[EQ].

Proving inexpressibility. Lemma 5.12 allows us to use Feferman-Vaught theorem, at least when considering languages that are restricted enough.

▶ **Lemma 5.13.** *There is no* FC*-formula that defines* $\{\mathtt{a}^n\mathtt{b}^n \mid n \geq 1\}$.

Moreover, we can show that regular constraints offer no help for defining this language.

▶ **Theorem 5.14.** FC[REG] *cannot express the equal length relation* $|x| = |y|$.

As FC[REG] has the same expressive power as $\mathsf{RGX}^{\mathsf{gcore}}$, this is the first inexpressible result for $\mathsf{RGX}^{\mathsf{gcore}}$ on non-unary alphabets. The proof has two parts, which both rely on the limited structure of the language $\mathtt{a}^n\mathtt{b}^n$. One part is using Lemma 5.13, wich applies the Feferman-Vaught theorem. The other is using Lemma 5.5 to eliminate the regular constraints, which is based on combinatorics on words. The authors expect that a more general inexpressibility method for FC (or even FC[REG]) would need to combine more advanced techniques from combinatorics on words (like those in [33]) with methods from logic.

6 Conclusions and future work

On words, concatenation is one of the most natural operations. But as seen for C, using concatenation with first-order logic quickly becomes undecidable. Restricting the universe to a word and all its factors changes the situation drastically. In contrast to C, the resulting logic FC has a meaningful distinction between satisfiability and model checking; and the latter is not only decidable, but we can use the structure of the formula to derive upper bounds in the same way as for FO over finite structures. In addition to this, FC can also replace FO[<] as "base" logic for characterizing complexity classes. Hence, while one might certainly make a case against the claim that FC is *the* finite model version of the theory of concatenation, the results leave little doubt that it is at least *a* valid approach.

FC also provides an extendable framework for querying and model checking words, in particular for scenarios that rely on expressing that factors appear multiple times. If more expressive power is needed, FC is easily extended with constraints, without affecting the lower bounds on evaluation and model checking. In particular, we can translate core and generalized core spanners to FC[REG] and then analyze or optimize these formulas with respect to parameters like width. To a degree, this was also possible the spanner logic SpLog, but FC is more elegant, easier to use, and behaves much more like FO on relational databases.

Future work

Many fundamental questions remain open, in particular for model checking and related problems, like evaluation and enumeration.

Compilation into tractable fragments. One promising direction is the compiling of formulas into equivalent formulas of a fragment where these problems can be solved more efficiently. For example, Theorem 4.2 shows that bounding the width of the formulas leads to tractable model checking. Theorem 4.4 then provides us with a sufficient criterion for formulas that can be rewritten into formulas with a lower width, by decomposing the pattern of word equations. It is likely that this approach can be further refined by not just rewriting single patterns, but taking the larger formula into account. This approach can also be used with other structure parameter for formulas (like acyclicity and bounded tree width), by developing a corresponding variant of Theorem 4.4. One example of this is [26], which adapts the concept of acyclic conjunctive queries to FC.

A more fundamental question is whether all tractable fragments of FC can be explained through the criterion of bounded width (or are subset of a larger tractable fragment that is explained through it). A good starting point for this line of investigation is the question whether all classes of pattern languages with a polynomial time membership problem can be explained through bounded width.

Model checking as parsing. Another approach – that is not investigated in the present paper – is the connection to parsing algorithms. This is a natural question, as model-checking EP-FC-formulas can be understood as a parsing problem (where variables are mapped to factors of the input word). Promising starting points for this are parsing algorithms for RCGs (recall Section 4.3) and related grammars, and the extraction grammars from [45].

Data structures. Model checking algorithms will likely benefit from specialized data structures. For example, a naive representation of all factors of a word of length n would contain about $O(n^2)$ elements, and if these are just represented directly as words, this would take $O(n^3)$ memory. But using data structures like suffix trees and suffix arrays, one can create in time $O(n)$ a data structure that allows us to enumerate all factors with constant delay (see [26], which also examines small word equations).

While these optimizations do not matter if one considers polynomial time efficient enough, it would be very useful to know which fragments can be model-checked in time $O(n^k)$ for small k, or even in sub-quadratic time.

Inexpressibility and satisfiability. Our results on inexpressibility also leave many questions open. Lemma 5.13 heavily relies on the limited structure of the language. This is the same situation as in Section 6.1 of [22], which describes an inexpressibility technique for EP-FC[REG]. Although these two approaches provide us with some means of proving inexpressibility, they only cover special cases, and much remains to be done. It seems likely that a more general method will need to combine approaches from finite model theory (like the Feferman-Vaught theorem that we used for Lemma 5.13) with techniques from combinatorics on words (like those in [33] that [22] uses). A related problem that is still open is whether EP-FC has the same expressive power as EP-C.

Of particular interest is finding a method to prove inexpressibility in FC^k for some $k > 0$. This problem relates to the open questions whether there are algorithms that minimize the width of a formula, and for which k the fragment FC^{k+1} is more expressive than FC^k. The authors conjecture that this holds for all $k \geq 0$, which would contrast with FO[<], where the fragment of formulas with width three has the same expressive power as the full logic. Finally, it remains open whether satisfiability is decidable for FC^1 or FC^2.

Beyond FC. Using FC as a logic for spanners (and other models, potentially) raises further questions. For example, while every tractable fragment of FC[REG] maps to a tractable fragment of core spanners (namely, those that are obtained by converting the formulas), there is no guarantee that the obtained fragment is natural. Hence, a more detailed investigation into conversions between FC[REG] and $\mathsf{RGX}^{\mathsf{core}}$ is justified.

There are many other possible directions. For example, one could easily define a second-order version of FC and adapt various results from SO. Moreover, FC could be examined from an algebra point of view, or related to rational and regular relations.

References

1 Serge Abiteboul, Richard Hull, and Victor Vianu. *Foundations of Databases*. Addison-Wesley, 1995. URL: `http://webdam.inria.fr/Alice/`.

2 Isolde Adler and Mark Weyer. Tree-width for first order formulae. *Log. Methods Comput. Sci.*, 8(1), 2012.

3 Antoine Amarilli, Pierre Bourhis, Stefan Mengel, and Matthias Niewerth. Constant-delay enumeration for nondeterministic document spanners. In *Proc. ICDT 2019*, pages 22:1–22:19, 2019.

4 Michael Benedikt, Leonid Libkin, Thomas Schwentick, and Luc Segoufin. Definable relations and first-order query languages over strings. *J. ACM*, 50(5):694–751, 2003.

5 Anthony J. Bonner and Giansalvatore Mecca. Sequences, datalog, and transducers. *J. Comput. Syst. Sci*, 57(3):234–259, 1998.

6 Pierre Boullier. Range concatenation grammars. In *New developments in parsing technology*, pages 269–289. Springer, 2004.

7 Laura Ciobanu, Volker Diekert, and Murray Elder. Solution sets for equations over free groups are EDT0L languages. *IJAC*, 26(5):843–886, 2016.

8 Marek Cygan, Fedor V. Fomin, Lukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michal Pilipczuk, and Saket Saurabh. *Parameterized Algorithms*. Springer, 2015.

9 Joel D. Day, Pamela Fleischmann, Florin Manea, Dirk Nowotka, and Markus L. Schmid. On matching generalised repetitive patterns. In *Proc. DLT 2018*, pages 269–281, 2018.

10 Joel D. Day, Vijay Ganesh, Paul He, Florin Manea, and Dirk Nowotka. The satisfiability of word equations: Decidable and undecidable theories. In *Proc. RP 2018*, pages 15–29, 2018.

11 Joel D. Day and Florin Manea. On the structure of solution sets to regular word equations. In *Proc. ICALP 2020*, pages 124:1–124:16, 2020.

12 Joel D. Day, Florin Manea, and Dirk Nowotka. The hardness of solving simple word equations. In *Proc. MFCS 2017*, pages 18:1–18:14, 2017.

13 Volker Diekert. Makanin's Algorithm. In M. Lothaire, editor, *Algebraic Combinatorics on Words*, chapter 12. Cambridge University Press, 2002.

14 Volker Diekert. More than 1700 years of word equations. In *Proc. CAI 2015*, 2015.

15 Johannes Doleschal, Benny Kimelfeld, Wim Martens, Yoav Nahshon, and Frank Neven. Split-correctness in information extraction. In *Proc. PODS 2019*, pages 149–163, 2019.

16 Johannes Doleschal, Benny Kimelfeld, Wim Martens, and Liat Peterfreund. Weight annotation in information extraction. In *Proc. ICDT 2020*, pages 8:1–8:18, 2020.

17 V. G. Durnev. Undecidability of the positive $\forall\exists^3$-theory of a free semigroup. *Siberian Mathematical Journal*, 36(5):917–929, 1995.

18 Heinz-Dieter Ebbinghaus and Jörg Flum. *Finite Model Theory*. Springer, 2nd edition, 1999.

19 Ronald Fagin, Benny Kimelfeld, Frederick Reiss, and Stijn Vansummeren. Document spanners: A formal approach to information extraction. *J. ACM*, 62(2):12:1–12:51, 2015.

20 Ronald Fagin, Benny Kimelfeld, Frederick Reiss, and Stijn Vansummeren. Declarative cleaning of inconsistencies in information extraction. *ACM Trans. Database Syst.*, 41(1):6:1–6:44, 2016.

21 Fernando Florenzano, Cristian Riveros, Martín Ugarte, Stijn Vansummeren, and Domagoj Vrgoc. Constant delay algorithms for regular document spanners. In *Proc. PODS 2018*, pages 165–177, 2018.

22 Dominik D. Freydenberger. A logic for document spanners. *Theory Comput. Syst.*, 63(7):1679–1754, 2019.

23 Dominik D. Freydenberger and Mario Holldack. Document spanners: From expressive power to decision problems. *Theory Comput. Syst.*, 62:854–898, 2018.

24 Dominik D. Freydenberger, Benny Kimelfeld, and Liat Peterfreund. Joining extractions of regular expressions. In *Proc. PODS 2018*, pages 137–149, 2018.

25 Dominik D. Freydenberger and Sam M. Thompson. Dynamic complexity of document spanners. In *Proc. ICDT 2020*, pages 11:1–11:21, 2020.

26 Dominik D. Freydenberger and Sam M. Thompson. Splitting spanner atoms: A tool for acyclic core spanners, 2021. arXiv:2104.04758.

27 Seymour Ginsburg and Xiaoyang Sean Wang. Regular sequence operations and their use in database queries. *J. Comput. Syst. Sci*, 56(1):1–26, 1998.

28 Gösta Grahne, Matti Nykänen, and Esko Ukkonen. Reasoning about strings in databases. *J. Comput. Syst. Sci*, 59(1):116–162, 1999.

29 Simon Halfon, Philippe Schnoebelen, and Georg Zetzsche. Decidability, complexity, and expressiveness of first-order logic over the subword ordering. In *Proc. LICS 2017*, pages 1–12, 2017.

30 Juris Hartmanis. On Gödel speed-up and succinctness of language representations. *Theor. Comput. Sci.*, 26:335–342, 1983.

31 Laura Kallmeyer. *Parsing Beyond Context-Free Grammars*. Cognitive Technologies. Springer, 2010.

32 Laura Kallmeyer, Wolfgang Maier, and Yannick Parmentier. An earley parsing algorithm for range concatenation grammars. In *Proc. ACL-IJCNLP 2009*, pages 9–12, 2009.

33 Juhani Karhumäki, Filippo Mignosi, and Wojciech Plandowski. The expressibility of languages and relations by word equations. *J. ACM*, 47(3):483–505, 2000.

34 Phokion G. Kolaitis and Moshe Y. Vardi. Conjunctive-query containment and constraint satisfaction. *J. Comput. Syst. Sci*, 61(2):302–332, 2000.

35 Martin Kutrib. The phenomenon of non-recursive trade-offs. *Int. J. Found. Comput. Sci.*, 16(5):957–973, 2005.

36 Leonid Libkin. *Elements of Finite Model Theory*. Texts in Theoretical Computer Science. Springer, 2004.

37 Katja Losemann. *Foundations of Regular Languages for Processing RDF and XML*. PhD thesis, University of Bayreuth, 2015. URL: https://epub.uni-bayreuth.de/2536/.

38 M. Lothaire. *Combinatorics on Words*. Addison-Wesley, Reading, MA, 1983.

39 Johann A. Makowsky. Algorithmic uses of the Feferman-Vaught theorem. *Annals of Pure and Applied Logic*, 126(1-3):159–213, 2004.

40 Florin Manea and Markus L. Schmid. Matching patterns with variables. In *Proc. WORDS 2019*, pages 1–27, 2019.

41 Francisco Maturana, Cristian Riveros, and Domagoj Vrgoc. Document spanners for extracting incomplete information: Expressiveness and complexity. In *Proc. PODS 2018*, pages 125–136, 2018.

42 Andrea Morciano, Martin Ugarte, and Stijn Vansummeren. Automata-based evaluation of AQL queries. Technical report, Université Libre de Bruxelles, 2016.

43 Yoav Nahshon, Liat Peterfreund, and Stijn Vansummeren. Incorporating information extraction in the relational database model. In *Proc. WebDB 2016*, page 6, 2016.

44 Dirk Nowotka and Aleksi Saarela. An optimal bound on the solution sets of one-variable word equations and its consequences. In *Proc. ICALP 2018*, pages 136:1–136:13, 2018.

45 Liat Peterfreund. Grammars for document spanners. In *Proc. ICDT 2021*, pages 7:1–7:18, 2021.

46 Liat Peterfreund, Dominik D. Freydenberger, Benny Kimelfeld, and Markus Kröll. Complexity bounds for relational algebra over document spanners. In *Proc. PODS 2019*, pages 320–334, 2019.

47 Liat Peterfreund, Balder ten Cate, Ronald Fagin, and Benny Kimelfeld. Recursive programs for document spanners. In *Proc. ICDT 2019*, pages 13:1–13:18, 2019.

48 W. V. Quine. Concatenation as a basis for arithmetic. *J. Symb. Log.*, 11(4):105–114, 1946.

49 Daniel Reidenbach and Markus L. Schmid. Patterns with bounded treewidth. *Inf. Comput.*, 239:87–99, 2014.

50 Aleksi Saarela. Hardness results for constant-free pattern languages and word equations. In *Proc. ICALP 2020*, pages 140:1–140:15, 2020.

51 Markus L. Schmid. Characterising REGEX languages by regular languages equipped with factor-referencing. *Inf. Comput.*, 249:1–17, 2016.

52 Markus L. Schmid and Nicole Schweikardt. A purely regular approach to non-regular core spanners. In *Proc. ICDT 2021*, pages 4:1–4:19, 2021.

53 Howard Straubing. *Finite Automate, Formal Logic, and Circuit Complexity*. Birkhäuser, 1994.

Uniform Elgot Iteration in Foundations

Sergey Goncharov
University Erlangen-Nürnberg, Germany

Abstract

Category theory is famous for its innovative way of thinking of concepts by their descriptions, in particular by establishing *universal properties*. Concepts that can be characterized in a universal way receive a certain quality seal, which makes them easily transferable across application domains. The notion of partiality is however notoriously difficult to characterize in this way, although the importance of it is certain, especially for computer science where entire research areas, such as *synthetic* and *axiomatic domain theory* revolve around it. More recently, this issue resurfaced in the context of (constructive) *intensional type theory*. Here, we provide a generic categorical iteration-based notion of partiality, which is arguably the most basic one. We show that the emerging free structures, which we dub *uniform-iteration algebras* enjoy various desirable properties, in particular, yield an *equational lifting monad*. We then study the impact of classicality assumptions and choice principles on this monad, in particular, we establish a suitable categorial formulation of the *axiom of countable* choice entailing that the monad is an *Elgot monad*.

2012 ACM Subject Classification Theory of computation → Categorical semantics; Theory of computation → Constructive mathematics

Keywords and phrases Elgot monad, partiality monad, delay monad, restriction category

Digital Object Identifier 10.4230/LIPIcs.ICALP.2021.131

Category Track B: Automata, Logic, Semantics, and Theory of Programming

Related Version *Full Version*: https://arxiv.org/abs/2102.11828

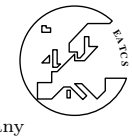
Funding *Sergey Goncharov*: Support by Deutsche Forschungsgemeinschaft (DFG) under project GO 2161/1-2 is gratefully acknowledged.

1 Introduction

Natural numbers form a prototypical domain for programming and reasoning. Both in *category theory* and in *type theory* they are characterized by a universal property, which consists of two parts: a definitional principle – *(structural) primitive recursion* and a reasoning principle – *induction*. Dualization yields respectively *co-natural numbers*, *co-recursion* and *co-induction*. Amid these two structuralist extremes, here, we analyse the challenging case of non-structural recursion in the form of iteration, which arises as follows. A map

$$h\colon S \to X + S$$

presents the simplest possible model of a computation process: with S regarded as a state space, h sends any state either to a successor *state* or to a terminal *value* in X. We wish to be able to form an object KX of *denotations* potentially reachable via such processes. Besides the values of X reachable in a finite number of steps, KX must also contain a designated value for divergence, generated by the right injection $h = \mathsf{inr}$. We then ask: what would be the generic universal characterization of KX and what properties it would imply? Somewhat surprisingly, this question has not been addressed yet on a level of generality, sufficiently close to the settings where the question can be posed, although many similar closely related questions have been addressed, mostly couched in type-theoretic terms.



The question trivializes whenever one of the two following perspectives is adopted.

- *intensional perspective:* the domain KX keeps track not only of results, but also of the number of steps needed to reach them. This leads to the identification of KX as the final coalgebra $DX = \nu\gamma.\ X + \gamma$, known as *Capretta's monad* or the *delay monad* [10].
- *non-constructive perspective:* assuming non-constructive principles, such as the *law of excluded middle*, leads to the identification of KX as the *maybe-monad* $X + 1$.

Here, we generally keep aloof from these interpretations of KX and work both extensionally and generically, using the language of the category theory to analyse the issue in the abstract, and keeping the potential class of models possibly large.

We introduce KX as a certain free structure, equipped with an iteration operator, which sends any $f\colon S \to KX + S$ to $f^\sharp\colon S \to KX$, and satisfies the following two basic and uncontroversial principles:

- *fixpoint:* $f^\sharp$ is in an obvious sense a fixpoint of f;
- *uniformity:* the structure of the state space S is ineffective (i.e. merging or adding new states done coherently does not influence the result).

We dub such structures *uniform-iteration algebras* and show that on a high level of generality (in any extensive category with finite limits and a stable natural number object) if KX exists then it satisfies a number of other properties: K extends to a monad $\mathbf{K}$, which is an equational lifting monad [9], the Kleisli category of K is enriched over partial orders and monotone maps, and the iteration operator is a least fixpoint operator w.r.t. this order; moreover, the iteration operator satisfies an additional principle, previously dubbed *compositionality* [2].

In some environments, such as *homotopy type theory (HoTT)*, $\mathbf{K}$ can be constructed directly, by using *higher inductive types*. One can then define a universal map from the delay monad $\mathbf{D}$ to $\mathbf{K}$ and regard it is a form of *extensional collapse*. However, proving $\mathbf{K}$ to be a quotient of $\mathbf{D}$ seems to be impossible without using (weak) choice principles [11, 5, 16]. We interpret this categorically, first by introducing a categorical *limited principle of omniscience (LPO)* under which $\mathbf{K}$ turns out to be isomorphic to the *maybe-monad* $(- + 1)$ and also turns out to be an *Elgot monad*. This generalizes slightly previous results [17] obtained for *hyper-extensive categories* [3]. Second, we identify other cases of $\mathbf{K}$ being a quotient of $\mathbf{D}$ and additionally being an (initial) Elgot monad, by introducing certain coequalizer preservation conditions, abstractly capturing the corresponding instances of the axiom of countable choice.

From the type-theoretic perspective, in our work we revisit the familiar waymarks of using/avoiding principles of classical/constructive mathematics in view of the tradoffs in expressive power of the corresponding constructions. Our present approach of uniform-iteration algebras as a fundamental primitive is entirely new, though. Moreover, we would like to emphasize that our results, being generic, apply to a wide range of categories, whose objects need not be like sets, or types in any conventional sense. This has a massive impact on the underlying proof methods. In topos theory, calculations are facilitated by existence of the subobject classifier Ω, which is used as a global parent space for propositions. Predicative theories, such as HoTT make do without Ω, but it is still possible to form predicative types of propositions per universe, implying that the style of proofs can to a significant extent be maintained, with Ω intuitively regarded as "scattered" over the cumulative universe hierarchy. Contrastingly, here we do not assume any kind of general reference spaces for propositions, which results in completely different proof methods. Nevertheless, we conjecture that our results can be implemented in HoTT. This is clear for the universe of sets, which in HoTT form a *pretopos* [30], and hence directly satisfy our assumptions. For types of higher homotopy levels this should be possible by using existing recipes of formalizing *precategories of types* [33].

Previous related work. We relate to the work on iteration theories, starting from a seminal paper of Elgot [15], who identified iteration as a fundamental unifying notion. Equational properties of Elgot iteration were extensively explored by Bloom and Ésik [8] with the initial iteration structure playing a prominent role, however, since the whole setup therein is inherently classical, most of our present agenda is essentially moot there. The uniformity property occurred under the name *functorial dagger implication* in Bloom and Ésik's monograph, and is an established and powerful principle, thus notably recognized in Simpson and Plotkin's work [32], in the context of generic recursion (as opposed to the present dual case of generic iteration). Adámek et al [2] introduced axioms of *(guarded) Elgot algebras*, and it follows from their results that these axioms are complete w.r.t. the algebras of the delay monad. Uniform-iteration algebras are generally a proper weakening of Elgot algebras, but we show that KX as a free uniform-iteration algebra over X is in fact also an Elgot algebra.

Another line of research we relate to is concerned with notions of partiality, via *dominances*, in particular the *Rosolini dominance* in *synthetic domain* theory [31], via *equational lifting monads* [9], and via *restriction categories* [13]. We remark that these approaches are rather concerned with specifying a notion of partiality than with defining it. This distinction is particularly significant in the context of constructive type theories, such as HoTT, which revitalized the interest to defining a notion of partiality both predicatively and constructively and to understanding the impact of (restricted) choice principles. Chapman et al [11] provided a construction of a partiality monad as a quotient of the delay monad assuming countable choice. Also, Uustalu and Veltri [35] explored universal properties of the obtained quotient as an initial *ω-complete pointed classifying monad*. Altenkirch et al [5] directly based on ω-complete partial orders to obtain a partiality monad in HoTT as a certain *quotient inductive-inductive type* without using any choice whatsoever, but established an equivalence with the delay monad quotient under countable choice. Chapman et al [12] subsequently used more basic *quotient inductive types* for the same purpose.

Recently, Escardó and Knapp [16] reinforced the issue of discrepancy between the quotient of the delay monad and partiality monads, by showing that the quotient precisely captures extensions of Turing computable values, whereas in the absence of any choice, the reasonable partiality monads seem to yield proply larger carriers. The latter view is particularly fine grained, and involves a monad, which is essentially our monad $\mathbf{K}$. According to them, showing the desired connection between $\mathbf{K}$ and the delay monad still amounts to (very weak) choice principles (albeit still not natively available in HoTT), while equivalence to more expressive monads would again require countable choice. Further relevant details of type-theoretic analysis of partiality can be found in recent theses [37, 25]. A comparison of various lifting monads in type theory using a unifying notion of *container* was recently provided by Uustalu and Veltri [36].

2 Categories and Monads

We assume familiarity with standard categorical concepts [28, 6]. In what follows, we generally work in an ambient extensive category $\mathbf{C}$ with finite products, a stable natural number object $\mathbb{N}$ and exponentials $X^{\mathbb{N}}$. By $|\mathbf{C}|$ we refer to the objects of $\mathbf{C}$. We often drop indices of natural transformations to avoid clutter. For the same purpose, we juxtaposition of morphisms as composition. Let us clarify this and fix some conventions.

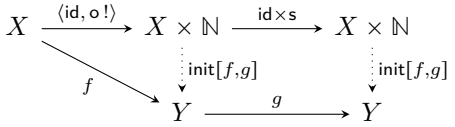
Extensive categories and pointful reasoning. Extensiveness means existence of disjoint finite coproducts and stability of them under pullbacks (which must exist). Every extensive category is *distributive*, that is, every morphism $[\mathsf{id}\times\mathsf{inl}, \mathsf{id}\times\mathsf{inr}]\colon X\times Y + X\times Z \to X\times(Y+Z)$ is an isomorphism whose inverse we denote $\mathsf{dstr}\colon X\times(Y+Z)\to X\times Y + X\times Z$. Let $\mathsf{dstl}\colon (X+Y)\times Z \to X\times Z + Y\times Z$ be the obvious dual to dstr.

In order to simplify reasoning, we occasionally use a rudimentary pointful notation for stating equalities in $\mathbf{C}$, most notably we use the case distinction operator case, e.g. we write

$$f(x) = \mathsf{case}\, g(x)\, \mathsf{of}\, \mathsf{inl}\, y \mapsto h(y);\ \mathsf{inr}\, z \mapsto u(z)$$

meaning $f = [h,u]\ g$ where $f\colon X\to W$, $g\colon X\to Y+Z$, $h\colon Y\to W$ and $u\colon Z\to W$.

Natural numbers and primitive recursion. A *stable natural number object* (NNO) in a Cartesian category $\mathbf{C}$, is an object $\mathbb{N}$ equipped with two morphisms $\mathsf{o}\colon 1\to\mathbb{N}$ (*zero*) and $\mathsf{s}\colon\mathbb{N}\to\mathbb{N}$ (*successor*) such that for any $X, Y\in|\mathbf{C}|$ and any $f\colon X\to Y$ and $g\colon Y\to Y$ there is unique $\mathsf{init}[f,g]\colon X\times\mathbb{N}\to Y$ such that



commutes. This combines two separate properties: there exists an initial $(1+-)$-algebra $(\mathbb{N}, [\mathsf{o},\mathsf{s}]\colon 1+\mathbb{N}\to\mathbb{N})$, and $(X\times\mathbb{N}, [\langle\mathsf{id},\mathsf{o}\,!\rangle, \mathsf{id}\times\mathsf{s}]\colon X + X\times\mathbb{N}\to X\times\mathbb{N})$ is an initial $(X+-)$-algebra. The latter property follows from the former in Cartesian closed categories.

More generally, we need the derivable Lawvere's internalization of *primitive recursion* [27]: Given $f\colon X\to Y$ and $g\colon Y\times X\times\mathbb{N}\to Y$ there is unique $\mathsf{p\text{-}rec}(f,g)\colon X\times\mathbb{N}\to Y$ such that

$$\mathsf{p\text{-}rec}(f,g)(x,\mathsf{o}) = f(x), \qquad \mathsf{p\text{-}rec}(f,g)(x,\mathsf{s}\,n) = g(\mathsf{p\text{-}rec}(f,g)(x,n),x,n).$$

We thus say that $\mathsf{p\text{-}rec}(f,g)$ is *defined* by (primitive) recursion, whereas *induction* is a *proof principle*, stating that $\mathsf{p\text{-}rec}(f,g) = w$ for any $w\colon X\times\mathbb{N}\to Y$ satisfying the same equations.

Exponentials $X^{\mathbb{N}}$ are adjoint to products $X\times\mathbb{N}$, meaning that there is an isomorphism $\mathsf{curry}\colon \mathbf{C}(X\times\mathbb{N}, Y)\to\mathbf{C}(X, Y^{\mathbb{N}})$ natural in X. This induces an evaluation morphism $\mathsf{ev} = \mathsf{curry}^{-1}\,\mathsf{id}\colon X^{\mathbb{N}}\times\mathbb{N}\to X$ with the standard properties.

Strong functors and monads. A functor T is strong if it is equipped with a natural transformation *strength* $\tau_{X,Y}\colon X\times TY\to T(X\times Y)$, satisfying standard coherence conditions w.r.t. the monoidal structure $(1,\times)$ of $\mathbf{C}$ [26]. This induces the obvious dual $\hat\tau_{X,Y}\colon TX\times Y\to T(X\times Y)$. A natural transformation $\alpha\colon F\to G$ between two strong functors is itself *strong* if it preserves strength in the obvious sense, i.e. $\alpha\ \tau = \tau\ (\mathsf{id}\times\alpha)$.

A monad $\mathbf{T}$ (in the form of a Kleisli triple) consists of an endomap $T\colon|\mathbf{C}|\to|\mathbf{C}|$, a family of morphisms $(\eta_X\in\mathbf{C}(X,TX))_{X\in|\mathbf{C}|}$ and a lifting operation $(-)^\star\colon\mathbf{C}(X,TY)\to\mathbf{C}(TX,TY)$, satisfying standard laws [29]. It then follows that T is an endofunctor with $Tf = (\eta\ f)^\star$, η extends to a natural transformation, and the *multiplication* transformation $\mu\colon TT\to T$ is definable as $\mathsf{id}^\star$. For every monad $\mathbf{T}$, whose underlying functor T is strong, η and μ are strong (with id being a strength of Id and $(T\tau)\ \tau$ being a strength of μ). Such monad $\mathbf{T}$ is then called *strong* if both η and μ are strong. A strong monad is *commutative* if $\tau^\star\ \hat\tau = \hat\tau^\star\ \tau$.

We adopt Moggi's perspective [29] to strong monads as carriers of computational effects, and thus say that a morphism $f\colon X\to TY$ *computes a value in* Y. Since, the only effect we deal with here is divergence, f can either produce a value or diverge (modulo the inherent linguistic inaccuracy of the excluded middle law baked into the natural language).

Functor algebras and monad algebras. For an endofunctor T, we distinguish T-algebras, which are pairs $(A, a\colon TA \to A)$, from $\mathbf{T}$-algebras, which can only be formed for monads $\mathbf{T}$ on T: a $\mathbf{T}$-algebra is a T-algebra (A, a), which additionally satisfies $a\ \eta = \mathsf{id}$ and $a\ \mu = a\ Ta$. Both T- and $\mathbf{T}$-algebras form categories under the standard structure preserving morphisms, the latter fully embeds into the former.

With our assumptions on $\mathbf{C}$, we mean to cover the following (classes of) categories.

1. Zermello-Fraenkel set theory with choice (ZFC) and further variants of set theory: ETCS, ZF, CZF, etc.
2. Toposes satisfying countable choice, e.g. the *topological topos* [23].
3. Toposes not satisfying countable choice, e.g. *nominal sets*.
4. Pretoposes, e.g. ΠW-pretoposes, compact Hausdoff spaces.
5. The category of topological spaces **Top**, and its subcategories, such as the category of directed complete sets **dCpo**.

3 Basic Properties of the Delay Monad

The final coalgebras $DX = \nu\gamma.\, X + \gamma$ jointly yield a monad $\mathbf{D}$, called the *delay monad* [10]. Capretta [10] showed that $\mathbf{D}$ is strong, which remains valid in our setting. By Lambek's lemma, the final coalgebra structure $\mathsf{out}\colon DX \to X + DX$ is an isomorphism. Its inverse $\mathsf{out}^{-1} = [\mathsf{now}, \mathsf{later}]\colon X + DX \to DX$ is composed of the morphisms, conventionally called now and later, of which the first one is the monad unit, and the effect of the second one is intuitively to postpone the argument computation by one time unit. In what follows, we will write $\triangleright$ instead of later for the sake of succinctness. As a final coalgebra, DX comes together with a *coiteration operator*: for any $f\colon Y \to X + Y$, $\mathsf{coit}\, f\colon Y \to DX$ is the unique morphism, such that $\mathsf{out}\ (\mathsf{coit}\, f) = (\mathsf{id} + \mathsf{coit}\, f)\ f$.

We denote $D1$ by $\bar{\mathbb{N}}$, and think of it as an object of co-natural or possibly infinite natural numbers. Note that the initial algebra structure $[\langle \mathsf{id}, \mathsf{o}\,! \rangle, \mathsf{id} \times \mathsf{s}]\colon X + X \times \mathbb{N} \to X \times \mathbb{N}$, is an isomorphism, and thus yields a D-coalgebra structure on $X \times \mathbb{N}$. This induces a unique D-coalgebra morphism $\iota_X\colon X \times \mathbb{N} \to DX$. Alternatively, we can regard ι_X as defined by primitive recursion:

$$\iota_X(x, \mathsf{o}) = \mathsf{now}(x) \qquad\qquad \iota_X(x, \mathsf{s}(n)) = \triangleright(\iota_X(x, n))$$

Let $\hat{\iota}\colon \mathbb{N} \to \bar{\mathbb{N}}$ be ι_1 modulo the obvious isomorphism.

In our setting, DX need not be postulated, for it is in fact definable as a retract of the object $(X + 1)^{\mathbb{N}}$ of *infinite streams*, which is elaborated in detail by Chapman et al [11]. This also entails that ι is a componentwise monic. Intuitively, DX consists of precisely those streams, which contain at most one element of the form $\mathsf{inl}\, x$. This intuition becomes precise in (possibly non-classical) set theory, where

$$\mathsf{now}\, x = (\mathsf{inl}\, x, \mathsf{inr}\, \star, \mathsf{inr}\, \star, \ldots) \qquad\qquad \triangleright\, (e_1, e_2, \ldots) = (\mathsf{inr}\, \star, e_1, e_2, \ldots)$$

This explains why classically, more precisely, under the *law of excluded*, DX is isomorphic to $X \times \mathbb{N} + 1$. We provide a stronger result to this effect further below. Let us record some general facts about $\mathbf{D}$ first.

▶ **Proposition 1.** *The monad* **D** *admits the following characterization:*

1. *unit* $\mathsf{now}\colon X \to DX$ *of* **D** *satisfies* $\mathsf{out} \;\; \mathsf{now} = \mathsf{inl}$*;*
2. *Kleisli lifting of* $f\colon X \to DY$ *is the unique morphism* $f^\star\colon DX \to DY$*, for which the diagram*

$$\begin{array}{ccc} DX & \xrightarrow{\quad f^\star \quad} & DY \\ {\scriptstyle \mathsf{out}}\downarrow & & \downarrow{\scriptstyle \mathsf{out}} \\ X + DX & \xrightarrow{[\mathsf{out}\, f,\ \mathsf{inr}\, f^\star]} & Y + DY \end{array}$$

   commutes;
3. *strength* $\tau\colon X \times DY \to D(X \times Y)$ *is a unique such morphism that the diagram*

$$\begin{array}{ccccc} X \times DY & & \xrightarrow{\qquad\qquad \tau \qquad\qquad} & & D(X \times Y) \\ {\scriptstyle \mathsf{id}\times\mathsf{out}}\downarrow & & & & \downarrow{\scriptstyle \mathsf{out}} \\ X \times (Y + DY) & \xrightarrow{\mathsf{dstr}} & X \times Y + X \times DY & \xrightarrow{\mathsf{id}+\tau} & Y + DY \end{array}$$

   commutes.

Proof. (1) and (2) follow from a more general characterization by Uustalu [34]; (3) is established in [19]. ◀

▶ **Proposition 2.** **D** *is commutative.*

Let us proceed with a characterization of the situations when $DX \cong X \times \mathbb{N} + 1$. Recall that a monic σ is called *complemented* if there exists $\sigma'\colon X' \hookrightarrow Y$, such that Y is a coproduct of X and X' with σ and σ' as coproduct injections. The *law of excluded middle* states that any monic is complemented. We involve a rather more specific property.

▶ **Proposition 3.** *The monic* $\hat{\iota}\colon \mathbb{N} \hookrightarrow \bar{\mathbb{N}}$ *is complemented iff* $DX \cong X \times \mathbb{N} + 1$*.*

Proof (Sketch). The necessity is obvious. Let us proceed with the sufficiency. Using extensiveness of **C** one can obtain the following pullback:

$$\begin{array}{ccc} X \times \mathbb{N} & \xrightarrow{\quad \mathsf{snd} \quad} & \mathbb{N} \\ {\scriptstyle \iota}\downarrow & \lrcorner & \downarrow{\scriptstyle \hat{\iota}} \\ DX & \xrightarrow{\quad D\,! \quad} & \bar{\mathbb{N}} \end{array}$$

By assumption, $\hat{\iota}$ is complemented, and since **C** is extensive, so is ι. We obtain that $DX \cong \mathbb{N} \times X + R$ for some R, and then it follows from finality of DX that $R \cong 1$. ◀

The property of $\hat{\iota}\colon \mathbb{N} \hookrightarrow \bar{\mathbb{N}}$ to be complemented is a categorical formulation of the *limited principle of omniscience (LPO)*, which is rejected in constructive mathematics. Informally, LPO states that every infinite bit-stream either contains 1 at some position or contains only 0 everywhere (the constraint that the stream contains *at most one* 1, does not make a difference). We say that **C** is an LPO category if $\hat{\iota}\colon \mathbb{N} \hookrightarrow \bar{\mathbb{N}}$ is complemented.

▶ **Corollary 4.** *Suppose that (i)* **C** *has countable products and (ii) given a family* $(\sigma_i\colon A_i \to A)_{i\in\omega}$ *of complemented pairwise disjoint monos, the induced universal morphism* $\coprod_i A_i \to A$ *is complemented. Then* **C** *satisfies LPO and hence* $DX \cong X \times \mathbb{N} + 1$*.*

Proof. It is folklore that in categories with countable products $\mathbb{N}$ is isomorphic to the sum of ω copies of 1. Thus $\hat{\iota}\colon \mathbb{N} \to \bar{\mathbb{N}}$ is the induced universal map, which is complemented by (ii). ◀

▶ **Example 5.** As expected, Proposition 3 does not apply to models, designed with constructivist principles in mind, such as intensional type theories, or realizability toposes, although, it is technically possible to design a realizability topos, satisfying LPO [7], in which thus $DX \cong X \times \mathbb{N} + 1$. Another class of examples to which Proposition 3 does not apply stems from topology. In **Top**, $\bar{\mathbb{N}}$ is a subspace of the *Cantor space* $2^{\mathbb{N}}$ whose topology is generated by the base of opens of the form $\{sr \mid r \in \{0,1\}^{\omega}\}$ with $s \in 2^{\star}$. Then $\bar{\mathbb{N}}$ is isomorphic to a *one-point compactification* of $\mathbb{N}$, i.e. it is the set $\mathbb{N} \cup \{\infty\}$, whose opens are all subsets of $\mathbb{N}$ and additionally all complements of finite subsets of $\mathbb{N}$ in $\mathbb{N} \cup \{\infty\}$. Clearly, $\bar{\mathbb{N}} \ncong \mathbb{N} + 1$. This kind of arguments is inherited by higher order topology-based models, such as Johnstone's *topological topos* [23], which is a Grothendieck topos not satisfying LPO.

▶ **Example 6.** Proposition 3 and Corollary 4 cover quite a few models constructed in the scope of classical mathematics. Every set theory satisfying the law of excluded middle satisfies LPO. Every presheaf topos (w.r.t. a classical set theory) inherits countable coproducts from **Set** and those satisfy (ii) of Corollary 4. As we indicated in Example 5, a Grothendieck topos generally need not satisfy LPO, but, e.g. *Schanuel topos* (aka the topos of nominal sets) does satisfy it, because this topos is Boolean. As we indicated in Example 5, **Top** does not satisfy LPO, but curiously the full subcategory of directed complete partial orders **dCpo** (under Scott topology) does. Both **Top** and **dCpo** have countable coproducts, but **Top** fails to satisfy condition (ii), of Corollary 4, while **dCpo** does satisfy it. This can be read as a manifestation of (undesirable) effects, which motivated synthetic domain theory [21].

Conditions (i) and (ii) in Corollary 4 are essentially the axioms of *hyper-extensive categories* by Adámek et al [3] (modulo our background extensiveness assumption). An example of an LPO category that fails (i) is Lawvere's ETCS. Another example of a Grothendieck topos that fails (ii) can be rendered as a certain category of *Jónsson-Tarski algebras* [3].

The above examples indicate that in models developed w.r.t. constructive foundations LPO fails by design, while in models developed w.r.t. classical foundations, depending on the purposes, constructively questioned principles may leak in from the metalogic level inside of the category, possibly in a weakened form, resulting in an explicit expression for DX.

4 Unguarded Elgot Algebras

Recall the following notion from [2] where the term *complete Elgot algebra over H* is used.

▶ **Definition 7** (Guarded Elgot Algebras). *Given an endofunctor H, an* (H-)guarded Elgot algebra *is a tuple $(A, a\colon HA \to A, (\text{–})^{\sharp})$ where the* iteration $f^{\sharp}\colon X \to A$ *for every given $f\colon X \to A + HX$, satisfies the following axioms:*

- (**Fixpoint**) *for every $f\colon X \to A + HX$, $f^{\sharp} = [\mathsf{id}, a\ Hf^{\sharp}]\ f$;*
- (**Uniformity**) *for every $f\colon X \to A + HX$ every $g\colon Y \to A + HY$ and every $h\colon X \to Y$, $(\mathsf{id} + Hh)\ f = g\ h$ implies $f^{\sharp} = g^{\sharp}\ h$;*
- (**Compositionality**) *for every $h\colon Y \to X + HY$ and $f\colon X \to A + HX$, $((f^{\sharp} + \mathsf{id})\ h)^{\sharp} = \big([(\mathsf{id} + H\,\mathsf{inl})\ f, \mathsf{inr}\ (H\,\mathsf{inr})]\ [\mathsf{inl}, h]\colon X + Y \to A + H(X + Y)\big)^{\sharp}\ \mathsf{inr}$.*

H-guarded Elgot algebras form a category together with iteration preserving *morphisms defined as follows: a morphism h from $(A, a, (\text{–})^{\sharp})$ to $(B, b, (\text{–})^{\sharp})$ is a morphism $h\colon A \to B$ between carriers, such that $h\ f^{\sharp} = ((h + \mathsf{id})\ f)^{\sharp}$ for every $f\colon X \to A + HX$ (this entails $h\ a = b\ (Hh)$ [2, Lemma 5.2]).*

The **Compositionality** axiom is the most sophisticated one. It intuitively states that running h in a loop over Y as the state space, and subsequently running f in a loop over X as the state space, equivalently corresponds to running a certain term constructed from f and g in a single loop over the combined state $X + Y$.

The axioms of guarded Elgot algebras are complete in the following sense.

▶ **Theorem 8** ([2, Theorem 5.4, Corollary 5.7, Theorem 5.8])**.** *For every X, a final coalgebra $\nu\gamma.\, X + H\gamma$ is a free H-guarded algebra over X, in particular, existence of final coalgebras is equivalent to existence of free H-guarded Elgot algebras. The categories of H-guarded Elgot algebras and algebras of the monad $\nu\gamma.\, X + H\gamma$ are isomorphic.*

By Theorem 8, free algebras of the delay monad are thus precisely the free Id-guarded Elgot algebras. We then introduce *un-guarded Elgot algebras* as a certain subcategory of Id-guarded ones.

▶ **Definition 9** (Unguarded Elgot Algebras)**.** *We call* Id*-guarded Elgot algebras of the form* $(A, \mathsf{id}\colon A \to A, (\text{–})^\sharp)$ unguarded Elgot algebras*, or simply* Elgot algebras *if no confusion arises. Given two Elgot algebras A and B, we call $f\colon X \times A \to B$ right iteration preserving if*

$$f(\mathsf{id} \times h^\sharp) = \big(X \times Z \xrightarrow{\mathsf{id}\times h} X \times (A + Z) \xrightarrow{\mathsf{dstr}} X \times A + X \times Z \xrightarrow{f+\mathsf{id}} B + X \times Z\big)^\sharp$$

for any $h\colon Z \to A + Z$. This generalizes Elgot algebra morphisms under $X = 1$.

We write simply "iteration preserving" instead of "right iteration preserving" in the sequel if the decomposition of $X \times A$ into the Elgot algebra part A and the parameter part X is clear from the context. Parametrization will be needed later for characterizing stability of free algebras (Lemma 18).

The unguarded Elgot algebras thus differ from the Id-guarded ones in that the Id-algebra structures $a\colon A \to A$ in the former case are forced to be trivial. This has an impact on forming the corresponding free structures: in the guarded case, the Id-algebra structures must be maximally unrestricted, which is the reason why we obtain a free Id-guarded Elgot algebra DX with the Id-algebra structure playing the role of delays. Intuitively, a free unguarded Elgot algebra must be a quotient of a free guarded one under removing delays, which is indeed what happens for LPO categories, as we show later. Otherwise, the situation is much more subtle, and it is one of our goals to demonstrate that free unguarded Elgot algebras are exactly the semantic carriers generated by unguarded iteration.

In the unguarded case **Compositionality** can be replaced by a simpler looking new law that we dub **Folding**:

▶ **Proposition 10.** *Given $A \in |\mathbf{C}|$, $(A, (\text{–})^\sharp)$ is an Elgot algebra iff $(\text{–})^\sharp$ satisfies*

- (**Fixpoint**) *for every $f\colon X \to A + X$, $f^\sharp = [\mathsf{id}, f^\sharp]\ f$;*
- (**Uniformity**) *for every $f\colon X \to A + X$ every $g\colon Y \to A + Y$ and every $h\colon X \to Y$, $(\mathsf{id} + h)\ f = g\ h$ implies $f^\sharp = g^\sharp\ h$;*
- (**Folding**) *for every $h\colon Y \to X + Y$ and $f\colon X \to A + X$, $(f^\sharp + h)^\sharp = [(\mathsf{id} + \mathsf{inl})\ f, \mathsf{inr}\ h]^\sharp$.*

The laws of Elgot algebras are summarised in Fig. 1 in the style of string diagrams, akin to those, which are used for axiomatizing traced symmetric monoidal categories [24]. In contrast to the latter, here we essentially can only form traces of morphisms of the form $X + Y \to A + Y$ where A is an Elgot algebra. Merging wires is to be interpreted as calling codiagonal morphisms $\nabla\colon X + X \to X$.

As expected, products and exponents of Elgot algebras can be formed in a canonical way.

▶ **Lemma 11.** *Given two Elgot algebras $(A, (\text{–})^\sharp)$ and $(B, (\text{–})^\sharp)$ and an object $X \in |\mathbf{C}|$,*

1. *$(A \times B, (\text{–})^{\hat\sharp})$ is an Elgot algebra with $h^{\hat\sharp} = \langle((\mathsf{fst} + \mathsf{id})\ h)^\sharp, ((\mathsf{snd} + \mathsf{id})\ h)^\sharp\rangle$ for any $h\colon Z \to A \times B + Z$.*
2. *If A^X exists then $(A^X, (\text{–})^{\hat\sharp})$ is an Elgot algebra with $h^{\hat\sharp} = \mathsf{curry}((\mathsf{ev} + \mathsf{id})\ \mathsf{dstl}\ (h \times \mathsf{id}))^\sharp$ for any $h\colon Z \to A^X + Z$.*

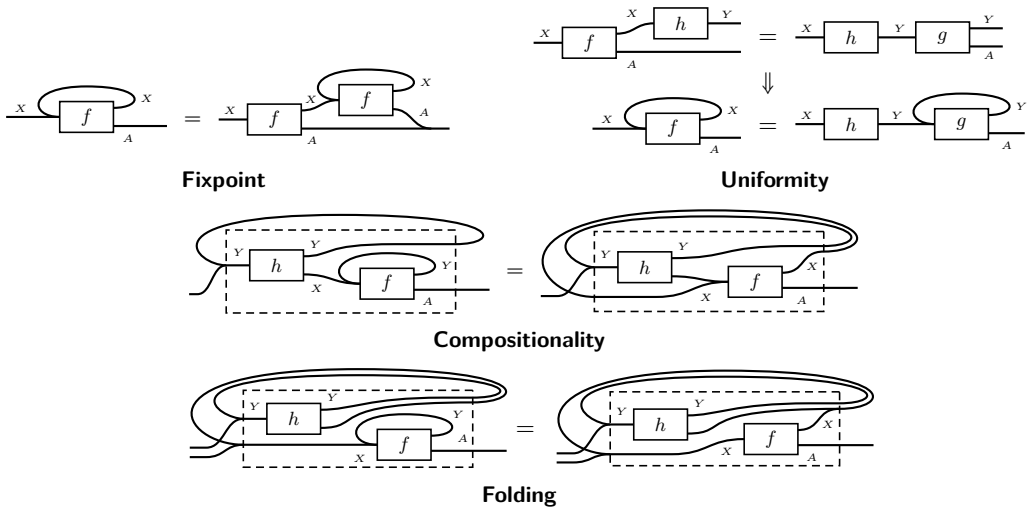


Figure 1 Laws of (unguarded) Elgot algebras.

Every Elgot algebra $(A, (-)^\sharp)$ comes together with a divergence constant $\bot\colon 1 \to A = (\mathsf{inr}\colon 1 \to A + 1)^\sharp$. Note that $\bot$ is automatically preserved by Elgot algebra morphisms.

By omitting the not entirely self-motivating **Compositionality** (or **Folding**) law, we obtain what we dub *uniform-iteration algebras*. As we see later, this law is automatic for free uniform-iteration algebras.

▶ **Definition 12** (Uniform-Iteration Algebras). *A* uniform-iteration algebra *is a tuple* $(A, (-)^\sharp)$ *as in Definition 9 but* $(-)^\sharp$ *is only required to satisfy* **Fixpoint** *and* **Uniformity**. *Morphisms of uniform-iteration algebras are defined in the same way.*

5 The Initial Pre-Elgot Monad

The goal of this section is to show that free uniform-iteration algebras coincide with free Elgot algebras (Theorem 29), and enjoy a number of other characteristic properties. In particular, we characterize the functor sending any X to a free uniform-iteration algebra on X as an initial pre-Elgot monad. We define pre-Elgot monads as follows.

▶ **Definition 13** (Pre-Elgot Monads). *We call a monad* $\mathbf{T}$ pre-Elgot *if every* TX *is equipped with an Elgot algebra structure, in such a way that* $h^\star\ f^\sharp = ((h^\star + \mathsf{id})\ f)^\sharp$ *for any* $f\colon Z \to TX + Z$ *and any* $h\colon X \to TY$. *A pre-Elgot monad* $\mathbf{T}$ *is* strong pre-Elgot *if* $\mathbf{T}$ *is strong as a monad and strength is iteration preserving.*

Pre-Elgot monads are to be compared with Elgot monads, which support a stronger type profile for the iteration operator, and satisfy more sophisticated axioms.

▶ **Definition 14** (Elgot Monads [15, 4]). *A monad* $\mathbf{T}$ *is an* Elgot monad *if it is equipped with an iteration operator sending each* $f\colon X \to T(Y + X)$ *to* $f^\dagger\colon X \to TY$ *and satisfying:*

- (**Fixpoint**) $f^\dagger = [\eta, f^\dagger]^\star\ f$;
- (**Naturality**) $g^\star\ f^\dagger = ([(T\,\mathsf{inl})\ g, \eta\ \mathsf{inr}]^\star\ f)^\dagger$ *for* $f\colon X \to T(Y + X)$, $g\colon Y \to TZ$;
- (**Codiagonal**) $(T[\mathsf{id}, \mathsf{inr}]\ f)^\dagger = f^{\dagger\dagger}$ *for* $f\colon X \to T((Y + X) + X)$;
- (**Uniformity**) $f\ h = T(\mathsf{id}+h)\ g$ *implies* $f^\dagger\ h = g^\dagger$ *for* $f\colon X \to T(Y+X)$, $g\colon Z \to T(Y+Z)$ *and* $h\colon Z \to X$.

If $\mathbf{T}$ *is additionally strong then* $\mathbf{T}$ *is strong Elgot if moreover:*

- (**Strength**) $\tau\ (\mathsf{id} \times f^\dagger) = ((T\,\mathsf{dstr})\ \tau\ (\mathsf{id} \times f))^\dagger$ *for any* $f\colon X \to T(Y + X)$.

▶ **Proposition 15.** *(Strong) Elgot monads are (strong) pre-Elgot under* $f^\sharp = ([T\,\mathsf{inl}, \eta\,\mathsf{inr}]\ f)^\dagger$.

It has been argued [17, 20] that strong Elgot monads are minimal semantic structures for interpreting effectful while-languages. In that sense, we acknowledge an expressivity gap between Elgot and pre-Elgot monads, which generally happen to be too weak. We will consider approaches to close this gap, in particular by drawing on some versions of the axiom of countable choice. Even though, in general, the gap presumably cannot be closed, we regard the initial pre-Elgot monad to be an important notion, which arises from first principles and carries a very clear operational intuition. The discrepancy between pre-Elgot monads and Elgot monads seems to represent a very basic form of discrepancy between operational and denotational semantics. We thus find it important to conceptually delineate between Elgot monads and pre-Elgot monads, no matter how desirable it is to have them to be equivalent.

▶ **Lemma 16.** *If for every* $X \in |\mathbf{C}|$ *a free uniform-iteration algebra* KX *exists then* K *extends to a monad* $\mathbf{K}$ *whose algebras are precisely uniform-iteration algebras.*

As in the case of natural numbers, one cannot make much progress without stability.

▶ **Definition 17** (Stable Free Uniform-Iteration Algebras)**.** *A free uniform-iteration algebra* KY *over* Y *is* stable *if for every* $X \in |\mathbf{C}|$, $\mathsf{fst}\colon X \times KY \to X$ *is a free uniform-iteration algebra in the slice category* $\mathbf{C}/X$.

▶ **Lemma 18.** *For* $Y \in |\mathbf{C}|$, KY *is stable iff for every uniform-iteration* A *and every* $f\colon X \times Y \to A$, *there is unique iteration preserving* $f^\sharp\colon X \times KY \to A$ *such that* $f = f^\sharp\ (\mathsf{id} \times \eta)$.

Using Lemma 11, it is easy to show that in Cartesian closed categories every KX is stable. For the rest of the section, we assume that all free uniform-iteration algebras KX exist and are stable.

▶ **Proposition 19.** *The monad* $\mathbf{K}$ *is strong, with the components of strength* $\tau\colon X \times KY \to K(X \times Y)$ *uniquely identified by the conditions:*

$$\tau\ (\mathsf{id} \times \eta) = \eta, \qquad \tau\ (\mathsf{id} \times h^\sharp) = ((\tau + \mathsf{id})\ \mathsf{dstr}\ (\mathsf{id} \times h))^\sharp \qquad (h\colon Z \to KY + Z)$$

Proof. In the notation of Lemma 18 we define strength of $\mathbf{K}$ as $(\eta\colon X \times Y \to K(X \times Y))^\sharp$. The axioms of strength are easy to verify. ◀

As a next step, we show that $\mathbf{K}$ is an equational lifting monad in the sense of Bucalo et al [9]. This means precisely that $\mathbf{K}$ is commutative and satisfies the equational law:

$$\tau\ \Delta = K\langle \eta, \mathsf{id}\rangle. \tag{1}$$

This law is rather restrictive, and roughly means that some form of non-termination is the only possible effect of the monad. Proving (1) is nontrivial. The key step is the following property, which allows for splitting a loop involving a product of algebras into two loops.

▶ **Lemma 20.** *Given uniform-iteration algebras* A *and* B, $f\colon Z \to A \times B + Z$ *and* $h\colon A \times B \to C$, $((h + \mathsf{id})\ f)^\sharp = ((h + \mathsf{id})\ \mathsf{dstr}\ (\mathsf{id} \times (\mathsf{snd} + \mathsf{id})\ f))^\sharp\ \langle((\mathsf{fst} + \mathsf{id})\ f)^\sharp, \mathsf{id}\rangle$.

▶ **Lemma 21.** *Given* $X, Z \in |\mathbf{C}|$, *and* $h\colon Z \to KX + Z$, *then* $\tau\ \langle h^\sharp, h^\sharp\rangle = ((\tau\ \Delta + \mathsf{id})\ h)^\sharp$.

Proof. It follows from Lemma 20 that $((\tau + \mathsf{id})\ \mathsf{dstr}\ (\mathsf{id} \times h))^\sharp\ \langle h^\sharp, \mathsf{id}\rangle = ((\tau\ \Delta + \mathsf{id})\ h)^\sharp$. On the other hand, by Proposition 19, $((\tau + \mathsf{id})\ \mathsf{dstr}\ (\mathsf{id} \times h))^\sharp\ \langle h^\sharp, \mathsf{id}\rangle = \tau\ \langle h^\sharp, h^\sharp\rangle$. By combining the last two identities, we obtain the goal. ◀

▶ **Theorem 22.** **K** *is an equational lifting monad.*

Proof. Let us sketch the proof of (1). Since $K\langle\eta,\mathsf{id}\rangle = (\eta\ \langle\eta,\mathsf{id}\rangle)^\star$, using the definition of Kleisli star for K, it suffices to show that $\tau\ \Delta$ is a unique iteration preserving morphism for which $\eta\ \langle\eta,\mathsf{id}\rangle = \tau\ \Delta\ \eta$. Indeed, $\tau\ \Delta\ \eta = \tau\ (\mathsf{id}\times\eta)\ \langle\eta,\mathsf{id}\rangle = \eta\ \langle\eta,\mathsf{id}\rangle$, and $\tau\ \Delta$ is iteration preserving by Lemma 21. ◀

The fact that **K** is an equational lifting monad has a number of implications, in particular, the Kleisli category of **K** is a *restriction category* [13]. That is, we can calculate the *domain (of definiteness)*, represented by an idempotent Kleisli morphism as follows: given $f\colon X\to KY$,

$$\mathsf{dom}\ f = (K\,\mathsf{fst})\ \tau\ \langle\mathsf{id}, f\rangle\colon X\to KX,$$

We additionally use the notation $f\mathbin{\downharpoonleft} g = \mathsf{fst}^\star\ \tau\ \langle f,g\rangle$, meaning: restrict f to the domain of g. It is easy to see that $\mathsf{dom}\ f = \eta\mathbin{\downharpoonleft} f$ and $f\mathbin{\downharpoonleft} g = f^\star\ (\mathsf{dom}\ g)$. Let $f\sqsubseteq g$ abbreviate $f = g\mathbin{\downharpoonleft} f$. Under this definition, every $\mathbf{C}(X,KY)$ is partially ordered, which is a general fact about restriction categories. In our case, moreover, this partial order additionally has a bottom element $\bot = \mathsf{inr}^\sharp$; $\mathsf{dom}(\eta\ f) = \eta$ for any $f\colon X\to KY$, and $\mathsf{dom}\ f\sqsubseteq\eta$ for any f.

▶ **Proposition 23.** *The Kleisli category of* **K** *is enriched over pointed partial orders and strict monotone maps. Moreover, strength preserves* $\bot$ *and* $\sqsubseteq$ *as follows:*

$$\tau\ (\mathsf{id}\times\bot) = \bot \qquad\qquad f\sqsubseteq g \quad \textit{implies} \quad \tau\ (\mathsf{id}\times f)\sqsubseteq\tau\ (\mathsf{id}\times g)$$

▶ **Corollary 24.** $K\emptyset\cong 1$.

Proof. Since $!\bot = \mathsf{id}\colon 1\to 1$ and $\bot! = \mathsf{id}\colon K\emptyset\to K\emptyset$, we obtain an isomorphism $K\emptyset\cong 1$. ◀

▶ **Proposition 25.** *The monad* **K** *is* copyable *and* weakly discardable *[18], i.e.:* $\hat\tau^\star\ \tau\ \Delta = K\Delta$ *and* $(K\,\mathsf{fst})\ \hat\tau^\star\ \tau\ \langle f,g\rangle\sqsubseteq f$ *for* $f\colon X\to KY$ *and* $g\colon X\to KZ$.

▶ **Definition 26** (Bounded Iteration)**.** *Let* A *be a* pointed object*, i.e. an object with a canonical map* $\bot\colon 1\to A$*. Then we define* bounded iteration $(\text{–})^{\langle\sharp\rangle}\colon\mathbf{C}(X,A+X)\to\mathbf{C}(X\times\mathbb{N},A)$ *by primitive recursion as follows:*

$$f^{\langle\sharp\rangle}(x,\mathsf{o}) = \bot \qquad\qquad f^{\langle\sharp\rangle}(x,\mathsf{s}\,n) = \mathsf{case}\ f(x)\ \mathsf{of}\ \mathsf{inl}\ a\mapsto a;\ \mathsf{inr}\ y\mapsto f^{\langle\sharp\rangle}(y,n).$$

Intuitively, $f^{\langle\sharp\rangle}(x,n)$ behaves as $f^\sharp(x)$ except that at each iteration the counter n is decreased, and $\bot$ is returned once $n=\mathsf{o}$. We next show that $f^\sharp(x)$ is in a suitable sense a limit of the $f^{\langle\sharp\rangle}(x,n)$ as n tends to infinity. This is, of course, a form of *Kleene fixpoint theorem*.

▶ **Theorem 27** (Kleene Fixpoint Theorem)**.** *Given* $f\colon X\to KY+X$*, and* $g\colon X\to KY$*, (i)* $f^{\langle\sharp\rangle}\sqsubseteq f^\sharp\ \mathsf{fst}$*, and (ii)* $f^{\langle\sharp\rangle}\sqsubseteq g\ \mathsf{fst}$ *implies* $f^\sharp\sqsubseteq g$.

▶ **Corollary 28.** *Given* $f\colon X\to KY+X$*,* $f^\sharp\colon X\to KY$ *is the least pre-fixpoint of the map* $[\mathsf{id},\text{–}]\ f\colon\mathbf{C}(X,KY)\to\mathbf{C}(X,KY)$.

Finally, we obtain

▶ **Theorem 29.** **K** *is an initial pre-Elgot monad and an initial strong pre-Elgot monad.*

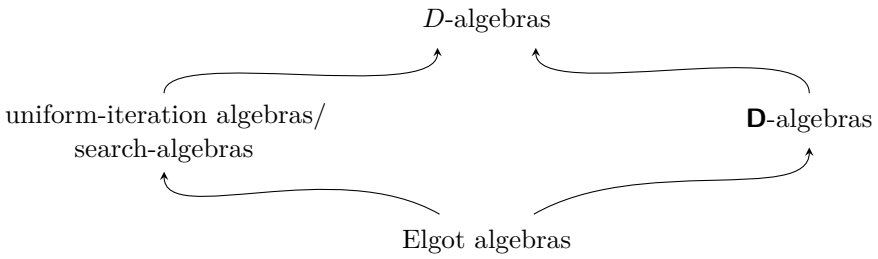


Figure 2 Connections between classes of D-algebras.

6 Quotienting the Delay Monad

By Theorem 8, Id-guarded Elgot algebras are precisely the **D**-algebras. We proceed to characterize uniform-iteration and Elgot algebras as certain D-algebras, which we dub *search-algebras*. Intuitively, modulo identification of DA with a set of streams from $(A+1)^{\mathbb{N}}$, a search-algebra structure $a\colon DA \to A$ is guaranteed to find the first element in the stream of the form $\mathsf{inl}\, a$ if it exists. We expect that this notion can be formulated more generally, but we do not pursue it here.

▶ **Definition 30** (Search-Algebra)**.** *We call a D-algebra $(A, a\colon DA \to A)$ a* search-algebra *if it satisfies the conditions: $a\ \mathsf{now} = \mathsf{id}$, $a \triangleright = a$. Search-algebras form a full subcategory of the category of all D-algebras.*

Uniform-iteration algebras capture the structure of search-algebras independently of the assumption that D exists. This and further connections between categories of D-algebras illustrated in Fig. 2 (arrows indicate full embeddings of categories) are formalized as follows.

▶ **Proposition 31.**

1. *The categories of uniform-iteration algebras and search-algebras are isomorphic under:*

$$(A, (-)^\sharp) \mapsto (A,\ \mathsf{out}^\sharp\colon DA \to A),$$
$$(A, a\colon DA \to A) \mapsto (A,\ a\ \ \mathsf{coit}(-)\colon \mathbf{C}(X, A+X) \to \mathbf{C}(X, A)).$$

2. *Elgot algebras are precisely those D-algebras, which are search-algebras and* **D***-algebras.*

▶ **Lemma 32.** *Every Elgot algebra $(DA, a\colon DA \to A)$ satisfies $a\ \iota^\star = a\ (D\,\mathsf{fst})$.*

We proceed to model the construction of quotienting **D** by weak bisimilarity $\approx$, previously described in type-theoretic terms [11]. Modulo identification of DX with the object of those streams $\sigma\colon \mathbb{N} \to X+1$ for which $\sigma(n) \neq \mathsf{inr}\star$ for at most one n, $\approx$ can be described as follows: $\sigma \approx \sigma'$ if for every a, $\sigma(n) = a$ for some n iff $\sigma'(n) = a$ for some n.

Recall the embedding $\iota\colon X \times \mathbb{N} \hookrightarrow DX$, and define the quotient of DX by the coequalizer

$$D(X \times \mathbb{N}) \underset{D\,\mathsf{fst}}{\overset{\iota^\star}{\rightrightarrows}} DX \xrightarrow{\ \rho_X\ } \widetilde{D}X \tag{2}$$

which we assume to exist and be preserved by products. It is then straightforward that $\widetilde{D}$ is a functor and ρ_X is natural in X. It also follows that $X \xrightarrow{\mathsf{now}} DX \xrightarrow{\rho} \widetilde{D}X$ is strong. Following tradition, we denote $\widetilde{D}1$ as Σ.

▶ **Lemma 33.** $\rho \triangleright = \rho$.

Defining ρ as a coequalizer of $\triangleright$ and id in the first place does not seem to be sufficient, though, in particular, for showing the following property. We leave open the question of identifying conditions under which it is possible.

▶ **Proposition 34.** *The following is a coequalizer:*

$$D(X + (X \times \mathbb{N} + X \times \mathbb{N})) \underset{[\eta,[\eta\ \ \mathsf{fst},\iota\ \ (\mathsf{id}\times\mathsf{s})]]^\star}{\overset{[\eta,[\iota\ \ (\mathsf{id}\times\mathsf{s}),\eta\ \ \mathsf{fst}]]^\star}{\rightrightarrows}} DX \xrightarrow{\rho_X} \widetilde{D}X \tag{3}$$

The last proposition brings the definition of ρ in accordance with the intuition; the coproduct $X + (X \times \mathbb{N} + X \times \mathbb{N})$ covers three alternatives for $\sigma \approx \sigma'$: either $\sigma = \sigma'$, or σ terminates earlier than σ' by a specified number, or the other way around. It can be verified that the embedding $D(X + (X \times \mathbb{N} + X \times \mathbb{N})) \hookrightarrow DX \times DX$ is an internal equivalence relation.

▶ **Theorem 35.** *The following conditions are equivalent:*

1. *for every X, coequalizer (2) is preserved by D;*
2. *every $\widetilde{D}X$ extends to a search-algebra, so that each ρ_X is a D-algebra morphism;*
3. *for every X, $(\widetilde{D}X, \rho\ \mathsf{now}\colon X \to \widetilde{D}X)$ is a stable free Elgot algebra on X, ρ_X is a D-algebra morphism and $\rho_X = ((\rho_X\ \ \mathsf{now} + \mathsf{id})\ \ \mathsf{out})^\sharp$;*
4. *$\widetilde{D}$ extends to a strong monad, so that ρ is a strong monad morphism.*

If the equivalent conditions of Theorem 35 are satisfied, we obtain an explicit construction of the initial pre-Elgot monad $\mathbf{K}$, which we explored previously. Let us consider concrete examples.

▶ **Example 36** (Maybe-Monad)**.** Suppose that $\mathbf{C}$ is an LPO category, and recall that DX is isomorphic to $X \times \mathbb{N} + 1$. It is then easy to check that (2) exists, it is preserved by products, $\widetilde{D}X \cong X + 1$ and $\rho = \mathsf{fst} + \mathsf{id}\colon X + 1 \to X \times \mathbb{N} + 1$. Since D is the composition of $(- \times \mathbb{N})$ and $(- + 1)$, and both these functors preserve coequalizers (first as a left adjoint, and second by extensiveness of $\mathbf{C}$), D preserves (2). We thus obtain that the maybe-monad is an initial pre-Elgot monad. This covers instances of LPO categories from Example 6. Moreover, the initial pre-Elgot monad is in fact an initial Elgot monad in this case: the profiles of the iteration operators $(-)^\sharp$ and $(-)^\dagger$ agree up to rearrangement of summands, and the axioms of Definition 14 become the axioms of Definition 13, except for **Codiagonal**, which can be checked directly.

Note that Example 36 entails that the maybe-monad is the initial Elgot monad in $\mathbf{dCpo}$. This is a result of our assumption that $\mathbf{dCpo}$ is developed w.r.t. a classical set theory, which entails that $\mathbf{dCpo}$ is an LPO category. This would not be the case if we defined $\mathbf{dCpo}$ internally to a non-classical environment, which is indeed the core idea of synthetic domain theory.

Another direction for obtaining an Elgot monad from (2) is by using a suitable instance of the *axiom of countable choice*. In our setting this takes the following form.

▶ **Theorem 37.** *Suppose that the coequalizers (2) are preserved by the exponentiation $(-)^{\mathbb{N}}$.*

1. *The equivalent conditions of Theorem 35 hold, in particular, $\mathbf{K}$ is an initial (strong) pre-Elgot monad.*
2. *If every (3) is an effective quotient, i.e. $D(X + (X \times \mathbb{N} + X \times \mathbb{N}))$ is a kernel pair of ρ_X, then $\mathbf{K}$ is a strong Elgot monad with $f^\dagger$ being the least fixpoint of $[\eta, -]^\star\ f\colon \mathbf{C}(X, TY) \to \mathbf{C}(X, TY)$ for any $f\colon X \to T(Y + X)$.*

The effectiveness assumption in clause 2. is satisfied in any exact category (e.g. in any pretopos) – by definition, every internal equivalence relation there is effective.

▶ **Example 38.** Theorem 37 applies to **Top**, yielding a concrete description for **K**. Recall that in **Top**, coequalizers are computed as in **Set** and are equipped with the quotient topology. Note that DX is the set $X \times \mathbb{N} \cup \{\infty\}$ whose base opens are $\{(x,n) \mid x \in O\}$ and $\{(x,k) \mid x \in X, k \geqslant n\} \cup \{\infty\}$ with $n \in \mathbb{N}$ and O ranging over the opens of X. The collapse $\widetilde{D}X$ computed with (2) is thus the set $X \cup \{\infty\}$, whose opens are those of X and additionally the entire space $X \cup \{\infty\}$, in particular, $\widetilde{D}1$ is the *Sierpiński space*.

To obtain that (2) is preserved by $(-)^{\mathbb{N}}$, it suffices to show that the opens of $(X \cup \{\infty\})^{\mathbb{N}}$ are precisely those, whose inverse images under $\rho^{\mathbb{N}}$ are open. This is in fact true for any regular epi in **Top**. The effectiveness condition in 2. is not vacuous for **Top**, which is not an exact category (and not even regular), but it can be checked manually.

In every pretopos, preservation of (2) by $(-)^{\mathbb{N}}$ is a proper instance of the *internal axiom of countable choice*, or *internal projectivity of* $\mathbb{N}$, which means preservation of epis by $(-)^{\mathbb{N}}$, roughly because every pretopos is *exact* and our quotienting morphism ρ is associated with an internal equivalence relation by Proposition 34. Theorem 37 can thus be related to the existing result in synthetic domain theory, that Rosolini dominance, i.e. our Σ, is indeed a dominance [31], which applies to Hyland's *effective topos* [22], as it satisfies countable choice. Contrastingly, we cannot apply Theorem 37 to nominal sets, which falsify countable choice, however, as a Boolean topos, nominal sets fall into the scope of Example 36.

We currently do not have a concrete example of **K** being definable, but not being an Elgot monad. Theorem 37 and Example 36 indicate that a category to witness this must neither support excluded middle nor the axiom of countable choice.

7 Conclusions and Further Work

Iteration and iteration theories emerged as unifying concepts for computer science semantics and reasoning. By interpreting iteration suitably, one obtains a basic extensible equational logic of programs, shown to be sound and complete across various models [8]. Elgot monads implement this inherently algebraic view in the general categorical realm of abstract data types and effects. The class of Elgot monads (over a fixed category) is stable under various categorical constructions (monad transformers), and thus one can build new Elgot monads from old, but the simplest Elgot monad, the initial one, does not arise in this way.

Here, we proposed an approach to defining an initial iteration structure from first principles, characterized it in various ways, analysed conditions, under which it can be concretely described, and to yield an Elgot monad. Unsurprisingly, these conditions generally cannot be lifted, as the previous research in type theory indicates. We consider broadening the scope in which results about notions of partiality apply, and unifying both classical and non-classical models, as an important part of our contribution. Universal properties play a central role in category theory, but many important concepts are not covered by them. One example is Sierpiński space, which is fundamental in topology, duality theory and domain theory. It follows from our results, that it is in fact a free uniform-iteration algebra on one generator. We believe that the structure of our results can be reused in more sophisticated setting, such as semantics of *hybrid systems*, which require a notion of partiality, combined with continuous evolution, and rise semantic issues, structurally similar to those, we considered here [14]. Another potential for taking further the present work is to consider more general shapes of the basic functor (instead of the current $(X + -)$), prospectively leading to more sophisticated (non-)structural recursion scenarios (see e.g. [1]).

References

1 Jiří Adámek, Stefan Milius, and Lawrence S. Moss. On well-founded and recursive coalgebras. In Jean Goubault-Larrecq and Barbara König, editors, *Foundations of Software Science and Computation Structures*, pages 17–36. Springer International Publishing, 2020.

2 Jirí Adámek, Stefan Milius, and Jiri Velebil. Elgot algebras. *Log. Meth. Comput. Sci.*, 2, 2006.

3 Jırı Adámek, Stefan Milius, and Jırı Velebil. Iterative algebras: How iterative are they. *Theory Appl. Categ*, 19:61–92, 2008.

4 Jiří Adámek, Stefan Milius, and Jiří Velebil. Elgot theories: a new perspective of the equational properties of iteration. *Math. Struct. Comput. Sci.*, 21:417–480, 2011.

5 Thorsten Altenkirch, Nils Danielsson, and Nicolai Kraus. Partiality, revisited - the partiality monad as a quotient inductive-inductive type. In Javier Esparza and Andrzej Murawski, editors, *Foundations of Software Science and Computation Structures, FOSSACS 2017*, volume 10203 of *LNCS*, pages 534–549, 2017.

6 Steve Awodey. *Category Theory*. Oxford University Press, Inc., New York, NY, USA, 2nd edition, 2010.

7 Andrej Bauer. An injection from the Baire space to natural numbers. *Mathematical Structures in Computer Science*, 25(7):1484–1489, 2015.

8 Stephen Bloom and Zoltán Ésik. *Iteration theories: the equational logic of iterative processes*. Springer, 1993.

9 Anna Bucalo, Carsten Führmann, and Alex K. Simpson. An equational notion of lifting monad. *Theor. Comput. Sci.*, 294(1/2):31–60, 2003.

10 Venanzio Capretta. General recursion via coinductive types. *Log. Meth. Comput. Sci.*, 1(2), 2005.

11 James Chapman, Tarmo Uustalu, and Niccolò Veltri. Quotienting the delay monad by weak bisimilarity. In *Theoretical Aspects of Computing, ICTAC 2015*, volume 9399 of *LNCS*, pages 110–125. Springer, 2015.

12 James Chapman, Tarmo Uustalu, and Niccolò Veltri. Quotienting the delay monad by weak bisimilarity. *Mathematical Structures in Computer Science*, 29(1):67–92, 2019.

13 J Robin B Cockett and Stephen Lack. Restriction categories I: categories of partial maps. *Theoretical computer science*, 270(1-2):223–259, 2002.

14 Tim Lukas Diezel and Sergey Goncharov. Towards Constructive Hybrid Semantics. In Zena M. Ariola, editor, *5th International Conference on Formal Structures for Computation and Deduction (FSCD 2020)*, volume 167 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 24:1–24:19, Dagstuhl, Germany, 2020. Schloss Dagstuhl–Leibniz-Zentrum für Informatik.

15 Calvin Elgot. Monadic computation and iterative algebraic theories. In *Logic Colloquium 1973*, volume 80 of *Studies in Logic and the Foundations of Mathematics*, pages 175–230. Elsevier, 1975.

16 Martín H. Escardó and Cory M. Knapp. Partial Elements and Recursion via Dominances in Univalent Type Theory. In Valentin Goranko and Mads Dam, editors, *26th EACSL Annual Conference on Computer Science Logic (CSL 2017)*, volume 82 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 21:1–21:16, Dagstuhl, Germany, 2017. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik.

17 Sergey Goncharov, Christoph Rauch, and Lutz Schröder. Unguarded recursion on coinductive resumptions. In *Mathematical Foundations of Programming Semantics, MFPS 2015*, ENTCS, 2015.

18 Sergey Goncharov and Lutz Schröder. A relatively complete generic Hoare logic for order-enriched effects. In *Proc. 28th Annual Symposium on Logic in Computer Science (LICS 2013)*, pages 273–282. IEEE, 2013.

19 Sergey Goncharov, Lutz Schröder, Christoph Rauch, and Julian Jakob. Unguarded recursion on coinductive resumptions. *Logical Methods in Computer Science*, 14(3), 2018.

20 Sergey Goncharov, Lutz Schröder, Christoph Rauch, and Maciej Piróg. Unifying guarded and unguarded iteration. In Javier Esparza and Andrzej Murawski, editors, *Foundations of Software Science and Computation Structures, FoSSaCS 2017*, volume 10203 of *LNCS*, pages 517–533. Springer, 2017.

21 J. M. E. Hyland. First steps in synthetic domain theory. In *Category Theory*, volume 1144 of *LNM*, pages 131–156. Springer, 1992.

22 J.M.E. Hyland. The effective topos. In A.S. Troelstra and D. van Dalen, editors, *The L. E. J. Brouwer Centenary Symposium*, volume 110 of *Studies in Logic and the Foundations of Mathematics*, pages 165–216. Elsevier, 1982.

23 Peter T Johnstone. On a topological topos. *Proceedings of the London mathematical society*, 3(2):237–271, 1979.

24 Andrè Joyal, Ross Street, and Dominic Verity. Traced monoidal categories. *Mathematical Proceedings of the Cambridge Philosophical Society*, 119:447–468, April 1996.

25 Cory M. Knapp. *Partial functions and recursion in univalent type theory*. PhD thesis, University of Birmingham, UK, 2018.

26 Anders Kock. Strong functors and monoidal monads. *Archiv der Mathematik*, 23(1):113–120, 1972.

27 Bill Lawvere. An elementary theory of the category of sets. *Proceedings of the National Academy of Sciences of the United States of America*, 52:1506–1511, 1964.

28 Saunders Mac Lane. *Categories for the Working Mathematician*. Springer, 1971.

29 Eugenio Moggi. Notions of computation and monads. *Inf. Comput.*, 93:55–92, 1991.

30 Egbert Rijke and Bas Spitters. Sets in homotopy type theory. *Mathematical Structures in Computer Science*, 25(5):1172–1202, 2015.

31 Guiseppe Rosolini. *Continuity and Effectiveness in Topoi*. PhD thesis, University of Oxford, 1986.

32 Alex Simpson and Gordon Plotkin. Complete axioms for categorical fixed-point operators. In *Logic in Computer Science, LICS 2000*, pages 30–41, 2000.

33 The Univalent Foundations Program. *Homotopy Type Theory: Univalent Foundations of Mathematics*. `https://homotopytypetheory.org/book`, Institute for Advanced Study, 2013.

34 Tarmo Uustalu. Generalizing substitution. *ITA*, 37:315–336, 2003.

35 Tarmo Uustalu and Niccolò Veltri. The delay monad and restriction categories. In Dang Van Hung and Deepak Kapur, editors, *Theoretical Aspects of Computing – ICTAC 2017*, pages 32–50. Springer International Publishing, 2017.

36 Tarmo Uustalu and Niccolò Veltri. Partiality and container monads. In Bor-Yuh Evan Chang, editor, *APLAS*, volume 10695 of *Lecture Notes in Computer Science*, pages 406–425. Springer, 2017.

37 Niccoló Veltri. *A Type-Theoretical Study of Nontermination*. PhD thesis, Tallinn University of Technology, 2017.

Powerset-Like Monads Weakly Distribute over Themselves in Toposes and Compact Hausdorff Spaces

Alexandre Goy
Université Paris-Saclay, CentraleSupélec, MICS, France

Daniela Petrişan
Université de Paris, IRIF, France

Marc Aiguier
Université Paris-Saclay, CentraleSupélec, MICS, France

Abstract

The powerset monad on the category of sets does not distribute over itself. Nevertheless a weaker form of distributive law of the powerset monad over itself exists and it essentially stems from the canonical Egli-Milner extension of the powerset to the category of relations. On the other hand, any regular category yields a category of relations, and some regular categories also possess a powerset-like monad, as is the Vietoris monad on compact Hausdorff spaces. We derive the Egli-Milner extension in three different frameworks : sets, toposes, and compact Hausdorff spaces. We prove that it corresponds to a monotone weak distributive law in each case by showing that the multiplication extends to relations but the unit does not. We provide an application to coalgebraic determinization of alternating automata.

2012 ACM Subject Classification Theory of computation → Formal languages and automata theory

Keywords and phrases Egli-Milner relation, weak extension, weak distributive law, weak lifting, powerset monad, Vietoris monad, topos, alternating automaton, generalized determinization

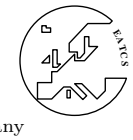
Digital Object Identifier 10.4230/LIPIcs.ICALP.2021.132

Category Track B: Automata, Logic, Semantics, and Theory of Programming

1 Introduction

Composing monads is usually achieved using distributive laws. Unfortunately, sometimes these do not exist. It is known since the work of Varacca [24] that there is no distributive law between the monad $\mathbf{D}$ of probability distributions and the powerset monad $\mathbf{P}$. The proof, attributed to Plotkin, relies on a manipulation of the naturality squares of the unit of $\mathbf{D}$. More recently, [14] showed that there is no distributive law of the powerset $\mathbf{P}$ over itself, and even more nor over its iterations. More negative results for other Set-based algebraic theories are presented in [26].

One way to circumvent such negative results is to compose monads using weaker forms of distributive laws. In the definition of a distributive law between monads we have four axioms specifying the interactions of the law with the units, respectively with the multiplications of the two monads. In a weak distributive law, an axiom involving the unit of one of the monads is dropped. In our previous paper [11] we exhibited a canonical weak distributive law between the monads $\mathbf{D}$ and $\mathbf{P}$. It comes as no surprise that the axiom that is dropped from the definition of such a law is the one involving the unit of $\mathbf{D}$ – on which the argument of Plotkin relied. Our work in turn, was based on Garner's results [10]. In loc. cit. he exhibited a weak distributive law between the powerset monad and the ultrafilter monad $\boldsymbol{\beta}$. This leads to a weak lifting of the powerset to the category of Eilenberg-Moore algebras of $\boldsymbol{\beta}$, that is,



to the category of compact Hausdorff spaces KHaus. This weak lifting is the Vietoris monad and is indeed the closest there is to a powerset-like monad on KHaus. In [10] it is also shown that the powerset monad on Set weakly distributes over itself.

The weak distributive laws in [10] and [11] were all of the form $\mathbf{TP} \to \mathbf{PT}$, where $\mathbf{P}$ is the powerset monad on Set. The recipe for obtaining these laws was based on the monotone weak extension of the monad $\mathbf{T}$ to the category of relations, this is, the Kleisli category of $\mathbf{P}$.

The motivation of the present paper is to understand composition of monads via weak distributive laws in other settings than for Set-based monads. In particular our aim here is to build the technology necessary for combining various forms of nondeterminism (ordinary and probabilistic) in a continuous setting.

An obvious starting point is to consider the category of compact Hausdorff spaces KHaus, where we have the Vietoris monad $\mathbf{V}$ whose Kleisli category $\mathsf{Kl}(\mathbf{V})$ can be seen as a category of relations satisfying additional continuity constraints. The first question we have asked ourselves is whether the Vietoris monad $\mathbf{V}$ weakly distributes over itself. Can we extend Garner's result from Set to KHaus? It turns out the answer is positive, see Theorem 24. In the process we have used various results from the literature, in particular, the work of [7] for extending functors on a regular category C to the category of relations $\mathsf{Rel}(\mathsf{C})$. Nevertheless, the results are far from immediate since $\mathsf{Kl}(\mathbf{V})$ is only a subcategory of $\mathsf{Rel}(\mathsf{KHaus})$, hence some additional work is needed to obtain the canonical extension of $\mathbf{V}$ to $\mathsf{Kl}(\mathbf{V})$. We also notice that despite the fact that the Vietoris functor does not preserve pullbacks (see also [6]), it nearly preserves pullbacks, and this is exactly the condition needed in [7] to provide the relational extension. Once the monotone extension of the monad $\mathbf{V}$ to $\mathsf{Kl}(\mathbf{V})$ is found, we can provide the weak distributive law via the same mechanism as in [10].

Another extension we provide is for the powerset-like monad on a topos C. This monad is defined on objects as Ω^X where Ω is the subobject classifier of the topos. Notice that we are not considering the contravariant powerset-like functor usually appearing in topos theory, but rather the monad $\mathbf{\exists}$ of [19]. Its Kleisli category is simply $\mathsf{Rel}(\mathsf{C})$ and, as far as the underlying functor is concerned, a monotone extension is obtained again by leveraging the work in [7]. To obtain the extension of the *monad* $\mathbf{\exists}$, we need to investigate the properties of its unit and of its multiplication. As far as the multiplication is concerned, we can internalize the proof from the Set case, using the internal logic of the topos. As far as the unit is concerned, we show that it has the required property for obtaining a strong extension of the monad, only when the topos is degenerate. In all other meaningful cases, we thus only obtain a weak extension of $\mathbf{\exists}$ to $\mathsf{Kl}(\mathbf{\exists})$, and hence a weak distributive law of $\mathbf{\exists}$ over itself, see Theorem 13.

The paper is structured as follows. In Section 2 we recall the necessary preliminaries on weak distributive laws and relations in regular categories. In Section 3 we recall the weak distributive law of the powerset over itself and provide an application to coalgebraic determinization of alternating automata. We find it instructive to understand the proofs first in Set, as they will serve as a basis for the generalization to toposes, performed in Section 4. In Section 5 we provide the weak distributive law of the Vietoris functor over itself and we conclude with a summary and directions for future work in Section 6.

2 Preliminaries

Notations

For a relation $R \subseteq X \times Y$ between two sets and $A \subseteq X$, $B \subseteq Y$, we write $R[A] = \{y \in Y \mid (x, y) \in R \text{ and } x \in A\}$ and $R^{-1}[B] = \{x \in X \mid (x, y) \in R \text{ and } y \in B\}$. The complement of $A \subseteq X$ is denoted by A^c. In the whole paper, C denotes a generic category, $F, G : \mathsf{C} \to \mathsf{C}$ denote functors and $\alpha : F \to G$ denotes a natural transformation. Identity morphisms, functors and natural transformations will all be denoted by 1.

2.1 (Weak) Extensions, (Weak) Distributive Laws, (Weak) Liftings

We assume the reader is familiar with the basic theory of monads, and fix here some notations. A *monad* is a triple $\mathbf{T} = (T, \eta^{\mathbf{T}}, \mu^{\mathbf{T}})$ where $T : \mathsf{C} \to \mathsf{C}$ is a functor, $\eta^{\mathbf{T}} : 1 \to T$ and $\mu^{\mathbf{T}} : TT \to T$ are natural transformations called respectively unit and multiplication and equations $\mu^{\mathbf{T}} \circ T\eta^{\mathbf{T}} = 1 = \mu^{\mathbf{T}} \circ \eta^{\mathbf{T}}T$, $\mu^{\mathbf{T}} \circ T\mu^{\mathbf{T}} = \mu^{\mathbf{T}} \circ \mu^{\mathbf{T}}T$ hold. In the following we fix two monads $\mathbf{T}$ and $\mathbf{S}$ on C. The *Kleisli category* of $\mathbf{S}$ is denoted by $\mathsf{Kl}(\mathbf{S})$. A morphism in $\mathsf{Kl}(\mathbf{S})$ will be denoted by $X \nrightarrow Y$ and corresponds to a morphism $X \to SY$ in C. The Kleisli free and forgetful functor are denoted respectively by $F_{\mathbf{S}} : \mathsf{C} \to \mathsf{Kl}(\mathbf{S})$ and $U_{\mathbf{S}} : \mathsf{Kl}(\mathbf{S}) \to \mathsf{C}$. The *Eilenberg-Moore category* of $\mathbf{T}$ is denoted by $\mathsf{EM}(\mathbf{T})$, objects are algebras (X, x) where X is an object of C and $x : TX \to X$ satisfies $x \circ \eta^{\mathbf{T}}_X = 1_X$ and $x \circ \mu^{\mathbf{T}}_X = x \circ Tx$. The Eilenberg-Moore free and forgetful functor are denoted respectively by $F^{\mathbf{T}} : \mathsf{C} \to \mathsf{EM}(\mathbf{T})$ and $U^{\mathbf{T}} : \mathsf{EM}(\mathbf{T}) \to \mathsf{C}$.

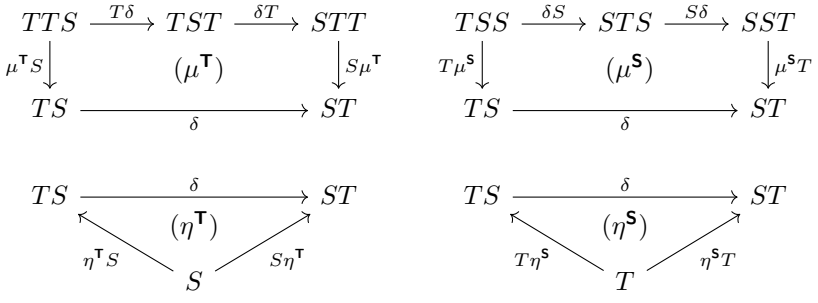
▶ **Example 1.** The *powerset monad* $\mathbf{P}$ on the category of sets and functions Set is defined as follows. The functor P maps a set X to the set of its subsets and acts on functions by taking direct images. Unit is given by the singleton operation $\eta^{\mathbf{P}}_X(x) = \{x\}$ and multiplication by union $\mu^{\mathbf{P}}_X(\mathcal{A}) = \bigcup \mathcal{A}$.

Monads are not stable under composition. However, Beck introduced the framework of distributive laws [1] as a tool to generate composite monads. Distributive laws are actually one face of a three-sided coin comprising also extensions and liftings. Interestingly enough, each component of this triptych can be weakened in such a way that the correspondence still stands between weak distributive laws, weak extensions and weak liftings [10]. The rest of this section aims at jointly recalling both the usual and the weakened framework.

An extension of F to $\mathsf{Kl}(\mathbf{S})$ is a functor $\overline{F} : \mathsf{Kl}(\mathbf{S}) \to \mathsf{Kl}(\mathbf{S})$ such that $\overline{F}F_{\mathbf{S}} = F_{\mathbf{S}}F$. Similarly, an extension of $\alpha : F \to G$ is a natural transformation $\overline{\alpha} : \overline{F} \to \overline{G}$ such that the equation $\overline{\alpha}F_{\mathbf{S}} = F_{\mathbf{S}}\alpha$ holds.

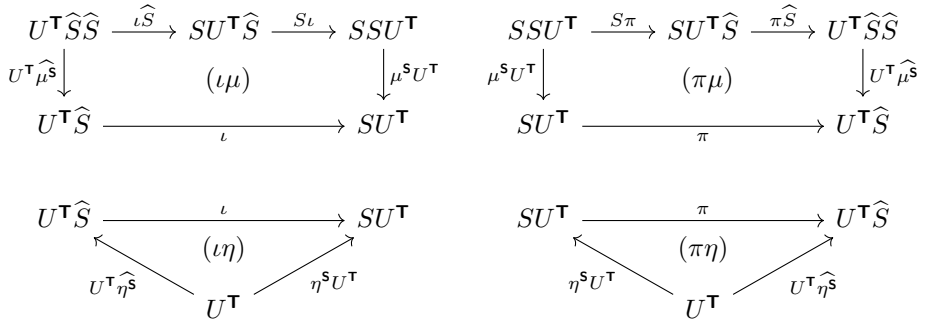
▶ **Definition 2** (Extension). *An* extension *of* $\mathbf{T}$ *to* $\mathsf{Kl}(\mathbf{S})$ *is a monad* $\overline{\mathbf{T}}$ *on* $\mathsf{Kl}(\mathbf{S})$ *whose functor, unit and multiplication are extensions of those of* $\mathbf{T}$*. A* weak extension *only requires the extension of the functor and of the multiplication of* $\mathbf{T}$*.*

▶ **Definition 3** (Distributive law). *A* distributive law *of type* $\mathbf{TS} \to \mathbf{ST}$ *is a natural transformation* $\delta : TS \to ST$ *such that the four following diagrams commute*



A weak distributive law *only requires the* $(\eta^{\mathbf{S}})$, $(\mu^{\mathbf{S}})$ *and* $(\mu^{\mathbf{T}})$ *diagrams.*

▶ **Definition 4** (Lifting). *A* lifting *of* $\mathbf{S}$ *to* $\mathsf{EM}(\mathbf{T})$ *is a monad* $\widehat{\mathbf{S}} : \mathsf{EM}(\mathbf{T}) \to \mathsf{EM}(\mathbf{T})$ *such that* $U^{\mathbf{T}}\widehat{S} = SU^{\mathbf{T}}$, $U^{\mathbf{T}}\widehat{\eta^{\mathbf{S}}} = \eta^{\mathbf{S}}U^{\mathbf{T}}$ *and* $U^{\mathbf{T}}\widehat{\mu^{\mathbf{S}}} = \mu^{\mathbf{S}}U^{\mathbf{T}}$. *A* weak lifting *of* $\mathbf{S}$ *on* $\mathbf{T}$ *is a monad* $\widehat{\mathbf{S}} : \mathsf{EM}(\mathbf{T}) \to \mathsf{EM}(\mathbf{T})$ *along with two natural transformations* $\pi : SU^{\mathbf{T}} \to U^{\mathbf{T}}\widehat{S}$, $\iota : U^{\mathbf{T}}\widehat{S} \to SU^{\mathbf{T}}$ *such that* $\pi \circ \iota = 1$ *and the following diagrams commute:*



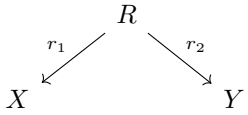
Recall that an idempotent morphism $e : X \to X$ *splits* if there is an object Y and morphisms $f : X \to Y$, $g : Y \to X$ such that $g \circ f = e$ and $f \circ g = 1_Y$.

▶ **Theorem 5** ([1, 10]). *There is a bijective correspondence between extensions of* $\mathbf{T}$ *to* $\mathsf{Kl}(\mathbf{S})$, *distributive laws of type* $\mathbf{TS} \to \mathbf{ST}$, *and liftings of* $\mathbf{S}$ *to* $\mathsf{EM}(\mathbf{T})$. *This extends to a bijective correspondence between weak extensions, weak distributive laws, and (if all idempotents split in* C*) weak liftings.*

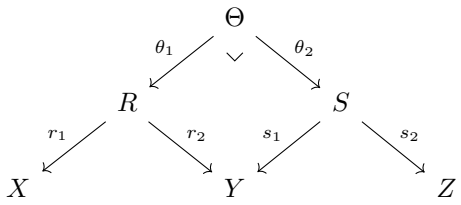
2.2 Relations in Regular Categories

From now on, we make the assumption that C is a *regular* category, this is, finitely complete with pullback-stable image factorizations. (Note that in a regular category, regular epis and strong epis coincide; notions will be phrased in terms of strong epis, as in [7].) In particular, C has all pullbacks: in this context, a *weak pullback* (resp. *near pullback*) is a commutative square such that the mediating morphism into the pullback is a split epi (resp. strong epi). The functor F is *weakly cartesian* (resp. *nearly cartesian*) if it maps pullbacks into weak pullbacks (resp. near pullbacks), and α is weakly cartesian (resp. nearly cartesian) if its naturality squares are weak pullbacks (resp. near pullbacks). Note that in the literature, strong epis are sometimes called *covers*, and within this terminology a nearly cartesian functor is a functor that *covers pullbacks* [22].

Regular categories have a (strong epi, mono) factorization system. In such a factorization $f = m \circ e$ we recall that the subobject defined by the mono m is the *image* of the morphism f. Note that these factorizations imply that idempotents split in C, so that Theorem 5 can be fully applied. As explained in [7], one can build a category $\mathsf{Rel}(\mathsf{C})$ whose morphisms will stand for relations. The objects of $\mathsf{Rel}(\mathsf{C})$ are the objects of C. A morphism $r : X \rightsquigarrow Y$ in $\mathsf{Rel}(\mathsf{C})$ is a subobject of $X \times Y$ in C and is called a relation – the notation $\rightsquigarrow$ tells relations apart from morphisms in C. We will not distinguish monomorphisms from their equivalence classes, hence a relation is equivalently a jointly monic span



The composition of relations $r = \langle r_1, r_2 \rangle : R \to X \times Y$ and $s = \langle s_1, s_2 \rangle : S \to Y \times Z$ is obtained using a pullback Θ, by taking the image of the morphism $\Theta \to X \times Z$ below:



This relational composition of r and s will be denoted by $s.r$. Identities are obtained via the diagonal monomorphism $\langle 1_X, 1_X \rangle : X \to X \times X$. There is a contravariant involution $-^\circ : \mathsf{Rel}(\mathsf{C})^{\mathsf{op}} \to \mathsf{Rel}(\mathsf{C})$ given by $X^\circ = X$ and $\langle r_1, r_2 \rangle^\circ = \langle r_2, r_1 \rangle$. The graph functor $\mathcal{G} : \mathsf{C} \to \mathsf{Rel}(\mathsf{C})$ is defined by $\mathcal{G}X = X$ and $\mathcal{G}f = \langle 1_X, f \rangle$ for any $f : X \to Y$. These two fundamental functors have a nice interplay, as for every $r : X \rightsquigarrow Y$ we have $r = \mathcal{G}r_2.(\mathcal{G}r_1)^\circ$. Most of the time, the mention of $\mathcal{G}$ will be omitted, e.g. the previous equation writes $r = r_2.r_1^\circ$ and functoriality of $\mathcal{G}$ writes $g.f = g \circ f$. Given two relations $r : R \to X \times Y$ and $s : S \to X \times Y$, the subobject order is defined by:

$$r \leq s \iff \exists h : R \to S, r = s \circ h \tag{1}$$

Accordingly, a functor $H : \mathsf{Rel}(\mathsf{C}) \to \mathsf{Rel}(\mathsf{C})$ is called *monotone* if $r \leq s \Rightarrow Hr \leq Hs$.

A *relational extension* of F is a monotone functor $\mathsf{Rel}(F) : \mathsf{Rel}(\mathsf{C}) \to \mathsf{Rel}(\mathsf{C})$ such that $\mathsf{Rel}(F)\mathcal{G} = \mathcal{G}F$. This actually forces $\mathsf{Rel}(F)X = FX$ and

$$\mathsf{Rel}(F)r = (Fr_2).(Fr_1)^\circ \tag{2}$$

so that F has at most one relational extension. Similarly, a relational extension of $\alpha : F \to G$ is a natural transformation $\mathsf{Rel}(\alpha) : \mathsf{Rel}(F) \to \mathsf{Rel}(G)$ such that $\mathsf{Rel}(\alpha)\mathcal{G} = \mathcal{G}\alpha$, and α has at most one such extension. Collecting results from the literature ([7, §4.3], [21, Corollary 1.5.7]) we get the following existence theorem

▶ **Theorem 6** (Existence of relational extensions). *A functor $F : \mathsf{C} \to \mathsf{C}$ on a regular category C has a (unique) relational extension if and only if F preserves strong epis and F is nearly cartesian. Provided these conditions hold for both F and G, a natural transformation $\alpha : F \to G$ has a (unique) relational extension if and only if α is nearly cartesian.*

Note that whenever $\mathsf{Rel}(F)$ and $\mathsf{Rel}(G)$ exist, then $\mathsf{Rel}(G)\mathsf{Rel}(F) = \mathsf{Rel}(GF)$ (see also [7, §4.4]).

3 Sets

There is no distributive law of type $\mathbf{PP} \to \mathbf{PP}$ on Set [14]. However, there is a weak distributive law recently described by Garner [10]. In this section, we detail how this law is obtained using Theorem 6. Our aim is twofold. First, we lay the ground for Sections 4 and 5 where we will generalize this reasoning in two different directions. Second, as an application we present how this weak distributive law allows to retrieve a known procedure that transforms alternating automata into non-deterministic automata [13].

In this section, C is the (regular) category Set of sets and functions and $\mathbf{S}$ is the powerset monad $\mathbf{P}$ defined in Example 1. It turns out that both $\mathsf{Kl}(\mathbf{P})$ and $\mathsf{Rel}(\mathsf{Set})$ can be identified to the category Rel of sets and relations. Under this identification we have $F_{\mathbf{P}} = \mathcal{G}$. In this

context a relation $R : X \nrightarrow Y$ is just a subset of the product $R \subseteq X \times Y$ and composition is defined by the usual formula $S \circ R = \{(x, z) \in X \times Z \mid (x, y) \in R \text{ and } (y, z) \in S\}$. A relational extension of a functor is nothing but a (Kleisli) extension that is monotone with respect to relation inclusion. The axiom of choice yields that all epis are split, henceforth any functor automatically preserves strong epis, and near pullbacks coincide with weak pullbacks. Theorem 6 therefore boils down to saying that F (resp. α) extends to Rel iff F (resp. α) is weakly cartesian – this is [10, Proposition 15].

Let $\mathbf{T}$ also be the powerset monad $\mathbf{P}$. The following example is essentially in [10] – however the proofs there are done with $\mathbf{T}$ being the *finite* powerset monad. An obvious consequence of the above paragraph is that $\mathbf{P}$ has a weak extension to Rel iff P and $\mu^{\mathbf{P}}$ are weakly cartesian, which both are known results, see e.g. [23] for P and [10] for $\mu^{\mathbf{P}}$. Further, the unit $\eta^{\mathbf{P}}$ is not nearly cartesian [10], so that the weak extension is not an extension. We recall these proofs here because they will be used in the next sections.

▶ **Proposition 7.** *The powerset functor P is weakly cartesian.*

Proof. Equivalently, P being nearly cartesian amounts to showing that for every $f : X \to Z$, $g : Y \to Z$ and $(A, B) \in PX \times PY$ such that $f[A] = g[B]$, there is $C \subseteq P := \{(x, y) \in X \times Y \mid f(x) = g(y)\}$ such that $\pi_1[C] = A$ and $\pi_2[C] = B$, where $\pi_1 : P \to X$, $\pi_2 : P \to Y$ are the projections from the pullback. One can easily check that $C = \pi_1^{-1}(A) \cap \pi_2^{-1}(B)$ completes the proof. ◀

▶ **Proposition 8.** *The unit $\eta^{\mathbf{P}}$ is not weakly cartesian.*

Proof. Consider the naturality square for the unique map $! : \{0, 1\} \to \{0\}$. The corresponding pullback $\{(0, \{0\}), (0, \{1\}), (0, \{0, 1\})\}$ has cardinality 3, so there cannot be a surjective map from $\{0, 1\}$ into it. ◀

▶ **Proposition 9.** *The multiplication $\mu^{\mathbf{P}}$ is weakly cartesian.*

Proof. For any $f : X \to Y$ and $(A, \mathcal{B}) \in PX \times PPY$ such that $f[A] = \bigcup \mathcal{B}$, we must find $\mathcal{A} \in PPX$ such that $\bigcup \mathcal{A} = A$ and $(Pf)[\mathcal{A}] = \mathcal{B}$. Take $\mathcal{A} = \{A \cap f^{-1}(B) \mid B \in \mathcal{B}\}$ and check

$$(Pf)[\mathcal{A}] = \{f[A \cap f^{-1}(B)] \mid B \in \mathcal{B}\} = \{f[A] \cap B \mid B \in \mathcal{B}\} = \{B \mid B \in \mathcal{B}\} = \mathcal{B}$$
$$\bigcup \mathcal{A} = A \cap f^{-1}\left(\bigcup \mathcal{B}\right) = A \cap f^{-1}(f[A]) = A$$ ◀

Computing the weak extension of $\mathbf{P}$ to Rel using equation (2) yields the well-known Egli-Milner relation

$$\overline{P}R = \{(A, B) \in PX \times PY \mid \forall x \in A, \exists y \in B, (x, y) \in R \text{ and } \forall y \in B, \exists x \in A, (x, y) \in R\} \quad (3)$$

The corresponding weak distributive law $\lambda : \mathbf{PP} \to \mathbf{PP}$ is given by

$$\lambda_X(\mathcal{A}) = \left\{B \in PX \mid B \subseteq \bigcup \mathcal{A} \text{ and } \forall A \in \mathcal{A}, A \cap B \neq \emptyset\right\} \quad (4)$$

The corresponding weak lifting of $\mathbf{P}$ to the category of complete join semi-lattices $\mathsf{EM}(\mathbf{P})$ is as follows: $\widehat{P}(X, x) = (S, s)$ has underlying set $S = \{A \in PX \mid \forall B \subseteq A, B \neq \emptyset \Rightarrow x(B) \in A\}$ with join given for every $\mathcal{A} \in PS$ by $s(\mathcal{A}) = \{x(\{a_A \mid A \in \mathcal{A}\}) \mid \forall A \in \mathcal{A}, a_A \in A\}$. The morphism $\pi_{(X,x)} : PX \to S$ sends a subset $A \subseteq X$ to its closure under non-empty join $\{x(B) \mid B \in P(A) \setminus \{\emptyset\}\}$, whereas $\iota_{(X,x)} : S \to PX$ is just the inclusion. Practically speaking, disposing of such a weak lifting allows to perform generalized determinization of PP-coalgebras as in [11, Section 5].

An *F-coalgebra* (X, c) is a morphism $c : X \to FX$, and a morphism of F-coalgebras $f : (X, c) \to (Y, d)$ is a morphism $f : X \to Y$ such that $Ff \circ c = d \circ f$. Let $\mathsf{Coalg}(F)$ be the category of F-coalgebras. Generalized determinization of PP-coalgebras is a transformation into a P-coalgebra via a process that factors through $\widehat{P}$-coalgebras as follows:

$$\begin{array}{ccccc} \mathsf{Coalg}(PP) & \longrightarrow & \mathsf{Coalg}(\widehat{P}) & \longrightarrow & \mathsf{Coalg}(P) \\ \downarrow & & \downarrow & & \downarrow \\ \mathsf{Set} & \xrightarrow{F^{\mathbf{P}}} & \mathsf{EM}(\mathbf{P}) & \xrightarrow{U^{\mathbf{P}}} & \mathsf{Set} \end{array} \tag{5}$$

More precisely, this construction maps a coalgebra $X \xrightarrow{c} PPX$ to

$$c^+ = PX \xrightarrow{Pc} PPPX \xrightarrow{\lambda_{PX}} PPPX \xrightarrow{P\mu^{\mathbf{P}}_X} PPX \tag{6}$$

so $c^+(A) = \left\{ \bigcup_{x \in A} B_x \mid \forall x \in A, B_x \in \overline{c(x)} \right\}$ where $\overline{c(x)}$ is the closure of $c(x)$ under non-empty unions. See Figure 1 for a concrete example. An interesting view is to interpret P-coalgebras as non-deterministic automata and PP-coalgebras as alternating automata, meaning a transition from a state $x \in X$ consists in choosing non-deterministically a set $U \in c(x)$, then going simultaneously into every state $y \in U$. Alternating automata have been introduced in [8] and have known some difficulties to be properly modelled as PP-coalgebras ([2, 13]). In particular, this non-standard transformation of alternating automata was already described in [13], but did not fit very well into any general framework. There, λ was only identified as a non-distributive law, because the $(\eta^{\mathbf{T}})$ diagram does not commute – which in this case is equivalent to $\eta^{\mathbf{P}}$ not being weakly cartesian. We hereby pinpoint that this automata transformation is canonical in the sense that it comes from the unique monotone weak extension of $\mathbf{P}$ to Rel.

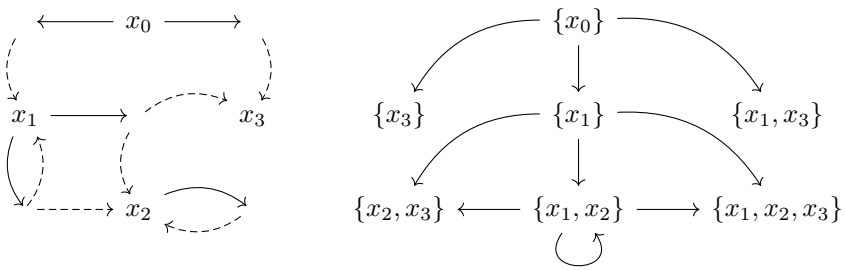


Figure 1 On the left, an alternating automaton: a transition consists in going in one of the solid lines, then in all of the available dashed lines. On the right, a portion of the non-deterministic automaton obtained after the process described in (5).

4 Toposes

The category Set is a special case of the more general notion of topos. Our contribution in this section is to generalize the results of Section 3 to arbitrary toposes. Some standard references about the theory of toposes are [12], [16], and our approach will be close to the one of [19]. Our presentation of toposes will be self-contained, with the exception of internal logic. In short, toposes are sufficiently set-behaved to internalize a logic in which one may reason as if they were picking elements in sets, and accommodate internally constructive proofs,

i.e., not using either the law of excluded middle or the axiom of choice. For more details about the internal logic, see [12, Part D], [16, Section VI.5] or the more accessible [20], [18, Chapters 14-16].

A *topos* is a finitely complete cartesian closed category with a subobject classifier Ω. Having a subobject classifier means that there is a morphism out of the terminal object $\mathsf{true} : 1 \to \Omega$ such that for every subobject $m : A \hookrightarrow X$, there is a unique morphism $\chi_m : X \to \Omega$ (called the characteristic morphism of m) such that the following diagram is a pullback:

$$\begin{array}{ccc} A & \xrightarrow{!_A} & 1 \\ {\scriptstyle m}\downarrow & \lrcorner & \downarrow{\scriptstyle \mathsf{true}} \\ X & \xrightarrow[\chi_m]{} & \Omega \end{array}$$

Note that $!_A$ denotes the unique morphism of type $A \to 1$. Toposes are finitely cocomplete. Every topos is regular, and conversely a regular category C is a topos if and only if the graph functor $\mathcal{G} : \mathsf{C} \to \mathsf{Rel}(\mathsf{C})$ has a right adjoint ([19, §6.1.1], [9, §1.911]). This adjunction yields a monad $\mathbf{\exists} : \mathsf{C} \to \mathsf{C}$ which is the generalization of the powerset monad on the topos Set – as hinted by the equality $\exists X = \Omega^X$ in C similar to $PX = 2^X$ in Set. In the internal logic of the topos, the data of $\mathbf{\exists}$ can be expressed as

$$\exists f(a) = \{y : Y \mid \exists x : X, x \in a \wedge f(x) = y\} \qquad \text{[20, Proposition 4.9]}$$
$$\eta^{\mathbf{\exists}}_X(x) = \{x' : X \mid x = x'\} \qquad \text{[20, Proposition 4.17]}$$
$$\mu^{\mathbf{\exists}}_X(t) = \{x : X \mid \exists s : \Omega^X, x \in s \wedge s \in t\} \qquad \text{[20, Proposition 4.19]}$$

Another view on $\eta^{\mathbf{\exists}}_X : X \to \Omega^X$ is that it is the exponential transpose of the characteristic morphism $X \times X \to \Omega$ of the diagonal monomorphism $\langle 1_X, 1_X \rangle : X \hookrightarrow X \times X$ ([12, page 86]).

The Kleisli category of $\mathbf{\exists}$ is nothing but the category $\mathsf{Rel}(\mathsf{C})$ ([19, §6.1.10]), with again the identification $F_{\mathbf{\exists}} = \mathcal{G}$ and relational extensions being exactly monotone (Kleisli) extensions. Given the similarities with Set, a natural question is whether the results of the previous section extend to any topos: is there a weak distributive law of type $\mathbf{\exists\exists} \to \mathbf{\exists\exists}$?

Some of the ingredients required for obtaining such a law are already in the literature:

▶ **Proposition 10** ([19, Proposition 6.5.1])**.** *The functor $\exists$ is weakly cartesian and preserves strong epis.*

De Moor [19] deduces that the *functor* $\exists$ is a *relator*, i.e., it has a monotone extension to $\mathsf{Rel}(\mathsf{C})$. One can compute this generalized Egli-Milner formula using equation (2) and the internal logic notations, although it is not relevant for the subsequent developments.

The extension $\overline{\exists}$ corresponds to a distributive law between the *functor* $\exists$ and the monad $\mathbf{\exists}$ of type $\exists\mathbf{\exists} \to \mathbf{\exists}\exists$, called *cross-operator* in [19], meaning that the $(\eta^{\mathbf{S}})$ and $(\mu^{\mathbf{S}})$ diagrams relative to the inner $\mathbf{\exists}$ commute. We provide the missing results:

▶ **Proposition 11.** *The unit $\eta^{\mathbf{\exists}}$ is nearly cartesian if and only if C is degenerate, i.e., the initial object 0 and the terminal object 1 are isomorphic.*

Proof sketch. If C is degenerate, then C is the category with a single object and a single arrow, and every natural transformation is nearly cartesian in such a category. Conversely, assume $\eta^{\mathbf{\exists}}$ is nearly cartesian. As $\eta^{\mathbf{\exists}}$ components are mono (see [16, Lemma 1 p.166]), this induces that $\eta^{\mathbf{\exists}}$ is cartesian, i.e., naturality squares are pullbacks. In particular, the left square below is a pullback. (Note that this square is the one that appears in the proof of

Proposition 8, because in Set we have $\{0\} \cong 1$ and $\{0,1\} \cong \Omega$.) Let $p : 1 \to \Omega^\Omega$ be the morphism that picks the maximal subobject $\Omega \subseteq \Omega$. The pasting law for pullbacks yields that $\Theta \cong \Phi$, where Θ and Φ are defined by the pullback squares on the right:

$$\begin{array}{ccc} \Omega & \xrightarrow{!_\Omega} & 1 \\ {\scriptstyle \eta^{\exists}_\Omega}\downarrow & \lrcorner & \downarrow{\scriptstyle \eta^{\exists}_1} \\ \Omega^\Omega & \xrightarrow[\exists !_\Omega]{} & \Omega \end{array} \qquad \begin{array}{ccc} \Theta & \longrightarrow & \Omega \\ {\scriptstyle !_\Theta}\downarrow & \lrcorner & \downarrow{\scriptstyle \eta^{\exists}_\Omega} \\ 1 & \xrightarrow[p]{} & \Omega^\Omega \end{array} \qquad \begin{array}{ccc} \Phi & \xrightarrow{!_\Phi} & 1 \\ {\scriptstyle !_\Phi}\downarrow & \lrcorner & \downarrow{\scriptstyle \eta^{\exists}_1} \\ 1 & \xrightarrow[\exists !_\Omega \circ p]{} & \Omega \end{array}$$

We can prove additionally that $\Theta \cong \Theta \times \Omega$ and $\Phi \cong 1$. Combining these results yields $\Omega \cong 1$, and this in turn implies that the topos is degenerate. ◀

▶ **Proposition 12.** *The multiplication $\mu^{\exists}$ is weakly cartesian.*

Proof. Mimicking the computation of Proposition 9 in the internal logic of C produces a valid proof, because the latter is a constructive intuitionistic proof, i.e., does not use either the axiom of choice or the law of excluded middle. ◀

▶ **Theorem 13.** *In any topos, there is a unique monotone weak distributive law of type $\exists\exists \to \exists\exists$, which is a distributive law exactly when the topos is degenerate.*

Proof. By Theorem 6, Propositions 10, 11, 12 and the fact that $F_{\exists} = \mathcal{G}$, the generalized Egli-Milner relation defines the unique monotone weak extension of $\exists$ to $\mathsf{Kl}(\exists)$, and is a monad extension iff the topos is degenerate. Applying Theorem 5 completes the proof. ◀

There is also a weak lifting of $\exists$ to the category $\mathsf{EM}(\exists)$ of internal complete join semi-lattices, implying in particular that the generalized determinization procedure described in Section 3 can be applied to $\exists\exists$-coalgebras in arbitrary toposes.

5 Compact Hausdorff Spaces

In this section, C is the category of compact Hausdorff spaces and continuous functions KHaus. As recalled in [10], KHaus is isomorphic to the Eilenberg-Moore category of the ultrafilter monad $\beta : \mathsf{Set} \to \mathsf{Set}$. This yields that KHaus is regular, complete, and that limits can be computed as in Set and given the initial topology afterwards. As KHaus is a pretopos (see, e.g., [17]), strong epis are just epis, that is, (continuous) surjections. Given an object X of KHaus, we denote its carrier set also by X and its topology by τ_X. Define VX to be the set of all closed subsets of X equipped with the *Vietoris topology*, i.e., the topology generated by the subbase

$$\Box U = \{A \in VX \mid A \subseteq U\} \qquad \Diamond U = \{A \in VX \mid A \cap U \neq \emptyset\} \tag{7}$$

where U ranges over τ_X. The mapping $X \mapsto VX$ extends into a monad $\mathbf{V}$ on KHaus called the *Vietoris monad* [10] in the same way as the powerset in Set. In concrete terms, V maps a continuous function $f : X \to Y$ to its direct image $Vf : VX \to VY$, $\eta^{\mathbf{V}} : 1 \to V$ takes singletons and $\mu^{\mathbf{V}} : VV \to V$ takes unions.

▶ Remark 14. It turns out that there is a monotone weak extension of β to $\mathsf{Kl}(\mathbf{P}) \cong \mathsf{Rel}$ and that the corresponding weak lifting is the Vietoris monad on $\mathsf{EM}(\beta) \cong \mathsf{KHaus}$. This is the main result of Garner's paper [10].

As KHaus is not a topos but only a pretopos, $\mathbf{V}$ cannot be obtained using the graph functor $\mathsf{KHaus} \to \mathsf{Rel}(\mathsf{KHaus})$ as a left adjoint. This is closely related to the fact that VX, being only part of a *weak* lifting, does not contain *all* subsets of X. Actually, $\mathsf{Rel}(\mathsf{KHaus})$ and $\mathsf{Kl}(\mathbf{V})$ correspond to the two essential ways of embedding KHaus into a relational category (see [4], where they are denoted by $\mathsf{KHaus}^{\mathsf{R}}$ and $\mathsf{KHaus}^{\mathsf{C}}$). Let X, Y be compact Hausdorff spaces and $R \subseteq X \times Y$ be a relation. Consider the following properties of R:

(i) $\forall A \in VX, R[A] \in VY$
(ii) $\forall B \in VY, R^{-1}[B] \in VX$
(iii) $\forall U \in \tau_Y, R^{-1}[U] \in \tau_X$

The relation R is *closed* if it satisfies properties (i) and (ii), or equivalently, if R is a closed subset of the product topology $\tau_{X\times Y}$. The relation R is *continuous* if it satisfies properties (i), (ii) and (iii). Note that these properties are preserved by the usual composition of relations and satisfied by identity relations. The following are straightforward results:

▶ **Proposition 15.** *The category of compact Hausdorff spaces and closed relations is isomorphic to* $\mathsf{Rel}(\mathsf{KHaus})$.

▶ **Proposition 16** (see [3, 4])**.** *The category of compact Hausdorff spaces and continuous relations is isomorphic to* $\mathsf{Kl}(\mathbf{V})$.

As a summary, we have the wide subcategory inclusions

$$\mathsf{KHaus} \xrightarrow[F_{\mathbf{V}}]{} \mathsf{Kl}(\mathbf{V}) \xrightarrow[\text{(iii)}]{\text{forget}} \mathsf{Rel}(\mathsf{KHaus}), \qquad \mathsf{KHaus} \xrightarrow{\mathcal{G}} \mathsf{Rel}(\mathsf{KHaus})$$

Any endofunctor on $\mathsf{Rel}(\mathsf{KHaus})$ that preserves continuous relations – i.e. preserves property (iii) – therefore restricts to an endofunctor on $\mathsf{Kl}(\mathbf{V})$. Similarly, given two such endofunctors on $\mathsf{Rel}(\mathsf{KHaus})$, any natural transformation whose components are continuous relations (e.g. every $\mathsf{Rel}(\alpha)$) automatically restricts to a natural transformation between their restrictions on $\mathsf{Kl}(\mathbf{V})$. Putting this together with Theorem 6 and the definition of monad extensions, we get

▶ **Proposition 17.** *Assume that T preserves continuous surjections, that T and $\mu^{\mathbf{T}}$ are nearly cartesian, and that $\mathsf{Rel}(T)$ preserves continuous relations. Then there is a monotone weak extension of $\mathbf{T}$ to $\mathsf{Kl}(\mathbf{V})$, and this is an extension if and only if $\eta^{\mathbf{T}}$ is nearly cartesian.*

Now we fix $\mathbf{T} = \mathbf{V}$ and proceed to verify the assumptions of Proposition 17.

▶ **Proposition 18.** *The Vietoris functor V preserves continuous surjections.*

Proof. For such a continuous surjective $f : X \to Y$, $Vf : VX \to VY$ is surjective because any $C \in VY$ satisfies $Vf(f^{-1}(C)) = C$, with $f^{-1}(C) \in VX$. ◀

▶ **Proposition 19.** *The Vietoris functor V is nearly cartesian.*

Proof. This is the same proof as for Proposition 7, with an additional check that if $A \in VA$ and $B \in VB$ then $C = \pi_1^{-1}(A) \cap \pi_2^{-1}(B)$ indeed is in VR, by continuity of π_1, π_2. ◀

▶ **Proposition 20.** *The unique relational extension $\mathsf{Rel}(V)$ preserves continuous relations.*

Proof. Assume that $R \subseteq X \times Y$ is a closed relation that satisfies condition (iii), denote its projections by $r_1 : R \to X$, $r_2 : R \to Y$ and show that $\mathsf{Rel}(V)R$ satisfies condition (iii). Sets of the form $\Box U_0 \cap \bigcap_{1\leq i\leq n} \Diamond U_i$ with $n \in \omega$ and $U_i \in \tau_Y$ form a base of τ_{VY}, therefore the identity (8) below suffices to conclude because R satisfying condition (iii) makes the right-hand side an element of τ_{VY}.

$$(\mathsf{Rel}(V)R)^{-1}\left[\Box U_0 \cap \bigcap_{1\leq i\leq n} \Diamond U_i\right] = \Box R^{-1}[U_0] \cap \bigcap_{1\leq i\leq n} \Diamond R^{-1}[U_0 \cap U_i] \tag{8}$$

We now prove (8). A subset $C \in VX$ belongs to the left-hand side iff there exists $D \in VY$ such that

1. $\forall x \in C, \exists y \in D, (x, y) \in R$
2. $\forall y \in D, \exists x \in C, (x, y) \in R$
3. $D \subseteq U_0$
4. $\forall i \in \{1, ..., n\}, D \cap U_i \neq \emptyset$

A subset $C \in VX$ belongs to the right-hand side iff

5. $\forall x \in C, \exists y \in U_0, (x, y) \in R$
6. $\forall i \in \{1, ..., n\}, \exists (x_i, y_i) \in R, x_i \in C, y_i \in U_0 \cap U_i$

Let $C \in VX$ and $D \in VY$ such that $1. - 4.$ are satisfied. For any $x \in C$, using 1. and 3., we can find $y \in U_0$ such that $(x, y) \in R$, so that 5. holds. For any $i \in \{1, ..., n\}$, using 3. and 4. we find $y_i \in U_0 \cap U_i$, then using 2. we find $x_i \in C$ such that $(x_i, y_i) \in R$, so that 6. holds.

For the other direction we note that every compact Hausdorff space is regular and use the following property

▶ **Lemma 21** ([25, Theorem 14.3]). *A topological space Y is regular if and only if for every $U \in \tau_Y$ and every $y \in U$, there is a $W \in \tau_Y$ such that $y \in W$ and $\overline{W} \subseteq U$.*

Let $C \in VX$ such that $5. - 6.$ hold. For every $x \in C$ we fix $y_x \in U_0$ such that $(x, y_x) \in R$. For every $i \in \{1, ..., n\}$ we fix $(x_i, y_i) \in R$ such that $x_i \in C$ and $y_i \in U_0 \cap U_i$. Apply Lemma 21 to get $W_x \in \tau_Y$ such that $y_x \in W_x$ and $\overline{W_x} \subseteq U_0$ and $W_i \in \tau_Y$ such that $y_i \in W_i$ and $\overline{W_i} \subseteq U_0 \cap U_i$. For every $x \in C$, $(x, y_x) \in R$ and $y_x \in W_x$ so $x \in R^{-1}[W_x]$, hence $C \subseteq \bigcup_{x\in C} R^{-1}[W_x]$. As C is closed in a compact space, it is compact and we can extract a finite subcover $C \subseteq \bigcup_{1\leq k\leq m} R^{-1}[W_{x_k}]$. Define the closed set

$$K = \bigcup_{1\leq i\leq n} \overline{W_i} \cup \bigcup_{1\leq k\leq m} \overline{W_{x_k}} \in VY \tag{9}$$

and consider $D = Vr_2((C \times K) \cap R) \in VY$. For any $x \in C$, there is $k \in \{1, ..., m\}$ and $y \in W_{x_k} \subseteq K$ such that $(x, y) \in R$, hence $y \in D$ and property 1. holds. Property 2. is immediate from the expression of D. As $\overline{W_i}, \overline{W_{x_k}} \subseteq U_0$, we have $D \subseteq K \subseteq U_0$ and property 3. holds. For any $i \in \{1, ..., n\}$, $(x_i, y_i) \in (C \times K) \cap R$ so $y_i \in D \cap U_i$ and property 4. holds. This achieves the proof. ◀

▶ **Proposition 22.** *The Vietoris unit $\eta^{\mathbf{V}}$ is not nearly cartesian.*

Proof. The counterexample of Proposition 8 still works by endowing the sets with the discrete topology. Note that the discrete topology on a finite set is always compact Hausdorff. ◀

A consequence is that there is no relational extension of $\mathbf{V}$ to $\mathsf{Rel}(\mathsf{KHaus})$, hence also no distributive law of type $\mathbf{VV} \to \mathbf{VV}$ can be obtained via Proposition 17. Further, there is no possible distributive law of this type because the counter-example for $\mathbf{PP} \to \mathbf{PP}$ presented in [14, Theorem 2.4] uses only finite sets: it is still valid in KHaus by endowing every set with the discrete topology again.

▶ **Proposition 23.** *The Vietoris multiplication $\mu^{\mathbf{V}}$ is nearly cartesian.*

Proof. Consider $f : X \to Y$ and let P be the pullback of its $\mu^{\mathbf{V}}$ naturality square. Surjectivity of the mediating map $h : VVX \to P$ amounts to proving that for every $C \in VX$ and $\mathcal{D} \in VVY$ such that $Vf(C) = {\mu^{\mathbf{V}}}_Y(\mathcal{D})$, there is a $\mathcal{C} \in VVX$ such that ${\mu^{\mathbf{V}}}_X(\mathcal{C}) = C$ and $VVf(\mathcal{C}) = \mathcal{D}$. However, our usual candidate $\mathcal{A} = \{C \cap f^{-1}(D) \mid D \in \mathcal{D}\}$ may not be a closed subset of VX. In its place we take

$$\mathcal{C} = (Vf)^{-1}(\mathcal{D}) \cap (\Diamond(C^c))^c \in VVX \tag{10}$$

Note that $\mathcal{C} = \{K \in VX \mid Vf(K) \in \mathcal{D} \text{ and } K \subseteq C\}$. Inclusions ${\mu^{\mathbf{V}}}_X(\mathcal{C}) \subseteq C$ and $VVf(\mathcal{C}) \subseteq \mathcal{D}$ are immediate. For the other ones, note that $\mathcal{A} \subseteq \mathcal{C}$, hence $C = {\mu^{\mathbf{P}}}_X(\mathcal{A}) \subseteq {\mu^{\mathbf{V}}}_X(\mathcal{C})$ and $\mathcal{D} = PPf(\mathcal{A}) \subseteq VVf(\mathcal{C})$. ◀

▶ **Theorem 24.** *There is a monotone weak distributive law of type* $\mathbf{VV} \to \mathbf{VV}$ *defined by*

$$\lambda_X(\mathcal{A}) = \left\{B \in VX \mid B \subseteq \bigcup \mathcal{A} \text{ and } \forall A \in \mathcal{A}, A \cap B \neq \emptyset\right\} \tag{11}$$

Proof. Use Proposition 17 together with Propositions 18, 19, 20, 22, 23 to get that there is a monotone weak extension of $\mathbf{V}$ to $\mathsf{Kl}(\mathbf{V})$ defined by

$$\overline{V}R = \{(A, B) \in VX \times VY \mid \forall x \in A, \exists y \in B, (x, y) \in R \text{ and } \forall y \in B, \exists x \in A, (x, y) \in R\}$$

By Theorem 5, this corresponds to a weak distributive law with the wanted expression. ◀

Again, we also have a weak lifting of $\mathbf{V}$ to the category $\mathsf{EM}(\mathbf{V})$ – i.e., to continuous lattices, see [10, § 2.3] – whose existence notably allows generalized determinization of VV-coalgebras.

▶ Remark 25. The Vietoris monad restricts to the full subcategory of Stone spaces and continuous functions $\mathsf{Stone} \hookrightarrow \mathsf{KHaus}$, which regularly attracts the interest of the coalgebraic community ([15], [6]). Equation (11) clearly still defines a weak distributive law in Stone, which may be useful to understand better double Vietoris coalgebras as described in [5].

6 Conclusion

In this article, we have detailed how to obtain a triptych weak extension - weak distributive law - weak lifting using powerset-like monads. First we assembled results in the literature to show that the usual method for obtaining weak extensions from Set to Rel can be adapted to obtain weak extensions from any regular category C to its category of relations. We proved that weak self-distributivity of the Set powerset monad – known since the paper of Garner [10] – can actually be understood at the deeper level of toposes. Then we treated compact Hausdorff spaces, for which the Vietoris monad plays the role of a powerset. This case is particularly interesting because the category of closed relations and the Kleisli category of the Vietoris monad do not have exactly the same morphisms. In every case, we find that the unique monotone weak extension can be expressed with an Egli-Milner-shaped formula. Using the corresponding weak distributive law, we provide an application to automata theory: generalized determinization of alternating automata into non-deterministic automata. Here alternating automata are to be understood as double powerset coalgebras, living in any category previously considered.

To our knowledge, all previous examples of interesting weak distributive laws that are not distributive laws were exhibited in the category Set. Our work provides some first instances of such laws outside of Set. It would be an interesting research direction to find useful weak

distributive laws that do not come from the extension result of Theorem 6, or that do not live into a set-like category. However, categories of relations being a rich framework, looking for more laws of the form $\mathbf{TP} \to \mathbf{PT}$ is also a promising direction. In particular, the category KHaus possesses a probability monad $\mathbf{R}$ called the Radon monad. Maybe is it possible to generalize the weak distributive law of type $\mathbf{DP} \to \mathbf{PD}$ presented in [11] (where $\mathbf{D}$ is the finitely supported distribution monad) to a continuous version $\mathbf{RV} \to \mathbf{VR}$ in KHaus. We leave this to future work.

Finally, we thank reviewers for bringing up the following remark. Although KHaus is not a topos, the Vietoris monad $\mathbf{V}$ may be thought of as classifying subobjects, in the sense that global elements of VX correspond to subobjects of X. It is an interesting question whether our weak distributive law can be generalized to (regular) categories admitting a monad classifying subobjects in this element-wise sense.

References

1 Jon Beck. Distributive laws. In B. Eckmann, editor, *Seminar on Triples and Categorical Homology Theory*, pages 119–140, Berlin, Heidelberg, 1969. Springer Berlin Heidelberg. doi:10.1007/BFb0083084.

2 Meven Bertrand and Jurriaan Rot. Coalgebraic determinization of alternating automata. *arXiv preprint*, 2018. arXiv:1804.02546.

3 Guram Bezhanishvili, Nick Bezhanishvili, and John Harding. Modal compact Hausdorff spaces. *Journal of Logic and Computation*, 25(1):1–35, 2015. doi:10.1093/logcom/exs030.

4 Guram Bezhanishvili, David Gabelaia, John Harding, and Mamuka Jibladze. Compact Hausdorff spaces with relations and Gleason spaces. *Applied Categorical Structures*, 27(6):663–686, 2019. doi:10.1007/s10485-019-09573-x.

5 Nick Bezhanishvili, Sebastian Enqvist, and Jim de Groot. Duality for instantial neighbourhood logic via coalgebra. In Daniela Petrişan and Jurriaan Rot, editors, *Coalgebraic Methods in Computer Science*, pages 32–54, Cham, 2020. Springer International Publishing. doi:10.1007/978-3-030-57201-3_3.

6 Nick Bezhanishvili, Gaëlle Fontaine, and Yde Venema. Vietoris bisimulations. *Journal of Logic and Computation*, 20(5):1017–1040, 2010. doi:10.1093/logcom/exn091.

7 Aurelio Carboni, G Max Kelly, and Richard J Wood. A 2-categorical approach to change of base and geometric morphisms I. *Cahiers de topologie et géométrie différentielle catégoriques*, 32(1):47–95, 1991.

8 Ashok K. Chandra, Dexter C. Kozen, and Larry J. Stockmeyer. Alternation. *J. ACM*, 28(1):114–133, 1981. doi:10.1145/322234.322243.

9 Peter J Freyd and Andre Scedrov. *Categories, allegories*. Elsevier, 1990.

10 Richard Garner. The Vietoris monad and weak distributive laws. *Applied Categorical Structures*, October 2019. doi:10.1007/s10485-019-09582-w.

11 Alexandre Goy and Daniela Petrişan. Combining probabilistic and non-deterministic choice via weak distributive laws. In *LICS '20: Proceedings of the 35th Annual ACM/IEEE Symposium on Logic in Computer Science*, pages 454–464, Saarbrücken, Germany, 2020. doi:10.1145/3373718.3394795.

12 Peter T Johnstone. *Sketches of an Elephant: A Topos Theory Compendium: Volume 1*, volume 1. Oxford University Press, 2002.

13 Bartek Klin and Jurriaan Rot. Coalgebraic trace semantics via forgetful logics. In *International Conference on Foundations of Software Science and Computation Structures*, pages 151–166. Springer, 2015. doi:10.1007/978-3-662-46678-0_10.

14 Bartek Klin and Julian Salamanca. Iterated covariant powerset is not a monad. *Electronic Notes in Theoretical Computer Science*, 341:261–276, 2018. Proceedings of the Thirty-Fourth Conference on the Mathematical Foundations of Programming Semantics (MFPS XXXIV). doi:10.1016/j.entcs.2018.11.013.

15 Clemens Kupke, Alexander Kurz, and Yde Venema. Stone coalgebras. *Theoretical Computer Science*, 327(1-2):109–134, 2004. doi:10.1016/j.tcs.2004.07.023.

16 Saunders MacLane and Ieke Moerdijk. *Sheaves in geometry and logic: A first introduction to topos theory*. Springer Science & Business Media, 2012.

17 Vincenzo Marra and Luca Reggio. A characterisation of the category of compact Hausdorff spaces. *Theory and Applications of Categories*, 35(51):1871–1906, 2020. URL: http://www.tac.mta.ca/tac/volumes/35/51/35-51.pdf.

18 Colin McLarty. *Elementary categories, elementary toposes*. Clarendon Press, 1992.

19 Oege de Moor. *Categories, relations and dynamic programming*. PhD thesis, University of Oxford, 1992.

20 Gerhard Osius. Logical and set theoretical tools in elementary topoi. In F. William Lawvere, Christian Maurer, and Gavin C. Wraith, editors, *Model Theory and Topoi*, pages 297–346, Berlin, Heidelberg, 1975. Springer Berlin Heidelberg. doi:10.1007/BFb0061299.

21 Christoph Schubert. *Lax Algebras: A Scenic Approach*. PhD thesis, Universität Bremen, 2006.

22 Sam Staton. Relating coalgebraic notions of bisimulation. In *International Conference on Algebra and Coalgebra in Computer Science*, pages 191–205. Springer, 2009. doi:10.2168/LMCS-7(1:13)2011.

23 Daniele Turi. *Functorial operational semantics and its denotational dual*. PhD thesis, Vrije Universiteit, Amsterdam, 1996. URL: https://www.cs.vu.nl/en/Images/D_Turi_06-06-1996_tcm210-258569.pdf.

24 Daniele Varacca. Probability, nondeterminism and concurrency: Two denotational models for probabilistic computation. Technical report, PhD thesis, Univ. Aarhus, 2003. BRICS Dissertation Series, 2003. URL: https://www.brics.dk/DS/03/14/BRICS-DS-03-14.pdf.

25 Stephen Willard. *General Topology*. Dover Publications, 2004.

26 M. Zwart and D. Marsden. No-go theorems for distributive laws. In *2019 34th Annual ACM/IEEE Symposium on Logic in Computer Science (LICS)*, pages 1–13, Los Alamitos, CA, USA, June 2019. IEEE Computer Society. doi:10.1109/LICS.2019.8785707.

Elementary Equivalence Versus Isomorphism in Semiring Semantics

Erich Grädel
RWTH Aachen University, Germany

Lovro Mrkonjić
RWTH Aachen University, Germany

Abstract

We study the first-order axiomatisability of finite semiring interpretations or, equivalently, the question whether elementary equivalence and isomorphism coincide for valuations of atomic facts over a finite universe into a commutative semiring. Contrary to the classical case of Boolean semantics, where every finite structure is axiomatised up to isomorphism by a first-order sentence, the situation in semiring semantics is rather different, and depends on the underlying semiring. We prove that for a number of important semirings, including min-max semirings, and the semirings of positive Boolean expressions, there exist finite semiring interpretations that are elementarily equivalent but not isomorphic. The same is true for the polynomial semirings that are universal for the classes of absorptive, idempotent, and fully idempotent semirings, respectively. On the other side, we prove that for other, practically relevant, semirings such as the Viterby semiring $\mathbb{V}$, the tropical semiring $\mathbb{T}$, the natural semiring $\mathbb{N}$ and the universal polynomial semiring $\mathbb{N}[X]$, all finite semiring interpretations are first-order axiomatisable, and thus elementary equivalence implies isomorphism.

2012 ACM Subject Classification Theory of computation → Finite Model Theory

Keywords and phrases Semiring semantics, elementary equivalence, axiomatisability

Digital Object Identifier 10.4230/LIPIcs.ICALP.2021.133

Category Track B: Automata, Logic, Semantics, and Theory of Programming

Related Version *Previous Version*: https://arxiv.org/abs/2102.05473

1 Introduction

Semiring semantics is based on the idea to evaluate logical statements not just by *true* or *false*, but by values in some commutative semiring $(K, +, \cdot, 0, 1)$. In this context, the standard semantics appears as the special case when the Boolean semiring $\mathbb{B} = (\{\bot, \top\}, \vee, \wedge, \bot, \top)$ is used. Valuations in other semirings provide additional information, beyond the truth or falsity of a statement: the Viterbi-semiring $\mathbb{V} = ([0,1]_{\mathbb{R}}, \max, \cdot, 0, 1)$ models *confidence scores*, the tropical semiring $\mathbb{T} = (\mathbb{R}^{\infty}_{+}, \min, +, \infty, 0)$ is used for *cost analysis*, and min-max-semirings $(K, \max, \min, a, b)$ for a totally ordered set $(K, <)$ can model, for instance, different *access levels*. More generally, semirings of polynomials, such as $\mathbb{N}[X]$ or $\mathbb{B}[X]$, allow us to track the role of specific atomic facts for the evaluation of a logical statement, to describe evaluation strategies for a formula, and to determine which combinations of literals prove the truth of a formula.

Some of the motivation for the study of semiring semantics comes from the successful development of semiring provenance in database theory and related fields (see e.g. [5, 6, 9, 14, 15, 18, 19, 20]), and the fact that the typical applications of provenance analysis, such as confidence scores, cost analysis, proof counting, and the understanding of evaluation strategies are of importance in many other areas of logic as well. However, semiring provenance analysis for database queries had originally been largely confined to positive query languages, such as conjunctive queries, positive relational algebra, and Datalog, and the treatment

of negation poses non-trivial algebraic problems. Only recently, provenance analysis via semiring semantics has been extended to logics with negation, and in particular to full first-order logic [12, 13], by means of new algebraic constructions based on quotient semirings. Semiring semantics has also been studied for other logics, including modal logics, description logics, guarded logics, and fixed-point logic [1, 2, 3, 4], and this paper is part of a larger project devoted to a systematic study of semiring semantics for various logics. An important objective in this context is the understanding of the *model theory of semiring semantics*, and the development of model-theoretic methods for semiring interpretations.

It turns out that this is much more involved and diverse than for Boolean semantics. In the standard semantics, a model $\mathfrak{A}$ assigns to each (instantiated) atomic formula a Boolean value, whereas K-interpretations π, for a suitable semiring K, generalise this by assigning to each literal a semiring value in K, where 0 is interpreted as *false* and all other semiring values as *nuances of true*. Interpreting disjunction by $+$ and conjunction by $\cdot$, we can extend π to provide semiring valuations $\pi[\![\varphi]\!] \in K$ for all first-order sentences φ, written in negation normal form. Semiring semantics thus gives a finer distinction of logical statements, and formulae that are equivalent in the Boolean sense (i.e. in the Boolean semiring) may have different valuations in other semirings. As a consequence, standard facts of classical (finite) model theory may lead to interesting and sometimes rather difficult questions in semiring semantics, and the answer may strongly depend on algebraic properties of the underlying semiring. Specific such questions that we study here concern the first-order axiomatisability of finite K-interpretations or, what amounts to the same, the relationship between isomorphism and elementary equivalence in this context.

It is a rather trivial fact of finite model theory that every finite structure $\mathfrak{A}$ (with a finite vocabulary τ) can be axiomatised, up to isomorphism, by a first-order sentence $\chi_{\mathfrak{A}}$. In particular, two finite τ-structures $\mathfrak{A}$ and $\mathfrak{B}$ are isomorphic if, and only if, they are elementarily equivalent, in short $\mathfrak{A} \equiv \mathfrak{B}$, which means that they cannot be distinguished by any first-order sentence. Is this also the case for semiring interpretations? Notice that standard notions such as isomorphism and elementary equivalence generalise in a natural way from τ-structures to semiring interpretations, which raises, for any given semiring K, the following

▶ **Questions.**

1. *Are elementarily equivalent, finite K-interpretations always isomorphic?*
2. *Is every finite K-interpretation π_A first-order axiomatisable, in the sense that there is a set of axioms $\Phi_A \subseteq$ FO such that whenever $\pi_B[\![\varphi]\!] = \pi_A[\![\varphi]\!]$ for all $\varphi \in \Phi_A$, then $\pi_B \cong \pi_A$?*
3. *Does every finite K-interpretation admit an axiomatisation by a* finite *set of axioms?*
4. *Can every finite K-interpretation be axiomatised by a single first-order sentence?*

Clearly, the first two questions are equivalent, and a positive answer to the third question implies also positive ones to the first two. The converse is not necessarily true, because a first-order axiomatisation of a finite semiring interpretation might require an infinite collection of sentences, and, contrary to the Boolean case, it is a priori also not clear that an axiomatisation by a finite set of sentences implies an axiomatisation by a single sentence, because from the value of a conjunction we cannot necessary infer the values of its components.

We shall prove that the answers to these questions strongly depend on the chosen semiring. There are in fact rather simple semirings, such as min-max semirings with at least three elements, for which one can construct examples of non-isomorphic K-interpretations which are, however, elementarily equivalent. The standard method for proving elementary equivalence in model theory, the Ehrenfeucht–Fraïssé method, seems not really available in

semiring semantics, an aspect that we shall discuss at the end of this paper. To establish elementary equivalence, we shall hence develop new methods based on classes of semiring homomorphisms and reduction arguments. Elementarily equivalent but non-isomorphic semiring interpretations also exist for powerful polynomial semirings, such as $\mathbb{S}[X]$ and $\mathbb{B}[X]$, which are universal for the classes of absorptive and idempotent semirings, respectively. On the other side, there are practically relevant semirings, such as the Viterby semring $\mathbb{V}$, the tropical semiring $\mathbb{T}$, the natural semiring $\mathbb{N}$ and the universal polynomial semiring $\mathbb{N}[X]$, for which any finite K-interpretation is first-order axiomatisable, thus elementary equivalence does indeed imply isomorphism. At least for $\mathbb{V}$ and $\mathbb{T}$, finite axiomatisations are always possible, but not axiomatisations by a single sentence, so there exist semirings where the answers to questions (3) and (4) are different.

2 Semiring Interpretations

We briefly summarise semiring semantics for first-order logic, as introduced in [12].

▶ **Definition 1** (Semiring). A commutative semiring $\mathbb{K} = (K, +, \cdot, 0, 1)$ is an algebraic structure with two binary operations such that $(K, +, 0)$ and $(K, \cdot, 1)$ are commutative monoids, multiplication distributes over addition and multiplication with zero annihilates elements. We may identify $\mathbb{K}$ with its universe K if the operations are clear from the context.

In this paper, we only consider commutative semirings and simply call them "semirings" for convenience. Any class of semirings is implicitly restricted to commutative semirings only.

Let τ denote a finite relational vocabulary. We write $\mathrm{Lit}_n(\tau)$ for the set of atoms $R\overline{z}$ and negated atoms $\neg R\overline{z}$ with $R \in \tau$ and where $\overline{z}$ is any tuple of variables taken from $\{x_1, \dots, x_n\}$. For a universe A, we write $\mathrm{Lit}_A(\tau)$ for the set of *instantiated* τ-literals $R\overline{a}$ and $\neg R\overline{a}$ with $\overline{a} \in A^{\mathrm{arity}(R)}$.

▶ **Definition 2** (K-interpretation). For a semiring K, a mapping $\pi\colon \mathrm{Lit}_A(\tau) \to K$ is called a *K-interpretation* over the universe A with signature τ. We call π *model-defining* if exactly one of the values $\pi(L)$ and $\pi(\overline{L})$ is zero for all pairs of opposing literals $L, \overline{L} \in \mathrm{Lit}_A(\tau)$. In that case, π *induces* the classical model $\mathfrak{A}_\pi$ with $\mathfrak{A}_\pi \models L$ if, and only if, $\pi(L) \neq 0$.

▶ **Definition 3** (Isomorphism). K-interpretations $\pi_A\colon \mathrm{Lit}_A(\tau) \to K$ and $\pi_B\colon \mathrm{Lit}_B(\tau) \to K$ are *isomorphic*, denoted as $\pi_A \cong \pi_B$, if there is a bijective mapping $\sigma\colon A \to B$ such that $\pi_A(L) = \pi_B(\sigma(L))$ for all $L \in \mathrm{Lit}_A(\tau)$, where $\sigma(L) \in \mathrm{Lit}_B(\tau)$ is defined by replacing each $a \in A$ occurring in L with $\sigma(a) \in B$. The mapping $\sigma\colon \pi_A \xrightarrow{\sim} \pi_B$ is called an *isomorphism*.

Given a K-interpretation $\pi\colon \mathrm{Lit}_A(\tau) \to K$, a formula $\varphi(\overline{x}) \in \mathrm{FO}(\tau)$ in negation normal form and an assignment $\overline{a} \subseteq A$, the semiring semantics $\pi[\![\varphi(\overline{a})]\!]$ is straightforwardly defined by induction on $\mathrm{FO}(\tau)$. Equalities are simply mapped to their truth values by $\pi[\![a = b]\!] := 1$ if $a = b$ and 0 otherwise, and vice versa for inequalities. Similarly to the semantics of weighted logics introduced in [7], disjunctions and existential quantifiers are interpreted as sums, and conjunctions and universal quantifiers as products:

$$\pi[\![\psi(\overline{a}) \vee \vartheta(\overline{a})]\!] := \pi[\![\psi(\overline{a})]\!] + \pi[\![\vartheta(\overline{a})]\!] \qquad \pi[\![\psi(\overline{a}) \wedge \vartheta(\overline{a})]\!] := \pi[\![\psi(\overline{a})]\!] \cdot \pi[\![\vartheta(\overline{a})]\!]$$
$$\pi[\![\exists x \vartheta(\overline{a}, x)]\!] := \sum_{a \in A} \pi[\![\vartheta(\overline{a}, a)]\!] \qquad \pi[\![\forall x \vartheta(\overline{a}, x)]\!] := \prod_{a \in A} \pi[\![\vartheta(\overline{a}, a)]\!].$$

Our goal is to analyse classical model-theoretic concepts under semiring semantics.

▶ **Definition 4** (Elementary Equivalence). Two K-interpretations $\pi_A\colon \mathrm{Lit}_A(\tau) \to K$ and $\pi_B\colon \mathrm{Lit}_B(\tau) \to K$ are *elementarily equivalent*, denoted as $\pi_A \equiv \pi_B$, if $\pi_A[\![\psi]\!] = \pi_B[\![\psi]\!]$ holds for all sentences $\psi \in \mathrm{FO}(\tau)$.

Clearly, the notions of isomorphism and elementary equivalence of K-interpretations are natural generalisations of the corresponding definitions for τ-structures. Further, it is obvious that, as in classical semantics, isomorphism implies elementary equivalence.

▶ **Lemma 5** (Isomorphism Lemma). *Let $\pi_A\colon \mathrm{Lit}_A(\tau) \to K$ and $\pi_B\colon \mathrm{Lit}_B(\tau) \to K$ be two K-interpretations, $\overline{a} \in A^k$ and $\overline{b} \in B^k$ be k-tuples and $\sigma\colon \pi_A \xrightarrow{\sim} \pi_B$ an isomorphism with $\sigma(\overline{a}) = \overline{b}$. Then, $\pi_A[\![\varphi(\overline{a})]\!] = \pi_B[\![\varphi(\overline{b})]\!]$ holds for all $\varphi(\overline{x}) \in \mathrm{FO}(\tau)$ with k free variables.*

Coarser definitions may be conceivable in semirings K with more than two elements, such as replacing equality by a congruence relation $\sim\, \subseteq K \times K$. However, Definition 4 indirectly covers these variants, since any non-trivial congruence relation $\sim$ on K induces a semiring homomorphism $h_\sim\colon K \to K/\!\sim$, which is compatible with FO-semantics as follows [12].

▶ **Lemma 6** (Fundamental Property). *Let $\pi\colon \mathrm{Lit}_A(\tau) \to K$ be a K-interpretation and $h\colon K \to L$ a semiring homomorphism. Then, $(h \circ \pi)$ is an L-interpretation such that $(h \circ \pi)[\![\varphi(\overline{a})]\!] = h(\pi[\![\varphi(\overline{a})]\!])$ holds for all $\varphi(\overline{x}) \in \mathrm{FO}(\tau)$ and $\overline{a} \subseteq A$.*

3 Polynomial Semirings and the Universal Property

Semiring homomorphisms and the fundamental property open the possibility to reduce semiring semantics to the evaluation of polynomials. For a finite set X of abstract provenance tokens that are used to track atomic facts, consider the semiring $\mathbb{N}[X]$ of multivariate polynomials with indeterminates from X and coefficients from $\mathbb{N}$, whose generality is formalised by the following *universal property* [14].

▶ **Lemma 7** (Universal Property). *For each commutative semiring K, every assignment $e\colon X \to K$ induces a unique homomorphism $h_e\colon \mathbb{N}[X] \to K$ with $h_e(x) = e(x)$ for $x \in X$.*

This property can be used to save computation resources, for example, if we would like to evaluate a sentence $\psi \in \mathrm{FO}$ under several interpretations $(\pi_i)_{i\in I}$ sharing the same set of true literals. In such a scenario, we may build an $\mathbb{N}[X]$-interpretation π that assigns a unique variable $x \in X$ to each positive literal. After computing the polynomial $p := \pi[\![\psi]\!]$ once, the semiring semantics $\pi_i[\![\psi]\!]$ for each $i \in I$ may be computed by plugging the positive literals' values into their corresponding variables and evaluating the polynomial p instead of starting from scratch.

The universality of $\mathbb{N}[X]$ also makes it relevant for model theory. Crucially, more restricted polynomial semirings can be used to capture smaller classes of semirings in the sense of the universal property. This is formalised in the following and summarised by Figure 1.

▶ **Definition 8** (Idempotence and Absorption). A semiring K is called *idempotent* if $a + a = a$ holds for all $a \in K$, that is, if addition is idempotent. It is *multiplicatively idempotent* if $a \cdot a = a$ for all $a \in K$. If both properties hold, we call K *fully idempotent*. Finally, a semiring K is *absorptive* if $a + ab = a$ holds for all $a, b \in K$.

- By dropping coefficients from $\mathbb{N}[X]$, we get the semiring $\mathbb{B}[X]$ whose elements are finite sets of monomials. It has the universal property for the class of idempotent semirings.

- By collapsing exponents in $\mathbb{B}[X]$, we get the semiring $\mathbb{W}[X]$ of finite sums of monomials that are linear in each argument. It is sometimes called the Why-semiring. While $\mathbb{W}[X]$ is not fully idempotent itself due to $(x+y)\cdot(x+y) = x+xy+y$, it still holds the universal property for fully idempotent semirings.
- For absorptive semirings, we require absorptive polynomials as introduced in [5]. An absorptive polynomial is a sum of distinct monomials over a finite set of variables X, with absorption among monomials: "shorter" monomials absorb "longer" monomials. More formally, m_1 absorbs m_2, denoted $m_1 \succeq m_2$, if it has smaller exponents, i.e. $m_2 = m \cdot m_1$ for some monomial m. For example, $xy^2 \succeq x^3y^2$ and $x \succeq xy$, but x^2y and xy^2 are incomparable. Applying absorption after each operation induces the semiring of absorptive polynomials $\mathbb{S}[X]$, which is universal for the class of all absorptive semirings.
- By collapsing exponents in $\mathbb{S}[X]$, we obtain yet another polynomial semiring $\mathsf{PosBool}[X]$, which is universal for the fully idempotent and absorptive semirings. Those are precisely the semirings induced by distributive lattices, as shown in [17]. Incidentally, $\mathsf{PosBool}[X]$ is the distributive lattice freely generated by the set X.

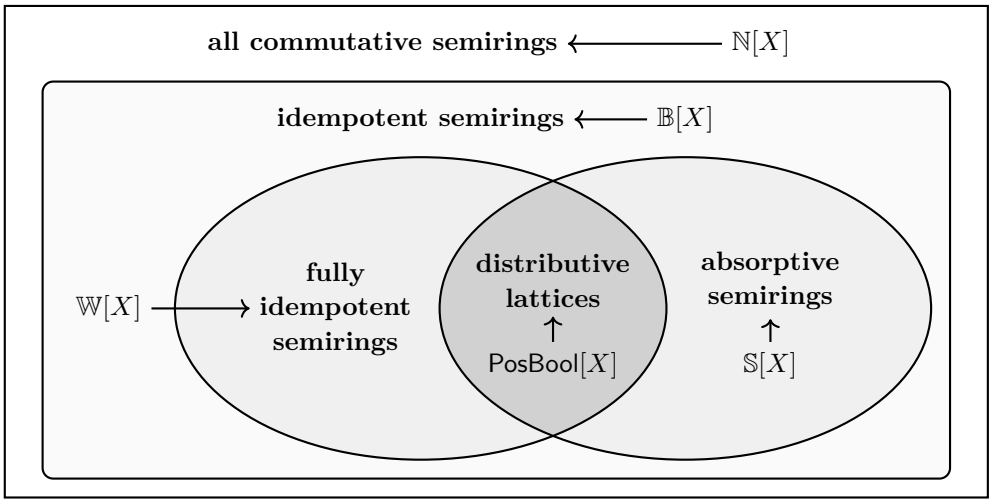


Figure 1 Relationships between some classes of commutative semirings and their respective universal polynomial semirings, adapted from [17].

4 Separating Elementary Equivalence from Isomorphism

We shall now, for certain semirings K, provide examples of finite, non-isomorphic K-interpretations that are, however, elementarily equivalent. We thus provide negative answers to Question (1) from the introduction, and hence also to Questions (2) to (4). For instance, we claim that the following two K_4-interpretations over the min-max-semiring with four elements, $K_4 = \{0,1,2,3\}$, are elementarily equivalent, but not isomorphic.

π_{PQ} :

A	P	Q	$\neg P$	$\neg Q$
a	1	3	0	0
b	2	1	0	0
c	3	2	0	0

π_{QP} :

A	P	Q	$\neg P$	$\neg Q$
a	3	1	0	0
b	1	2	0	0
c	2	3	0	0

Observe that π_{QP} is obtained from π_{PQ} by permuting the relations P and Q, or visually, by permuting the "columns". Moreover, for both interpretations, the Q-column can be obtained by permuting the P-column. Informally, these properties ensure that the two interpretations are "sufficiently similar" so that no first-order sentence can distinguish them.

Clearly, π_{PQ} is not isomorphic to π_{QP}, as intended. However, the only tool we presented so far for proving elementary equivalence under semiring semantics is the Isomorphism Lemma itself, which is not directly applicable for obvious reasons. Hence, we shall develop another tool for proving elementary equivalence that enables the indirect use of the Isomorphism Lemma after switching to a different semiring via homomorphisms.

4.1 Separating Pairs of Homomorphisms

The central idea of the reduction technique is to "decompose" the semiring K via homomorphisms. Observe that if $\pi_A \not\equiv \pi_B$, then there is a witnessing sentence $\psi \in \mathrm{FO}(\tau)$ with $\pi_A[\![\psi]\!] \neq \pi_B[\![\psi]\!]$, hence a pair of distinct elements $a, b \in K$ with $\pi_A[\![\psi]\!] =: a \neq b := \pi_B[\![\psi]\!]$ exists. If we can find two homomorphisms $h_A, h_B \colon K \to K'$ with $h_A(a) \neq h_B(b)$, but we are sure that the corresponding K'-interpretations $(h_A \circ \pi_A)$ and $(h_B \circ \pi_B)$ are elementarily equivalent, then we can exclude (a, b) as a witness for $\pi_A \not\equiv \pi_B$. If we are able to provide enough pairs of homomorphisms so that each distinct pair (a, b) can be excluded, then $\pi_A \equiv \pi_B$ must hold. The following definition formalises the required properties.

▶ **Definition 9** (Separating Homomorphism Pairs)**.** A set $S \subseteq \mathrm{Hom}^2(K, K')$ of homomorphism pairs $h_A, h_B \colon K \to K'$ is called *separating* if for all $a, b \in K$ with $a \neq b$, there is a pair $(h_A, h_B) \in S$ such that $h_A(a) \neq h_B(b)$. S is called *diagonal* if $h_A = h_B$ for all pairs $(h_A, h_B) \in S$. In that case, we may write S as a subset of $\mathrm{Hom}(K, K')$.

Note that a single *injective* homomorphism $h \colon K \to K'$ induces the diagonal separating set $S := \{(h, h)\}$ with just one element. Moreover, some semirings, such as $\mathsf{PosBool}[X]$, can be completely decomposed into $K' := \mathbb{B}$ using a diagonal separating set of semiring homomorphisms as follows. Any subset $Y \subseteq X$ induces a unique homomorphism $h_Y \colon \mathsf{PosBool}[X] \to \mathbb{B}$ by $h_Y(x) = \top$ for $x \in Y$ and $h_Y(x) = \bot$ for $x \in X \setminus Y$. Clearly, for any $p \in \mathsf{PosBool}[X]$, we have that $h_Y(p) = \top$ if, and only if, p contains a monomial with only variables from Y.

▶ **Lemma 10.** *The set $S := \{h_Y \mid Y \subseteq X\} \subseteq \mathrm{Hom}(\mathsf{PosBool}[X], \mathbb{B})$ is a diagonal separating set of homomorphisms.*

Proof. Consider $p, q \in \mathsf{PosBool}[X]$ such that $p \neq q$. Among the monomials that appear in one of the two polynomials p, q but not in the other, let m be one whose set Y of variables is minimal. By symmetry, we can assume that m appears in p but not in q. It follows that $h_Y(p) = \top$. We claim that $h_Y(q) = \bot$. Otherwise q must contain a monomial m' with only variables from Y. Since m' has less variables than m, m' must also be contained in p. But m' absorbs m, so m does not occur in p, a contradiction. ◀

On the other side, $\mathbb{N}[X]$, among other semirings, cannot be decomposed into $\mathbb{B}$ by a diagonal separating set. For example, $h(x + xy) = h(x) \vee (h(x) \wedge h(y)) = h(x)$ for all homomorphisms $h \colon \mathbb{N}[X] \to \mathbb{B}$, but $x + xy \neq x$. The reason why a decomposition into $\mathbb{B}$ would be useful for model theory is given by the following reduction technique.

▶ **Proposition 11** (Reduction Technique)**.** *Let $\pi_A \colon \mathrm{Lit}_A(\tau) \to K$ and $\pi_B \colon \mathrm{Lit}_B(\tau) \to K$ be two K-interpretations, $\overline{a} \in A^k$ and $\overline{b} \in B^k$ be k-tuples and $S \subseteq \mathrm{Hom}^2(K, K')$ a separating set of homomorphism pairs. Then, for any formula $\varphi(x_1, \ldots, x_k) \in \mathrm{FO}(\tau)$, we have that whenever $(h_A \circ \pi_A)[\![\varphi(\overline{a})]\!] = (h_B \circ \pi_B)[\![\varphi(\overline{b})]\!]$ for all $(h_A, h_B) \in S$, then also $\pi_A[\![\varphi(\overline{a})]\!] = \pi_B[\![\varphi(\overline{b})]\!]$.*

Proof. We show the contraposition. Suppose that $\pi_A[\![\varphi(\overline{a})]\!] \neq \pi_B[\![\varphi(\overline{b})]\!]$. Then, by definition of S, there exists a pair $(h_A, h_B) \in S$ such that $h_A(\pi_A[\![\varphi(\overline{a})]\!]) \neq h_B(\pi_B[\![\varphi(\overline{b})]\!])$. Applying the fundamental property yields $(h_A \circ \pi_A)[\![\varphi(\overline{a})]\!] \neq (h_B \circ \pi_B)[\![\varphi(\overline{b})]\!]$. ◀

▶ **Corollary 12.** *For S as above, $(h_A \circ \pi_A) \equiv (h_B \circ \pi_B)$ for all $(h_A, h_B) \in S$ implies $\pi_A \equiv \pi_B$.*

With the target semiring $K' \coloneqq \mathbb{B}$, the corollary shows that proving equivalence in K may be reduced to proving equivalence in $\mathbb{B}$, which permits using results from standard semantics.

4.2 Applications to PosBool$[X]$ and $\mathbb{W}[X]$

Consider the following two PosBool$[X]$-interpretations π_{xy}, π_{yx} with $X \coloneqq \{x, y\}$ over the universe $A \coloneqq \{a, b, c, d\}$ with four elements and a signature $\tau \coloneqq \{P, Q\}$ with two unary relation symbols.

π_{xy} :

A	P	Q	$\neg P$	$\neg Q$
a	0	y	x	0
b	x	0	0	y
c	y	x	0	0
d	0	0	y	x

π_{yx} :

A	P	Q	$\neg P$	$\neg Q$
a	y	0	0	x
b	0	x	y	0
c	x	y	0	0
d	0	0	x	y

Thanks to Lemma 10, the four homomorphisms $S = \{h_\varnothing, h_{\{x\}}, h_{\{y\}}, h_{\{x,y\}}\}$ induce a separating set on PosBool$[X]$ and with Corollary 12, it suffices to show that $(h \circ \pi_{xy}) \equiv (h \circ \pi_{yx})$ in $\mathbb{B}$ for all $h \in S$ in order to prove $\pi_{xy} \equiv \pi_{yx}$ on PosBool$[X]$. Indeed, Figure 2 demonstrates that $(h \circ \pi_{xy}) \cong (h \circ \pi_{yx})$ holds for all $h \in S$.

Thus, we conclude that $\pi_{xy} \equiv \pi_{yx}$ and due to $\pi_{xy} \ncong \pi_{yx}$, this shows that for finite PosBool$[X]$-interpretations, elementary equivalence does not necessarily imply isomorphism. Moreover, similar examples can be constructed for any distributive lattice semiring K thanks to the universal property of PosBool$[X]$, by assigning $r, s \in K$ to the variables x, y.

π_{rs} :

A	P	Q	$\neg P$	$\neg Q$
a	0	s	r	0
b	r	0	0	s
c	s	r	0	0
d	0	0	s	r

π_{sr} :

A	P	Q	$\neg P$	$\neg Q$
a	s	0	0	r
b	0	r	s	0
c	r	s	0	0
d	0	0	r	s

Clearly, $\pi_{rs} \equiv \pi_{sr}$ holds as above, and the only requirement for $\pi_{rs} \ncong \pi_{sr}$ is that r and s must be distinct. This yields the following theorem.

▶ **Theorem 13.** *For any distributive lattice semiring K with at least three elements, there is a pair of finite K-interpretations π_{rs}, π_{sr} over a universe with four elements and a signature with two unary relation symbols such that $\pi_{rs} \equiv \pi_{sr}$, but $\pi_{rs} \ncong \pi_{sr}$.*

Note that the two K_4-interpretations π_{PQ} and π_{QP} from the opening example of this section can be shown to be elementarily equivalent using a similar technique as above. In fact, the above theorem even shows that the opening example was not minimal and a counterexample with only three semiring elements in $K_3 = \{0, 1, 2\}$ exists.

We shall strengthen the result of Theorem 13 to the class of all fully idempotent semirings by simply regarding π_{xy} and π_{yx} as $\mathbb{W}[X]$-interpretations instead of PosBool$[X]$-interpretations. However, the proof that $\pi_{xy} \equiv \pi_{yx}$ becomes more involved, since a diagonal separating set for $\mathbb{W}[X]$ into $\mathbb{B}$ does not exist. Nevertheless, a suitable separating set can

$h_\varnothing \circ \pi_{xy}$:

A	P	Q	$\neg P$	$\neg Q$
a	$\bot$	$\bot$	$\bot$	$\bot$
b	$\bot$	$\bot$	$\bot$	$\bot$
c	$\bot$	$\bot$	$\bot$	$\bot$
d	$\bot$	$\bot$	$\bot$	$\bot$

$h_\varnothing \circ \pi_{yx}$:

A	P	Q	$\neg P$	$\neg Q$
a	$\bot$	$\bot$	$\bot$	$\bot$
b	$\bot$	$\bot$	$\bot$	$\bot$
c	$\bot$	$\bot$	$\bot$	$\bot$
d	$\bot$	$\bot$	$\bot$	$\bot$

$h_{\{x\}} \circ \pi_{xy}$:

A	P	Q	$\neg P$	$\neg Q$
a	$\bot$	$\bot$	$\top$	$\bot$
b	$\top$	$\bot$	$\bot$	$\bot$
c	$\bot$	$\top$	$\bot$	$\bot$
d	$\bot$	$\bot$	$\bot$	$\top$

$h_{\{x\}} \circ \pi_{yx}$:

A	P	Q	$\neg P$	$\neg Q$
a	$\bot$	$\bot$	$\bot$	$\top$
b	$\bot$	$\top$	$\bot$	$\bot$
c	$\top$	$\bot$	$\bot$	$\bot$
d	$\bot$	$\bot$	$\top$	$\bot$

$h_{\{y\}} \circ \pi_{xy}$:

A	P	Q	$\neg P$	$\neg Q$
a	$\bot$	$\top$	$\bot$	$\bot$
b	$\bot$	$\bot$	$\bot$	$\top$
c	$\top$	$\bot$	$\bot$	$\bot$
d	$\bot$	$\bot$	$\top$	$\bot$

$h_{\{y\}} \circ \pi_{yx}$:

A	P	Q	$\neg P$	$\neg Q$
a	$\top$	$\bot$	$\bot$	$\bot$
b	$\bot$	$\bot$	$\top$	$\bot$
c	$\bot$	$\top$	$\bot$	$\bot$
d	$\bot$	$\bot$	$\bot$	$\top$

$h_X \circ \pi_{xy}$:

A	P	Q	$\neg P$	$\neg Q$
a	$\bot$	$\top$	$\top$	$\bot$
b	$\top$	$\bot$	$\bot$	$\top$
c	$\top$	$\top$	$\bot$	$\bot$
d	$\bot$	$\bot$	$\top$	$\top$

$h_X \circ \pi_{yx}$:

A	P	Q	$\neg P$	$\neg Q$
a	$\top$	$\bot$	$\bot$	$\top$
b	$\bot$	$\top$	$\top$	$\bot$
c	$\top$	$\top$	$\bot$	$\bot$
d	$\bot$	$\bot$	$\top$	$\top$

Figure 2 Illustrations of $h \circ \pi_{xy}$ next to $h \circ \pi_{yx}$ for all $h \in S = \{h_\varnothing, h_{\{x\}}, h_{\{y\}}, h_{\{x,y\}}\}$.

be obtained by exploiting homomorphisms into $\mathbb{W}[X]$ itself. Consider any permutation $\sigma\colon X \to X$ of the variables. Surely, it induces an automorphism h_σ of $\mathbb{W}[X]$. In the previous example, if σ swaps the variables x and y, then applying h_σ to π_{xy} yields an interpretation that is isomorphic to π_{yx}, as illustrated below.

$(h_\sigma \circ \pi_{xy})$:

A	P	Q	$\neg P$	$\neg Q$
a	0	x	y	0
b	y	0	0	x
c	x	y	0	0
d	0	0	x	y

π_{yx} :

A	P	Q	$\neg P$	$\neg Q$
a	y	0	0	x
b	0	x	y	0
c	x	y	0	0
d	0	0	x	y

With this insight, we can construct a suitable separating set $S \subseteq \mathrm{Hom}^2(\mathbb{W}[X], \mathbb{W}[X])$, starting with the pair $(h_\sigma, h_{\mathrm{id}}) \in S$. This pair alone does not separate $\mathbb{W}[X]$, since we have $x \neq y$ in $\mathbb{W}[X]$, but $h_\sigma(x) = y = h_{\mathrm{id}}(y)$. Hence, we add more homomorphisms by annihilating some variables, similarly to the construction for $\mathsf{PosBool}[X]$ in Lemma 10. Fixing a permutation $\sigma\colon X \to X$, we want to construct a homomorphism h^Y_σ that annihilates all variables in $X \setminus Y$ and permutes the variables in Y. Observe that for each $x \in Y$ there is a minimal number $r(x) \geq 1$ such that $\sigma^{r(x)}(x) \in Y$. Formally, we define $\sigma^Y\colon X \to Y \cup \{0\}$ by setting $\sigma^Y(x) \coloneqq \sigma^{r(x)}(x)$ for $x \in Y$, and $\sigma^Y(x) \coloneqq 0$ for $x \in X \setminus Y$. Note that σ^Y induces a homomorphism $h^Y_\sigma\colon \mathbb{W}[X] \to \mathbb{W}[X]$.

▶ **Lemma 14.** *$S \coloneqq \{(h^Y_\sigma, h^Y_{\mathrm{id}}) \mid Y \subseteq X\} \subseteq \mathrm{Hom}^2(\mathbb{W}[X], \mathbb{W}[X])$ is a separating set of homomorphism pairs.*

Proof. Suppose that $p \neq q$ for a pair $p, q \in \mathbb{W}[X]$. A monomial in $\mathbb{W}[X]$ can be identified with the set of its variables. Thus, without loss of generality, there is some $Y \subseteq X$ with $Y \in p$ and $Y \notin q$. Surely, $h^Y_\sigma(p)$ contains the monomial $h^Y_\sigma(Y) = Y$, but $h^Y_{\mathrm{id}}(q)$ only contains monomials from q, hence it does not contain Y and $h^Y_\sigma(p) \neq h^Y_{\mathrm{id}}(q)$. ◀

Before applying the reduction technique to obtain $\pi_{xy} \equiv \pi_{yx}$ from Corollary 12, it only remains to show that $(h^Y_\sigma \circ \pi_{xy}) \equiv (h^Y_{\mathrm{id}} \circ \pi_{yx})$ for all $Y \subseteq X := \{x, y\}$. Since we have already illustrated $(h^X_\sigma \circ \pi_{xy}) \cong (h^X_{\mathrm{id}} \circ \pi_{yx})$, we only need to consider the cases where $Y \subsetneq X$. But then, at most one variable is contained in Y and the remaining variables are annihilated by h^Y_σ and h^Y_{id}, thus $(h^Y_\sigma \circ \pi_{xy}) \cong (h^Y_{\mathrm{id}} \circ \pi_{yx})$ clearly follows. The reduction technique then implies that $\pi_{xy} \equiv \pi_{yx}$ on $\mathbb{W}[X]$, which can naturally be lifted to all fully idempotent semirings thanks to the universal property.

▶ **Theorem 15.** *For any fully idempotent semiring K with at least three elements, there is a pair of finite K-interpretations π_{rs}, π_{sr} over a universe with four elements and a signature with two unary relation symbols such that $\pi_{rs} \equiv \pi_{sr}$, but $\pi_{rs} \not\cong \pi_{sr}$.*

In conclusion, the proof of $\pi_{xy} \equiv \pi_{yx}$ on $\mathsf{PosBool}[X]$ illustrates how elementary equivalence in semiring semantics can be reduced to elementary equivalence on $\mathbb{B}$ by completely decomposing $\mathsf{PosBool}[X]$ to $\mathbb{B}$ with homomorphisms. Moreover, the proof of elementary equivalence on $\mathbb{W}[X]$ shows that it may even pay off to use separating sets of homomorphisms from $\mathbb{W}[X]$ to $\mathbb{W}[X]$ itself.

5 Characteristic Sentences

The results of the previous section raise the question whether it is possible to construct a similar example of non-isomorphic, but elementarily equivalent interpretations also for the most general semiring $\mathbb{N}[X]$, and lift it to all commutative semirings with at least three elements. In order to show that this is not possible, we draw inspiration from classical semantics, where for each finite τ-structure $\mathfrak{A}$ with universe $A = \{a_1, \ldots, a_n\}$ one can construct a *characteristic sentence* $\chi_{\mathfrak{A}}$ such that $\mathfrak{B} \models \chi_{\mathfrak{A}}$ if, and only if, $\mathfrak{A} \cong \mathfrak{B}$. The characteristic sentence is explicitly defined as

$$\chi_{\mathfrak{A}} := \exists x_1 \ldots \exists x_n (\varphi(\overline{x}) \wedge \psi(\overline{x})) \quad \text{with}$$
$$\varphi(\overline{x}) := \bigwedge_{1 \leq i < j \leq n} x_i \neq x_j \wedge \forall y \bigvee_{i \leq n} y = x_i \quad \text{and} \quad \psi(\overline{x}) := \bigwedge \{L(\overline{x}) \in \mathrm{Lit}_n(\tau) \mid \mathfrak{A} \models L(\overline{a})\}.$$

The subformula $\varphi(\overline{x})$ of this sentence asserts that the universe has precisely n elements assigned to the variables $\overline{x}$. Since $\varphi(\overline{x})$ uses only equalities and inequalities it can be used as-is for semiring semantics in any semiring.

▶ **Lemma 16.** *For every K-interpretation $\pi_B \colon \mathrm{Lit}_B(\tau) \to K$ into an arbitrary semiring K and every tuple $\overline{b} = (b_1, \ldots, b_n)$, we have that $\pi_B[\![\varphi(\overline{b})]\!] = 1$ if $B = \{b_1, \ldots, b_n\}$ and $b_i \neq b_j$ for $i \neq j$, and $\pi_B[\![\varphi(\overline{b})]\!] = 0$, otherwise.*

Proof. Semiring interpretations evaluate equalities and inequalities to 0 and 1, so

$$\pi_B[\![\varphi(\overline{b})]\!] = \prod_{i<j} \pi_B[\![b_i \neq b_j]\!] \cdot \prod_{b \in B} \Big(\sum_{i \leq n} \pi_B[\![b = b_i]\!]\Big)$$

evaluates to 1 if $b_1, \ldots, b_n$ is a distinct enumeration of all elements of B, and to 0 otherwise. ◀

On the other side, $\psi(\overline{a})$ is the conjunction of all true literals in $\mathfrak{A}$. Since $\mathfrak{A}$ satisfies precisely one literal out of each pair of opposing literals L and $\overline{L}$, it is clear that $\psi(\overline{a})$ describes $\mathfrak{A}$ up to isomorphism. However, this approach does not lift to arbitrary semiring interpretations π_A, since different literals in π_A may have different non-zero values, but conjunctions are interpreted as products, and it is in general impossible to trace the result back to the contributions of the literals.

5.1 The Viterbi Semiring

The Viterbi semiring $\mathbb{V} = ([0,1]_{\mathbb{R}}, \max, \cdot, 0, 1)$ is used in confidence analysis, probabilistic parsing, and Hidden Markov Models (see [8, 10]). It is isomorphic to the tropical semiring $\mathbb{T} = (\mathbb{R}_+^\infty, \min, +, \infty, 0)$, used for instance for cost analysis and performance evaluation, via $x \mapsto e^{-x}$. Hence, all results that we establish for the Viterbi semiring also hold for the tropical semiring. We can illustrate the shortcomings of the characteristic sentences in their classical form by very simple $\mathbb{V}$-interpretations with one element.

π_{19} :

A	P	Q	$\neg P$	$\neg Q$
a	0.1	0.9	0	0

π_{91} :

A	P	Q	$\neg P$	$\neg Q$
a	0.9	0.1	0	0

They are clearly not isomorphic, but trying to construct χ_{19} from π_{19} as above would yield $\chi_{19} = \exists x(\varphi(x) \wedge \psi(x))$ with $\psi(x) = Px \wedge Qx$, hence $\pi_{19}[\![\chi_{19}]\!] = 0.1 \cdot 0.9 = 0.9 \cdot 0.1 = \pi_{91}[\![\chi_{19}]\!]$.

However, under semiring semantics, and especially on the Viterbi semiring $\mathbb{V}$, multiplication need not be idempotent; hence we can hope to distinguish two interpretations by simply repeating one of the literals. In the given example, we can set $\psi(x) \coloneqq Px \wedge (Qx)^2$, which is short for $Px \wedge Qx \wedge Qx$, to obtain $\pi_{19}[\![\chi_{19}]\!] = 0.1 \cdot 0.9^2 \neq 0.9 \cdot 0.1^2 = \pi_{91}[\![\chi_{19}]\!]$. We now generalise this idea to arbitrary finite $\mathbb{V}$-interpretations. We shall associate with every finite $\mathbb{V}$-interpretation $\pi_A \colon \mathrm{Lit}_A(\tau) \to \mathbb{V}$ and every $\varepsilon \in \mathbb{R}^+$ a characteristic sentence

$$\chi_{\pi_A,\varepsilon} \coloneqq \exists x_1 \ldots \exists x_n(\varphi(\overline{x}) \wedge \psi_\varepsilon(\overline{x})),$$

with $n \coloneqq |A|$ and $\varphi(x)$ as before, but a more involved construction of $\psi_\varepsilon(\overline{x})$:

Let $\overline{a} = (a_1, \ldots, a_n)$ be some fixed order on A and $L_1(\overline{a}), \ldots, L_k(\overline{a})$ an arbitrary enumeration of the "true" literals in $\mathrm{Lit}_A(\tau)$ with $\pi_A(L_i(\overline{a})) \neq 0$. Further, fix a sequence $f(1), \ldots, f(k)$ of "exponents" in $\mathbb{N}$, where $f(1) = 1$ and $f(i+1)$ is chosen large enough so that

$$(*) \qquad (1-\varepsilon)^{f(i+1)} < \varepsilon^{f(1)+\cdots+f(i)}.$$

Then, put $\psi_\varepsilon(\overline{x}) \coloneqq \bigwedge_{i=1}^k L_i(\overline{x})^{f(i)}$, where "exponentiation" denotes repetition of a literal.

The idea is that $\chi_{\pi_A,\varepsilon}$ should characterise π_A up to isomorphism by repeating the literals in $\psi_\varepsilon(\overline{x})$ "sufficiently often" so that the contribution of each literal can be distinguished and changing the value for one literal surely alters the final value of $\psi_\varepsilon(\overline{x})$. Since the elements of $\mathbb{V}$ are from $[0,1]_{\mathbb{R}}$, the values of the literals can change by an arbitrarily small amount, hence the "exponents" $f(i)$ must depend on the "smallest possible change" ε. This intuition is formalised as follows.

▶ **Proposition 17.** *Let $\pi_A \colon \mathrm{Lit}_A(\tau) \to \mathbb{V}$ and $\pi_B \colon \mathrm{Lit}_B(\tau) \to \mathbb{V}$ be two finite, model-defining $\mathbb{V}$-interpretations, which induce the finite set of values*

$$V \coloneqq \{\pi_A(L) \mid L \in \mathrm{Lit}_A(\tau)\} \cup \{\pi_B(L) \mid L \in \mathrm{Lit}_B(\tau)\}.$$

Then, for every $\varepsilon \in \mathbb{R}$ bounded by $0 < \varepsilon \leq \min\{|r-s| \mid r,s \in V, r \neq s\}$, we have that $\pi_A[\![\chi_{\pi_A,\varepsilon}]\!] = \pi_B[\![\chi_{\pi_A,\varepsilon}]\!]$ implies $\pi_A \cong \pi_B$.

Proof. Assume $\pi_A[\![\chi_{\pi_A,\varepsilon}]\!] = \pi_B[\![\chi_{\pi_A,\varepsilon}]\!]$. By construction, $\pi_A[\![\chi_{\pi_A,\varepsilon}]\!] > 0$, so $\pi_B[\![\chi_{\pi_A,\varepsilon}]\!] > 0$ as well. By Lemma 16, together with the fact that the existential quantifiers $\exists x_1 \dots \exists x_n$ in $\chi_{\pi_A,\varepsilon}$ are interpreted as max in the Viterbi semiring $\mathbb{V}$, this implies that $|A| = |B|$, and that we have enumerations $\overline{a} = (a_1, \dots, a_n)$ and $\overline{b} = (b_1, \dots, b_n)$ of the elements of A and B, such that

$$\pi_A[\![\chi_{\pi_A,\varepsilon}]\!] = \pi_A[\![\psi_\varepsilon(\overline{a})]\!] = \pi_B[\![\psi_\varepsilon(\overline{b})]\!] = \pi_B[\![\chi_{\pi_A,\varepsilon}]\!].$$

Recall that $\pi_A[\![\psi_\varepsilon(\overline{a})]\!] = \prod_{i=1}^{k} \pi_A(L_i(\overline{a}))^{f(i)} > 0$, hence $\pi_A(L_1(\overline{a})), \dots, \pi_A(L_k(\overline{a}))$ are all positive. Accordingly, $\pi_B[\![\psi_\varepsilon(\overline{b})]\!] = \prod_{i=1}^{k} \pi_B(L_i(\overline{b}))^{f(i)} > 0$, so that $\pi_B(L_1(\overline{b})), \dots, \pi_B(L_k(\overline{b}))$ are positive as well. Given that π_A and π_B share the same signature and universe size, any permutations $\overline{a}$, $\overline{b}$ of their elements yield the same number of positive literals, which is k by definition. We infer that all remaining literals in both interpretations are mapped to zero. Hence, for $i = 1, \dots, k$, let $r_i \coloneqq \pi_A(L_i(\overline{a})) > 0$ and $s_i \coloneqq \pi_B(L_i(\overline{b})) > 0$ be the values of the positive literals, then it only remains to show that $r_i = s_i$ for all $i \leq k$ in order to conclude that $\overline{a} \mapsto \overline{b}$ is indeed an isomorphism from π_A to π_B.

Towards a contradiction, assume that this is not the case and let j be the maximal index among $1, \dots, k$ with $r_j \neq s_j$. We can assume that $r_j < s_j$. Since the difference between the two values is at least ε and since $s_j \leq 1$, it follows that $r_j \leq s_j - \varepsilon \leq s_j - \varepsilon s_j = (1-\varepsilon)s_j$. Further, we have $\varepsilon \leq s_i, r_i \leq 1$ for all i. It follows that

$$r_1^{f(1)} \cdots r_j^{f(j)} \leq r_j^{f(j)} \leq (1-\varepsilon)^{f(j)} s_j^{f(j)} \overset{*}{<} \varepsilon^{f(1)+\cdots+f(j-1)} \cdot s_j^{f(j)} \leq s_1^{f(1)} \cdots s_j^{f(j)}.$$

However, since $r_i = s_i$ for $i = j+1, \dots, k$, this would imply that

$$\pi_A[\![\psi_\varepsilon(\overline{a})]\!] = \prod_{i \leq k} r_i^{f(i)} \neq \prod_{i \leq k} s_i^{f(i)} = \pi_B[\![\psi_\varepsilon(\overline{b})]\!]$$

and hence $\pi_A[\![\chi_{\pi_A,\varepsilon}]\!] \neq \pi_B[\![\chi_{\pi_A,\varepsilon}]\!]$. ◀

Notice that none of the sentences $\chi_{\pi_A,\varepsilon}$ characterises π_A alone, but the countable set $X_{\pi_A} \coloneqq \{\chi_{\pi_A,\varepsilon} \mid \varepsilon \in \mathbb{Q}^+\}$ does so. No infinite $\mathbb{V}$-interpretation π_B agrees with π_A on any of the ε-characteristic sentences $\chi_{\pi_A,\varepsilon}$ due to $\varphi(\overline{x})$, whereas for each finite $\mathbb{V}$-interpretation π_B, one can calculate an $\varepsilon \in \mathbb{Q}$ to apply the proposition just proved.

▶ **Theorem 18.** *For finite $\mathbb{V}$-interpretations π_A and π_B, $\pi_A \equiv \pi_B$ implies $\pi_A \cong \pi_B$.*

As a consequence, there are indeed interesting semirings beyond the Boolean semiring $\mathbb{B}$, where elementary equivalence implies isomorphism on finite interpretations.

5.2 Finite Axiomatisability

The characteristic set X_{π_A} raises the question whether a finite set of sentences suffices to characterise a $\mathbb{V}$-interpretation π_A. We will answer this question positively using two observations. By Proposition 17, we observe that $\chi_{\pi_A,\varepsilon}$ characterises π_A up to isomorphism inside the class of $\mathbb{V}$-interpretations that only use values in $V = \{\pi_A(L) \mid L \in \mathrm{Lit}_A(\tau)\}$, with $\varepsilon \coloneqq \min\{|r-s| \mid r, s \in V, r \neq s\}$. Hence, π_A can be characterised by adding sentences to ensure that no values outside of V are used.

We will show that this is possible by building sentences that fix particular values $\pi_A(L)$.

▶ **Definition 19.** Let $\pi\colon \mathrm{Lit}_A(\tau) \to \mathbb{V}$ be a finite $\mathbb{V}$-interpretation over a universe A with n elements and $\varphi(\overline{x}) \in \mathrm{FO}(\tau)$ a formula with $k \in \mathbb{N}$ free variables. The sequence $(s^i_{\pi,\varphi})_{1 \leq i \leq n^k}$ is defined as the non-increasingly sorted sequence of the values $\pi[\![\varphi(\overline{a})]\!]$ for $\overline{a} \in A^k$.

In particular, $s^1_{\pi,\varphi}$ is the largest possible value $\pi[\![\varphi(\overline{a})]\!]$; further, $s^2_{\pi,\varphi} \leq s^1_{\pi,\varphi}$ is either the second largest one, or equal to $s^1_{\pi,\varphi}$ if the maximal value is shared by two distinct tuples $\overline{a}, \overline{b} \in A^k$, and so on. We construct a series of sentences that fix the values $(s^i_{\pi,\varphi})_{1 \leq i \leq n^k}$.

▶ **Lemma 20** (Sorting Lemma). *For $\varphi(\overline{x}) \in \mathrm{FO}(\tau)$ with k free variables and $1 \leq i \leq n^k$, let*

$$\psi^i_\varphi \coloneqq \exists \overline{x}_1 \ldots \exists \overline{x}_i \Big(\bigwedge_{1 \leq j < \ell \leq i} \overline{x}_j \neq \overline{x}_\ell \wedge \bigwedge_{j=1}^{i} \varphi(\overline{x}_j) \Big)$$

where $\overline{x}_1, \ldots \overline{x}_i$ are k-tuples of variables. Then, for any $\mathbb{V}$-interpretation $\pi \colon \mathrm{Lit}_A(\tau) \to \mathbb{V}$ over a universe with n elements, we have that

$$\pi[\![\psi^i_\varphi]\!] = \prod_{1 \leq j \leq i} s^j_{\pi,\varphi} \quad \text{for } 1 \leq i \leq n^k.$$

Proof. Recall that existential quantifiers are interpreted as max on $\mathbb{V}$. Due to monotonicity of multiplication, the maximum $\pi[\![\psi^i_{\pi,\varphi}]\!]$ is achieved by picking the i pairwise distinct tuples $\overline{a}_1, \ldots, \overline{a}_i$ that yield the largest values $\pi[\![\varphi(\overline{a}_j)]\!] = s^j_{\pi,\varphi}$ and inserting them for $\overline{x}_1, \ldots, \overline{x}_i$. Clearly, this yields $\pi[\![\psi^i_\varphi]\!] = \prod_{1 \leq j \leq i} s^j_{\pi,\varphi}$. ◀

By observing that $\mathbb{V}$ is cancellative, i.e. $ab = ac$ implies $b = c$ for all $a, b, c \in \mathbb{V}$ with $a \neq 0$, we may disentangle the products $\prod_{1 \leq j \leq i} s^j_{\pi,\varphi}$ to draw the following conclusion.

▶ **Corollary 21.** *Let $\varphi(\overline{x})$ be as above, and consider two $\mathbb{V}$-interpretations $\pi \colon \mathrm{Lit}_A(\tau) \to \mathbb{V}$ and $\pi' \colon \mathrm{Lit}_B(\tau) \to \mathbb{V}$ with $|A| = |B| = n$. If π and π' agree on $\Psi \coloneqq \{\psi^i_\varphi \mid 1 \leq i \leq n^k\}$, then $s^i_{\pi,\varphi} = s^i_{\pi',\varphi}$ for all $1 \leq i \leq n^k$. In other words, the values $\pi[\![\varphi(\overline{a})]\!]$ for $\overline{a} \in A^k$ and $\pi'[\![\varphi(\overline{b})]\!]$ for $\overline{b} \in B^k$ are the same, up to permutation.*

If $R \in \tau$ is a k-ary relation, pick $\varphi(\overline{x}) \coloneqq R\overline{x}$ and construct $\Psi_R \coloneqq \{\psi^i_{R\overline{x}} \mid 1 \leq i \leq n^k\}$ according to the Sorting Lemma. Similarly, construct $\Psi_{\neg R}$ from $\varphi(\overline{x}) \coloneqq \neg R\overline{x}$. Then, define

$$\Psi_\tau \coloneqq \bigcup_{R \in \tau} \Psi_R \cup \bigcup_{R \in \tau} \Psi_{\neg R}.$$

Clearly, any two $\mathbb{V}$-interpretations over τ that agree on Ψ_τ use the same set of values from $\mathbb{V}$. Putting this together with the characteristic sentences $\chi_{\pi_A,\varepsilon}$ from Proposition 17 provides a finite axiomatisation of any $\mathbb{V}$-interpretation.

▶ **Theorem 22.** *Let $\pi \colon \mathrm{Lit}_A(\tau) \to \mathbb{V}$ be a finite $\mathbb{V}$-interpretation. Then, $\Psi_\tau \cup \{\chi_{\pi,\varepsilon}\}$ is a finite axiomatisation of π up to isomorphism, where $\varepsilon \coloneqq \min\{|r - s| \mid r, s \in \pi(\mathrm{Lit}_A(\tau)), r \neq s\}$.*

Proof. Let $\pi' \colon \mathrm{Lit}_B(\tau) \to \mathbb{V}$ agree with π on all sentences in $\Psi_\tau \cup \{\chi_{\pi,\varepsilon}\}$. Due to the construction of $\chi_{\pi,\varepsilon}$, π' is finite and $|A| = |B|$. Since π' agrees with π on Ψ_τ, we have $\{\pi(L) \mid L \in \mathrm{Lit}_A(\tau)\} = \{\pi'(L) \mid L \in \mathrm{Lit}_B(\tau)\}$. Thus, we can invoke Proposition 17 by observing that $V = \pi(\mathrm{Lit}_A(\tau))$ and conclude that $\pi[\![\chi_{\pi,\varepsilon}]\!] = \pi'[\![\chi_{\pi,\varepsilon}]\!]$ implies $\pi \cong \pi'$. ◀

Under classical semantics, any finite axiom system $\Phi \subseteq \mathrm{FO}(\tau)$ can be collapsed to a single axiom $\psi \coloneqq \bigwedge \Phi$, but this is not the case in semiring semantics. To illustrate this, we shall show that there are $\mathbb{V}$-interpretations that cannot be axiomatised up to isomorphism by a single sentence.

▶ **Proposition 23.** *There exist $\mathbb{V}$-interpretations $\pi \colon \mathrm{Lit}_A(\tau) \to \mathbb{V}$ such that, for every sentence $\psi \in \mathrm{FO}(\tau)$, there exists an interpretation $\pi' \colon \mathrm{Lit}_A(\tau) \to \mathbb{V}$ such that $\pi \ncong \pi'$, but $\pi[\![\psi]\!] = \pi'[\![\psi]\!]$.*

Proof. Take an interpretation with just two atoms Pa and Qa and with values $\pi(Pa) = p$ and $\pi(Qa) = q$ such that $0 < p, q < 1$ are multiplicatively independent real numbers, i.e. $k = \ell = 0$ is the only solution to $p^k q^\ell = 1$ with $k, \ell \in \mathbb{Z}$. Let $\pi_{\mathbb{B}}$ be the corresponding $\mathbb{B}[x, y]$-interpretation, with $\pi_{\mathbb{B}}(Pa) = x$ and $\pi_{\mathbb{B}}(Qa) = y$. A sentence $\psi \in \mathrm{FO}$ is evaluated under $\pi_{\mathbb{B}}$ to a polynomial $\pi_{\mathbb{B}}[\![\psi]\!] \in \mathbb{B}[x, y]$, and by the universal property for idempotent semirings, the homomorphism $h\colon \mathbb{B}[x, y] \to \mathbb{V}$ induced by $h(x) = p$ and $h(y) = q$ maps $\pi_{\mathbb{B}}[\![\psi]\!]$ to $\pi[\![\psi]\!]$. Writing $\pi_{\mathbb{B}}[\![\psi]\!]$ as a sum of monomials $m = x^i y^j$, we conclude that $\pi[\![\psi]\!] = p^i q^j$ is the maximal value $m(p, q)$ for the monomials m occuring in $\pi_{\mathbb{B}}[\![\psi]\!]$. Since p, q are multiplicatively independent, no other monomial can take the same value, i.e. $m'(p, q) < m(p, q)$ for all other monomials m' in $\pi_{\mathbb{B}}[\![\psi]\!]$. We now can certainly find a value $r \neq p$ that is sufficiently close to p, and a value s such that $r^i s^j = p^i q^j$, i.e. $m(r, s) = m(p, q)$, but $m'(r, s) < m(r, s)$ for all other monomials m' in $\pi_{\mathbb{B}}[\![\psi]\!]$. For the $\mathbb{V}$-interpretation π' with $\pi'(Pa) = r$ and $\pi'(Qa) = s$ this implies that $\pi'[\![\psi]\!] = r^i s^j = p^i q^j = \pi[\![\psi]\!]$, but clearly, $\pi' \not\cong \pi$. ◀

This result can be strengthened in many directions. It holds, in fact, for almost all $\mathbb{V}$-interpretations, as long as they do not map all literals to either 0 or 1. Further, we shall exploit the isomorphism of $\mathbb{V}$ and $\mathbb{T}$ in order to prove explicit lower bounds on the number of axioms that are needed to characterise an interpretation, depending on the number of literals mapped to multiplicatively independent values.

5.3 Lower Bound for Axiomatisations of $\mathbb{T}$- and $\mathbb{V}$-interpretations

Recall that $\mathbb{V} = ([0, 1]_{\mathbb{R}}, \max, \cdot, 0, 1)$ is isomorphic to $\mathbb{T} = (R_+^\infty, \min, +, \infty, 0)$ via isomorphisms $\sigma_{\mathbb{V}\to\mathbb{T}}(a) = -\log_b(a)$ for any fixed base $b \in \mathbb{R}_{>1}$, and the corresponding inverse isomorphisms $\sigma_{\mathbb{T}\to\mathbb{V}}(a) = b^{-a}$ for any fixed $b \in \mathbb{R}_{>1}$. We formulate our result in terms of $\mathbb{T}$.

▶ **Theorem 24.** *Let $\pi\colon \mathrm{Lit}_A(\tau) \to \mathbb{T}$ be any finite, model-defining $\mathbb{T}$-interpretation with $|A| = n$ and $|\mathrm{Lit}_A(\tau)| = 2\ell$, such that its finite values $\pi(\mathrm{Lit}_A(\tau)) \setminus \{\infty\}$ are linearly independent over $\mathbb{Q}$. Then, for any set of sentences $\Psi \subseteq \mathrm{FO}(\tau)$ with $|\Psi| < \ell$, there is an interpretation $\pi'\colon \mathrm{Lit}_A(\tau) \to \mathbb{T}$ such that $\pi[\![\psi]\!] = \pi'[\![\psi]\!]$ for all $\psi \in \Psi$, but $\pi \not\cong \pi'$.*

Proof. Since π is model-defining, there are ℓ literals L in $\mathrm{Lit}_A(\tau)$ with $\pi(L) \neq \infty$, which we call the positive literals. Choose $X \coloneqq \{x_1, \ldots, x_\ell\}$ and construct the $\mathbb{B}[X]$-interpretation $\pi_{\mathbb{B}}\colon \mathrm{Lit}_A(\tau) \to \mathbb{B}[X]$ by assigning a unique variable to each of the positive literals. Clearly, there is a homomorphism $h\colon \mathbb{B}[X] \to \mathbb{T}$ with $h \circ \pi_{\mathbb{B}} = \pi$ induced by mapping each variable to the original value $\pi(L)$ of the corresponding literal.

Enumerate $\Psi = \{\psi_1, \ldots, \psi_j\}$ arbitrarily with $j < \ell$ and construct the polynomials $p_i \coloneqq \pi_{\mathbb{B}}[\![\psi_i]\!] \in \mathbb{B}[X]$ for $1 \leq i \leq j$. By the fundamental property, $\pi[\![\psi_i]\!] = h(p_i)$ holds for $1 \leq i \leq j$.

We assume without loss of generality that $p_i \neq 0$ for all $1 \leq i \leq j$, and we will construct π' with the same positive literals as π. Thus, p_i contains a monomial m_i so that $h(m_i)$ is minimal among $\{h(m) : m \in p_i\}$. This monomial is unique thanks to the linear independence of the values of π, which guarantees that $h(m) \neq h(m')$ for $m \neq m'$. Suppose for a contradiction that $h(m) = h(m')$. We may write

$$h(m) = h\left(\prod_{i=1}^{\ell} x_i^{m(x_i)}\right) = \sum_{i=1}^{\ell} m(x_i) \cdot h(x_i),$$

which implies that $h(m)$ is a linear combination of the values of π. In particular, $h(m) = h(m')$ implies that $h(m - m') = 0$, hence $m(x_i) - m'(x_i) = 0$ for all $1 \leq i \leq j$ due to linear independence.

We conclude that there is a sufficiently small $\varepsilon \in \mathbb{R}_{>0}$ such that changing the numbers $h(x_i)$ by less than ε does not affect the monomial order. In other words, view the values $\overline{v} \coloneqq (h(x_1), \dots, h(x_\ell)) \in \mathbb{R}^\ell_{\geq 0}$ as a vector and notice that any $\overline{w} \in \mathbb{R}^\ell_{\geq 0}$ with $|\overline{v} - \overline{w}| < \varepsilon$ preserves the monomial order, so that if we construct $h' \colon \mathbb{B}[X] \to \mathbb{T}$ induced by $h'(x_i) = w_i$ for $1 \leq i \leq \ell$, we have $h(m) < h(m')$ if, and only if, $h'(m) < h'(m')$.

To complete the proof, it remains to ensure that $h'(p_i) = h(p_i)$ stays the same for all $1 \leq i \leq j$. By the above considerations, it suffices to ensure that $h'(m_i) = h(m_i)$ for the corresponding maximal monomials $m_1, \dots, m_j$. Each of these monomials induces one condition $h(m_i) - h'(m_i) = 0$, which translates to a linear equation

$$h(m_i) - h'(m_i) = \sum_{i=1}^{\ell} m_i(x_i)(h(x_i) - h'(x_i)) = \sum_{i=1}^{\ell} m_i(x_i) \cdot (v_i - w_i) = 0$$

on $(\overline{v} - \overline{w})$.

Since there are only $j < \ell$ equations and ℓ variables, the solution space is at least one-dimensional, meaning that we can pick $\overline{w} \neq \overline{v}$ adequately with $|\overline{v} - \overline{w}| < \varepsilon$ to satisfy all equations and obtain $h'(p_i) = h(p_i)$ for all $1 \leq i \leq j$. Note that due to linear independence, none of the entries from $\overline{v}$ was zero, hence it is possible to ensure that $\overline{w}$ only has positive entries. We thus can pick $\pi' \coloneqq h' \circ \pi_{\mathbb{B}}$ with the desired properties. ◀

This result translates to $\mathbb{V}$ thanks to isomorphism. Linear independence of values from $\mathbb{T}$ as $\mathbb{Q}$-vectors translates to multiplicative independence of the corresponding values from $\mathbb{V}$.

5.4 The Semirings $\mathbb{N}$ and $\mathbb{N}[X]$

We will now provide a similar analysis of axiomatisablity for the most general semiring $\mathbb{N}[X]$ by taking a detour via $\mathbb{N}$. For the construction of the characteristic sentences for $\mathbb{N}$, we shall need the following combinatorial lemma.

▶ **Lemma 25.** *For any two natural numbers k, c with $c > 1$, there exists a exponent e such that, for any two non-decreasing sequences $r_1 \leq r_2 \leq \dots \leq r_k$ and $s_1 \leq s_2 \leq \dots \leq s_k$ of k natural numbers, with $r_k, s_k < c$, the equation $r_1^e + \dots + r_k^e = s_1^e + \dots + s_k^e$ implies that the two sequences are the same, i.e. $r_i = s_i$ for all $i \leq k$.*

Proof. Choose e large enough so that $(c/(c-1))^e > k$. Towards a contradiction, assume that there are two *distinct* sequences $r_1 \leq r_2 \leq \dots r_k < c$ and $s_1 \leq s_2 \leq \dots s_k < c$ such that $r_1^e + \dots + r_k^e = s_1^e + \dots + s_k^e$. Let j be the maximal index with $r_j \neq s_j$. Thanks to additive cancellation, we can remove the summands with index $i > j$ to obtain that $r_1^e + \dots + r_j^e = s_1^e + \dots + s_j^e$. By symmetry we can assume that $r_j < s_j$. Since $s_j < c$, it follows that $s_j > (c/(c-1))r_j$ and hence $s_j^e > k \cdot r_j^e$. But this implies that

$$r_1^e + \dots + r_j^e \leq j \cdot r_j^e \leq k \cdot r_j^e < s_j^e \leq s_1^e + \dots + s_j^e,$$

contradicting the equation above. ◀

▶ **Lemma 26.** *Let $(r_1, \dots r_k), (s_1, \dots, s_k) \in \mathbb{N}^k$ be strictly bounded by c, that is $r_i, s_i < c$ for all $i \leq k$. Then, there is an exponent e depending only on c and k such that*

$$\sum_{i=1}^{k} r_i^e = \sum_{i=1}^{k} s_i^e$$

implies that there is a permutation $\sigma \in S_k$ such that $r_i = s_{\sigma(i)}$ for all $1 \leq i \leq k$.

Proof. Sort both sequences non-decreasingly, that is, permute them with $\rho, \tau \in S_k$ so that $r_{\rho(1)} \leq \ldots \leq r_{\rho(k)}$ and $s_{\tau(1)} \leq \ldots \leq s_{\tau(k)}$. By the previous Lemma, there is a suitable e such that $r_{\rho(i)} = s_{\tau(i)}$ for all $i \leq k$. Then, $\sigma \coloneqq \tau \circ \rho^{-1}$ is the desired permutation. ◀

We are now ready to construct characteristic sentences for finite $\mathbb{N}$-interpretations $\pi_A \colon \mathrm{Lit}_A(\tau) \to \mathbb{N}$. For $n = |A|$, let $L_1(\overline{x}), \ldots, L_k(\overline{x})$ be an enumeration of all literals in $\mathrm{Lit}_n(\tau)$. For any constant $q \in \mathbb{N}$ we define the *q-characteristic sentence* $\chi_{\pi_A,q}$ as

$$\chi_{\pi_A,q} \coloneqq \exists x_1 \ldots \exists x_n (\varphi(\overline{x}) \wedge \psi_q(\overline{x}))^e, \quad \text{with } \psi_q(\overline{x}) \coloneqq \bigvee_{i=1}^{k} q^{i-1} \cdot L_i(\overline{x}),$$

with $\varphi(\overline{x})$ as given before and e is an exponent that depends on q, n and τ, according to Lemma 26. The notation $q^{i-1} \cdot L_i(\overline{x})$ denotes a disjunctive repetition of the literal $L_i(\overline{x})$ for q^{i-1} times.

The idea of this construction is similar to the one for the Viterbi semiring. While ε-characteristic sentences work for $\mathbb{V}$-interpretations where the differences of two distinct values are at least ε, q-characteristic sentences work for $\mathbb{N}$-interpretations with values less than q. With this in mind, we can explain the construction of $\psi_q(\overline{x})$ as follows. If all values in π_A are less than q, we can picture the value

$$\pi_A[\![\psi_q(\overline{a})]\!] = \sum_{i=1}^{k} q^{i-1} \pi_A(L_i(\overline{a})),$$

to be in a number system with radix q, hence the values $\pi_A(L_i(\overline{a}))$ can be seen as digits. Thus, it is immediately clear that for any $\mathbb{N}$-interpretation π_B with universe enumerated by $\overline{b}$ and values less than q, $\pi_A[\![\psi_q(\overline{a})]\!] = \pi_B[\![\psi_q(\overline{b})]\!]$ implies that $\overline{a} \xrightarrow{\sim} \overline{b}$ is an isomorphism between π_A and π_B, since the corresponding "digits" $\pi_A(L_i(\overline{a}))$ and $\pi_B(L_i(\overline{b}))$ for each $1 \leq i \leq k$ have to be the same.

The only remaining problem is the fact that existential quantifiers $\exists x_1 \ldots \exists x_n$ from $\chi_{\pi_A,q}$ are interpreted as a sum in $\mathbb{N}$. Thus, the value $\pi_A[\![\chi_{\pi_A,q}]\!]$ is not induced by a single variable assignment $\overline{a}$. The exponent e is used to separate the contributions of different variable assignments to the sum on the basis of Lemma 26.

▶ **Theorem 27.** *Let π_A and π_B are finite $\mathbb{N}$-interpretations with values less than q. Then $\pi_A[\![\chi_{\pi_A,q}]\!] = \pi_B[\![\chi_{\pi_A,q}]\!]$ implies that $\pi_A \cong \pi_B$.*

Proof. Clearly, $\varphi(\overline{x})$ takes care of the number of elements, hence we can assume $\overline{a}$ and $\overline{b}$ enumerate the universes of π_A and π_B. Now, we have

$$\pi_A[\![\chi_{\pi_A,q}]\!] = \sum_{\sigma \in S_n} \pi_A[\![\psi_q(\sigma(\overline{a}))]\!]^e = \sum_{\sigma \in S_n} \pi_B[\![\psi_q(\sigma(\overline{b}))]\!]^e = \pi_B[\![\chi_{\pi_A,q}]\!].$$

Recall that $\psi_q(\overline{x})$ is constructed as a number with k "digits", where the digits are the values of the literals $\pi_A(L_i(\overline{a}))$ and $\pi_B(L_i(\overline{b}))$, which are bounded by q. Hence, $\pi_A[\![\psi_q(\sigma(\overline{a}))]\!]$ and $\pi_B[\![\psi_q(\sigma(\overline{b}))]\!]$ are less than $c \coloneqq q^k$, which only depends on q, n and τ. By Lemma 26, there is a sufficiently large e so that $\sum_{\sigma \in S_n} \pi_A[\![\psi_q(\sigma(\overline{a}))]\!]^e = \sum_{\sigma \in S_n} \pi_B[\![\psi_q(\sigma(\overline{b}))]\!]^e$ implies that both sums share the same summands. In particular, there are permutations $\sigma_A, \sigma_B \in S_n$ such that $\pi_A[\![\psi_c(\sigma_A(\overline{a}))]\!] = \pi_B[\![\psi_c(\sigma_B(\overline{b}))]\!]$.

Thanks to the construction of $\psi_c(\overline{x})$, this yields $\pi_A(L_i(\sigma_A(\overline{a})) = \pi_B(L_i(\sigma_B(\overline{b}))$ for all literals L_i of $\mathrm{Lit}_n(\tau)$. Thus, $\sigma_B \circ (\overline{a} \mapsto \overline{b}) \circ \sigma_A^{-1}$ is an isomorphism from π_A to π_B. ◀

We can further use the q-characteristic sentences also for $\mathbb{N}[X]$-interpretations instead of $\mathbb{N}$-interpretations. Let $X_k = \{x_1, \dots, x_k\}$, and let $\mathbb{N}[X_k](C, n)$ denote the set of polynomials $p \in \mathbb{N}[X_k]$ with coefficients less than C and exponents less than n. If we choose a suitable variable assignment $X_k \to \mathbb{N}$, we can obtain a homomorphism that assigns unique values to all polynomials in $\mathbb{N}[X_k](C, n)$.

▶ **Lemma 28.** *The variable assignment $x_i \mapsto C^{n^{i-1}}$ for $1 \leq i \leq k$ defines a homomorphism $h\colon \mathbb{N}[X_k] \to \mathbb{N}$ which induces a bijection from $\mathbb{N}[X_k](C, n)$ to $\{0, \dots, c-1\} \subseteq \mathbb{N}$ where $c \coloneqq C^{n^k}$.*

Proof. We proceed by induction on the number of variables $k \in \mathbb{N}$. The base case $k = 0$ is trivial, since $\mathbb{N}[\varnothing] \cong \mathbb{N}$ and the empty assignment induces the corresponding isomorphism. For $k > 0$, notice that $\mathbb{N}[X_k] \cong \mathbb{N}[X_{k-1}][x_k]$. Hence, each $p \in \mathbb{N}[X_k](C, n)$ may be written as

$$p = \sum_{i=0}^{n-1} q_i x_k^i, \quad \text{where } q_i \in \mathbb{N}[X_{k-1}](C, n).$$

Thus, applying the induced homomorphism h yields

$$h(p) = \sum_{i=0}^{n-1} h(q_i) h(x_k)^i = \sum_{i=0}^{n-1} h'(q_i)(C^{n^{k-1}})^i,$$

where $h'\colon \mathbb{N}[X_{k-1}] \to \mathbb{N}$ is induced by $x_i \mapsto C^{n^{i-1}}$ for $1 \leq i < k$. By induction hypothesis the restriction $h'|_{\mathbb{N}[X_{k-1}](C,n)}$ is a bijection from $\mathbb{N}[X_{k-1}](C, n)$ to $\{0, \dots, C^{n^{k-1}} - 1\}$. Clearly, $h(p)$ may be seen as a number with n digits $h'(q_i) \in \{0, \dots, C^{n^{k-1}} - 1\}$ for $0 \leq i < n$ in the number system with radix $C^{n^{k-1}}$. Thus, any number in $\{0, \dots, C^{n^k} - 1\}$ can be uniquely represented as $h(p)$ for $p \in \mathbb{N}[X_k](C, n)$, which completes the proof. ◀

▶ **Corollary 29.** *For finite $\mathbb{N}[X]$-interpretations π_A and π_B whose values are contained in $\mathbb{N}[X](C, n)$, $\pi_A[\![\chi_{\pi_A,c}]\!] = \pi_B[\![\chi_{\pi_A,c}]\!]$ implies $\pi_A \cong \pi_B$ with $c \coloneqq C^{n^{|X|}}$.*

Proof. Transform π_A and π_B to $\mathbb{N}$-interpretations $\pi'_A \coloneqq h \circ \pi_A$ and $\pi'_B \coloneqq h \circ \pi_B$ by applying the homomorphism from above. The fundamental property yields $\pi'_A[\![\chi_{\pi_A,c}]\!] = \pi'_B[\![\chi_{\pi_A,c}]\!]$. Since $h|_{\mathbb{N}[X](C,n)}$ is a bijection from $\mathbb{N}[X](C, n)$ to $\{0, \dots, c\}$, the values of π'_A and π'_B are less than c, hence we can invoke Theorem 27 to conclude $\pi'_A \cong \pi'_B$. Now, the injectivity of h on $\mathbb{N}[X](C, n)$ yields $\pi_A \cong \pi_B$. ◀

Similarly to the implications of Proposition 17 on the Viterbi semiring $\mathbb{V}$, we conclude that finite $\mathbb{N}[X]$-interpretations π_A are characterised by a the set $X_{\pi_A} \coloneqq \{\chi_{\pi_A,c} \mid c \in \mathbb{N}, c > 1\}$ of characteristic sentences. The obvious consequence is that no counterexamples exist on $\mathbb{N}[X]$.

▶ **Theorem 30.** *For finite $\mathbb{N}[X]$-interpretations π_A and π_B, $\pi_A \equiv \pi_B$ implies $\pi_A \cong \pi_B$.*

These results highlight the importance of cancellation for the construction of characteristic sentences. $\mathbb{V}$ and $\mathbb{N}[X]$ allow multiplicative cancellation, and $\mathbb{N}[X]$ even allows additive cancellation. We shall use this observation in the search for counterexamples on semirings that do *not* admit cancellation.

6 Cancellation

There are indeed counterexamples for a large class of semirings that break cancellation, including the polynomial semirings $\mathbb{B}[X]$ and $\mathbb{S}[X]$.

▶ **Definition 31.** *Let K be an idempotent semiring. A witness that K breaks cancellation is a triple $a, b, c \in K \setminus \{0\}$ such that*
(1) $a + b = a + c = a$ *and*
(2) $ab = ac$, *but* $b \neq c$.[1]

For any such triple, we define the following two non-isomorphic K-interpretations.

π_b :

A	R	$\neg R$
d	a	0
e	b	0

π_c :

A	R	$\neg R$
d	a	0
e	c	0

▶ **Lemma 32.** *The K-interpretations π_b and π_c are elementarily equivalent.*

Proof. Consider the $\mathbb{B}[X]$ interpretation

π :

A	R	$\neg R$
d	x	0
e	y	0

Let $h_b, h_c \colon \mathbb{B}[X] \to K$ be the unique homomorphisms induced by $x \mapsto a, y \mapsto b$ and $x \mapsto a, y \mapsto c$ respectively. Obviously, $\pi_b = h_b \circ \pi$ and $\pi_c = h_c \circ \pi$, hence, for each sentence $\psi \in \mathrm{FO}(\{R\})$, the fundamental property yields $\pi_b[\![\psi]\!] = h_b(\pi[\![\psi]\!])$ and $\pi_c[\![\psi]\!] = h_c(\pi[\![\psi]\!])$. In fact, if we set $p \coloneqq \pi[\![\psi]\!]$, we have $\pi_b[\![\psi]\!] = h_b(p)$ and $\pi_c[\![\psi]\!] = h_c(p)$, hence both interpretations evaluate the same polynomial p under their own homomorphism.

It remains to show that $h_b(p) = h_c(p)$. The automorphism $\overline{h}$ of $\mathbb{B}[X]$ induced by swapping the variables x and y yields the $\mathbb{B}[X]$-interpretation

$\overline{h} \circ \pi = \overline{\pi}$:

A	R	$\neg R$
d	y	0
e	x	0

Clearly, $\pi \cong \overline{\pi}$ by swapping d and e, hence $p = \pi[\![\psi]\!] = \overline{\pi}[\![\psi]\!] = \overline{h}(\pi[\![\psi]\!]) = \overline{h}(p)$. In other words, p is invariant under swapping variables, so for each pair i, j we have that $x^i y^j \in p$ if, and only if, $x^j y^i \in p$. In particular, $x^i \in p \Leftrightarrow y^i \in p$ $(*)$.

Since p is finite and all exponents are less than some $d \in \mathbb{N}$, we may write p as

$$p = \sum_{i,j<d, x^i y^j \in p} x^i y^j = \sum_{i<d, x^i \in p} x^i + \sum_{j<d, y^j \in p} y^j + \sum_{0<i,j<d, x^i y^j \in p} x^i y^j.$$

Set $I \coloneqq \{i < d \mid x^i \in p\} \overset{*}{=} \{j < d \mid y^j \in p\}$ and $M \coloneqq \{x^i y^j \mid 0 < i, j < d, x^i y^j \in p\}$. Then,

$$p = \sum_{i \in I} (x^i + y^i) + \sum_{m \in M} m.$$

For each $m \in M$, $h_b(m) = a^i b^j = a^i c^j = h_c(m)$ due to condition (2) and $i > 0$. More precisely, if $i > 0$, we can invoke (2) inductively to transform ab^j into ac^j due to commutativity of multiplication. We now invoke condition (1) for each $i \in I$. For $z \in \{b, c\}$ and using idempotence of K (i), this yields

[1] Strictly speaking, condition (2) suffices for breaking cancellation. Condition (1) imposes a further restriction on the witness, which is needed for the proof of Lemma 32.

$$a^i + z^i \overset{(1)}{=} (a+z)^i + z^i = \sum_{j=0}^{i} a^{i-j} z^j + z^i = \sum_{j=0}^{i-1} a^{i-j} z^j + z^i + z^i$$
$$\overset{(i)}{=} \sum_{j=0}^{i-1} a^{i-j} z^j + z^i = a^i.$$

Hence, $h_b(x^i + y^i) = a^i + b^i = a^i = a^i + c^i = h_c(x^i + y^i)$ for each $i \in I$ holds as well. Together, we have

$$h_b(p) = \sum_{i \in I} h_b(x^i + y^i) + \sum_{m \in M} h_b(m) = \sum_{i \in I} h_c(x^i + y^i) + \sum_{m \in M} h_c(m) = h_c(p),$$

which completes the proof, since ψ was arbitrary and $\pi_b[\![\psi]\!] = h_b(p) = h_c(p) = \pi_c[\![\psi]\!]$. ◀

Since $\pi_b \not\cong \pi_c$ clearly holds for π_b and π_c from Definition 31, we can apply Lemma 32 to construct counterexamples in many idempotent semirings, such as $\mathbb{B}[X]$ itself and $\mathbb{S}[X]$. The only requirement for this approach is the existence of an appropriate witness (a, b, c).

▶ **Theorem 33.** *For $X \supseteq \{x, y\}$, there exists a pair of elementarily equivalent, but non-isomorphic K-interpretations in the shape of π_b and π_c for both $K = \mathbb{B}[X]$ and $K = \mathbb{S}[X]$.*

Proof. For $\mathbb{B}[X]$, choose $a \coloneqq x + y + x^2 + xy + y^2$, $b \coloneqq x^2 + y^2$ and $c \coloneqq x^2 + xy + y^2$ to obtain the following pair of $\mathbb{B}[X]$-interpretations.

π_b :

A	R	$\neg R$
d	$x + y + x^2 + xy + y^2$	0
e	$x^2 + y^2$	0

π_c :

A	R	$\neg R$
d	$x + y + x^2 + xy + y^2$	0
e	$x^2 + xy + y^2$	0

To prove the desired properties, we only have to check conditions (1) and (2) from Definition 31 and then invoke Lemma 32. Condition (1) is obvious, and for (2), it suffices to expand the products ab and ac to calculate that

$$ab = ac = x^3 + xy^2 + x^2y + y^3 + x^4 + x^3y + x^2y^2 + xy^3 + y^4.$$

In $\mathbb{S}[X]$, we may use the "absorptive versions" $a \coloneqq x + y$, $b \coloneqq x^2 + y^2$ and $c \coloneqq x^2 + xy + y^2$ and apply the same argument. ◀

Finally, we will apply Lemma 32 to the Łukasiewicz semiring $\mathbb{L} = ([0,1]_{\mathbb{R}}, \max, \star, 0, 1)$, which is used in many-valued logic. A witness that $\mathbb{L}$ breaks cancellation is easy to obtain due to the multiplication $\star$ being defined as $a \star b \coloneqq \max\{0, a + b - 1\}$, which "cuts off" at 0.

▶ **Theorem 34.** *There exists a pair of elementarily equivalent, but non-isomorphic $\mathbb{L}$-interpretations.*

Proof. $\mathbb{L}$ is idempotent and the triple $a \coloneqq 1/2, b \coloneqq 1/3, c \coloneqq 1/4 \in \mathbb{L}$ satisfies the conditions (1) and (2) from Definition 31. Together with Lemma 32, we conclude that the $\mathbb{L}$-interpretations π_b and π_c, as constructed in the definition from the triple (a, b, c), are elementarily equivalent, but non-isomorphic. ◀

7 Conclusion and Outlook

Our analysis of first-order axiomatisations and elementary equivalence of finite semiring interpretations has revealed some remarkable differences between semiring semantics and classical Boolean semantics. Depending on the underlying semiring, there may exist finite semiring interpretations that are elementarily equivalent without being isomorphic. Indeed, this phenomenon happens already in very simple cases such as for min-max semirings with three elements. On the other side, there are relevant semirings, used for instance in provenance analysis in databases such as the tropical semiring or the Viterbi semiring, where every finite interpretation is first-order axiomatisable, and in fact even by a finite set of axioms. However, and this is again an interesting difference to Boolean semantics, a finite axiomatisation does not imply an axiomatisation by a single axiom.

Also for the semirings of polynomials, fundamental for a general provenance analysis that reveals which combinations of atomic facts are responsible for the truth of a logical statement, the picture is not unique. While the most general semiring $\mathbb{N}[X]$, freely generated by X, admits axiomatisations of all finite interpretations, so that elementary equivalence implies isomorphism, this is not the case for the semirings $\mathsf{PosBool}[X]$, $\mathbb{S}[X]$, $\mathbb{B}[X]$ and $\mathbb{W}[X]$ which are universal for important subclasses of semirings.

In the study of elementary equivalence for semiring semantics, it turns out that there is no *straightforward* adaptation of Ehrenfeucht–Fraïssé games, or their generalisations such as Hellas bijective pebble game [16], to semiring interpretations. Whatever the specific protocol of possible moves in such games may be, they always result in a *localisation*, in the sense that some tuples in the two structures are picked that are indistinguishable on the atomic level. As shown by the very simple example of the interpretations π_{PQ} and π_{QP} at the beginning of Sect. 4, this is not possible in semiring semantics. Although the two interpretations are elementarily equivalent, no element of the first "looks the same" as any element of the second, so any kind of localisation would result in a winning play of the Spoiler. It is an intriguing open question how elementary equivalence of semiring interpretations can be captured by a different notion of comparison games or back-and-forth systems à la Fraïssé. This not being available (yet), we have established elementary equivalence by different methods, based on homomorphisms, which we believe to be of independent interest.

There are many other model-theoretic issues that deserve to be studied in semiring semantics. While we have limited ourselves here to *finite* semiring interpretations, the study of semiring semantics over infinite universes is of course very interesting as well. It requires certain restrictions on the underlying semirings, concerning existence and appropriate algebraic properties of infinite sums and products, but there are useful semirings satisfying such properties. A particularly interesting question is what kind of compactness and preservation results are possible in such contexts.

We finally remark that an altogether different approach to semiring interpretations would consider them as two-sorted structures, one sort being a finite or infinite structure (or just a set), the second one consisting of the semiring, with functions from the first to the second sort giving the semiring interpretation of the literals. This is very similar to the approach of metafinite model theory [11], and to get a reasonable logical theory it is important that the elements of the second sort, here the semiring, are treated differently than the elements of the first sort. In particular, quantifiers should range only over the first sort, and operations on the second sort are just the algebraic semiring operations and their aggregates, together with equalities between terms. Such an approach is certainly useful for a number of questions and permits the study of semiring interpretations via classical Boolean semantics. In particular,

once semiring values are directly accessible in the logic, the construction of characteristic sentences, axiomatising a finite structure up to isomorphism, can easily be translated into such a setting. However, such an internalisation of the semiring values, from the meta-level of truth values into the structures under consideration, does not really capture the essence of semiring semantics.

References

1 C. Bourgaux, A. Ozaki, R. Peñaloza, and L. Predoiu. Provenance for the description logic ELHr. In *Proceedings of IJCAI 2020*, pages 1862–1869, 2020. doi:10.24963/ijcai.2020/258.

2 K. Dannert and E. Grädel. Provenance analysis: A perspective for description logics? In C. Lutz et al., editor, *Description Logic, Theory Combination, and All That*, Lecture Notes in Computer Science Nr. 11560. Springer, 2019. doi:10.1007/978-3-030-22102-7_12.

3 K. Dannert and E. Grädel. Semiring provenance for guarded logics. In *Hajnal Andréka and István Németi on Unity of Science: From Computing to Relativity Theory through Algebraic Logic*, Outstanding Contributions to Logic. Springer, 2020.

4 K. Dannert, E. Grädel, M. Naaf, and V. Tannen. Semiring provenance for fixed-point logic. In *Proceedings of CSL 2021*, 2021.

5 D. Deutch, T. Milo, S. Roy, and V. Tannen. Circuits for datalog provenance. In *Proc. 17th International Conference on Database Theory ICDT*, pages 201–212, 2014.

6 J. Doleschal, B. Kimelfeld, W. Martens, and L. Peterfreund. Weight annotation in information extraction. In *International Conference on Database Theory, ICDT 2020*, 2020. doi:10.4230/LIPIcs.ICDT.2020.8.

7 M. Droste and P. Gastin. Weighted automata and weighted logics. In *International Colloquium on Automata, Languages, and Programming*, pages 513–525. Springer, 2005.

8 M. Droste and W. Kuich. Semirings and formal power series. In *Handbook of Weighted Automata*, pages 3–28. Springer, 2009.

9 F. Geerts and A. Poggi. On database query languages for K-relations. *Journal of Applied Logic*, 8(2):173–185, 2010.

10 J. Goodman. Semiring parsing. *Computational Linguistics*, 25:573–605, 1999.

11 E. Grädel and Y. Gurevich. Metafinite model theory. *Information and Computation*, 140:26–81, 1998.

12 E. Grädel and V. Tannen. Semiring provenance for first-order model checking. arXiv:1712.01980 [cs.LO], 2017. arXiv:1712.01980.

13 E. Grädel and V. Tannen. Provenance analysis for logic and games. *Moscow Journal of Combinatorics and Number Theory*, 9(3):203–228, 2020. doi:10.2140/moscow.2020.9.203.

14 T. Green, G. Karvounarakis, and V. Tannen. Provenance semirings. In *Principles of Database Systems PODS*, pages 31–40, 2007.

15 T. Green and V. Tannen. The semiring framework for database provenance. In *Proceedings of PODS*, pages 93–99, 2017.

16 L. Hella. Logical hierarchies in PTIME. In *Proceedings of LICS 92*, pages 360–368, 1992.

17 M. Naaf. *Semirings for Provenance Analysis of Fixed Point Logics and Games*. Msc thesis, RWTH Aachen University, 2019.

18 A. Ozaki and R. Peñaloza. Provenance in ontology-based data access. In *Description Logics 2018*, volume 2211 of *CEUR Workshop Proceedings*, 2018.

19 M. Raghothaman, J. Mendelson, D. Zhao, M. Naik, and B. Scholz. Provenance-guided synthesis of datalog programs. *Proc. ACM Program. Lang.*, 4:62:1–62:27, 2020. doi:10.1145/3371130.

20 P. Senellart. Provenance and probabilities in relational databases: From theory to practice. *SIGMOD Record*, 46(4):5–15, 2017.

Logarithmic Weisfeiler-Leman Identifies All Planar Graphs

Martin Grohe
RWTH Aachen University, Germany

Sandra Kiefer
University of Warsaw, Poland
RWTH Aachen University, Germany

Abstract

The Weisfeiler-Leman (WL) algorithm is a well-known combinatorial procedure for detecting symmetries in graphs and it is widely used in graph-isomorphism tests. It proceeds by iteratively refining a colouring of vertex tuples. The number of iterations needed to obtain the final output is crucial for the parallelisability of the algorithm.

We show that there is a constant k such that every planar graph can be identified (that is, distinguished from every non-isomorphic graph) by the k-dimensional WL algorithm within a logarithmic number of iterations. This generalises a result due to Verbitsky (STACS 2007), who proved the same for 3-connected planar graphs.

The number of iterations needed by the k-dimensional WL algorithm to identify a graph corresponds to the quantifier depth of a sentence that defines the graph in the $(k+1)$-variable fragment C^{k+1} of first-order logic with counting quantifiers. Thus, our result implies that every planar graph is definable with a C^{k+1}-sentence of logarithmic quantifier depth.

2012 ACM Subject Classification Theory of computation → Finite Model Theory; Mathematics of computing → Graph theory

Keywords and phrases Weisfeiler-Leman algorithm, finite-variable logic, isomorphism testing, planar graphs, quantifier depth, iteration number

Digital Object Identifier 10.4230/LIPIcs.ICALP.2021.134

Category Track B: Automata, Logic, Semantics, and Theory of Programming

Related Version `https://arxiv.org/abs/2106.16218`

Funding *Sandra Kiefer*: supported by the European Research Council under the European Union's Horizon 2020 research and innovation programme (ERC consolidator grant LIPA, agreement no. 683080).

1 Introduction

The Weisfeiler-Leman (WL) algorithm is a combinatorial procedure for detecting symmetries in graphs. It is widely used in approaches to tackle the graph-isomorphism problem, both from a theoretical ([4, 5, 24]) and from a practical perspective ([7, 23, 31, 32]). The algorithm is derived from a technique called *naïve vertex classification* (or *Colour Refinement*), which may be viewed as the 1-dimensional version WL^1 of the WL algorithm. For every $k \geq 1$, the k-dimensional WL algorithm (WL^k) iteratively colours k-tuples of vertices of a graph by propagating local information until it reaches a *stable colouring*. Weisfeiler and Leman introduced the 2-dimensional version WL^2, today known as the *classical WL algorithm*, in [37]. The algorithm WL^k can be implemented to run in time $O(n^{k+1} \log n)$ on graphs of order n [22].

The algorithm has striking connections to numerous areas of mathematics and computer science, which surely is a reason why research on it has been active since its introduction over half a century ago. For example, there are tight connections to linear and semidefin-

ite programming [2, 3, 20], homomorphism counting [8, 10], and the algebra of coherent configurations [6]. Most recently, the WL algorithm has been applied in several interesting machine-learning contexts [1, 16, 33, 34, 39].

A very strong and highly exploited link between the algorithm and logic was established by Immerman and Lander [22] and Cai, Fürer, and Immerman [5]: WL^k assigns the same colour to two k-tuples of vertices if and only if these tuples satisfy the same formulas of the $(k+1)$-variable fragment C^{k+1} of first-order logic with counting quantifiers. Cai, Fürer, and Immerman [5] used this correspondence and an Ehrenfeucht-Fraïssé game that characterises equivalence for the logic C^{k+1} to prove that, for every k, there are non-isomorphic graphs of order $O(k)$ that are not distinguished by WL^k. Here we say that WL^k *distinguishes* two graphs if WL^k computes different stable colourings on them, that is, there is some colour such that the numbers of k-tuples of that colour differ in the two graphs.

We say that WL^k *identifies* a graph G if it distinguishes G from all graphs G' that are not isomorphic to G. It has been shown that for suitable constants k, the algorithm WL^k identifies all planar graphs [13], all graphs of bounded tree width [18], and all graphs in many other natural graph classes [12, 14, 15, 17, 19]. For some of these classes, fairly tight bounds for the optimal value of k, called the *Weisfeiler-Leman (WL) dimension*, are known. Notably, interval graphs have WL dimension 2 [12], graphs of tree width k have WL dimension in the range $\lceil k/2 \rceil - 3$ to k [26], and, most relevant for us, planar graphs have WL dimension 2 or 3 [27].

Another parameter of the WL algorithm that has received recent attention is the number of iterations it needs to reach its final, stable colouring. Since a set of size n^k can only be partitioned $n^k - 1$ times, a natural upper bound on the number of iterations to reach the final output is $n^k - 1$ (n always denotes the number of vertices of the input graph). This bound cannot be improved for WL^1, since there are infinitely many graphs on which the algorithm takes $n - 1$ iterations to compute its final output [25]. However, for WL^2, it was shown that the bound $\Theta(n^2)$ is asymptotically not tight [28]. Currently, the best upper bound on the iteration number for WL^2 is $O(n \log n)$ [30].

The number of iterations of WL^k is crucial for the parallelisability of the algorithm: for $\ell \geq \log n$, it holds that ℓ iterations of WL^k can be simulated in $O(\ell)$ steps on a PRAM with $O(n^k)$ processors [21, 29]. In particular, if for a class $\mathcal{C}$ of graphs, all $G, G' \in \mathcal{C}$ (of order n) can be distinguished by WL^k in $O(\log n)$ iterations, then the isomorphism problem for graphs in $\mathcal{C}$ is in the complexity class AC^1. Grohe and Verbitsky [21] proved that this is the case for all classes of graphs of bounded tree width and all maps (graphs embedded into a surface together with a rotation system specifying the embedding), and Verbitsky [36] proved it for the class of 3-connected planar graphs.

Our results

We say that WL^k distinguishes two graphs *in ℓ iterations* if the colouring obtained by WL^k in the ℓ-th iteration differs among the two graphs, and we say WL^k identifies a graph *in ℓ iterations* if it distinguishes the graph from every non-isomorphic graph in ℓ iterations.

▶ **Theorem 1.** *There is a constant k such that WL^k identifies every n-vertex planar graph in $O(\log n)$ iterations.*

The correspondence between WL^k and the logic C^{k+1} can be refined to a correspondence between the number of iterations and the quantifier depth: WL^k assigns the same colour to two k-tuples of vertices in the ℓ-th iteration if and only if these two k-tuples satisfy the same C^{k+1}-formulas of quantifier depth ℓ. Thus, the following theorem is equivalent to Theorem 1.

▶ **Theorem 2.** *There is a constant k such that for every n-vertex planar graph G, there is a C^k-sentence of quantifier depth $O(\log n)$ that identifies G (that is, characterises G up to isomorphism).*

We exploit the logical characterisation of the WL algorithm in our proof, so it is actually Theorem 2 that we prove. We first show that every planar graph G has a tree decomposition of logarithmic height where each bag consists of at most four 3-connected components of G and the adhesion is at most 6. Then we inductively construct a formula to identify G by ascending through the tree, encoding all information about isomorphism types of the parsed subgraphs in subformulas. At each node of the tree, we use Verbitsky's result to deal with the 3-connected components.

2 Preliminaries

All graphs in this paper are finite, simple, and undirected. For a graph G, we denote by $V(G)$ and $E(G)$ its set of vertices and edges, respectively. The *order* of G is $|G| \coloneqq |V(G)|$. We write edges without parenthesis, as in vw. For $v \in V(G)$, we let $N_G(v) \coloneqq \{w \mid vw \in E(G)\}$.

A *subgraph* of G is a graph H with $V(H) \subseteq V(G)$ and $E(H) \subseteq E(G)$. We set $N_G(H) \coloneqq \bigcup_{v \in V(H)} N_G(v) \setminus V(H)$. We call a graph H a *topological subgraph* of G if a subdivision of H (i.e., a graph obtained from H by replacing some edges with paths) is a subgraph of G. For $W \subseteq V(G)$, we let $G[W] \coloneqq (W, E(G) \cap \{uv \mid u, v \in W\})$ and, for arbitrary sets W, we let $G \setminus W \coloneqq G[V(G) \setminus W]$.

A graph G is *k-connected* if $|G| > k$ and there is no set $S \subseteq V(G)$ with $|S| \leq k-1$ such that $G \setminus S$ is disconnected.

2.1 Logic

We denote by C the extension of first-order logic FO by *counting quantifiers* $\exists^{\geq m} x$ with the obvious meaning. C is only a syntactical extension of FO, because $\exists^{\geq m} x \varphi(x)$ is equivalent to $\exists x_1 \dots \exists x_m \Big(\bigwedge_{i \neq j} x_i \neq x_j \wedge \bigwedge_i \varphi(x_i) \Big)$. However, we are mainly interested in the fragments C^k of C consisting of all formulae with at most k variables (which can, however, be reused within the formula). If $m > k$, then $\exists^{\geq m} x$ cannot be expressed in the k-variable fragment of FO, this is why we add the counting quantifiers.

We write $\varphi(x_1, \dots, x_\ell)$ to indicate that the free variables of φ are among $x_1, \dots, x_\ell$. Then for a graph G and vertices $u_1, \dots, u_\ell \in V(G)$, we write $G \models \varphi(u_1, \dots, u_\ell)$ to denote that G satisfies φ if, for all i, the variable x_i is interpreted by u_i. Moreover, we write $\varphi[G, u_1, \dots, u_i, x_{i+1}, \dots, x_\ell]$ to denote the set of all $(\ell - i)$-tuples $(u_{i+1}, \dots, u_\ell)$ such that $G \models \varphi(u_1, \dots, u_\ell)$.

The *quantifier depth* $\operatorname{qd}(\varphi)$ of a formula $\varphi \in \mathsf{C}$ is its depth of quantifier nesting. More formally,

- if φ is atomic, then $\operatorname{qd}(\varphi) = 0$.
- $\operatorname{qd}(\neg\varphi) = \operatorname{qd}(\varphi)$.
- $\operatorname{qd}(\varphi_1 \vee \varphi_2) = \operatorname{qd}(\varphi_1 \wedge \varphi_2) = \max\{\operatorname{qd}(\varphi_1), \operatorname{qd}(\varphi_2)\}$.
- $\operatorname{qd}(\exists^{\geq p} x \varphi) = \operatorname{qd}(\varphi) + 1$.

We denote the set of all C^k-formulas of quantifier depth at most ℓ by C^k_ℓ.

It will often be convenient to use asymptotic notation, such as $\mathsf{C}^{O(1)}_{O(\log n)}$. The parameter n always refers to the order of the input graph, and we will typically make assertions such as: *For every n, there exists a $\mathsf{C}^{O(1)}_{O(\log n)}$-formula $\varphi^{(n)}(x)$ such that for all graphs G of order $|G| = n$ and all $v \in V(G)$, [something holds].* What this means is that *there is a constant k and a function $\ell(n) \in O(\log n)$ such that for every n, there exists a $\mathsf{C}^k_{\ell(n)}$-formula $\varphi^{(n)}(x)$ such that for all graphs G of order $|G| = n$ and all $v \in V(G)$, [something holds].*

Throughout this paper, we will have to express properties of graphs and their vertices using $\mathsf{C}^{O(1)}_{O(\log n)}$-formulas. The basic building blocks that we use are connectivity statements with formulas of logarithmic quantifier depth, as illustrated in the following example.

▶ **Example 3.** For every $k \geq 0$, we define a $\mathsf{C}^3_{\lceil \log n \rceil}$-formula $\mathsf{dist}_{\leq k}$ such that for every graph G of order at most n and all vertices $u, u' \in V(G)$, it holds that $G \models \mathsf{dist}_{\leq k}(u, u')$ if and only if u and u' have distance at most k in G. We let

$$\mathsf{dist}'_{\leq k}(x, x') \coloneqq \begin{cases} x = x' & \text{if } k = 0 \\ E(x, x') \vee x = x' & \text{if } k = 1 \\ \exists y_k \big(\mathsf{dist}'_{\leq \lfloor \frac{k}{2} \rfloor}(x, y_k) \wedge \mathsf{dist}'_{\leq \lceil \frac{k}{2} \rceil}(y_k, x') \big) & \text{otherwise.} \end{cases}$$

Thus, for $k \leq n$, the quantifier depth of $\mathsf{dist}'_{\leq k}$ is bounded by $\lceil \log n \rceil$. Now, it suffices to note that we can actually get by with the three variables x, x', y_k by reusing them in the subformulas that are defined inductively. We hence obtain the desired $\mathsf{C}^3_{\lceil \log n \rceil}$-formula $\mathsf{dist}_{\leq k}$. Note that, for $k \geq 1$, the $\mathsf{C}^3_{\lceil \log n \rceil}$-formula $\mathsf{dist}_{=k}(x, x') \coloneqq \mathsf{dist}_{\leq k}(x, x') \wedge \neg \mathsf{dist}_{\leq k-1}(x, x')$ states that x and x' have distance exactly k. Moreover, in every graph of order at most n, the $\mathsf{C}^3_{\lceil \log n \rceil}$-formula $\mathsf{comp}(x, x') \coloneqq \mathsf{dist}_{\leq n-1}(x, x')$ states that x and x' lie in the same connected component and the $\mathsf{C}^3_{\lceil \log n \rceil}$-sentence $\mathsf{conn}_n \coloneqq \forall x \forall x' \mathsf{dist}_{\leq n-1}(x, x')$ states that the graph is connected. ⌟

2.2 The WL Algorithm

We briefly review the WL algorithm. For details, we refer to the recent survey [24].

Let $k \geq 1$. The *atomic type* $\mathrm{atp}(G, \bar{u})$ of a k-tuple $\bar{u} = (u_1, \ldots, u_k)$ of vertices of a graph G is the set of all atomic facts satisfied by these vertices, that is, all adjacencies and equalities between the vertices. Hence, tuples $\bar{u} = (u_1, \ldots, u_k)$ and $\bar{v} = (v_1, \ldots, v_k)$ of vertices of graphs G, H, respectively, have the same atomic type if and only if the mapping $u_i \mapsto v_i$ is an isomorphism from the graph $G[\{u_1, \ldots, u_k\}]$ to $H[\{v_1, \ldots, v_k\}]$.

The algorithm WL^k (the *k-dimensional Weisfeiler-Leman algorithm*) takes a graph G as input and computes the following sequence of *colourings* wl^k_i of $V(G)^k$ for $i \geq 0$, until it returns $\mathsf{wl}^k_\infty \coloneqq \mathsf{wl}^k_i$ for the smallest i such that, for all $\bar{u}, \bar{v}$, it holds that $\mathsf{wl}^k_i(\bar{u}) = \mathsf{wl}^k_i(\bar{v}) \iff \mathsf{wl}^k_{i+1}(\bar{u}) = \mathsf{wl}^k_{i+1}(\bar{v})$. Set $\mathsf{wl}^k_0(\bar{u}) \coloneqq \mathrm{atp}(G, \bar{u})$. In the $(i+1)$-st *iteration*, the colouring wl^k_{i+1} is defined by $\mathsf{wl}^k_{i+1}(\bar{u}) \coloneqq \big(\mathsf{wl}^k_i(\bar{u}), M_i(\bar{u})\big)$, where, for $\bar{u} = (u_1, \ldots, u_k)$, we let $M_i(\bar{u})$ be the multiset

$$\Big\{\!\!\Big\{ \big(\mathrm{atp}(G, (u_1, \ldots, u_k, v)), \mathsf{wl}^k_i(u_1, \ldots, u_{k-1}, v), \\ \mathsf{wl}^k_i(u_1, \ldots, u_{k-2}, v, u_k), \ldots, \mathsf{wl}^k_i(v, u_2, \ldots, u_k) \big) \mid v \in V \Big\}\!\!\Big\}$$

The algorithm WL^k *distinguishes* two graphs G, H *in ℓ iterations* if there is a colour c in the range of wl^k_ℓ such that the number of tuples $\bar{u} \in V(G)^k$ with $\mathsf{wl}^k_\ell(\bar{u}) = c$ is different from the number of tuples $\bar{v} \in V(H)^k$ with $\mathsf{wl}^k_\ell(\bar{v}) = c$. In this case, we say WL^k_ℓ *distinguishes G and H*. Moreover, WL^k_ℓ *identifies* G if it distinguishes G from all graphs H that are not isomorphic to G.

▶ **Theorem 4** ([5, 22])**.** *Let $k \in \mathbb{N}$. Let G and H be graphs with $|G| = |H|$ and let $\bar{u} \coloneqq (u_1, \ldots, u_k) \in V(G)^k$ and $\bar{v} \coloneqq (v_1, \ldots, v_k) \in V(H)^k$. Then, for all $i \in \mathbb{N}$, the following are equivalent.*

1. $\mathsf{wl}^k_i(\bar{u}) = \mathsf{wl}^k_i(\bar{v})$.
2. $G \models \varphi(u_1, \ldots, u_k) \iff H \models \varphi(v_1, \ldots, v_k)$ *holds for every C^{k+1}_i-formula $\varphi(x_1, \ldots, x_k)$.*

3 3-Connected Planar Graphs

Verbitsky [36] proved that $\mathsf{WL}^{O(1)}_{O(\log n)}$ distinguishes any two 3-connected planar graphs. Before we discuss the specific version of this result that we need here, let us briefly review some background on planar graphs. Intuitively, a *plane graph* is a graph drawn into the plane with no edges crossing. A *planar graph* is an abstract graph G isomorphic to a plane graph; an isomorphism from G to a plane graph is a *planar embedding* of G. Now suppose G is a plane graph. If we cut the plane along all edges of the graph, the pieces that remain are the *faces* of G (note that one of these faces is unbounded). The closed walk along the vertices and edges in the boundary of a face is the *facial walk* associated with this face. If G is 2-connected, then every facial walk is a cycle. If G is 3-connected, we can describe the facial cycles combinatorially: a cycle C is a facial cycle of G if and only if C is an induced subgraph of G and $G \setminus V(C)$ is connected. (This is the statement of *Whitney's Theorem* [38].) This implies that all planar embeddings of a 3-connected planar graph have the same facial cycles, which can be interpreted as saying that, combinatorially, all planar embeddings of the graph are the same. Another way of describing a planar embedding combinatorially is by specifying, for each vertex, the cyclic order in which the edges incident to this vertex appear. This is what is known as a *rotation system*. It is easy to see that a rotation system determines all facial walks, and, conversely, the facial walks determine the rotation system. One last fact that we need to know about plane graphs is *Euler's formula*: if G is a connected plane graph with n vertices, m edges, and f faces, then $n - m + f = 2$. (For details and more background, we refer the reader to [9].)

Let us now turn to the version of Verbitsky's theorem about 3-connected planar graphs that we need here. It says that, in a 3-connected planar graph, we can find three vertices such that once these vertices are fixed, we can identify every other vertex by a $\mathsf{C}^{O(1)}_{O(\log n)}$-formula.

▶ **Theorem 5** ([36]). *Let $n \in \mathbb{N}$ and let G be a 3-connected planar graph of order $|G| \leq n$ and $v_1v_2 \in E(G)$. Then there is a $v_3 \in N_G(v_2)$ and for every $w \in V(G)$ a $\mathsf{C}^{O(1)}_{O(\log n)}$-formula $\mathsf{id}_w(x_1, x_2, x_3, y)$ such that $G \models \mathsf{id}_w(v_1, v_2, v_3, w)$ and $G \not\models \mathsf{id}_w(v_1, v_2, v_3, w')$ for all $w' \in V(G) \setminus \{w\}$.*

The key step in Verbitsky's proof is to define the rotation system underlying the unique planar embedding of a 3-connected planar graph. To state this formally, we use the terminology of [13, 15]. An *angle* of a plane graph G *at* a vertex v is a triple (w, v, w') of vertices such that vw and vw' are successive edges in a facial walk of G. Two angles (v_1, v_2, v_3) and (w_1, w_2, w_3) are *aligned* if $w_1 = v_2$ and $w_2 = v_3$ and both angles appear in the same facial walk. Observe that, if we know all angles at a vertex v, we can define the cyclic permutation of the edges incident with v induced by the embedding. If we know all angles of G and the alignment relation between them, we can define the rotation system. By Whitney's Theorem, all planar embeddings of a 3-connected planar graph G have the same angles; we call them *the angles of* G. Similarly, we can define abstractly if two angles of a 3-connected planar graph are aligned.

▶ **Lemma 6** ([36]). *There are $\mathsf{C}^{O(1)}_{O(\log n)}$-formulas $\mathsf{ang}^{(n)}(x_1, x_2, x_3)$ and $\mathsf{aln}^{(n)}(x_1, x_2, x_3, x_4)$ such that for all 3-connected planar graphs G of order $|G| = n$ and all $v_1, v_2, v_3, v_4 \in V(G)$, we have*

$$G \models \mathsf{ang}^{(n)}(v_1, v_2, v_3) \iff (v_1, v_2, v_3) \text{ is an angle of } G,$$
$$G \models \mathsf{aln}^{(n)}(v_1, \ldots, v_4) \iff (v_1, v_2, v_3), (v_2, v_3, v_4) \text{ are aligned angles of } G.$$

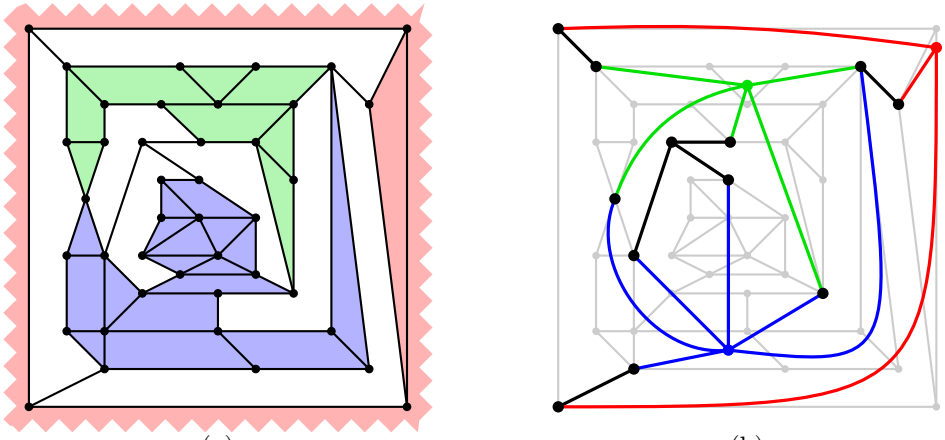


Figure 1 Defining the faces of a 3-connected planar graph: (a) shows a 3-connected planar graph G with 3 regions formed by faces with at most 6 edges in their boundary; (b) shows the derived graph $G^{(1)}$; the faces of $G^{(1)}$ are in one-to-one correspondence to the white faces of G.

This lemma is an easy consequence of the results in [36, Section 4]. The terminology there is different, the notion corresponding to (aligned) angles is that of a *layout system*. Verbitsky's proof is based on a careful (and tedious) analysis of how two paths between the neighbours of a vertex may intersect.

To give the reader some intuition about the lemma, we sketch an alternative proof, which is based on ideas from [14] (also see [15, Section 10.4]). Let G be a 3-connected planar graph, and let us think of G as being embedded in the plane. It follows from Euler's formula that in every plane graph of minimum degree 3, a constant fraction of the edges is contained in facial walks of length at most 6. Using Whitney's Theorem, we can define the set of all 6-tuples that determine a facial cycle of length at most 6 using a C^9-formula of logarithmic quantifier depth. This gives us all the angles associated with these cycles and the alignment relation on these angles. The faces corresponding to these facial cycles of size at most 6 can be partitioned into *regions*, where two faces belong to the same region if their boundaries share an edge (see Figure 1(a)).

We define a new graph $G^{(1)}$ as follows: for every region R of G, we delete all vertices contained in the interior of R, all vertices on the boundary of R that have no neighbours outside the region, and all edges that are either in the interior or on the boundary of the region. Then we add a fresh vertex v_R and edges from v_R to all vertices that remain in the boundary of the region R (see Figure 1(b)). Each face of $G^{(1)}$ corresponds to a face of G that we have not found yet. Applying Euler's formula again, we can prove that a constant fraction of the edges of G that remain edges of $G^{(1)}$ are contained in facial walks of $G^{(1)}$ that contain at most six vertices of degree ≥ 3. We can define the facial walks of the corresponding edges in G, again using Whitney's Theorem to test if a cycle is facial. Note that, for this, we do not need $G^{(1)}$ to be 3-connected (in general, it is not); we always define facial cycles in the original graph G. The new facial cycles together with those found in the first step give us new regions (covering more faces of G), and from these, we construct a graph $G^{(2)}$. Iterating the construction, we obtain a sequence of graphs $G^{(i)}$. The construction stops once we have found all facial walks of G. Since we always use a constant fraction of the edges, this happens after at most logarithmically many iterations. This completes our proof sketch of Lemma 6.

Proof of Theorem 5. Let G be a 3-connected planar graph of order $|G| = n$. For angles $\overline{v} = (v_1, v_2, v_3)$, $\overline{w} = (w_1, w_2, w_3)$, we write $\overline{v} \curvearrowright \overline{w}$ if $\overline{v}, \overline{w}$ are aligned, and we write $\overline{v} \between \overline{w}$ if $w_1 = v_3$ and $w_2 = v_2$ and $w_3 \neq v_1$. Note that, for every angle $\overline{v}$, there is a unique $\overline{w}$ such that $\overline{v} \curvearrowright \overline{w}$, because, by the 3-connectedness of G, every angle is in the boundary of a unique face, and the aligned angle belongs to the same face. There is also a unique $\overline{w}'$ such that $\overline{v} \between \overline{w}'$, determined by the cyclic order of the edges and faces around a vertex. An *angle walk* is a sequence $\overline{v}_0, \dots, \overline{v}_\ell$ of angles such that for all $i \in [\ell]$, we have $\overline{v}_{i-1} \curvearrowright \overline{v}_i$ or $\overline{v}_{i-1} \between \overline{v}_i$. The *direction* of the angle walk $\overline{v}_0, \dots, \overline{v}_\ell$ is the tuple $\overline{\delta} = (\delta_1, \dots, \delta_\ell) \in \{\curvearrowright, \between\}^\ell$ such that for every $i \in [\ell]$, we have $\overline{v}_{i-1}\delta_i\overline{v}_i$. Using Lemma 6, it is straightforward to prove that for every $\overline{\delta} \in \{\curvearrowright, \between\}^{\leq n}$, there is a $\mathsf{C}^{O(1)}_{O(\log n)}$-formula $\mathsf{awalk}^{(n)}_{\overline{\delta}}(\overline{x}, \overline{y})$ such that for all $\overline{v}, \overline{w} \in V(G)^3$, we have $G \models \mathsf{awalk}^{(n)}_{\overline{\delta}}(\overline{v}, \overline{w})$ if and only if there is an angle walk of direction $\overline{\delta}$ from $\overline{v}$ to $\overline{w}$. Now let $v_1v_2 \in E(G)$. Then there is a v_3 such that (v_1, v_2, v_3) is an angle. Let $\overline{v} := (v_1, v_2, v_3)$. Note that, for every $w \in V(G) \setminus \{v_1, v_2, v_3\}$, there is an angle walk of length at most n from $\overline{v}$ to some $\overline{w} = (w_1, w_2, w_3)$ with $w_3 = w$, simply because every path in G can be extended to an angle walk. Let $\Delta(w)$ be the set of all directions $\overline{\delta}$ of length at most n such that there is an angle walk of direction $\overline{\delta}$ from $\overline{v}$ to some $\overline{w} = (w_1, w_2, w_3)$ with $w_3 = w$. Note that the sets $\Delta(w)$ for $w \in V(G) \setminus \{v_1, v_2, v_3\}$ are mutually disjoint. Let $\mathsf{id}_{\overline{\delta}}(x_1, x_2, x_3, y) := \exists y_1 \exists y_2 \mathsf{awalk}^{(n)}_{\overline{\delta}}(x_1, x_2, x_3, y_1, y_2, y)$. Then for $\overline{\delta} \in \Delta(w)$, we have $G \models \mathsf{id}_{\overline{\delta}}(v_1, v_2, v_3, w)$ and $G \not\models \mathsf{id}_{\overline{\delta}}(v_1, v_2, v_3, w')$ for all $w' \neq w$. ◀

4 Decomposition into Blocks

Let G be a graph. A *tree decomposition* of G is a pair (T, β) where T is a tree and $\beta \colon V(T) \to 2^{V(G)}$ is a function such that for every $v \in V(G)$, the set $\{t \in V(T) \mid v \in \beta(t)\}$ is non-empty and induces a connected subgraph in T, and for every $e \in E(G)$, there is a $t \in V(T)$ such that $e \subseteq \beta(t)$. For $t \in V(T)$, we call $\beta(t)$ a *bag* of (T, β). The *adhesion* of (T, β) is $\mathrm{ad}(T, \beta) := \max\{|\beta(t) \cap \beta(u)| \mid tu \in E(T)\}$ (or 0 if $E(T) = \emptyset$). The *width* of (T, β) is $\mathrm{wd}(T, \beta) := \max_{t \in V(T)} |\beta(t)| - 1$.

We denote the root of a rooted tree T by r^T. For better readability, if the rooted tree is referred to as T^*, we set $r^* := r^{T^*}$. The *height* of T is the maximum length of a path from r^T to a leaf of T. We denote the descendant order of T by $\trianglelefteq^T$. That is, $t \trianglelefteq^T u$ if t occurs on the path from r^T to u. A *rooted tree decomposition* is a tree decomposition where the tree is rooted.

▶ **Lemma 7** (Folklore). *Let T be a tree and $\chi \colon V(T) \to \mathbb{R}_{\geq 0}$. Then there is a node $t \in V(T)$ such that for every connected component C of $T \setminus \{t\}$, it holds that*

$$\sum_{t \in V(C)} \chi(t) \leq \frac{1}{2} \sum_{t \in V(T)} \chi(t).$$

Proof. Orient all edges towards the larger sum of χ-weights in the connected components that the removal of the edge would induce, breaking ties arbitrarily. There will be a node such that all incident edges are oriented towards it. This node has the desired property. ◀

The following lemma is known in its essence (for example, [11]), though we are not aware of a reference where it is stated in this precise form, which we will need later.

▶ **Lemma 8.** *Let T be a tree, and let $B \subseteq V(T)$ be a set of size $|B| \leq 3$. Then there is a rooted tree decomposition (T^*, β^*) of T with $B \subseteq \beta^*(r^*)$ and the following additional properties:*

(i) *The height of* T^* *is at most* $2\log|T|$.
(ii) *The width of* (T^*, β^*) *is at most* 3.
(iii) *The adhesion of* (T^*, β^*) *is at most* 3.
(iv) *For every* $t^* \in V(T^*)$ *and every child* u^* *of* t^**, the graph* $T\left[\left(\bigcup_{v^* \trianglerighteq^{T^*} u^*} \beta^*(v^*)\right) \setminus \beta^*(t^*)\right]$ *is connected.*

Proof. Condition (iv) is something that we can easily achieve for every rooted tree decomposition: if, for the rooted subtree at some node, the subgraph induced by the bags in this subtree is not connected, we simply create one copy of the subtree for each connected component and only keep the vertices of that connected component in the copy. Moreover, the adhesion of a tree decomposition of width 3 can only be larger than 3 if there are adjacent nodes with the same bag. If this is the case, we can simply contract the edge between the nodes. Repeating this, we can turn the decomposition into a decomposition of adhesion at most 3. So we only need to take care of Conditions (i) and (ii).

The proof is by induction on $n := |T|$. We prove a slightly stronger statement; in addition to $B \subseteq \beta^*(r^*)$, we require $|\beta^*(r^*) \setminus B| \leq 1$.

The base case $n \leq 4$ is easy: for $n = 1$, the 1-node tree decomposition of height 0 has all the desired properties, and for $2 \leq n \leq 4$, we can take a 2-node tree decomposition of height 1 where the root bag is B and the leaf bag is $V(T)$.

For the inductive step, suppose $n > 4$.

Case 1: $|B| < 3$.

By Lemma 7, there is a node $b \in V(T)$ such that for every connected component C of $T \setminus \{b\}$, it holds that $|V(C)| \leq \frac{n}{2}$.

Let $C_1, \ldots, C_m$ be the vertex sets of the connected components of $T \setminus \{b\}$. For every $i \in [m]$, let c_i be the unique neighbour of b in C_i, and let $B_i := (B \cap V(C_i)) \cup \{c_i\}$. Note that $|B_i| \leq 3$.

By the induction hypotheses, for every i, there is a rooted tree decomposition (T_i, β_i) of C_i with the desired properties. In particular, the height of T_i is at most $2\log(n/2) = 2\log n - 2$.

For every i, let r_i be the root of T_i. We form a new tree T^* by taking the disjoint union of all the T_i, adding fresh nodes r^* and r_i^* for $i \leq m$, and adding edges $r^* r_i^*$, $r_i^* r_i$ for all $i \in [m]$. We define $\beta^* \colon V(T^*) \to 2^{V(T)}$ by

$$\beta^*(t) := \begin{cases} B \cup \{b\} & \text{if } t = r^*, \\ B_i \cup \{b\} & \text{if } t = r_i^*, \\ \beta_i(t) & \text{if } t \in V(T_i). \end{cases}$$

Then (T^*, β^*) is a tree decomposition of T of width at most 3 and height at most $2\log n$.

Case 2: $|B| = 3$.

By Lemma 7 applied to the characteristic function of B, there is a node $b \in V(T)$ such that for every connected component C of $T \setminus \{b\}$, it holds that $|V(C) \cap B| \leq 1$.

Let $C_1, \ldots, C_\ell$ be the connected components of $T \setminus \{b\}$, and for every i, let $B_i := B \cap V(C_i)$. Then $|B_i| \leq 1$.

▷ Claim 9. For every $i \in [\ell]$, there is a tree decomposition (T_i, β_i) of width at most 3 such that the height of T_i is at most $2\log n - 1$ and for the root r_i of T_i it holds that $B_i \subseteq \beta_i(r_i)$ and $|\beta(r_i)| \leq 2$.

Proof. Let $\in [\ell]$ and $n_i := |C_i|$. By Lemma 7, there is a $c \in V(C_i)$ such that for every connected component D of $C_i \setminus \{c\}$, it holds that $|D| \leq n_i/2$. Choose such a c and let $D_1, \ldots, D_m$ be the connected components of $C_i \setminus \{c\}$. For every $j \in [m]$, let d_j be the unique neighbour of c in D_j. Let $B_{ij} := (B_i \cap D_j) \cup \{d_j\}$. Then $|B_{ij}| \leq 2$.
By the induction hypotheses, for every j, there is a rooted tree decomposition (T_{ij}, β_{ij}) of D_j of width 3 such that the height of T_{ij} is at most $2 \log |D_i| \leq 2 \log(n_i/2) \leq 2 \log n - 2$. Furthermore, for the root r_{ij} of T_{ij}, it holds that $B_{ij} \subseteq \beta_{ij}(r_{ij})$ and $|\beta_{ij}(r_{ij}) \setminus B_{ij}| \leq 1$. This implies $|\beta_{ij}(r_{ij})| \leq 3$.
We form a new tree T_i by taking the disjoint union of all the T_{ij} for $j \in [m]$, adding a fresh node r_i, and adding edges r_{ij} for all $j \in [m]$. We define $\beta_i \colon V(T_i) \to 2^{V(C_i)}$ by

$$\beta_i(t) := \begin{cases} B_i \cup \{c\} & \text{if } t = r_i, \\ \beta_{ij}(r_{ij}) \cup \{c\} & \text{if } t = r_{ij}, \\ \beta_{ij}(t) & \text{if } t \in V(T_{ij}) \setminus \{r_{ij}\}. \end{cases}$$

Then (T_i, β_i) is a tree decomposition of C_i with the desired properties. ◁

To complete the proof of the lemma, we form a new tree T^* by taking the disjoint union of the T_i of Claim 9 for $i \in [\ell]$, adding a fresh node r^*, and adding edges $r^* r_i$ for all $i \in [\ell]$. We define $\beta^* \colon V(T^*) \to 2^{V(T)}$ by

$$\beta^*(t) := \begin{cases} B \cup \{b\} & \text{if } t = r^*, \\ \beta(r_i) \cup \{b\} & \text{if } t = r_i, \\ \beta_i(t) & \text{if } t \in V(T_i) \setminus \{r_i\}. \end{cases}$$

Then (T^*, β^*) is a tree decomposition of T of width at most 3 and height at most $2 \log n$. ◀

Let us now turn to decompositions of a graph into its 3-connected components. We need a few more definitions. In the following, let G be a connected graph and $X \subseteq V(G)$. The *torso* of X is the graph $G[\![X]\!]$ with vertex set X and edge set

$$\left\{ vw \in \binom{X}{2} \;\middle|\; vw \in E(G) \text{ or } v, w \in N_G(C) \text{ for some connected component } C \text{ of } G \setminus X \right\}.$$

The *adhesion* of X is the maximum of $|N_G(C)|$ for all connected components C of $G \setminus X$. It is easy to see that if the adhesion of X is at most 2, then the torso $G[\![X]\!]$ is a topological subgraph of G and if the adhesion of X is at most 1, then the torso $G[\![X]\!]$ is just the induced subgraph $G[X]$.

A *block*[1] of G is a set $B \subseteq V(G)$ such that

- either $G[\![B]\!]$ is 3-connected and the adhesion of B is at most 2,
- or $G[\![B]\!]$ is a complete graph of order 3 and the adhesion of B is at most 2,
- or $G[\![B]\!]$ is a complete graph of order 2 and the adhesion of B is at most 1.

We call blocks with 3-connected torsos *proper blocks* and blocks of cardinality at most 3 *degenerate blocks* of *order* 3 and 2, respectively. It is easy to see that for distinct blocks B, B', neither $B \subseteq B'$ nor $B' \subseteq B$ holds and, furthermore, $|B \cap B'| \leq 2$. A *block separator*

[1] Our usage of the term "block" is non-standard. If anything, what we call a "block" might better be called "2-block". But just using "block" is more convenient.

is a set $S \subseteq V(G)$ such that there are distinct blocks B, B' with $S = B \cap B'$, and the two sets $B \setminus S$ and $B' \setminus S$ belong to different connected components of $G \setminus S$. Note that by the definition of blocks, block separators have cardinality at most 2.

Observe that the torsos of all blocks of a graph are topological subgraphs. As all topological subgraphs of a planar graph are planar, the torsos of the blocks of a planar graph are planar. In particular, the torsos of proper blocks are 3-connected planar graphs. This will be important later.

Call a tree decomposition (T, β) *small* if for all distinct nodes $t, u \in V(T)$, it holds that $\beta(t) \not\subseteq \beta(u)$.

▶ **Lemma 10** ([35]). *Every connected graph G has a small tree decomposition (T, β) of adhesion at most 2 such that for all $t \in V(T)$, the bag $\beta(t)$ is a block of G.*

The decomposition in this lemma is essentially Tutte's well-known decomposition of a graph into its 3-connected components described in a slightly non-standard way. The two main differences are that, normally, the decomposition is only described for 2-connected graphs, whereas arbitrary connected graphs are first decomposed into their 2-connected components. We merge these decompositions into one. The second difference is that Tutte decomposes a 2-connected graph into 3-connected pieces (our proper blocks) and cycles. Instead of cycles, we only allow triangles, i.e., degenerate blocks of order 3. This is possible because every cycle can be decomposed into triangles. What we lose with our form of decomposition is the canonicity: a graph may have several structurally different decompositions of the form described in the lemma.

In the following, we apply Lemma 8 to the tree of the decomposition of Lemma 10 and obtain a decomposition of logarithmic height that is still essentially a decomposition into 3-connected components.

▶ **Lemma 11.** *Every connected graph G has a rooted tree decomposition (T^*, β^*) with the following properties.*

- **(i)** *The height of T^* is at most $2 \log |G|$.*
- **(ii)** *For every $t^* \in V(T^*)$, there are sets $B_1, \dots, B_4$ (not necessarily distinct or disjoint) such that $\beta^*(t^*) = \bigcup_{i=1}^{4} B_i$ and each B_i is either a block or a block separator.*
- **(iii)** *The adhesion of (T^*, β^*) is at most 6.*
- **(iv)** *For every $t^* \in V(T^*)$ and every child u^* of t^*, the induced subgraph*

$$G\left[\Big(\bigcup_{v^* \trianglerighteq^{T^*} u^*} \beta^*(v^*)\Big) \setminus \beta^*(t^*)\right]$$

is connected.

Proof. Let (T, β) be the decomposition of G into its blocks obtained from Lemma 10. Let (T^*, β^*_T) be the rooted tree decomposition of T obtained from Lemma 8. Let r^* be the root of T^*, and let $\trianglelefteq^* := \trianglelefteq^{T^*}$ be the partial descendant order associated with T^*. For every $t^* \in V(T^*)$, let

$$\gamma^*_T(t^*) := \bigcup_{u^* \trianglerighteq^* t^*} \beta^*_T(u^*)$$

and

$$\sigma^*_T(t^*) := \begin{cases} \emptyset & \text{if } t^* = r^*, \\ \beta^*_T(s^*) \cap \beta^*_T(t^*) & \text{for the parent } s^* \text{ of } t^* \text{ in } T^*, \text{ otherwise}. \end{cases}$$

For every $t \in V(T)$, we let $\min^*(t)$ be the unique $\trianglelefteq^*$-minimal node $t^* \in V(T^*)$ such that $t \in \beta_T^*(t^*)$. The uniqueness follows from the fact that the set of all $t^* \in V(T^*)$ with $t \in \beta_T^*(t^*)$ is connected in T^*.

Let us call $t \in V(T)$ *active* in $t^* \in V(T^*)$ if $t \in \beta_T^*(t^*)$ and $t^* \neq \min^*(t)$ and there is a $u \in N_T(t)$ such that $t^* \trianglelefteq \min^*(u)$. We call u an *activator* of t in t^*.

▷ Claim 12. Suppose that $t \in V(T)$ is active in $t^* \in V(T^*)$. Then there is a unique activator of t in t^*.

Proof. Since $t \in \beta_T^*(t^*)$ and $t^* \neq \min^*(t)$, we have $\min^*(t) \triangleleft t^*$ and $t \in \beta_T^*(\min^*(t)) \cap \beta_T^*(t^*) \subseteq \sigma_T^*(t^*)$. Moreover, for every activator u of t, it holds that $t^* \trianglelefteq \min^*(u)$, which implies $u \in \gamma_T^*(t^*) \setminus \sigma_T^*(t^*)$.

Suppose towards a contradiction that t has two activators u_1, u_2 in t^*. Then $u_1, u_2 \in N_T(t) \cap \big(\gamma_T^*(t^*) \setminus \sigma_T^*(t^*)\big)$. By Lemma 8(iv), the induced subgraph $T[\gamma_T^*(t^*) \setminus \sigma_T^*(t^*)]$ is connected. Thus, there is a path from u_1 to u_2 in $T[\gamma_T^*(t^*) \setminus \sigma_T^*(t^*)]$. As $u_1, u_2 \in N_T(t)$ and $t \in \sigma_T^*(t^*)$, there is a cyle in T, which is a contradiction. ◁

Hence, in the following we can speak of *the* activator of a node. Observe that if t is active in t^*, then t is also active in all u^* with $\min^*(t) \triangleleft u^* \triangleleft t^*$, with the same activator.

Now we are ready to define our tree decomposition (T^*, β^*) of G. The tree T^* is the same as in the decomposition (T^*, β_T^*) of T. We define $\beta^* \colon V(T^*) \to 2^{V(G)}$ by letting $\beta^*(t^*)$ for $t^* \in V(T^*)$ be the union of the following sets:

- for all $t \in \beta_T^*(t^*)$ such that $t^* = \min^*(t)$: the block $\beta(t)$, and
- for all $t \in \beta_T^*(t^*)$ such that t is active in t^* with activator u: the block separator $\beta(t) \cap \beta(u)$.

▷ Claim 13. (T^*, β^*) is a tree decomposition of G.

Proof. Every edge $e \in E(G)$ is contained in some bag $\beta(t)$, and $\beta(t) \subseteq \beta^*(\min^*(t))$.

Now consider a vertex $v \in V(G)$. Let

$$S_v \coloneqq \{t \in V(T) \mid v \in \beta(t)\},$$
$$S_v^* \coloneqq \{t^* \in V(T^*) \mid S_v \cap \beta_T^*(t^*) \neq \emptyset\}.$$

Since (T, β) is a tree decomposition, S_v is connected in T, and as (T^*, β_T^*) is a tree decomposition, S_v^* is connected in T^*. Thus, there is a unique $\trianglelefteq^*$-minimal node s^* in S_v^*. Let $s \in S_v \cap \beta_T^*(s^*)$. Then $s^* = \min^*(s)$ and therefore $v \in \beta^*(s^*)$.

Let $t^* \in V(T^*)$ such that $v \in \beta^*(t^*)$. We shall prove that $v \in \beta^*(v^*)$ for all v^* on the path from t^* to s^*. This will prove that the set of all t^* for which $v \in \beta^*(t^*)$ holds is connected in T^*.

By the definition of β^*, since $v \in \beta^*(t^*)$, there is a $t \in \beta_T^*(t)$ such that $v \in \beta(t)$ and either $t^* = \min^*(t)$ or t is active in t^*. We choose such a t. Then $t \in S_v$ and therefore $t^* \in S_v^*$. By the minimality of s^*, this implies $s^* \trianglelefteq^* t^*$.

The proof that $v \in \beta^*(v^*)$ holds for all v^* on the path from t^* to s^* is by induction on the distance d between t^* and s^*. The base case $d = 0$ is trivial. So let us assume that $d \geq 1$. It follows from the definition of β^* that $v \in \beta^*(v^*)$ holds for all v^* on the path from t^* to $\min^*(t)$. Thus, without loss of generality, we may assume that $t^* = \min^*(t)$.

Let $t = t_1, \ldots, t_m = s$ be the path from t to s in T. Note that $v \in \beta(t_i)$ holds for all $i \in [m]$. The edge $tt_2 = t_1t_2$ must be covered by some bag $\beta_T^*(u^*)$ that contains both t and t_2. Since $t^* = \min^*(t)$, we have $t^* \trianglelefteq^* u^*$. As the pre-image of the path $t_1, \ldots, t_m$ in T^* is connected and $s^* \trianglelefteq^* t^* \trianglelefteq^* u^*$, there is an $i > 1$ such that $t_i \in \beta^*(t^*)$. If $\min^*(t_i) = t^*$, we find a $j > i$ such that $t_j \in \beta^*(t)$, and, repeating this, we eventually arrive at a $t_k \in \beta^*(t)$ such

that $\min^*(t_k) \lhd t^*$. Arguing as above, we find that $v \in \beta^*(v^*)$ holds for all v^* on the path from t^* to $\min^*(t_k)$. Since $\min^*(t_k)$ is closer to s^* than t^*, we can now apply the induction hypothesis to conclude that $v \in \beta^*(v^*)$ holds for all v^* on the path from $\min^*(t_k)$ to s^*. ◁

Let us turn to proving that the tree decomposition (T^*, β^*) has the desired properties.

Since (T, β) is a small decomposition, we have $|T| \leq |G|$. Thus, Condition (i) follows from Lemma 8(i). Also, Condition (ii) follows from Lemma 8(ii) and the definition of $\beta^*(t)$.

To prove Condition (iii), let u^* be a child of t^*. Let us assume that $\beta^*_T(t^*) = \{t_1, \dots, t_4\}$ and $\beta^*_T(u^*) = \{u_1, \dots, u_4\}$ with $t_1 = u_1, t_2 = u_2$, and $t_3 = u_3$ and $t_4 \neq u_i, u_4 \neq t_i$ for $i \in [4]$. The cases of smaller bags $\beta^*_T(t^*)$, $\beta^*_T(u^*)$ or a smaller intersection between them can be dealt with similarly.

Let us first deal with the common elements $t_i = u_i$ for $i \in [3]$. Note that $\min^*(t_i) \trianglelefteq t^* \lhd u^*$. If t_i is not active in u^*, then it does not contribute to the $\beta^*(u^*)$ and hence not to the intersection of the two bags. If t_i is active in u^*, say, with activator v_i, then the block separator $S_i \coloneqq \beta(t_i) \cap \beta(v_i)$ is contained in $\beta^*(u^*)$. To simplify the notation, in the following, we let $S_i \coloneqq 0$ if t_i is not active in u^*.

Either t_i is active in t^* as well with the same activator and we have $S_i \subseteq \beta^*(t^*)$, or $t^* = \min^*(t_i)$ and $S_i \subseteq \beta(t_i) \subseteq \beta^*(t^*)$. In both cases,

$$S_i \subseteq \beta^*(t^*) \cap \beta(u^*). \tag{1}$$

Next, let us look at the contribution of t_4 and u_4. The contribution of t_4 to $\beta^*(t^*)$ is contained in $\beta(t_4)$, and the contribution of u_4 to $\beta^*(u^*)$ is contained in $\beta(u_4)$. Since the only neighbour of t_i in $\gamma^*_T(u^*) \setminus \sigma^*_T(u^*) = \gamma^*_T(u^*) \setminus \{t_1, t_2, t_3\}$ is v_i (if t_i is active in u^*, otherwise there is no neighbour), all paths from t_i to u_4 go through v_i. This implies that

$$\beta(t_i) \cap \beta(u_4) \subseteq \beta(t_i) \cap \beta(v_i) = S_i. \tag{2}$$

All paths from t_4 to u_4 go through t_1, t_2, t_3, and therefore

$$\beta(t_4) \cap \beta(u_4) \subseteq \bigcup_{i=1}^{3} \beta(t_i) \cap \beta(u_4) \subseteq S_1 \cup S_2 \cup S_3. \tag{3}$$

Thus, overall, we have $\beta^*(t^*) \cap \beta^*(u^*) \subseteq S_1 \cup S_2 \cup S_3$.

To prove that Condition (iv) holds, let $t^* \in V(T^*)$ and and let u^* be a child of t^*. To simplify the notation, let $\sigma^*(u^*) \coloneqq \beta(u^*) \cap \beta^*(t^*)$ and

$$\gamma^*(u^*) \coloneqq \bigcup_{v^* \trianglerighteq u^*} \beta^*(v^*). \tag{4}$$

We need to prove that $G[\gamma^*(u^*) \setminus \sigma^*(u^*)]$ is connected. The key observation is that

$$\gamma^*(u^*) \setminus \sigma^*(u^*) = \bigcup_{t \in \gamma^*_T(u^*) \setminus \sigma^*_T(u^*)} \beta(t). \tag{5}$$

The reason for this is that, for all $t \in \gamma^*_T(u^*) \setminus \sigma^*_T(u^*)$, it holds that $u^* \trianglelefteq \min^*(t)$, which implies that $\beta(t) \subseteq \beta^*(\min^*(t))$ appears on the right-hand side of (4). It follows from Lemma 8(iv) that the set $\gamma^*_T(u^*) \setminus \sigma^*_T(u^*)$ is connected in T, and this implies that the union on the right-hand side of (5) is connected. ◀

Our next goal will be to define the decomposition in the logic $\mathsf{C}^{O(1)}_{O(\log n)}$. The following lemma yields a way to define blocks via triplets of vertices.

▶ **Lemma 14.** *Let G be a graph, and let B be a proper block of G. Let $b_1, b_2, b_3 \in B$ be pairwise distinct vertices. Then B is the set of all $v \in V(G)$ such that there is no set $S \subseteq V(G) \setminus \{v\}$ of cardinality at most 2 separating v from $\{b_1, b_2, b_3\}$.*

Proof. Let $v \in B$. Since $G[\![B]\!]$ is 3-connected, there are paths $P_i \subseteq G[\![B]\!]$ from v to b_i that are internally disjoint, that is, $V(P_i) \cap V(P_j) = \{v\}$ for $i \neq j$. As $G[\![B]\!]$ is a topological subgraph of G, these paths can be expanded to paths P'_i from v to b_i in G, and the P'_i are still internally disjoint. Since every $S \subseteq V(G) \setminus \{v\}$ of cardinality at most 2 has an empty intersection with at least one of the paths P'_i, it does not separate v from $\{b_1, b_2, b_3\}$.

Conversely, let $v \in V(G) \setminus B$, and let C be the connected component of $G \setminus B$ with $v \in V(C)$, and let $S := N_G(C)$. Then $|S| \leq 2$. Then S separates v from $\{b_1, b_2, b_2\}$. ◀

Let G be a graph and $S, X \subseteq V(G)$. We say that S *separates* X if there are two distinct connected components C_1, C_2 of $G \setminus S$ such that $X \cap V(C_i) \neq \emptyset$ for both $i = 1, 2$.

▶ **Lemma 15.** *Let G be a graph, and let $b_1, b_2, b_3 \in V(G)$ be mutually distinct. Then there is a proper block B with $b_1, b_2, b_3 \in B$ if and only if there is a vertex $b_4 \in V(G) \setminus \{b_1, b_2, b_3\}$ such that no set $S \subseteq V(G)$ of cardinality at most 2 separates $\{b_1, b_2, b_3, b_4\}$.*

Proof. For the forward direction, suppose that B is a proper block with $b_1, b_2, b_3 \in B$. Let $b_4 \in B \setminus \{b_1, b_2, b_3\}$. Then it follows from Lemma 14 that there is no S of cardinality at most 2 that separates $\{b_1, b_2, b_3, b_4\}$.

For the backward direction, let B be the set of all $v \in V(G)$ such that no set $S \subseteq V(G) \setminus \{v\}$ of cardinality at most 2 separates v from $\{b_1, b_2, b_3\}$. Then $b_1, b_2, b_3 \in B$ and $|B| \geq 4$. It is easy to prove that B is a block. ◀

▶ **Lemma 16.** *For all $n \in \mathbb{N}$, there exist $\mathsf{C}^{O(1)}_{O(\log n)}$-formulas $\mathsf{block}^{(n)}(x_1, x_2, x_3, y)$ and $\mathsf{torso}^{(n)}(x_1, x_2, x_3, y, z)$ such that for all graphs G of order at most n and all $b_1, b_2, b_3, v \in V(G)$, we have*

$$G \models \mathsf{block}^{(n)}(b_1, b_2, b_3, v)$$

if and only if one of the following holds:

- *either $\{b_1, b_2, b_3\}$ is a degenerate block and $v \in \{b_1, b_2, b_3\}$,*
- *or b_1, b_2, b_3 are mutually distinct and there is a proper block B such that $b_1, b_2, b_3, v \in B$.*

Moreover, for all $b_1, b_2, b_3, v, w \in V(G)$, we have

$$G \models \mathsf{torso}^{(n)}(b_1, b_2, b_3, v, w)$$

if and only if $G \models \mathsf{block}^{(n)}(b_1, b_2, b_3, v)$ and $G \models \mathsf{block}^{(n)}(b_1, b_2, b_3, w)$ and vw is an edge of the torso of the block determined by b_1, b_2, b_3.

Proof. It is easy to express in $\mathsf{C}^{O(1)}_{O(\log n)}$ that $\{b_1, b_2, b_3\}$ is a degenerate block. For proper blocks, we use Lemmas 14 and 15. ◀

As an immediate consequence, we obtain a formula to define a block separator.

▶ **Corollary 17.** *For all $n \in \mathbb{N}$, there exists a $\mathsf{C}^{O(1)}_{O(\log n)}$-formula $\mathsf{blocksep}^{(n)}(x_1, x_2)$ such that for all graphs G of order at most n and all $s_1, s_2 \in V(G)$, we have*

$$G \models \mathsf{blocksep}^{(n)}(s_1, s_2)$$

if and only if $\{s_1, s_2\}$ is a block separator of G.

We are ready to define the formula that yields the decomposition from Lemma 11.

▶ **Lemma 18.** *For all $h \geq 0$, $n \geq 1$, there is a $\mathsf{C}^{O(1)}_{O(h+\log n)}$-formula $\mathsf{dec}^{(n)}_h(x^j_i, y_k \mid i \in [4], j \in [3], k \in [6])$ such that the following holds. Let G be a graph of order $|G| \leq n$ and $b^j_i, s_k \in V(G)$ for $i \in [4], j \in [3], k \in [6]$ (not necessarily distinct). Then*

$$G \models \mathsf{dec}^{(n)}_h(b^j_i, s_k \mid i \in [4], j \in [3], k \in [6])$$

if and only if the following conditions are satisfied.

- **(i)** *For all $i \in [4]$, either $B_i := \{b^1_i, b^2_i, b^3_i\}$ is a block separator or $B_i := \{b^1_i, b^2_i, b^3_i\}$ is a degenerate block or b^1_i, b^2_i, b^3_i are mutually distinct and there is a (unique) block B_i that contains b^1_i, b^2_i, b^3_i.*
  Let $B := B_1 \cup \ldots \cup B_4$.
- **(ii)** *$S := \{s_1, \ldots, s_6\} \subset B$.*
- **(iii)** *There is a (unique) connected component C of $G \setminus S$ such that $B \subseteq S \cup V(C)$.*
- **(iv)** *The induced subgraph $G[S \cup V(C)]$ has a rooted tree decomposition (T^*, β^*) of height at most h with $B = \beta^*(r^*)$ for the root r^* of T^*.*
- **(v)** *The tree decomposition (T^*, β^*) satisfies Conditions* (ii)–(iv) *of Lemma 11, where all blocks are blocks of the graph G (rather than of the subgraph $G[S \cup C]$).*

Proof. The proof is by induction on $h \geq 0$.

However, before we begin the induction, we observe that using Lemma 16 and Corollary 17, we can write a formula in the variables x^j_i that expresses Condition (i). It is straightforward to express Condition (ii), and, again using Lemma 16, to express Condition (iii). So in the induction, we will focus on Conditions (iv) and (v).

For the case $h = 0$, observe that a decomposition of height 0 consists of a single node that covers the whole graph. So we need to express that for the component C we obtain in (iii), we have $V(C) \cup S = B$. Then the 1-node tree decomposition of $G[B]$ satisfies (iv) and (v).

For a 1-node decomposition, Conditions (iii) and (iv) of Lemma 11 are void, and Condition (ii) of Lemma 11 follows from Condition (i) of this lemma.

For the inductive step $h \to h+1$, suppose we have a graph G and elements b^j_i, s_k satisfying Conditions (i)–(iii) for suitable sets B, S, C. It suffices to express that for each connected component C' of $G[S \cup V(C)] \setminus B$, we can find a decomposition of height h that covers C' and attaches to B in a way that satisfies the conditions of Lemma 11.

So let $G' := G[S \cup V(C)]$, and let C' be a connected component of $G' \setminus B$. Let $S' := N_G(C')$. If $|S'| > 6$, then there definitely is no decomposition with the desired properties. Suppose that $S' = \{s'_1, \ldots, s'_6\}$. Then, if there are $b'^j_i \in S' \cup V(C')$ such that $G \models \mathsf{dec}^{(n)}_h(b'^j_i, s'_k \mid i \in [4], j \in [3], k \in [6])$, the desired decomposition that covers C' exists by the induction hypothesis. If this is the case for all C', we can combine the decompositions to form the desired decomposition of G'. Conversely, if there is a decomposition of $G[S' \cup V(C')]$ of height h in the sense of Lemma 11 such that for the root u^*, the bag $\beta^*(u^*)$ contains S', then there are blocks or block separators $B'_1, \ldots, B'_4$ such that $\beta^*(u^*) = B'_1 \cup \ldots \cup B'_4$. From the B'_i, we obtain b'^j_i such that $G \models \mathsf{dec}^{(n)}_h(b'^j_i, s'_k \mid i \in [4], j \in [3], k \in [6])$, again by the induction hypothesis.

To conclude, in addition to the subformulas taking care of Conditions (i)–(iii), the formula $\mathsf{dec}^{(n)}_{h+1}$ must have a subformula stating that, for all connected components C' of $G' \setminus B$, there exist $s'_k \in B$ for $k \in [6]$ and $b'^j_i \in S' \cup V(C')$ for $i \in [4]$, $j \in [3]$ such that $\{s'_1, \ldots, s'_6\} = N_G(C')$ and $\mathsf{dec}^{(n)}_h(b'^j_i, s'_k \mid i \in [4], j \in [3], k \in [6])$

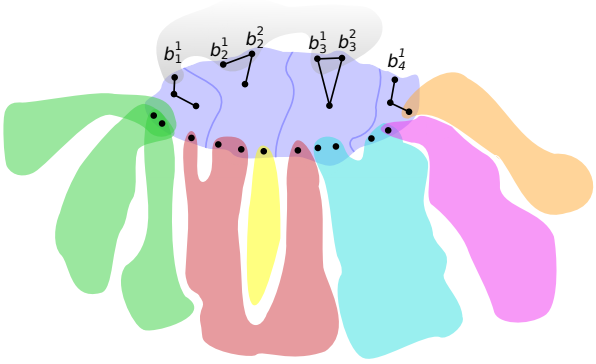


Figure 2 A (simplified) schematic visualisation of the rooted tree decomposition (T^*, β^*) from Lemma 11. For simplicity, all B_i in the bag of the purple node are depicted as distinct proper blocks.

Note that, in each step $h \to h+1$, we need to use formulas of quantifier depth $O(\log n)$ to express the desired connectivity conditions, for example to speak about components C', and to express that the b_i^j define blocks. However, the formula $\mathsf{dec}_h^{(n)}$ occurs only in the scope of constantly many (19, to be precise) quantifiers ranging over an element of the component(s) C' and the $b_i'^j, s_k'$. Overall, the quantifier depth will be $O(h) + O(\log n)$. ◀

5 Canonisation

In this section, we finally prove Theorems 1 and 2. By the logical characterisation of the WL algorithm given in Theorem 4, we obtain Theorem 1 as a corollary from Theorem 2, which we prove below.

In the following, for a graph G and a list of vertices $v_1, \dots, v_\ell \in V(G)$, we denote by $(G, v_1, \dots, v_\ell)$ the graph G with *individualised* vertices $v_1, \dots, v_\ell$. That is, $(G, v_1, \dots, v_\ell)$ and $(G', v_1', \dots, v_{\ell'}')$ have the same isomorphism type if and only if $\ell = \ell'$ and there is an isomorphism from G to G' that maps v_i to v_i' for every $i \in [\ell]$.

▶ **Lemma 19.** *For all $h \geq 0$, $n \geq 1$ and all connected planar graphs G of order $|G| \leq n$, and all $b_i^j, s_k \in V(G)$ for $i \in [4], j \in [3], k \in [6]$ (not necessarily distinct) such that*

$$G \models \mathsf{dec}_h^{(n)}(b_i^j, s_k \mid i \in [4], j \in [3], k \in [6]),$$

there is a $\mathsf{C}^{O(1)}_{O(h+\log n)}$-formula $\mathsf{iso}_h^{(n)}(x_i^j, y_k \mid i \in [4], j \in [3], k \in [6])$ (which depends on the b_i^j and the s_k) such that the following holds. Let H be a connected graph of order $|H| \leq n$ and $b_i'^j, s_k' \in V(H)$ for $i \in [4], j \in [3], k \in [6]$ (not necessarily distinct) and assume $H \models \mathsf{dec}_h^{(n)}(b_i'^j, s_k' \mid i \in [4], j \in [3], k \in [6])$. Then

$$H \models \mathsf{iso}_h^{(n)}(b_i'^j, s_k' \mid i \in [4], j \in [3], k \in [6])$$

if and only if for the connected components C_G, C_H that Lemma 18 yields for G and H, it holds that $\big(H[\{s_1', \dots, s_6'\} \cup V(C_H)], (b_i'^j, s_k' \mid i \in [4], j \in [3], k \in [6])\big) \cong \big(G[\{s_1, \dots, s_6\} \cup V(C_G)], (b_i^j, s_k \mid i \in [4], j \in [3], k \in [6])\big)$.

Proof. For the following arguments, see also Figure 2 for a better intuition.

Let $n \in \mathbb{N}$ and let G be a connected planar graph with $|G| \leq n$. The proof is by induction on $h \geq 0$.

First, given a second connected graph H of order at most $|G|$ that satisfies the $\mathsf{dec}_h^{(n)}$-formula, we can assume that the first four triplets of vertices form the same types of blocks and block separators (of corresponding sizes), respectively, in H as in G, since otherwise we can distinguish the graphs using the formulas from Lemma 16 and Corollary 17.

Note that there is a formula $\mathsf{bag}^{(n)}(x_1^1,\dots,x_4^3,y) \in \mathsf{C}^{O(1)}_{O(\log n)}$ such that for all graphs H of order at most n and all $b_1'^1, b_1'^2, b_1'^3, \dots, b_4'^1, b_4'^2, b_4'^3, v \in V(H)$, it holds that $H \models \mathsf{bag}^{(n)}(b_1'^1,\dots,b_4'^3,v)$ if and only if each set $\{b_i'^j \mid j \in [3]\}$ for $i \in [4]$ is a block separator B_i or a degenerate block B_i or contained in a proper block B_i of H and v is in $B \coloneqq \bigcup_{i=1}^4 B_i$.

The case that $h = 0$ follows analogously as the formula for the isomorphism type of the root bag in the inductive step. We therefore focus on the inductive step. Assume that for every list of vertices $(b_i'^j, s_k' \mid i \in [4], j \in [3], k \in [6]) \in V(G)^{18}$, where

$$G \models \mathsf{dec}_h^{(n)}(b_i'^j, s_k' \mid i \in [4], j \in [3], k \in [6]),$$

there is a $\mathsf{C}^{O(1)}_{O(h+\log n)}$-formula

$$\mathsf{iso}_{G,(b_i'^j, s_k' \mid i\in[4], j\in[3], k\in[6])}(x_1'^1,\dots,x_4'^3,y_1',\dots,y_6')$$

that defines the isomorphism type of $(G[\{s_1',\dots,s_6'\} \cup V(C')], (b_i'^j, s_k' \mid i \in [4], j \in [3], k \in [6]))$, where C' is the connected component from Parts (iii)–(v) in Lemma 18.

Let $(b_i^j, s_k \mid i \in [4], j \in [3], k \in [6]) \in V(G)^{18}$ be a list of vertices such that

$$G \models \mathsf{dec}_{h+1}^{(n)}(b_i^j, s_k \mid i \in [4], j \in [3], k \in [6]).$$

For $B_1, B_2, B_3, B_4, B, C, S$ as described in Lemma 18, let (T^*, β^*) be the rooted tree decomposition from Condition (iv) in Lemma 18. Let r^* be the root of T^*. By Condition (iv) in Lemma 18, it holds that $\beta^*(r^*) = B = \bigcup_{i=1}^4 B_i$. Consider a B_i with $|B_i| \geq 4$. Then B_i is a proper block, in which, by Theorem 5, we can find vertices v_i^1, v_i^2, v_i^3 such that for all $w \in B_i$, there is a $\mathsf{C}^{O(1)}_{O(\log n)}$-formula $\mathsf{id}'_w(x_i^1, x_i^2, x_i^3, y)$ such that $G[[B_i]] \models \mathsf{id}'_w(v_i^1, v_i^2, v_i^3, w)$ and $G[[B_i]] \not\models \mathsf{id}'_w(v_i^1, v_i^2, v_i^3, w')$ for every $w' \in B_i \setminus \{v\}$. (In every B_i with $|B_i| \leq 3$, such vertex-identifying formulas with four free variables exist trivially and they also identify the vertex the entire graph G.)

For simplicity, first assume that for all i with $|B_i| \geq 4$, the vertex v_i^j equals b_i^j for $j \in [3]$. Then by replacing in $\mathsf{id}'_v(x_i^1, x_i^2, x_i^3, y)$ every subformula of the form $\exists^{\geq k} x \psi$ with $\exists^{\geq k} x(\psi \wedge \mathsf{block}^{(n)}(x_i^1, x_i^2, x_i^3, x))$ and every $E(x,y)$ with $\mathsf{torso}^{(n)}(x_i^1, x_i^2, x_i^3, x, y)$, we easily obtain for each $v \in B$ a $\mathsf{C}^{O(1)}_{O(\log n)}$-formula $\mathsf{id}_v(x_1^1,\dots,x_4^3,y)$ with $\mathsf{id}_v[G, b_1^1,\dots,b_4^3, y] = \{v\}$.

Now we can use these formulas to address each vertex individually. More formally, we can define the edge relation of $G[B]$ by setting, for $v, w \in B$ with $v \neq w$,

$$\varphi_{v,w}(x,y) \coloneqq \begin{cases} E(x,y) & \text{if } vw \in E(G), \\ \neg E(x,y) & \text{otherwise.} \end{cases}$$

Then the $\mathsf{C}^{O(1)}_{O(\log n)}$-formula

$$\begin{aligned} \mathsf{iso}_B(x_1^1,\dots,x_4^3) \coloneqq \bigwedge_{v,w\in B} \exists^{=1} x \Big(\mathsf{id}_v(x_1^1,\dots,x_4^3,x) \wedge \exists^{=1} x' \big(\mathsf{id}_w(x_1^1,\dots,x_4^3,x') \wedge \varphi_{v,w}(x,x') \big) \Big) \\ \wedge \neg \exists x \Big(\mathsf{bag}^{(n)}(x_1^1,\dots,x_4^3,x) \to \bigwedge_{w\in B} \neg \mathsf{id}_w(x_1^1,\dots,x_4^3,x) \Big) \wedge \\ \bigwedge_{v\neq w\in B} \neg \exists x \Big(\mathsf{id}_v(x_1^1,\dots,x_4^3,x) \wedge \mathsf{id}_w(x_1^1,\dots,x_4^3,x) \Big) \end{aligned}$$

defines the isomorphism type of $(G[B], b_1^1,\dots,b_4^3)$ (see the purple bag in Figure 2).

We now construct a formula that describes how the connected components of $G[S \cup V(C)] \setminus B$ are attached to $G[B]$. Let $G' \coloneqq G[S \cup V(C)]$. By Condition (iii) in Lemma 11, for every connected component C' of $G' \setminus B$, it holds that $|N_G(C')| \leq 6$ (see the coloured shapes attached to the purple one in Figure 2). Hence, we iterate over all tuples $(s'_1, \dots, s'_6) \in B^6$: let $\mathcal{M}^{s'_1,\dots,s'_6}$ be the multiset of isomorphism types of the graphs $(G[S' \cup C'], s'_1, \dots, s'_6)$, where $S' \coloneqq \{s'_i \mid i \in [6]\}$ and C' is a connected component of $G' \setminus B$ with $N_G(C') = S'$.

Since $G \models \mathsf{dec}^{(n)}_{h+1}(b^j_i, s_k \mid i \in [4], j \in [3], k \in [6])$, for every $(s'_1, \dots, s'_6) \in B^6$ and every connected component C' of $G' \setminus B$ with $N_G(C') = \{s'_1, \dots, s'_6\}$, there exist vertices $(b'^j_i \mid i \in [4], j \in [3]) \in (S' \cup V(C'))^{12}$ such that

$$G[S' \cup V(C')] \models \mathsf{dec}^{(n)}_h(b'^j_i, s'_k \mid i \in [4], j \in [3], k \in [6]).$$

So, by the induction hypothesis, there is a formula $\mathsf{iso}_M(x'^1_1, \dots, x'^3_4, y'_1, \dots, y'_6) \in \mathsf{C}^{O(1)}_{O(h+\log n)}$ for the isomorphism type M of $(G[\{s'_1, \dots, s'_6\} \cup V(C')], (b'^j_i, s'_k \mid i \in [4], j \in [3], k \in [6]))$. Note that by Condition (ii) in Lemma 18, at least one of the vertices b'^j_i will lie outside B. Using the counting quantifiers, we can use the iso_M to make sure that every isomorphism type appears with the correct multiplicity. More precisely, we first group all components with equal isomorphism types. The fact that they are of the same size enables us to define their number (e.g. the three green shapes in Figure 2). This then allows us to build a formula $\mathsf{iso}'_{\mathcal{M}}(y'_1, \dots, y'_6)$ which identifies the graph $(G[S' \cup \bigcup_{C' : N_G(C') = S'} V(C')], s'_1, \dots, s'_6)$ (where $\mathcal{M} \coloneqq \mathcal{M}^{s'_1,\dots,s'_6}$ and the C' are connected components of $G' \setminus B$). Using the $\mathsf{dec}^{(n)}_h$-formula, we can turn $\mathsf{iso}'_{\mathcal{M}}(y'_1, \dots, y'_6)$ into a formula $\mathsf{iso}_{\mathcal{M}}(x^1_1, \dots, x^3_4, y_1, \dots, y_6, y'_1, \dots, y'_6)$ that ensures that $\mathsf{iso}_{\mathcal{M}}(b^1_1, \dots, b^3_4, s_1, \dots, s_6, s'_1, \dots, s'_6)$ describes for $S' \coloneqq \{s'_1, \dots, s'_6\}$ the subgraph $(G[S' \cup \bigcup_{C' : N_G(C') = S'} V(C')], s'_1, \dots, s'_6)$, where the C' are connected components of $G' \setminus B$, up to isomorphism.

Hence, it suffices to conjugate $\mathsf{iso}_B(x^1_1, \dots, x^3_4)$ with a conjunction over all $(s'_1, \dots, s'_6) \in B^6$ of the following formula

$$\exists y'_1 \dots \exists y'_6 \left(\bigwedge_{i=1}^{6} \mathsf{id}_{s'_i}(x^1_1, \dots, x^3_4, y'_i) \wedge \mathsf{iso}_{\mathcal{M}}(x^1_1, \dots, x^3_4, y_1, \dots, y_6, y'_1, \dots, y'_6) \right),$$

where $\mathcal{M} \coloneqq \mathcal{M}^{s_1,\dots,s_6}$, to obtain the desired $\mathsf{iso}_{G,(b^j_i, s_k \mid i \in [4], j \in [3], k \in [6])}(x^1_1, \dots, x^3_4, y_1, \dots, y_6)$.

We now consider the general case where it does not necessarily hold for all i, j that $v^j_i = b^j_i$. We assume for notational simplicity that for all i, the b^1_i, b^2_i, b^3_i define a block. It is easy to adapt the following construction to the situation that block separators are present.

We introduce one nested existential quantifier $\exists \tilde{x}^j_i$ for each of the v^j_i so that our resulting formula $\mathsf{iso}_{G,(b^j_i, s_k \mid i \in [4], j \in [3], k \in [6])}(x^1_1, \dots, x^3_4, y_1, \dots, y_6)$ looks as follows:

$$\exists \tilde{x}^1_1 \dots \exists \tilde{x}^3_4 \Bigg(\bigwedge_{j=1}^{3} \bigwedge_{i=1}^{4} \mathsf{block}^{(n)}(x^1_i, x^2_i, x^3_i, \tilde{x}^j_i) \wedge \mathsf{iso}_B(\tilde{x}^1_1, \dots, \tilde{x}^3_4) \wedge \bigwedge_{(s'_1,\dots,s'_6) \in B^6} \exists y'_1 \dots \exists y'_6 \Bigg(\bigwedge_{i=1}^{6} \mathsf{id}_{s'_i}(\tilde{x}^1_1, \dots, \tilde{x}^3_4, y'_i) \wedge \mathsf{iso}_{\mathcal{M}}(x^1_1, \dots, x^3_4, y_1, \dots, y_6, y'_1, \dots, y'_6) \Bigg) \Bigg).$$

The bounds on the quantifier depth and the number of variables follow similarly as in the proof of Lemma 18. ◀

Applying Lemma 8, we can deduce Theorem 2.

Proof of Theorem 2. Let $n \in \mathbb{N}$ and let G be a planar graph with order $|G| = n$. If G is not connected, we construct one formula for each connected component of G (as described in the following) and join them to obtain the identifying sentence.

So suppose G is connected. Then by Lemma 11, G has a rooted tree decomposition (T^*, β^*) of logarithmic height and adhesion at most 6 for which every bag is a union of four (not necessarily distinct) blocks or block separators and also Condition (iv) of the lemma holds. Let $b_1^1, \dots, b_4^3$ be vertices that determine the blocks and block separators in the root bag B of (T^*, β^*).

If there is a vertex $s \in B$ such that there is a unique connected component C of $G \setminus \{s\}$ with $B \subseteq \{s\} \cup V(C)$, then there are vertices b_i^j, s_k for $i \in [4], j \in [3], k \in [6]$ (e.g. $s_k = s$ for all k) such that G satisfies $\mathsf{dec}^{(n)}_{2\log|G|}(b_i^j, s_k \mid i \in [4], j \in [3], k \in [6])$. Then the sentence

$$\exists x_1^1 \dots \exists x_4^3 \exists y_1 \dots \exists y_6 \mathsf{iso}_{G,(b_i^j, s_k \mid i\in[4], j\in[3], k\in[6])}(x_1^1, \dots, x_4^3, y_1, \dots, y_6)$$

identifies G, where $\mathsf{iso}_{G,(b_i^j, s_k \mid i\in[4], j\in[3], k\in[6])}$ is the formula from Lemma 19.

Otherwise, let $s \in B$ be a vertex such that $G \setminus \{s\}$ has multiple connected components C_i and let $G_i \coloneqq G[V(C_i) \cup \{s\}]$. Then the restriction of (T^*, β^*) to each G_i still satisfies the conditions of Lemma 11, because the block structure of G_i is just the block structure induced by G on $V(G_i)$ (that is, the blocks of G_i are precisely those blocks of G contained in $V(G_i)$, and similarly for the block separators). This yields by Lemma 19 an identifying formula $\varphi_i(y)$ for each (G_i, s), which we can join by isomorphism type of (G_i, s) to obtain an identifying sentence. ◀

We can directly deduce Theorem 1.

Proof of Theorem 1. The theorem follows from Theorems 2 and 4. ◀

6 Conclusion

We prove that planar graphs are identified by the WL algorithm with constant dimension in a logarithmic number of iterations, thereby completing a project started by Verbitsky fourteen years ago with his proof of the same result in the special case of 3-connected planar graphs. Our proof is based on the careful analysis of a novel logarithmic-depth decomposition of graphs into their 3-connected components.

It is unclear which dimension of the WL algorithm is necessary to identify planar graphs in logarithmically many iterations and if there is a (provable) trade-off between dimension and iteration number. This is not only interesting for planar graphs, and many questions remain open.

We leave it as another interesting open project whether our result can be extended to graph classes of bounded genus. As it stands, our proof heavily relies on properties of 3-connected planar graphs that are not shared by 3-connected graphs of higher genus. Similarly, we pose as a challenge to find good bounds on the iteration number of the WL algorithm on other parameterised graph classes, such as graphs with a certain excluded minor or graphs of bounded rank width.

References

1 B. Ahmadi, K. Kersting, M. Mladenov, and S. Natarajan. Exploiting symmetries for scaling loopy belief propagation and relational training. *Machine Learning Journal*, 92(1):91–132, 2013. doi:10.1007/s10994-013-5385-0.

2 A. Atserias and E. N. Maneva. Sherali-Adams relaxations and indistinguishability in counting logics. *SIAM Journal on Computing*, 42(1):112–137, 2013. doi:10.1137/120867834.

3 A. Atserias and J. Ochremiak. Definable ellipsoid method, sums-of-squares proofs, and the isomorphism problem. In *Proceedings of the 33rd Annual ACM/IEEE Symposium on Logic in Computer Science (LICS '18)*, pages 66–75, 2018. doi:10.1145/3209108.3209186.

4 L. Babai. Graph isomorphism in quasipolynomial time. In *Proceedings of the 48th Annual ACM Symposium on Theory of Computing (STOC '16)*, pages 684–697, 2016. doi:10.1145/2897518.2897542.

5 J. Cai, M. Fürer, and N. Immerman. An optimal lower bound on the number of variables for graph identification. *Combinatorica*, 12:389–410, 1992. doi:10.1007/BF01305232.

6 G. Chen and I. Ponomarenko. Lectures on coherent configurations. Lecture notes available at http://www.pdmi.ras.ru/~inp/ccNOTES.pdf, 2019.

7 P. T. Darga, M. H. Liffiton, K. A. Sakallah, and I. L. Markov. Exploiting structure in symmetry detection for CNF. In *Proceedings of the 41st Design Automation Conference (DAC '04)*, pages 530–534. ACM, 2004. doi:10.1145/996566.996712.

8 H. Dell, M. Grohe, and G. Rattan. Lovász meets Weisfeiler and Leman. In *Proceedings of the 45th International Colloquium on Automata, Languages, and Programming (ICALP '18)*, pages 40:1–40:14, 2018. doi:10.4230/LIPIcs.ICALP.2018.40.

9 R. Diestel. *Graph Theory*. Springer Verlag, 5th edition, 2016.

10 Z. Dvorák. On recognizing graphs by numbers of homomorphisms. *Journal of Graph Theory*, 64(4):330–342, 2010. doi:10.1002/jgt.20461.

11 M. Elberfeld, A. Jakoby, and T. Tantau. Logspace versions of the theorems of Bodlaender and Courcelle. In *Proceedings of the 51st Annual IEEE Symposium on Foundations of Computer Science (FOCS '10)*, pages 143–152, 2010. doi:10.1109/FOCS.2010.21.

12 S. Evdokimov, I. N. Ponomarenko, and G. Tinhofer. Forestal algebras and algebraic forests (on a new class of weakly compact graphs). *Discrete Mathematics*, 225(1-3):149–172, 2000. doi:10.1016/S0012-365X(00)00152-7.

13 M. Grohe. Fixed-point logics on planar graphs. In *Proceedings of the 13th IEEE Symposium on Logic in Computer Science (LICS '98)*, pages 6–15, 1998. doi:10.1109/LICS.1998.705639.

14 M. Grohe. Isomorphism testing for embeddable graphs through definability. In *Proceedings of the 32nd ACM Symposium on Theory of Computing (STOC '00)*, pages 63–72, 2000. doi:10.1145/335305.335313.

15 M. Grohe. *Descriptive Complexity, Canonisation, and Definable Graph Structure Theory*, volume 47 of *Lecture Notes in Logic*. Cambridge University Press, 2017. doi:10.1017/9781139028868.

16 M. Grohe. The logic of graph neural networks. In *Proceedings of the 36th ACM-IEEE Symposium on Logic in Computer Science (LICS '21))*, 2021. arXiv version at arXiv:2104.14624.

17 M. Grohe and S. Kiefer. A linear upper bound on the Weisfeiler-Leman dimension of graphs of bounded genus. In *Proceedings of the 46th International Colloquium on Automata, Languages, and Programming (ICALP '19)*, pages 117:1–117:15, 2019. doi:10.4230/LIPIcs.ICALP.2019.117.

18 M. Grohe and J. Mariño. Definability and descriptive complexity on databases of bounded tree-width. In *Proceedings of the 7th International Conference on Database Theory (ICDT '99)*, volume 1540 of *Lecture Notes in Computer Science*, pages 70–82. Springer, 1999. doi:10.1007/3-540-49257-7_6.

19 M. Grohe and D. Neuen. Canonisation and definability for graphs of bounded rank width. In *Proceedings of the 34th Annual ACM/IEEE Symposium on Logic in Computer Science (LICS '19)*, pages 1–13, 2019. doi:10.1109/LICS.2019.8785682.

20 M. Grohe and M. Otto. Pebble games and linear equations. *Journal of Symbolic Logic*, 80(3):797–844, 2015. doi:10.1017/jsl.2015.28.

21 M. Grohe and O. Verbitsky. Testing graph isomorphism in parallel by playing a game. In *Proceedings of the 33rd International Colloquium on Automata, Languages and Programming (ICALP '06)*, pages 3–14, 2006. doi:10.1007/11786986_2.

22 N. Immerman and E. Lander. Describing graphs: A first-order approach to graph canonization. In *Complexity theory retrospective*, pages 59–81. Springer-Verlag, 1990.

23 T. A. Junttila and P. Kaski. Engineering an efficient canonical labeling tool for large and sparse graphs. In *Proceedings of the 9th Workshop on Algorithm Engineering and Experiments (ALENEX '07)*. SIAM, 2007. doi:10.1137/1.9781611972870.13.

24 S. Kiefer. The Weisfeiler-Leman algorithm: An exploration of its power. *ACM SIGLOG News*, 7(3):5–27, 2020. doi:10.1145/3436980.3436982.

25 S. Kiefer and B. D. McKay. The iteration number of Colour Refinement. In *Proceedings of the 47th International Colloquium on Automata, Languages, and Programming (ICALP '20)*, volume 168 of *LIPIcs*, pages 73:1–73:19. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPIcs.ICALP.2020.73.

26 S. Kiefer and D. Neuen. The power of the Weisfeiler-Leman algorithm to decompose graphs. In *Proceedings of the 44th International Symposium on Mathematical Foundations of Computer Science (MFCS '19)*, volume 138 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 45:1–45:15. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2019. doi:10.4230/LIPIcs.MFCS.2019.45.

27 S. Kiefer, I. Ponomarenko, and P. Schweitzer. The Weisfeiler-Leman dimension of planar graphs is at most 3. *J. ACM*, 66(6):44:1–44:31, 2019. doi:10.1145/3333003.

28 S. Kiefer and P. Schweitzer. Upper bounds on the quantifier depth for graph differentiation in first-order logic. *Log. Methods Comput. Sci.*, 15(2), 2019. doi:10.23638/LMCS-15(2:19)2019.

29 J. Köbler and O. Verbitsky. From invariants to canonization in parallel. In *Proceedings of the 3rd International Computer Science Symposium in Russia (CSR '08)*, volume 5010 of *Lecture Notes in Computer Science*, pages 216–227. Springer, 2008. doi:10.1007/978-3-540-79709-8_23.

30 M. Lichter, I. Ponomarenko, and P. Schweitzer. Walk refinement, walk logic, and the iteration number of the Weisfeiler-Leman algorithm. In *Proceedings of the 34th Annual ACM/IEEE Symposium on Logic in Computer Science (LICS '19)*, pages 1–13. IEEE, 2019. doi:10.1109/LICS.2019.8785694.

31 B. D. McKay. Practical graph isomorphism. *Congressus Numerantium*, 30:45–87, 1981.

32 B. D. McKay and A. Piperno. Practical graph isomorphism, II. *J. Symb. Comput.*, 60:94–112, 2014. doi:10.1016/j.jsc.2013.09.003.

33 C. Morris, M. Ritzert, M. Fey, W. Hamilton, J. E. Lenssen, G. Rattan, and M. Grohe. Weisfeiler and Leman go neural: Higher-order graph neural networks. In *Proceedings of the 33rd AAAI Conference on Artificial Intelligence*, 2019. doi:10.1609/aaai.v33i01.33014602.

34 N. Shervashidze, P. Schweitzer, E. J. van Leeuwen, K. Mehlhorn, and K. M. Borgwardt. Weisfeiler-Lehman graph kernels. *Journal of Machine Learning Research*, 12:2539–2561, 2011.

35 W. T. Tutte. *Graph Theory*. Addison-Wesley, 1984.

36 O. Verbitsky. Planar graphs: Logical complexity and parallel isomorphism tests. In *Proceedings of the 24th Annual Symposium on Theoretical Aspects of Computer Science (STACS '07)*, pages 682–693, 2007. doi:10.1007/978-3-540-70918-3_58.

37 B. Weisfeiler and A. Leman. The reduction of a graph to canonical form and the algebra which appears therein. *NTI, Series 2*, 1968. English translation by G. Ryabov available at https://www.iti.zcu.cz/wl2018/pdf/wl_paper_translation.pdf.

38 H. Whitney. Congruent graphs and the connectivity of graphs. *American Journal of Mathematics*, 54:150–168, 1932.

39 K. Xu, W. Hu, J. Leskovec, and S. Jegelka. How powerful are graph neural networks? In *Proceedings of the 7th International Conference on Learning Representations (ICLR '19)*, 2019.

Kernelization, Proof Complexity and Social Choice

Gabriel Istrate ✉
West University of Timişoara, Romania

Cosmin Bonchiş
West University of Timişoara, Romania

Adrian Crăciun
West University of Timişoara, Romania

Abstract

We display an application of the notions of kernelization and data reduction from parameterized complexity to proof complexity: Specifically, we show that the existence of data reduction rules for a parameterized problem having (a). a small-length reduction chain, and (b). small-size (extended) Frege proofs certifying the soundness of reduction steps implies the existence of subexponential size (extended) Frege proofs for propositional formalizations of the given problem.

We apply our result to infer the existence of subexponential Frege and extended Frege proofs for a variety of problems. Improving earlier results of Aisenberg et al. (ICALP 2015), we show that propositional formulas expressing (a stronger form of) the Kneser-Lovász Theorem have quasipolynomial size Frege proofs for each constant value of the parameter k.

Another notable application of our framework is to impossibility results in computational social choice: we show that, for any fixed number of agents, propositional translations of the Arrow and Gibbard-Satterthwaite impossibility theorems have subexponential size Frege proofs.

2012 ACM Subject Classification Theory of computation → Proof complexity

Keywords and phrases Kernelization, Frege proofs, Kneser-Lovász Theorem, Arrow's theorem, Gibbard-Satterthwaite theorem

Digital Object Identifier 10.4230/LIPIcs.ICALP.2021.135

Category Track B: Automata, Logic, Semantics, and Theory of Programming

Related Version *Full Version*: `https://arxiv.org/abs/2104.13681`

1 Introduction

The central task of *proof complexity* [10, 30] is that of understanding (and distinguishing) the relative power of various propositional proof systems. Proving lower bounds for stronger and stronger proof systems might (in principle) be a way to eventually confirm the various conjectures of computational complexity. Yet we are far from being able to prove exponential lower bounds for some concrete problems in strong proof systems.

One of the most important open problems in this area, explicitly raised by Bonet, Buss and Pitassi [6] is that of separating the complexity of Frege proof systems ("textbook style propositional proofs") from that of extended Frege proof systems (which in addition can introduce new variables as substitutes for arbitrary propositional formulas). That is, we would like to find explicit classes of propositional formulas that have extended Frege proofs of polynomial size but have exponential lower bounds on the size of the shortest Frege proofs.

Many classes of problems that are candidates for separating the two systems have been proposed, e.g. statements based on linear algebra [24,36], propositional encodings of the Paris-Harrington independence results [11], Ramsey's theorem [31], central theorems from extremal combinatorics [1, 34], or the Kneser-Lovász formula from combinatorial topology [2, 27]).

So far most of the proposed examples have turned out to have sub-exponential Frege proofs (only a couple of candidate formula classes for the purported separation have been advanced, such as local improvement principles [28], or truncations of the octahedral Tucker lemma [2]). On the other hand many of the tractability results listed above have been obtained using techniques that are highly problem-specific, with relatively little transferability to more general classes of formulas. The existence of such general methods would be highly desirable: such general results could guide the search for examples witnessing the desired separation by pointing to structural properties one needs to avoid in order to construct them.

The purpose of this paper is to present a more general approach for proving sub-exponential upper bounds for the Frege and extended Frege proof complexity of some classes of propositional formulas. We point out that concepts from the theory of parameterized complexity [18], specifically those of *data reduction* and *kernelization* [19] may be relevant to proof complexity as well[1]. We give a metatheorem that translates a data reduction for the original problem whose soundness can be witnessed by polynomial size (extended) Frege proofs into subexponential proofs for the corresponding propositional translation of the original problem. The exact size of these proofs is controlled by three factors: the length of the data reduction chain, the nature and size of the proofs witnessing the soundness of the reduction rules, and the size of proofs of unsatisfiability for the formulas in the kernel.

We give several applications of our metatheorem. The use of kernelization techniques does not only allow to tackle the complexity of new problems, but also to improve existing results: In [2] it was shown that propositional formulas $Kneser_{n,k}$ expressing a principle from topological combinatorics known as *the Kneser-Lovász theorem* have, for every fixed value of parameter k, quasipolynomial size Frege proofs. We improve this result by showing quasipolynomial upper bounds for a principle stronger than the Kneser-Lovász theorem, known as Schrijver's theorem. Other applications of our metatheorem concern several (mostly graph-theoretic) problems whose kernelization had previously been studied in the theory of parameterized algorithms. The problems we study are well-known examples satisfying two conditions: First, their negative instances have natural formulations as unsatisfiable CNF formulas. Second, they have efficient kernelizations, often with a small kernel. The problems we have chosen illustrate an important point: *several techniques used in the literature to prove the existence of a kernelization can often be efficiently simulated by (extended) Frege proofs.* Perhaps the most interesting set of applications of our general metatheorem comes, however, from the theory of computational social choice [7]: as it was recently observed, various propositional formalizations of impossibility principles in the theory of computational social choice have computer-assisted proofs that reduce the task of mathematically proving these theorems to the verification of a finite number of cases of unsatisfiability of propositional formulas (using SAT solvers; see [22] for a fairly recent overview)[2]. **We obtain subexponential upper bounds on the complexity of Frege proofs for propositional formulations of Arrow's theorem and the Gibbard-Satterthwaite theorem: quasipolynomial in general, polynomial for a fixed number of agents.**

[1] This is not the first time a connection between parameterized complexity and proof complexity was made; see e.g. [5, 16]. However, our concerns are rather different.

[2] A similar phenomenon had been independently uncovered for the Kneser-Lovász theorem [2].

2 Preliminaries

We assume basic familiarity with concepts from three distinct areas: proof complexity, parameterized algorithms and computational social choice. We refer the reader to [7, 19, 30] for book-length treatments of these topics. Nevertheless, for purposes of readability we review a couple of relevant notions in the sequel:

▶ **Definition 1.** *A* Frege proof system *is a sound and complete propositional proof systems having a finite number of axioms and inference rules. An* extended Frege proof *augments Frege proofs by allowing new variables to substitute complex formulas.*

All Frege proof systems are equivalent up to polynomial transformations [15]. Therefore, for concreteness, we will employ a standard "textbook proof style" system having *modus ponens* as the unique inference rule. In both cases we measure the length of a proof by the number of steps in it. Thus the effect of the extension rule in extended Frege proofs is reducing proof length.

We use the shorthand $[m]$ for the set $\{1, 2, \ldots, m\}$, $[i : j]$ for $\{i, i+1, \ldots, j\}$, and write $A \cong B$ when sets A, B have the same cardinality. Function $f(\cdot)$ is called *quasipolynomial* if there exists $k > 0$ such that $f(n) = O(2^{O(\log^k(n))})$. We will need the following simple

▶ **Lemma 2.** *Suppose C is a CNF formula and $Z_1, \ldots, Z_m$ are literals s.t. $C \wedge (Z_1 \wedge Z_2 \wedge \ldots \wedge Z_m)$ is unsatisfiable, as witnessed by a resolution (Frege) proof of length k. Then one can derive from C clause $\overline{Z_1} \vee \overline{Z_2} \vee \ldots \vee \overline{Z_m}$ via a resolution (Frege) proof of size at most k.*

▶ **Definition 3** (Parameterized problem)**.** *Let Σ be an alphabet. L is a* parametrized problem *over Σ^* iff $L \subseteq \Sigma^* \times \mathbb{N}$. Define the support of L, by $supp(L) = \{x \in \Sigma^* | (\exists k \in \mathbb{N}) : (x, k) \in L\}$.*

Let L be a parameterized problem in co-NP. Let ϕ be a "canonical" reduction of L to $\overline{SAT}$. When ϕ is clear from the context, we identify L with the set of pairs $\phi(L) := \{(\phi(x, k), k) : (x, k) \in L\}$, slightly abusing notation, and writing L instead of $\phi(L)$.

▶ **Example 4** (Graph colorability)**.** Let $COL = \{(G, i) \mid \chi(G) \leq i\}$. We can encode instances (G, k) of COL as SAT instances $(\phi(G, k), k)$ by the reduction ϕ informally defined by:

- For $v \in V(G)$ and $1 \leq i \leq k$ define boolean $X_{v,i}$ =TRUE iff v is colored with color i.
- For every pair of distinct vertices $v, w \in G$ we define variable $Y_{v,w}$. The semantics is that $Y_{v,w} = TRUE$ means that v and w are connected by an edge. Thus, for all sets $\{v, w\}$ that correspond to an edge we add to $\phi(G, k)$ the unit clause $Y_{v,w}$. On the other hand, for sets $\{v, w\}$ that correspond to non-edges we add to $\phi(G, k)$ the unit clause $\overline{Y_{v,w}}$.
- For every $v \in V$ add $X_{v,1} \vee X_{v,2} \vee \ldots \vee X_{v,k}$. ("$v$ must be colored with one of colors 1 to k")
- For $v \in V$ and $1 \leq i < j \leq k$ add $\overline{X_{v,i}} \vee \overline{X_{v,j}}$. ("$v$ cannot be colored with both i and j")
- For every set $v, w \in V$ and $i \in 1 \ldots k$, add clause $\overline{Y_{v,w}} \vee \overline{X_{v,i}} \vee \overline{X_{w,i}}$. ("if v and w are connected then they cannot both be colored with color i")

▶ **Definition 5.** *Define graph, $Kn_{n,k}$, parameterized by an integer $k \geq 1$: The vertex set of $Kn_{n,k}$ is $\binom{n}{k}$, the set of subsets of $\{1, 2, \ldots, n\}$ with k elements. Two sets A, B represent adjacent vertices iff $A \cap B = \emptyset$.*

The Kneser-Lovász theorem (see e.g. [17]) is a statement about the chromatic number of $Kn_{n,k}$, equivalently restated as $\chi(Kn_{n,k}) > n - 2k + 1$[3]. It is expressed as a parameterized problem as follows: $L_{Kn} = \left\{(Kn^k_n, i) : n \geq 2k > 1,\ i \leq n - 2k + 1\right\}$. Note that $L_{Kn} \subseteq$

[3] actually $\chi(Kn_{n,k}) = n - 2k + 2$. However, the existence of a $(n - 2k + 2)$-coloring is easy [17].

$\overline{COL}$, hence we can use the translation from Example 4 to canonically translate L_{Kn} as a set of unsatisfiable propositional formulas. Similar propositional translations of other constraint satisfaction problems appear e.g. in [25, 33].

The next problem is just the graph coloring problem, but with a different parameterization:

▶ **Definition 6.** *An instance of the Dual Coloring problem is a pair (G, k), where G is a graph with n vertices and k is an integer. To decide: is $\chi(G) \leq n - k$? That is, let $DualCol = \{(G, k) : \chi(G) \leq n - k\}$. We have $(G, k) \in DualCOL \Leftrightarrow (G, n - k) \in COL$. For this reason the translation of $\overline{DualCOL}$ into $\overline{SAT}$ modifies the one from $\overline{COL}$ to $\overline{SAT}$ in Example 4 in an obivious way.*

Given graph G, a *vertex cover in* G is a set $S \subseteq V(G)$ such that for every edge $e = (v, w)$, $v \in S$ or $w \in S$. We denote by $vc(G)$ the size of the smallest vertex cover of G.

▶ **Example 7** (Vertex Cover). Let $\overline{VC} = \{(G, i) \mid i < vc(G)\}$ be the set of unsatisfiable instances of Vertex Cover. We can encode (negative) instances (G, k) of VC as instances $\phi(G, k)$ of SAT by the reduction ϕ informally defined as follows:

- For every $v \neq w \in V$, $(v, w) \in E$ add new unit clause $Y_{v,w}$ to the formula. For $(v, w) \notin E$ add new unit clause $\overline{Y_{v,w}}$, to the formula.
- For $v \in V(G)$ and $i \in 1 \dots k$ define boolean variable $X_{v,i}$ with the informal semantics $X_{v,i}$ is TRUE when vertex v is the i'th vertex in a vertex cover of size k. To encode this semantics add to the formula, for every $v \in V$ and $i \in 1, \dots, k$, clause $\overline{X_{v,i}} \vee (\bigvee_{w \neq v} Y_{v,w})$. This ensures that if v is chosen in the vertex cover then it covers some edge (v, w). With some extra technical complications one can do away with adding these clauses.
- For every $i = 1, \dots, k$ we add to the formula clause $\bigvee_{v \in V} X_{v,i}$.
- For every $v \neq w \in V$ and $1 \leq i \leq k$ we add to the formula clause $\overline{X_{v,i}} \vee \overline{X_{w,i}}$.
- For every $v \in V$ and $1 \leq i < j \leq k$ we add to the formula clause $\overline{X_{v,i}} \vee \overline{X_{v,j}}$.
- For $v \neq w \in V$ add to the formula clause $\overline{Y_{v,w}} \vee X_{v,1} \vee \ldots \vee X_{v,k} \vee X_{w,1} \vee \ldots \vee X_{w,k}$.

▶ **Definition 8** (Kernelization). *Let L be a parametrized problem. A* kernelization algorithm *(or, shortly, kernelization) Ker for the problem L is an algorithm that works as follows: on input (x, k), Ker outputs (in time polynomial in $|(x, k)|$) a pair (x', k'), such that the following are true: $(x, k) \in L$ iff $(x', k') \in L$, and $|x'|, k' \leq g(k)$, where g is a computable function. Pair (x', k') is called* the kernel of (x, k), *while $g(k)$ is called* the size of the kernel.

One can convert a kernelization into an algorithm by solving kernel instances by other means (e.g. brute force). A kernelization is often the reflexive, transitive closure of a finite set of *data reduction rules*: we apply the rules as long as possible, until we are left with an instance, the kernel, to which no rule can be applied anymore.

▶ **Definition 9** (Data reduction rule). *Let L be a parameterized problem. A* ***data reduction rule for L*** *is an algorithm A that maps (in time polynomial in $|x| + k$) an instance (x, k) of L to an instance (x', k') such that $(x, k) \in L$ iff $(x', k') \in L$ (we say that the two instances are* equivalent*, or that the reduction rule is* safe*), and $|x'| \leq |x|$. In practice, a data reduction rule may be well-defined only for $|x| \geq f(k)$, for some function $f(\cdot)$, as we can simply extend it to smaller instances (x, k) by defining $A(x, k) = (x, k)$. All kernelizations in this paper have this nature, and we will assume this to be true for all the results we give in the sequel.*

▶ **Definition 10** (Data reduction chain). *Given parameterized problem L kernelizable via data reductions $(A_1, A_2, \dots, A_r)$, a* ***data reduction chain*** *for instance (x, k) of L is a sequence $(x_0, k_0), (x_1, k_1), \dots, (x_m, k_m)$, where $(x_0, k_0) = (x, k)$, $A_t(x_m, k_m) = (x_m, k_m)$, for all $t = 1, \dots r$ and, for all $i = 1, \dots, m$ there exists $j \in 1, \dots, r$ such that $(x_i, k_i) = A_j(x_{i-1}, k_{i-1})$.*

▶ **Example 11** (Data reduction for Kneser instances:). Reductions $(Kn^2_n, a) \to (Kn^2_{n-1}, a-1)$ and $(Kn^3_n, a) \to (Kn^3_{n-1}, a-1)$ were used in [27] to give polynomial size extended Frege upper bounds for Kneser formulas for $k = 2, 3$.

- For $k \geq 2$ there exists $N(k) \leq k^4$ such that for $n > N(k)$ $(\mathcal{K}^k_n, a) \to (\mathcal{K}^k_{n-1}, a-1)$. This was used in [2] to give polynomial size extended Frege upper bounds for Kneser formulas.
- For $k \geq 2$ there exists $N(k) \leq k^4$ such that for $n > N(k)$ $(Kn^k_n, a) \to (Kn^k_{n-\frac{n}{2k}}, a - \frac{n}{2k})$. This was used in [2] to give quasipolynomial size Frege proofs for Kneser formulas.

Our results will work by formalizing *data reductions* which solve the decision for an instance X_1 of a parameterized problem by reducing it to a smaller instance X_2 of the same problem. This will map, propositionally, to witnessing the soundness of an implication $\Phi_1 \vdash \Phi_2$. However, to make Φ_2 a propositional tranlation of an instance of the same problem, we will need to use variable substitutions $Y = \Xi[X]$. That is, the reduction we witness is actually $\Phi_1[X] \vdash \Phi_2[Y]$. When we will refer to the proof complexity of witnessing an implication $\Phi_1 \vdash \Phi_2$ using (extended) Frege proofs, what we mean is that one can derive the clauses of $\Phi_2[Y]$ using the clauses of $\Phi_1[X]$ as axioms. Since they use substitutions, these are extended Frege proofs. To convert them into Frege proofs one needs to unwind the definitions of newly defined variables.

▶ **Definition 12.** *A **crown decomposition of a graph G** (see e.g. Fig. 1 (b).) is a decomposition of $V(G)$ into three subsets C, H, R, $C \neq \emptyset$ such that (1). C is an independent set. (2). No vertex in C is adjacent to a vertex in R. (3). There exists a matching of H in C, i.e. a set of disjoint edges covering H with the other endpoint in C.*

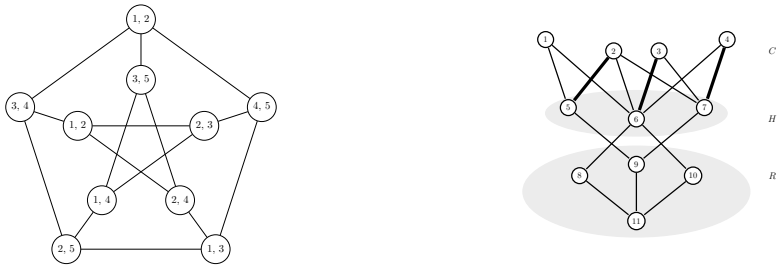


Figure 1 (a). The Kneser graph $Kn_{5,2}$. (b). A crown decomposition of a graph.

Given a set S and $T \subseteq S$, we will denote, as in [37,38], by S_{-T} the set $S \setminus T$. We will also write S_{-a} instead of $S_{-\{a\}}$. When $S = [m]$, of course $[m]_{-T} \cong [m - |T|]$ for every $T \subseteq [m]$.

▶ **Definition 13.** *Given a set of m objects, identified with the set $[m]$, a* preference profile *is a linear ordering of $[m]$, i.e. a permutation $\pi \in S_m$. Given $a, b \in [m]$ we say that a* is preferred to *b (written $a <_\pi b$) iff $\pi^{-1}(a) < \pi^{-1}(b)$. **Note that preferred objects are lower in the ordering.** We denote by $top(\pi)$ the object $\pi^{-1}(1)$, i.e. the object that is preferred in π to all others. Given a preference profile π and $T \subseteq [m]$, denote by π_{-T} the restriction of π to $[m]_{-T}$, and by π^{+T} the preference profile derived from π by making all elements $a \in T$ less preferred than any other $b \in [m]$ (with an arbitrary fixed order among them, e.g. the order induced on $[m]$ by the identical permutation).*

▶ **Definition 14.** *Given a set of m objects, identified with the set $[m]$ and a set of n agents, a **social choice function** (SCF) is a mapping $s : S^n_m \to Z$. Z is a set equal to S_m (for Arrow's theorem) and to $[m]$ (for the Gibbard-Satterthwaite theorem). A SCF is **dictatorial***

if there exists $i \in [m]$ such that for all $R_1, R_2, \ldots, R_n \in S_m$, $s(R_1, R_2, \ldots, R_n) = R_i$ ($s(R_1, R_2, \ldots, R_n) = top(R_i)$ for the Gibbard-Satterthwaite theorem). A SCF is ***unanimous*** *if whenever a is preferred to b in all profiles $R_1, R_2, \ldots, R_n$ then a is preferred to b in profile $s(R_1, R_2, \ldots, R_n)$. SCF s satisfies the* ***independence of irrelevant alternatives (IIA) axiom*** *(for Arrow's theorem) if whenever $a, b \in [m]$ are two different objects and $(R_1, R_2, \ldots, R_n) \in S_m^n$ and $(R'_1, R'_2, \ldots, R'_n) \in S_m^n$ are two vectors of preference profiles such that, for all $i = 1, \ldots m$, R_i and R'_i agree in their relative preference of a or b, then $s(R_1, R_2, \ldots, R_n)$ and $s(R'_1, R'_2, \ldots, R'_n)$ agree in their relative preference of a or b. A SCF is* ***onto*** *iff it is onto as a function. Finally, for every pair (R, o), $R = (R_1, \ldots, R_n)$ and player $1 \leq i \leq m$, denote by $pr(i, o, R)$ the set of objects o' s.t. $R_i^{-1}(o) \leq R_i^{-1}(o')$ (i.e. i weakly prefers o to o' in R_i). A SCF s is* ***strategyproof*** *(for the Gibbard-Satterthwaite theorem) iff, for every strategy profile R, if o is the outcome of preference profile R then i cannot misrepresent its preferences as $\pi \in S_m, \pi \neq R_i$ so that the social choice for the resulting profile $s(i, R, \pi)$ is an o' that i strictly prefers to o.*

Given an *SCF* $W : [m]^n \to Z$ and $B \subseteq [m]$ we define function $W_{-B} : [m]^n_{-B} \to Z$ to be defined as follows: $W_{-B}(R_1, R_2, \ldots, R_n) = W(R_1^{+B}, R_2^{+B}, \ldots, R_n^{+B})_{-B}$. In other words, we extend profiles $R_1, R_2, \ldots, R_n$ by making objects in B less preferred than all other objects, apply W on the resulting profiles, then drop objects from B from the result.

3 Main (Meta)Theorem and Applications

In the next definition we formalize the complexity of simulating data reduction steps by (extended) Frege proofs. Clearly, we want to encode the scenario where each such step can be simulated by efficient proofs. Our main result will allow a slightly more general setting, where the safety of each reduction step can be established by a "case by case argument with a limited number of cases". This will lead not to a chain but to a *tree* of logical reductions:

▶ **Definition 15.** *Given reduction rule $\mathcal{A}$ for problem L and function $h(\cdot)$,* ***the soundness of $\mathcal{A}$ has (extended) Frege proofs of size $h(\cdot)$*** *iff there is an integer $R \geq 1$ s.t. for every $(x, k) \notin L$ and every step $(x_i, k_i) \to (x_{i+1}, k_{i+1})$ in the reduction chain the following are true:*

- *There exists $r'_i \leq R$, tautology $\Xi_i := \bigvee_{t=1}^{r'_i} \Xi_{i,t}$ and formulas $\eta_{i,1}, \ldots \eta_{i,r'_i}$ isomorphic (up to a variable renaming) to $\Phi(x_{i+1}, k_{i+1})$ s.t. for $t = 1, \ldots, r'_i$, $\Phi(x_i, k_i) \wedge \Xi_{i,t} \vdash \eta_{i,t}$.*
- *Proving the soundness of Ξ_i and of all reductions $\Phi(x_i, k_i) \wedge \Xi_{i,t} \vdash \eta_{i,t}$ can be accomplished by (extended) Frege proofs of* ***total*** *size at most $h(|\Phi(x, k)|)$.*

In other words, the kernelization may encode a case-by-case construction with a constant number, at most R, of cases. Each case will lead to a different reduced formula. The consistency of all those reductions is witnessed by proofs with a common polynomial upper bound. Given this definition, our main (meta)theorem is:

▶ **Theorem 16.** *Let L be a parameterized problem that is kernelizable via a finite number of data reduction rules $(A_1, A_2, \ldots, A_r)$ with kernel size $g(\cdot)$.*

1. *Assume that negative instance (x, k) of L has a data reduction chains of length $C(x, k)$, and that the soundness of each reduction rule $A_1, A_2, \ldots, A_r$ can be witnessed using extended Frege proofs of size at most $h(|\Phi(x, k)|)$, for some function $h(\cdot)$. Then L has extended Frege proofs of size $O((\sum_{i=0}^{C(x,k)} R^i)[h(|\Phi(x, k)|) + 2^{O(poly(g(k)))}])$. In particular, if $R = 1$ and for every fixed k we have $C(x, k) = O(poly(|\Phi(x, k)|))$ then, for every fixed k, negative instances $\Phi(x, k)$ of L have extended Frege proofs of size polynomial in $|\Phi(x, k)|$.*

2. *Assume that negative instances (x,k) of L have data reduction chains of length $C(x,k) = O(1)$ ($O(log(|\Phi(x,k)|))$, respectively), where the constant may depend on k, and that the safety of each reduction $\Phi(x_i,k_i) \vdash \Phi(x_{i+1},k_{i+1})$ is witnessed by Frege proofs of size $\leq p(|\Phi(x,k)|)$, for some fixed polynomial $p(\cdot)$ Then for every fixed k, negative instances $(\Phi(x,k),k)$ of L have Frege proofs of size polynomial (quasipolynomial) in $|\Phi(x,k)|$.*

Proof.

1. Given an instance (x,k) of L, and data reduction chain $(x,k) = (x_0,k_0),(x_1,k_1),,\ldots(x_m,k_m)$, an extended Frege proof for $\Phi(x,k)$ is obtained by concatenating the proofs for statements $\Phi(x_{i-1},k_{i-1}) \vdash \Phi(x_i,k_i)$ with an extended Frege proof of the kernel instance. There is one complication, though, induced by the fact that we allow at most R cases in the reduction: the reduction chain maps to a **tree** of propositional proofs, since for each node $\Phi(x_i,k_i)$ we have $r_i' \leq R$ children $\Xi_{i,t}$, all isomorphic to $\Phi(x_{i+1},k_{i+1})$ (but different). The total number of nodes in this tree is at most $\sum_{t=0}^{C(x,k)} R^t$.

   Then the whole chain of reductions from $\Phi(x,k)$ to $\Phi(x_m,k_m)$ can be proved to be sound by proofs of length $(\sum_{t=0}^{C(x,k)} R^t) \cdot h(|\Phi(x,k)|)$. There are at most $R^{C(x,k)}$ copies of the kernel instance. Each of them can be proved (in brute force) in size $O(2^{|\Phi(x_m,k_m)|}) = O(2^{poly(g(k))})$, since $(x_m,k_m) \in ker(L)$ and any unsatisfiable formula Ξ with n variables has Frege proofs of size $O(2^n)$.

   The length of the total proof is thus $O(\sum_{t=0}^{C(x,k)} R^t \cdot [h(|\Phi(x,k)|) + 2^{poly(g(k))})]$. We infer the desired result when $C(x,k) = O(poly(|\Phi(x,k)|))$.

2. We unwind the substitions implicit in the extended Frege proofs. For $R = 1$ (i.e. a reduction chain), arguing that the blow-up due to making substitutions is quasipolynomial as long as the chain length is logarithmic is identical to similar arguments made in [9], [2] for other problems, and we omit further details.

   In our case the complication arises since we no longer have a chain but a tree. However, we can upper bound the complexity of Frege proofs by $R^{C(x,k)}$ times the complexity of a single chain (a root-to-leaf path in this tree). As long as $C(x,k) = O(\log(|\Phi(x,k)|))$, the term $R^{C(x,k)}$ has a magnitude polynomial in $|\Phi(x,k)|$. Multiplying this polynomial by the quasipolynomial complexity of each chain still yields a proof of complexity quasipolynomial in $|\Phi(x,k)|$. ◀

▶ **Observation 17.** *There is an important uniformity aspect of kernelization that we haven't used in the preceding proof:* the fact that data reductions are specified by polynomial time algorithms. *This issue will be important in applying the above result: often the existence of a data reduction is proved by an algorithm whose soundness (for all instances) would be rather cumbersome to simulate in propositional proofs. This is the case when results involve general techniques for developing kernelizations, such as the Crown Decomposition Lemma or the Sunflower Lemma. As long as we do not insist, however, on* actually generating *the proof, but merely on* proving its existence*, we can get away with proving the soundness of individual instances. That is, if we can prove the soundness of* an individual application of a propositional reduction rule*, $\Phi(x,k) \vdash \Phi(x',k')$, taking for granted the existence/definition of (x',k'), we can prove the existence of efficient proofs,* ***without actually having to generate a propositional proof of the soundness of the reduction techniques.***

Next we highlight some applications of our main (meta)theorem:

3.1 Proof Complexity of (Dual) Coloring

▶ **Theorem 18.** *There exists a kernelization that reduces instances (G,k) of DUALCOL to a kernel of size at most $3k-2$. The length of reduction chain in this kernelization is $O(k)$. The soundness of each reduction step can be witnessed by polynomial size Frege proofs. Hence, for every fixed k, negative instances (G,k) of DUALCOL have Frege proofs of size polynomial in $|\Phi_{G,k}|$.*

Proof. The kernelization is a variant of the classical one from the parameterized complexity literature, based on *crown decompositions* (Definition 12). It consists of three data reductions:

(a). Let $All(G)$ be the set of vertices v adjacent to all other vertices in G. If $All(G) \neq \emptyset$ then $(G,k) \in DualCol \Leftrightarrow (G \setminus All(G), k - |All(G)|) \in DualCol$.

(b). If $All(G) = \emptyset$ but $\overline{G}$ has a matching of size k, $x_1, y_1, \ldots, x_k, y_k$, with x_i being matched to y_i for $i = 1, \ldots, k$, then $(G,k) \in DualCol$ (so reduce it to an arbitrary positive instance).

(c). Assume that rules (a),(b) do not apply. Let (C,H,R) be a crown decomposition of the graph $\overline{G}$. Reduce (G,k) to (G',k'), by deleting $H \cup C$ from G and $k' = k - |H|$.

Without loss of generality, we will only apply rule (c). to crown decompositions where $|H| \neq \emptyset$ and all nodes in C are matched to some node in H. This is possible for the following reason: if $|H| \neq \emptyset$ and the original crown decomposition had other vertices in C, just move them to R. If, on the other hand $|H| = \emptyset$ then all vertices in C would be connected to all vertices in $C \cup R$, hence to all vertices of G. But this cannot happen, since the case $All(G) \neq \emptyset$ is covered by the first data reduction rule.

The Crown Decomposition Lemma (Lemma 4.5 of [19]) makes sure that at least one of reduction rules (a),(b),(c) applies to every graph with more than $3k-2$ vertices.

The safety of reduction rule (c) can be informally justified as follows: since vertices in a crown decomposition of $\overline{G}$ are matched in a matching m, vertices $v \in H$ and $m(v)$ is C are not connected in G, hence they can be colored with the same color. At the same time, $m(v)$ is connected in G to all the vertices of G', hence must assume a color different from all the colors of vertices of G'. Also $m(v_1)$ and $m(v_2)$ are connected, so must assume distinct colors. In conclusion, vertices of C must use $|C|$ different colors, and G is $n-k$ colorable if and only if G' is $n-k-|C|$ colorable. But $|G'| = n - 2|C|$, so G is $n-k$ colorable if and only if G' is $|G'| - (k - |C|)$ colorable.

Rule (b). does not apply to unsatisfiable instances of DualCol. Hence we have to argue about the size of Frege proofs witnessing the soundness of rules (a) and (c), namely: Let $\Phi_{v,1}[\overline{Y}]$ be the formula $\wedge_{w \neq v} Y_{v,w}$ (informally, $v \in All(G)$). We need to provide proofs that witness that

$$\Phi(G,k) \wedge \Phi_{v,1}[\overline{Y}] \vdash \Phi(G \setminus \{v\}, k-1), \text{ and}$$

$$\Phi(G,k) \vdash \Phi(G',k').$$

For the first implication, define new variables $Z_{w,i}$ via the substitution, for $w \neq v \in V(G)$, $Z_{w,i'} \leftrightarrow X_{w,i} \wedge X_{v,l}$, where $i' = i$ for $i < l$, $i' = i-1$ for $i > l$.

We start by deriving, by resolving unit literals $Y_{v,w}$ (which are part of the formula), for all $w \neq v \in V(G)$ and i, clauses $\overline{X_{v,i}} \vee \overline{X_{w,i}}$. Then we derive, for every $w \neq v \in V(G)$ and i, clauses $\overline{X_{v,i}} \vee (\vee_{j \neq i} X_{w,j})$. This is done by resolving $(\vee_{j=1}^{k} X_{w,j})$ and $\overline{X_{v,i}} \vee \overline{X_{w,i}}$. We then derive clauses $\overline{X_{v,i}} \vee (\vee_{i'=1}^{k-1} Z_{w,i'})$. By resolving all these clauses against $\vee_{i=1}^{k} X_{v,i}$ we derive $(\vee_{i'=1}^{k-1} Z_{w,i'})$. Similar tricks allow deriving clauses $\overline{Z_{w,i}} \vee \overline{Z_{w,j}}$ and $\overline{Y_{v,w}} \vee \overline{Z_{v,i}} \vee \overline{Z_{w,i}}$ from the corresponding clauses in the X variables.

As for the second reduction rule, intuitively we want to encode the fact that if a vertex $v \in G'$ is colored with color i in G, then coloring it with color $i - less(v)$, where $less(v)$ is the number of nodes in H colored with a color smaller than i, yields a legal coloring of G'. This is true since all nodes in H must get colors (in G) different from all colors in G'.

For an arbitrary vertex $v \in G'$, let $less(v)$ be the number of nodes $w \in C$ (here C refers to the class of the crown decomposition of $\overline{G}$) such that $col(w) < col(v)$. One can compute the binary representation of number $less(v)$ using Frege proofs as follows: we create a boolean variable $T_{v,w}$ which will be true iff $col(w) < col(v)$. One can compute $T_{v,w}$ as

$$T_{v,w} := \bigvee_{i<j} X_{w,i} \wedge X_{v,j}.$$

Now we simply use the predicate $COUNT(T_{v,w})_{w \in C}$ to compute the binary representation of $less(v)$. Here COUNT is the Buss counting predicate [8]. We will also derive the following formulas:

$$\overline{X_{v,i}} \vee \bigvee_{t=1}^{i} [less[v] = t] \tag{1}$$

To accomplish that, we use the pigeonhole principle PHP_i^{i+1} to prove that

$$\overline{X_{v,i}} \vee [COUNT((X_{w,j})_{w \in C, j<i}) \leq i] \tag{2}$$

Indeed, assuming $X_{v,i} = TRUE$ we can derive any disjunction of length $i+1$ consisting of literals of type $\overline{X_{w,j}}$, with $w \in C$, $j < i$. This is because for all $w_1 \neq w_2 \in C$, $k_1 \neq k_2$ $\overline{X_{w_1,k_1}} \vee \overline{X_{w_2,k_1}}$ and $\overline{X_{w_1,k_1}} \vee \overline{X_{w_1,k_2}}$ are clauses of $\Phi(G,k)$. By Lemma 2 we can derive equation (2). Next, simple arguments along the lines of [8] establishes the equivalence between formulas $U \leq i$ and $\bigvee_{k=1}^{i}[U = k]$. Here U is a bit vector of appropriate length to represent i. We use (2) and this to derive (1).

Now, for every $v \in V(G') = R$ we define a new variable $Z_{v,i}$, designed to be true iff the color of v in the induced coloring on G' is j. We will enforce this by making the substitutions

$$Z_{v,j} := \bigvee_{j=1}^{i} X_{v,i} \wedge [less(v) = i - j] \tag{3}$$

First note that Z respects the color classes of G: if v_1, v_2 have the same color in G then they have the same color in G'. Furthermore, the substitution does not collapse two different color classes of G into a single color class in G': it simply relabels the colors of vertices in G' with elements of $1, 2, \ldots, k'$. Therefore, if $Z_{v_1,j} = Z_{v_2,j} = TRUE$ then there exists an unique i_0 such that $X_{v_1,i_0} = X_{v_2,i_0} = TRUE$.

We need to derive clauses $\bigvee_{t=1}^{k'} Z_{v,t}$ as well as, for $vw \in E(G')$, $\overline{Z_{v,j}} \vee \overline{Z_{w,j}}$. Deriving the first type of clauses is easy: we use formulas (1) and $X_{v,1} \vee X_{v,2} \vee \ldots \vee X_{v,k}$.

As for the second one, nota that all clauses $\overline{X_{v,i}} \vee \overline{X_{w,i}}$ are part of $\Phi(G,k)$. Given the observation we made above and this fact, assuming $Z_{v,j} = Z_{w,j} = TRUE$ we can derive a contradiction. By Lemma 2 we can, therefore, derive (with the same complexity) clause $\overline{Z_{v,j}} \vee \overline{Z_{w,j}}$.

◀

3.2 Proof Complexity of Schrijver's Theorem

In this section we deal with the proof complexity of a stronger version of the Kneser-Lovász theorem known as Schrijver's Theorem [35]. This is a statement about the chromatic number of the so-called *stable Kneser graph* $SKn_{n,k}$, defined as follows:

▶ **Definition 19.** *Call a set* $A \subseteq \binom{n}{k}$ stable *if* A *does not contain two elements that are consecutive (we also consider* n *and* 1 *as consecutive). Denote the set of stable sets by* $\binom{n}{k}_{st}$. *The* stable Kneser graph $SKn_{n,k}$ *is the subgraph of* $Kn_{n,k}$ *induced by the set* $\binom{n}{k}_{st}$.

Schrijver's theorem asserts that the chromatic number of the stable Kneser graph $SKn_{n,k}$ is $n-2k+2$. We are, of course, interested mainly in the harder part of this result, the lower bound $\chi(SKn_{n,k}) > n-2k+1$. Since $SKn_{n,k}$ is the subset of the Kneser graph (see e.g. Figure 1, where the central star is the stable Kneser graph $SKn_{5,2}$), this strengthens the (harder part of the) Kneser-Lovász theorem. The propositional translation of Schrijver's theorem is immediate, and the resulting unsatisfiable formulas, that we will denote by $\text{Schrijver}_{n,k}$ are subformulas of formulas $Kneser_{n,k}$. We have the following:

▶ **Theorem 20.** *For every fixed* k, *formulas* $Schrijver_{n,k}$ *have Frege proofs of size quasipolynomial in* n.

For the (somewhat more involved) proof of this theorem, we refer the reader to [26].

3.3 Buss Meets Buss: the Proof Complexity of Vertex Cover

In this subsection we study the proof complexity of Vertex Cover, the "drosophila of parameterized complexity" [20]. We apply our result to a variation of the standard kernelization of VC (called in [20] *the Buss reduction*, hence the title of this subsection) to prove:

▶ **Theorem 21.** *Instances* (G,k) *of VC have a kernelization with a data reduction chain of length* $O(k)$ *to a kernel with at most* k^2 *vertices. The soundness of each step in this data reduction can be witnessed by Frege proofs of size polynomial in* $|\Phi(G,k)|$. *Hence, for every fixed* k *negative instances* $\Phi(G,k)$ *of VC have Frege proofs of size polynomial in* $|\Phi(G,k)|$.

Proof. Informally, we will use the following two data reduction rules:

(a). if G has a vertex v of degree larger than k then G has a VC of size $\leq k$ if and only if $G \setminus \{v\}$ has a VC of size $\leq k-1$. Indeed, v must be part of any VC of G of size $\leq k$.

(b). if $Isolated(G)$ denotes the set vertices v in G that are isolated then G has a VC of size $\leq k$ iff $G \setminus Isolated(G)$ has a VC of size $\leq k$.

The kernel of these two reduction rules, the set of instances (G,k) of VC such that none of the two rules applies is composed of graphs of at most k^2 vertices only [19].

To encode the soundness of these rules by polynomial-size Frege proofs we use the predicate $COUNT^n_k(x_1,x_2,\ldots,x_n)$ from [8]. Formula $COUNT^n_k(x_1,x_2,\ldots,x_n)$ is TRUE if and only if at least k of the variables $x_1,x_2,\ldots,x_n$ are true. For every fixed k, $COUNT^n_k$ can be computed by polynomial size Frege proofs.

We will define a sequence of formulas:

1. For $v \in V$, $\Phi_{v,1}(\overline{Y}) = COUNT^{n-1}_k((Y_{v,w})_{w \neq v \in V})$. Informally, $\Phi_{v,1}$ is true in graph G iff the degree of v is at least k.
2. For $v \in V$, $\Phi_{v,2}(\overline{X},\overline{Y}) = (\bigwedge_{i=1}^{k} \overline{X_{v,i}}) \wedge \Phi_{v,1}(\overline{Y}) \wedge \Phi_{VC}(G,k)[\overline{X},\overline{Y}]$.

For every neighbor w of v, by resolving $Y_{v,w}$ with clause $\overline{Y_{v,w}} \vee X_{v,1} \vee \ldots \vee X_{v,k} \vee X_{w,1} \vee \ldots \vee X_{w,k}$. of $\Phi_{v,2}$ we derive clause $X_{v,1} \vee \ldots \vee X_{v,k} \vee X_{w,1} \vee \ldots \vee X_{w,k}$. By resolving successively with $\overline{X_{v,1}},\ldots,\overline{X_{v,k}}$ we derive clause $X_{w,1} \vee \ldots \vee X_{w,k}$.

Formula $\bigwedge_{w \in N(v)} (X_{w,1} \vee \ldots \vee X_{w,k})$ is isomorphic to the Pigeonhole Principle $PHP^k_{|N(v)|}$ which has polynomial-size Frege refutations [8]. Plugging in this proof of this statement into our argument, we conclude that that the implication $\Phi_{v,2}(X, \overline{Y}) \vdash \Box$ can be witnessed by polynomial size Frege proofs, hence, by Lemma 2, so does the implication $\Phi_{v,1}(\overline{Y}) \wedge \Phi_{VC}(G,k)[\overline{X}, \overline{Y}] \vdash \bigvee_{i=1}^{k} X_{v,i}$.

As for the second reduction rule, it is just as easy: for every vertex $v \in V$ which is isolated and every $i = 1, \ldots, k$, we first derive by resolution (using negative clauses $\overline{Y_{v,w}}$ and clause $\overline{X_{v,i}} \vee (\bigvee_{w \neq v} Y_{v,w})$ unit clauses $\overline{X_{v,i}}$. We then use these clauses to resolve away every other occurrence of $X_{v,i}$ from the formula, obtaining a formula isomorphic to $\Phi(G \setminus Isolated(G), k)$. ◀

3.4 Proof Complexity of Edge Clique Cover

In this section we study the proof complexity of the following problem:

▶ **Definition 22** (Edge Clique Cover). *Given graph G and integer k, to decide is whether one can find sets of vertices $V_1, V_2, \ldots, V_k \subseteq V$ s.t. each V_i induces a clique, and for every edge $e = (v,w) \in E$ there exists $1 \leq i \leq k$ s.t. $v, w \in V_i$ ("each edge is covered by some clique"). We represent instance (G,k) of Edge Clique Cover by propositional formula $\Phi_{G,k}$ as follows:*

- *For every pair of distinct vertices $v, w \in V$ define a variable $Y_{v,w}$. For every edge $(v,w) \in E(G)$ add unit clause $Y_{v,w}$. For $(v,w) \notin E(G)$ add unit clause $\overline{Y_{v,w}}$.*
- *For $v \in V$ and $1 \leq i \leq k$ define boolean variable $X_{v,i} = TRUE$ iff $v \in V_i$.*
- *For $v, w \in V$ and $1 \leq i \leq k$ add $\overline{X_{v,i}} \vee \overline{X_{w,i}} \vee Y_{v,w}$ ("if $v, w \in V_i$ then $vw \in E(G)$") and $\overline{Y_{v,w}} \vee (\vee_{j=1}^{k}(X_{v,j} \wedge X_{w,j}))$. Of course, as written above the latter formula is not CNF, but it can be converted easily by expanding the last disjunction.*

The following is our result for the Edge Clique Cover problem. The main technical novelty is reducing the length of the data reduction chain (compared to the usual kernelization) from linear to logarithmic, so that we can get quasipolynomial-size Frege proofs:

▶ **Theorem 23.** *There exists a kernelization that reduces instances (G,k) of problem EDGE CLIQUE COVER with graph G having n vertices to a kernel with at most 2^k nodes. The length of the data reduction chain is $O(\log_{1+\frac{1}{2^k-1}}(n))$. The soundness of each reduction step can be witnessed by polynomial size Frege proofs. Consequently, for fixed k, negative instances (G,k) of EDGE CLIQUE COVER have extended Frege proofs of polynomial size and Frege proofs of quasipolynomial size in $|\Phi_{G,k}|$.*

Proof. We use the following data reduction rules:

(a). If $|Isolated(G)| \geq \frac{n}{2^k}$ then reduce (G,k) to $(G \setminus Isolated(G), k)$.

(b). If there exists a set $S \subseteq V$, $|S| \geq \frac{n}{2^k}$ such that vertices in S induce a clique in G, and for all $v, w \in S$ we have $N[v] = N[w]$, where $N[v]$ stands for *the closed neighborhood* of v, then reduce G to (G', k'), where G' is the graph obtained by identifying vertices v, w, and $k' = k$ whenever $N[v] = N[w] \neq \emptyset$, $k' = k - 1$, otherwise.

The soundness of the first reduction rule, $\Phi(G,k) \vdash \Phi(G \setminus Isolated(G), k)$ can be witnessed by efficient Frege proofs similar to those for the vertex cover problem.

As for the second rule, the formula

$$\Xi_S(G) := \wedge_{w,v \in S} \wedge_{r \in V} (Y_{v,r} \leftrightarrow Y_{w,r})$$

(where, of course, $A \leftrightarrow B$ can be equivalently rewritten as $(\overline{A} \vee B) \wedge (A \vee \overline{B})$) expresses the fact that $N[v] = N[w]$ for all $v, w \in S$. So we need to prove the soundness of the rule

$$\Phi(G,k) \wedge \Xi_S(G) \vdash \Phi(G',k'). \tag{4}$$

Without loss of generality we will only deal with the case $N[v] \neq \emptyset$ for all $v \in S$, (the other case, $N[v] = \emptyset$ for all $v \in S$, can be handled with minor modifications to this argument). By slightly abusing notation, we will denote by S the vertex of G' obtained by contraction. Let $s \in S$ be an arbitrary vertex.

We define substitutions: $Y'_{v,w} := Y_{v,w}$ for all $v, w \in G'$, $v, w \neq S$. If, say, $v = S$ we define $Y'_{S,w} := Y_{s,w}$. Also define $X'_{v,i} := X_{v,i}$ for $v \neq S$, $X'_{S,i} := X_{s,i}$. The substitution yields a formula isomorphic to $\Phi(G',k')$, and the proof of the safety is basically trivial.

To obtain the result note that the number of vertices goes down geometrically, by a ratio of $1 - \frac{1}{2^k}$ at each step.

▶ **Lemma 24.** *Rules (a). (b). are safe. Also, for every graph G with $n > 2^k$ vertices one of rules (a). (b). applies.*

Proof. Let G be a graph to which rules (a). (b). do not apply and which has an edge clique cover of size k. Consider an encoding $b(v)$ of every vertex v on k bits such that for every $v \in V$, $b(v)$ is a bit vector whose i'th bit is one iff v is a part of the i'th clique.

There must be a set of vertices $S \subset V$, $|S| \geq \frac{n}{2^k}$ such that for all $u, v \in S$, $b(u) = b(v) = b$, for some $b \in \{0,1\}^k$.

If $b = 0^k$ then, since every edge in G must be covered by one of the k cliques, it follows that every $v \in S$ is an isolated vertex. Hence rule (a). applies.

If, on the other hand $b \neq 0^k$, say $b_i \neq 0$, then every $v \in S$ must belong to the i'th clique. Hence S induces a clique in G. ◀

◀

3.5 Proof Complexity of the Hitting set problem

In the d-Hitting Set problem we are given an universe U and a family $\mathcal{A}$ of subsets of U, all of cardinality at most d, as well as an integer k. To decide is whether there exists a set $H \subseteq U$ containing at most k elements, such that H intersects every $P \in \mathcal{A}$.

A formalization of the d-Hitting set problem as an instance of SAT is obtained as follows:

▶ **Example 25.** Let $P = (U, \mathcal{A}, k)$ be an instance of d-Hitting set. Define formula Φ_P by:

- For $i \in U$, $j = 1, \ldots, k$ add variable $X_{i,j}$, TRUE iff i is the j'th chosen element.
- For $i \neq i' \in U$, $1 \leq j \leq k$ add clauses $\vee_{i \in U} X_{i,j}$ ("some i is the j'th chosen element") and $\overline{X_{i,j}} \vee \overline{X_{i',j}}$ ("at most one i can be the j'th chosen element").
- For $A \in \mathcal{A}$ add $(\vee_{i \in A}(\vee_{j=1,\ldots k} X_{i,j}))$ ("some element of A is among the k chosen elements")

Our result, which only guarantees polynomial size *extended* Frege proofs, is:

▶ **Theorem 26.** *There exists a kernelization mapping instances $(U, \mathcal{A}, k)$ of d-HittingSet with $|U| = n$ elements to a kernel with at most $d \cdot d! \cdot k^d$ sets (hence at most $d^2 \cdot d! \cdot k^d$ elements). The data reductions chains in this kernelization have length $O(n^d/k)$, and their soundness can be witnessed by polynomial-size Frege proofs. Hence for every fixed k, d, unsatisfiable instances $\Phi_{(U,\mathcal{A},k)}$ of d-HittingSet have extended Frege proofs of size $O(poly(|\Phi_{(U,\mathcal{A},k)}|))$.*

Proof. We employ the standard kernelization of d-Hitting set based on sunflowers [19]:

▶ **Definition 27.** *A* sunflower with k-petals and core Y *is a colection of sets $S_1, \ldots, S_k$, all different from Y, such that for $1 \leq i < j \leq k$, $S_i \cap S_j = Y$.*

We are going to propositionally encode the following informally stated data reduction rule: let $(U, \mathcal{A}, k)$ be an instance of the d-Hitting set such that $\mathcal{A}$ contains a sunflower $\mathcal{S} = \{S_1, S_2, \ldots, S_{k+1}\}$ of cardinality $k+1$ with core Y. We reduce $(U, \mathcal{A}, k)$ to the instance $(U', \mathcal{A}', k)$, where $\mathcal{A}' = (\mathcal{A} \setminus \mathcal{S}) \cup \{Y\}$ and $U' = \cup_{X \in \mathcal{A}'} X$. Indeed, consider a hitting set H for $(U, \mathcal{A}, k)$. By definition, H meets every element of $\mathcal{A} \setminus \mathcal{S}$. If H did not meet Y then it would have to meet each of the $k+1$ disjoint petals $S_j \setminus Y$. Hence $|H| \leq k$ iff H meets Y.

To simulate this argument propositionally, define for $i \in U'$ substitutions $X'_{i,j} := X_{i,j}$. We need to (a). derive clause $(\vee_{i \in Y}(\vee_{j=1,\ldots k} X_{i,j}))$ (b). for every $j = 1, \ldots, k$, derive clauses $\vee_{i \in U'} X_{i,j}$.

For the first clause we show that $\Phi(U, \mathcal{A}, k) \wedge (\wedge_{i \in Y}(\wedge_{j=1,\ldots k} \overline{X_{i,j}})) \vdash \emptyset$ and then we invoke Lemma 2. To do that we first derive, for $l = 1, \ldots, k+1$ (by resolving literals $\overline{X_{i,j}}$) clauses $(\vee_{i \in S_l \setminus Y}(\vee_{j=1,\ldots k} X_{i,j}))$.

Substituting each variable $X_{i,j}$, where $i \in S_j \setminus Y$ to a new variable $Y_{i,j}$ yields a formula isomorphic to PHP^k_{k+1} which has polynomial size Frege proofs [8]. Putting all these things together we get polynomial-size Frege proofs witnessing the soundness of one step of the data reduction.

The number of clauses drops at every reduction step by k, so the length of the data reduction chain is $O(n^d/k)$. ◀

4 Proof Complexity of principles in Computational Social Choice

A great number of applications come from the theory of Social Choice [7]: motivated by pioneering work of [38], a significant amount of research in Artificial Intelligence has investigated the provability of such results in logical settings (see [22] for a recent survey). We show that the most interesting of these results (Arrow's theorem and the Gibbard-Satterthwaite theorem) have proof complexity counterparts: the unsatisfiability of formulas encoding them can be certified by Frege proofs of subexponential length. A first example of application is Arrow's Theorem. The formulas encoding the nonexistence of a social welfare function satisfying the conditions of Arrow's theorem are rather large. Nevertheless, such an encoding exists, and was used explicitly in [38] to give a computer-assisted proof of Arrow's theorem[4]:

▶ **Definition 28.** *Consider an instance with n agents and m objects to rank. There are $(m!)^n$ possible profiles for the complete rankings of the m objects, and $m!$ possible aggregate orderings of the m objects. Formula $Arrow_{m,n}$ (unsatisfiable for $m, n \geq 3$) has $(m!)^{n+1}$ variables $X_{R,\pi}$, one for each possible pair (R, π) consisting of ranking profile R, and an aggregate ordering $\pi \in S_m$. The constraints are the following:*

- *For every $R \in \mathcal{R}$ and $\pi_1 \neq \pi_2 \in S_m$ add clauses $\bigvee_{\pi \in S_m} X_{R,\pi}$ ("every profile is aggregated to some ordering") and $\overline{X_{R,\pi_1}} \vee \overline{X_{R,\pi_2}}$ ("no profile is aggregated to more than one ordering")*
- *For $i = 1, \ldots, n$ we add to $Arrow_{m,n}$ clauses $\bigvee_{R \in \mathcal{R}} \overline{X_{R,R_i}}$. These forbid aggregations that always output the ordering given by the i'th agent, i.e. dictatorial rank aggregations.*
- *For every two objects a, b let $S^m_{a,b}$ be the set of orderings π where for all $i = 1, \ldots n$, $\pi^{-1}(a) < \pi^{-1}(b)$ (i.e. a is preferred to b in ordering π). Let $\mathcal{R}_{a,b}$ be the set of profiles such that for every $i = 1, \ldots, n$, $R_i \in S^m_{a,b}$ (i.e. all agents prefer a to b). For every $R \in \mathcal{R}_{a,b}$ add to $Arrow_{m,n}$ clauses $\bigvee_{\pi \in S^m_{a,b}} X_{R,\pi}$. These constraints encode unanimity (if all agents prefer object a to b then a is preferred to b in the aggregated ranking).*

[4] For encodings of Arrow's Theorem in more powerful logical frameworks see [14, 23].

- *For all profiles $R, R' \in \mathcal{R}$ and objects a, b such that all players rank a, b in the same way in both R, R' and all pairs $\pi_1, \pi_2 \in S^m$ that rank a, b in a different way (i.e. $\pi_1^{-1}(a) < \pi_1^{-1}(b)$ but $\pi_2^{-1}(a) > \pi_2^{-1}(b)$ or viceversa) we add to $Arrow_{m,n}$ clauses $\overline{X_{R,\pi_1}} \vee \overline{X_{R',\pi_2}}$. These encode independence of irrelevant alternatives (if R, R' coincide with respect to the relative ordering of a, b then their aggregate orderings also rank a, b in the same way).*

Results in [38] yield a kernelization for $Arrow_{m,n}$ with a reduction chain of length $O(m+n)$. We improve them by providing a kernelization with reduction chains whose length only depends on n, implying the existence of polynomial size Frege proofs for constant values of n:

▶ **Theorem 29.** *Formulas $Arrow_{m,n}$ have a kernelization with data reduction chains of length $\leq C(n+1)$, with constant C independent from m, n, whose safety is witnessed by polynomial-size Frege proofs. Hence (a) formulas $Arrow_{m,n}$ have Frege proofs of size quasipolynomial in $|Arrow_{m,n}|$). (b). For every fixed $n \geq 3$ there exists a polynomial $p_n(\cdot)$ such that for all $m \geq 3$ formulas $Arrow_{m,n}$ have Frege proofs of size at most $p_n(|Arrow_{m,n}|)$.*

Proof. The kernelization has two data reduction rules, described informally as follows:

(a). If $n \geq 2, m \geq 6$ and $W : [S_m]^n \to [S_m]$ is a function that is non-dictatorial, IIA and unanimous then there exists an $T \subseteq [m]$, $|T| = m - 5$ such that $W_{-T} : [S_{[m]_{-T}}]^n \to [S_{[m]_{-T}}]$ (see the end of Section 2) has the same properties. In other words, one can reduce in one step the set of alternatives from $[m]$ (which has m elements) to $[m]_{-T}$ (which has 5).

(b). See [38]: If $n, m \geq 3$ and $W : [S_m]^n \to [S_m]$ is non-dictatorial, IIA and unanimous then at least one of the functions $W_{1,2}, W_{1,3}, W_{2,3} : [S_m]^{n-1} \to [S_m]$ defined as follows: $W_{i,j}(R_1, R_2, \ldots, \widehat{R_i}, \ldots, R_n) = W(R'_1, \ldots, R'_n)$ is non-dictatorial, IIA and unanimous. Here $R'_i = R_j$, $R'(k) = R_k, k \neq i$. In other words, one can reduce in one step the number of agents by one.

▶ **Lemma 30.** *If W is unanimous, IIA and non-dictatorial then for every $B \subseteq [m]$, function W_{-B} is unanimous and IIA.*

Proof. Suppose that $a, a' \in [m]_{-B}$ and $a <_{R_i} a'$ for all $i \in [m]_{-B}$. Then $a <_{R_i^{+B}} a'$. By unanimity of W, $a <_{W(R_1^{+B}, \ldots, R_n^{+B})} a'$. Since a, a' were arbitrary, it follows that W_{-B} is unanimous. As for IIA, let $a, a' \in [m]_{-B}$ and $(R_1, R_2, \ldots, R_n)$ and $(R'_1, R'_2, \ldots, R'_n)$ be preference profiles such that, for every $i = 1, \ldots, n$, R_i and R'_i agree with respect to the relative ordering of a, a'. Then for every $i = 1, \ldots, n$, R_i^{+B} and $R_i'^{,+B}$ agree with respect to the relative ordering of a, a'. By the IIA axiom for W, $W(R_1^{+B}, R_2^{+B}, \ldots, R_n^{+B})$ and $W(R_1'^{,+B}, R_2'^{,+B}, \ldots, R_n'^{,+B})$ agree with respect to the relative ranking of a, a'. Hence so do $W_B(R_1, R_2, \ldots, R_n)$ and $W_B(R'_1, R'_2, \ldots, R'_n)$. ◀

▶ **Lemma 31.** *Reduction (a). is safe.*

Proof. Consider an arbitrary set $T \subseteq [m]$ of cardinality $m - 6$, e.g. $T = \{7, \ldots, m\}$. Let $x \notin T$, e.g. $x = 6$ and $U = T \cup \{x\}$. If W_{-U} is non-dictatorial we are done. Otherwise, assume w.l.o.g. that agent 1 is a dictator for W_{-U}. Since 1 is not a dictator for W, there must exist indices $c \neq d \in [m]$ and preference profiles $<_1, \ldots, <_n$ on $[m]$ such that $c <_1 d$ but $d <_{W(<_1, \ldots, <_n)} c$. Let $y, a, b \notin T$, different from x, c, d, and let $V \subseteq [m]$, $|V| = m - 5$, $a, b, c, d, x \notin V$, $y \in V$. Such a V exists, since $m \geq 6$. Clearly $V \neq U$, since $y \in V \setminus U$. We claim that function W_{-V} is not dictatorial.

First note that 1 cannot be a dictator for W_{-V}. Indeed, consider $<_{i,-V}$ the restriction of $<_i$ to $[m]_{-V}$. We have $c <_{1,-V} d$ but $d <_{W_{-V}(<_{1,-V}, \ldots, <_{n,-V})} c$. The first relation holds because $c <_1 d$ and $c, d \notin V$. The second relation holds because to compare c, d according to

$W_{-V}(<_{1,-V},\ldots,<_{n,-V})$ we apply function W on $(<^{+V}_{1,-V},\ldots,<^{+V}_{n,-V})$. But since $c,d\notin V$, $<^{+V}_{i,-V}$ coincides with $<_i$ with respect to the ordering of c,d for $i=1,\ldots,n$. Invoking the IIA property of W for tuples $(<_1,\ldots,<_n)$ and $(<^{+V}_{1,-V},\ldots,<^{+V}_{n,-V})$ justifies the second relation.

Assume now that another agent, say 2, were a dictator for W_{-V}. Let $<_1,<_2$ be preference profiles on $[m]$ s.t. $a<_1 b$ but $b<_2 a$, and $<_3,\ldots,<_n$ be arbitrary preference profiles. First, invoking the first relation, the fact that W_{-U} is computed using W, and that 1 is a dictator for W_{-U} we get that $a<_{W_{-U}(<_{1,-U},<_{2,-U},\ldots,<_{n,-U})} b$. Invoking the IIA property of W on tuples $(<_{1,-U},<_{2,-U},\ldots,<_{n,-U})$ and $(<_1,<_2,\ldots,<_n)$ we get that $a<_{W(<_1,<_2,\ldots,<_n)} b$. Using a similar reasoning for the function W_{-V} we get that $b<_{W(<_1,<_2,\ldots,<_n)} a$, a contradiction. ◀

The safety of reduction (b) was (mathematically) proved in [38]. Next we outline how to simulate these mathematical arguments using polynomial-size Frege proofs. First, note that formula $Arrow_{m,n}$ has $(m!)^{n+1}$ variables, all of them appearing explicitly in the formula. But $n=O(log((m!)^{n+1}))$, so indeed a reduction chain of length $O(n)$ has length logarithmic in $|Arrow_{m,n}|$. Invoking our metatheorem yields a proof of point (a) of Theorem 29. Point (b) follows by invoking point 2 of the same metatheorem.

First reduction rule: Define, for $Q\subseteq[m]$, $|Q|=m-5$, and $i=1,\ldots,n$ formulas

$$Nondict_{-Q,i}:=\bigvee_{R\in\mathcal{R}_{-Q}}\overline{X_{R^{+Q},R_i^{+Q}}}$$

(informally, formula $Nondict_{-Q,i}$ is true iff i is not a dictator for W_{-Q}).

$$Unanimous_{-Q}:=\bigwedge_{a,b\in[m]_{-Q}}\bigvee_{\substack{R\in\mathcal{R}_{a,b,-Q}\\ \pi\in S^m_{a,b}}}X_{R^{+Q},\pi^{+Q}}.$$

$$IIA_{-Q}:=\bigwedge_{(R,R',\pi,\pi')\in\mathcal{R}_{-Q}}(\overline{X_{R^{+Q},\pi_1^{+Q}}}\vee\overline{X_{R'^{,+Q},\pi_2^{+Q}}})$$

(where, for simplicity, we have ommitted the IIA restrictions on R,R',π_1,π_2, see Definition 28). Note that $Arrow_{m,n}=Unanimous_{-\emptyset}\wedge IIA_{-\emptyset}\wedge\bigwedge_{i=1}^n Nondict_{-\emptyset,i}$.
We will prove that

$$Arrow_{m,n}\wedge\bigwedge_{i=1}^n Nondict_{-[6:m],i}\vdash Arrow_{5,n}\tag{5}$$

and, for $i=1,\ldots,n$

$$Arrow_{m,n}\wedge\bigvee_{i=1}^n\bigwedge_{R\in\mathcal{R}_{-[6:m]}}X_{R^{+[6:m]},R_i^{+[6:m]}}\vdash Arrow_{5,n}\tag{6}$$

Employing tautology $\bigwedge_{i=1}^n Nondict_{-[6:m],i})\vee\bigvee_{i=1}^n(\bigwedge_{R\in\mathcal{R}_{-[6:n]}}X_{R^{+[6:m]},top(R_i^{+[6:m]})})$ and substitutions implicit in (5) and (6) we conclude that $Arrow_{m,n}\vdash P_1\vee P_2$, where both P_1,P_2 are formulas isomorphic to $Arrow_{5,n}$. Thus we are in the framework of our metatheorem with $R=2$.
We need to specify the substitutions implicit in (5) and (6). First, formalizing the mathematical argument in Lemma 30 we show that $Arrow_{m,n}\vdash IIA_{-Q}\wedge Unanimous_{-Q}$. The propositional content of this implication is trivial: the clauses of $Unanimous_{-Q}$ and IA_{-Q} are simply subclauses of $Arrow_{m,n}$.
Because of this, the substitution witnessing implication (5) is quite simple: it replaces a restricted variable $X'_{R,\pi}$ of $Arrow_{5,n}$ with the variable $X_{R^{+[6;m]},\pi^{+[6:m]}}$ of $Arrow_{m,n}$.

As for (6), define, for all $c < d \in [5]$ and $i = 1, \ldots, n$, formulas

$$Witness_{c,d,i} := \bigvee_{\substack{R \in \mathcal{R}: R_i \in S^m_{c,d} \\ \pi \in S^m_{d,c}}} X_{R,\pi} \tag{7}$$

Informally, formula $Witness_{c,d,i}$ is true when pair (c, d) acts as a witness that agent i is not a dictator for W, since $c <_{R_i} d$ but $d <_{W(R)} c$.

A next step is to prove that $Arrow_{m,n} \wedge \bigwedge_{R \in \mathcal{R}_{-[6:m]}} X_{R^{+[6:m]}, R_i^{+[6:m]}} \vdash \bigvee_{c<d \in [5]} Witness_{c,d,i}$. This is easy: we use literals $X_{R^{+[6:m]}, R_i^{+[6:m]}}$ to prove (by resolution)

$$Arrow_{m,n} \wedge \bigwedge_{R \in \mathcal{R}_{-[6:m]}} X_{R^{+[6:m]}, R_i^{+[6:m]}} \wedge \bigwedge_{c<d \in [5]} \bigwedge_{\substack{R \in \mathcal{R}: R_i \in S^m_{c,d} \\ \pi \in S^m_{d,c}}} \overline{X_{R,\pi}} \vdash \square \tag{8}$$

(the last conjunction negates formulas $Witness_{c,d,i}$) and then invoke Lemma 2 to get a proof of the same length of the implication we claimed.

Now we prove, for all $j = 1, \ldots, n$ that $Arrow_{m,n} \wedge Witness(c, d, i) \vdash Nondict_{-V,j}$, where V is defined as in the proof of Lemma 31.

The proof of the implication for $j = i$ uses unit literal X_{R,R_j} (negation of one from $Nondict_{-V,i}$) and $\overline{X_{R,\pi}} \vee \overline{X_{R^{+V}_{-V}, \pi_2}}$ (part of the IIA part of $Arrow_{m,n}$) to derive clauses $\overline{X_{R^{+V}_{-V}, \pi_2}}$ for all π_2 that rank c, d in a different way than π. Resolving away these literals from clause $\bigvee_{\pi \in S_m} X_{R^{+V}_{-V}, \pi}$ (part of $Arrow_{m,n}$) derives clause $Nondict_{-V,i}$.

As for the case $j \neq i$, we want to show that

$$Arrow_{m,n} \wedge Witness(c, d, i) \wedge \bigwedge_{R \in \mathcal{R}_{[5]}} X_{R^{+[6:m]}, R_j^{+[6:m]}} \vdash \square \tag{9}$$

By Lemma 2 this will imply $Nondict_{-V,j}$. Let $R = (R_1, R_2, \ldots, R_n)$ be a profile on $[5]$ such that $a <_{R_1} b$ but $b <_{R_2} a$ ($a, b \in [5]$ are defined as in Lemma 31). Combining the derivations of $Nondict_{-V,j}$ in (9) with the ones of $Unanimous_{-V}$ and IIA_{-V} (outlined before), plus a bijective identification of of $[m]_{-V}$ and $[5]$ yields a substitution that proves $Arrow_{5,n}$, completing the proof of (6).

Second reduction rule: We refer to Lemma 2 of [38] for (mathematical) details of the reduction. What is important is that the soundness of the statement that at least one of $W_{1,2}, W_{2,3}, W_{1,3}$ is non-dictatorial is established by a case-by-case analysis. It is first proved that it cannot be that all these functions have the same dictator. Then it is established that if i is the dictator of $W_{1,2}$, j the dictator of $W_{1,3}$, k the dictator of $W_{2,3}$ then $i \in \{2, 3\}$, $j \in \{2, 3\}$, $k \in \{1, 3\}$. For all eight possible cases for triplets (i, j, k) we obtain a contradiction: either we explicitly provide a profile R showing that triplet (i, j, k) cannot represent the set of dictators for the three function, or we employ an argument similar to the one in the case $i = j = k$. ◀

As for the Gibbard-Satterthwaite theorem, we use the following formalization:

▶ **Definition 32.** *Consider an instance with n agents and m objects. There are $(m!)^n$ possible profiles for the complete rankings of the m objects, and m possible outcomes. Formula $GS_{m,n}$ has $(m!)^n \times m$ variables $X_{R,o}$, one for each possible pair consisting of a strategy profile R and a value $o \in [m]$, the value of the SCF on profile R. The constraints are the following:*

- *For $R \in \mathcal{R}$ add clauses $\bigvee_{o\in[m]} X_{R,o}$ ("every joint profile is aggregated to some object") and $\overline{X_{R,o_1}} \vee \overline{X_{R,o_2}}$ ("no joint profile is aggregated to more than one object").*
- *For $i = 1, \ldots, n$ we add to $GS_{m,n}$ clauses $\bigvee_{R\in\mathcal{R}} \overline{X_{R,top(R_i)}}$. They forbid social choice functions that always output the top preference of the i'th agent, i.e. dictatorial aggregations.*
- *For $i = 1, \ldots, n$ add $\bigvee_{R\in\mathcal{R}} X_{R,o}$. This eliminates social choice functions that are not onto.*
- *Add, for every pair (R, o), player $1 \leq i \leq m$ and $\pi \in S_m$, $\overline{X_{R,o}} \vee (\bigvee_{o'\in pr(i,o,R)} X_{s(i,R,\pi),o'})$. These clauses state that the social choice function is strategyproof.*

▶ **Theorem 33.** *For every fixed m formulas $GS_{n,m}$ have a kernelization of length $O(n)$ whose soundness has polynomial time Frege proofs. Hence, formulas $GS_{n,m}$ expressing the Gibbard-Satterthwaite theorem have (a). Frege proofs of size quasipolynomial in $|GS_{n,m}|$, (b). for every fixed n, Frege proofs of size polynomial in $|GS_{n,m}|$.*

Proof. The argument is very similar to that of Arrow's theorem: We use the following two data reductions: (a). for $n \geq 2, m \geq 4$, if $W : [m]^n \to [m]$ is a social choice function that is onto, non-dictatorial and strategy-proof then there exists $T \subseteq [m]$, $|T| = m - 3$ such that function $W_{-T} : ([m]_{-T})^n \to [m]_{-T}$ is onto, non-dictatorial and strategyproof, and (b). [37] if $W : [m]^n \to [m]$ is a social choice function that is onto, non-dictatorial and strategy-proof then one of functions $W_{1,2}, W_{1,3}, W_{2,3} : [S_m]^{n-1} \to [m]$ defined by $W_{a,b}(R_1, R_2, \ldots, \widehat{R_a}, \ldots, R_n) = W(R'_1, \ldots, R'_n)$ is non-dictatorial, onto and strategy-proof. Here $R'_a = R_b$, $R'(k) = R_k$ for $k \neq a$.

First, it is not obvious that, as formulated in the paragraph, W_{-B} is well-defined. The reason is that $W_{-B}(R)$ invokes W on profile R^{+B}, and it is not obvious that if R is a profile on $[m]_{-B}$ the the outcome of W is an element of $[m]_{-B}$, as needed by the definition.

Suppose that $W_{-B}(R) = W(R^{+B}) \in B$ for some profile R on $[m]_{-B}$. We claim that $W(S) \in B$ for every profile $S = (S_1, S_2, \ldots, S_n)$, contradicting the hypothesis that W is onto. Indeed, if $W_{-B}(R) = W(R^{+B}) \in B$ then $W_{-B}(R_{-1}, S_1) = W(R_{-1}^{+B}, S_1^{+B}) \in B$, otherwise agent 1 would have an opportunity to manipulate at profile R^{+B} by misrepresenting its preference as S_1^{+B}. Applying this argument inductively for agents $2, 3, \ldots, n$ (replacing R_i by S_i) we infer that $W(S) \in B$, which is what we claimed.

▶ **Lemma 34.** *If $W : S_m^n \to [m]$ is onto, strategyproof and non-dictatorial then for every $B \subseteq [m]$, function W_{-B} is onto and strategyproof.*

Proof. Suppose there exists some profile R and agent $i \in [n]_{-B}$ such that i could manipulate $W_{-B}(R)$ by misrepresenting its profile as R'_i. This means that i could manipulate W on profile R^{+B} by misrepresenting its profile as $R_i'^{,+B}$, contradicting the fact that W is strategy-proof. Hence W_{-B} is strategy-proof.

Suppose now that $a \in [m]_{-B}$. Since W is onto, there must exist a profile R such that $W(R) = a$. Consider the profile R'_i that modifies R by moving a to the top of preference profile R_i. We claim that $W(R'_i) = a$. Indeed, if this was not the case then i could manipulate on profile R'_i by misrepresenting its preferences. Continuing the argument inductively for all agents we infer that if $\overline{R}$ is the profile that modifies R by moving a to the top of all profile preferences then $W(\overline{R}) = a$. This means that W is unanimous, hence W_{-B} also is. But then there is a profile $\underline{R}$ such that $W_{-B}(\underline{R}) = a$: simply make a the top of all profiles. Since a was arbitrary, it follows that W_{-B} is onto. ◀

We next prove the safety of reduction rule (a) (for (b) see [37]). We show that there exists $T \subseteq [m]$, $|T| = m-3$ such that W_{-T} is non-dictatorial. Together with Lemma 34 this establishes the safety of rule a.

Step 1. Given two different sets T_1, T_2 of size $m-3$, $|T_1 \cap T_2| = m-4$ (in other words, $|\overline{T_1}| = |\overline{T_2}| = 3$, $|\overline{T_1} \cap \overline{T_2}| = 2$), we show that functions W_{-T_1}, W_{-T_2} must have the same dictator, if they have one.

Suppose, indeed, that W_{-T_1} has dictator i, W_{-T_2} has dictator $j \neq i$. Let $d \in T_2 \setminus T_1$, $c \in T_1 \setminus T_2, a, b \in \overline{T_1} \cap \overline{T_2}$. Define profiles

$$R_s = a < b < \ldots < sorted(T_2) < c, \text{ for } s \neq j$$

$$R_j = b < a < \ldots < sorted(T_2) < c.$$

$W(R) = a$, since i is a dictator for W_{-T_1}. Now, if we replace R_i by

$$R'_i = a < b < \ldots < c < sorted(T_2).$$

obtaining profile R', then $W(R') = a$, otherwise agent i could manipulate by reporting profile R_i instead. We continue changing iteratively profiles $R_s, s \neq j$ to $R'_s = a < b < \ldots < c < sorted(T_2)$, one profile at a time, until all profiles R_s except R_j have been replaced by R'_s. Call this profile $\underline{R}$. That is, $\underline{R}_s = a < b < \ldots < c < sorted(T_2)$ for $s \neq j$, while $\underline{R}_j = b < a < \ldots < sorted(T_2) < c$.

Since no agent $s \neq j$ had an opportunity to manipulate, it must be that $W(\underline{R}) = a$. Consider now profile $\underline{R}'_j = b < a < \ldots < c < sorted(T_2)$. Since W_{-T_2} has agent j as a dictator, $W(\underline{R}_1, \ldots, \underline{R}'_j, \ldots, \underline{R}_n) = W_{-T_2}(\underline{R}_{1,-T_2}, \ldots, \underline{R}'_{j,-T_2}, \underline{R}_{n,-T_2}) = b$. So agent j has an opportunity to manipulate at profile $(\underline{R}_1, \ldots, \underline{R}'_j, \ldots, \underline{R}_n)$ by reporting instead $\underline{R}'_j$.

Step 2. Either there exists a set T of size $m-3$ such that W_{-T} is not dictatorial, or all functions $W_{-T}, |T| = m-3$ must have the same dictator. Indeed, we can "interpolate" between any two sets of cardinality $m-3$ by a sequence of sets falling under step 1.

Step 3. We show that it is not possible that all functions W_{-T}, $|T| = m-3$ have the same dictator i. Since W is not dictatorial, there exists a profile R such that $b = W(R)$ is different from $a = top(R_i)$. Let $T \subseteq [m], |T| = m-3, a, b \notin T$ and consider a profile R' that modifies R by moving $b, sorted(T)$ to the bottom of all preferences (in this order), that is $R' = (R_{-b}^{+b})_{-sorted(T)}^{+sorted(T)}$.

We have $W(R') = W_{-T}(R'_{-T}) = top(R'_i) = a$, since W_{-T} has i as dictator and $top(R'_i) = a$. Let us create a path between R and R' by changing one profile R_s at a time to R'_s, the last move being R_i. The value of W does not change at any step s, since W_{-T} has i as dictator, and the relative orders of elements in $[m]_{-T}$ does not change at any profile as a result of a change $R_s \to R'_s$, $s \neq i$, or agent s would have an opportunity of manipulation at one of the two profiles, using R_s, R'_s, whichever yields a result ranked lower in R_s, R'_s. But this yields a contradiction, since $W(R) = b$ and $W(R') = a \neq b$.

For some details on the propositional simulations we refer to [26]. ◀

5 Conclusions and open problems

We believe that the most important contribution of our paper is to show that several techniques for proving kernelization can sometimes be simulated by efficient extended Frege proofs. The proof techniques in this list includes: crown decomposition, the sunflower lemma, ad-hoc methods. It is an interesting challenge to enlarge the list of methods and problems that have such a simulation or, conversely, show limits of some of these methods.

It is rewarding to note that methods from algebraic topology can be used to reframe and extend results in both Topological Combinatorics and Computational Social Choice: Arrow's theorem has topological proofs [4, 12, 13]. On the other hand the original results on Kneser's conjecture [32] have been strengthened using more advanced topological methods, e.g. [3] (see [29] for a book-length treatment). The results in [2, 38] can be interpreted as stating that in both cases one can bypass topological arguments by purely combinatorial arguments (plus computer-assisted verification of finitely many cases). It is an interesting question whether this is still true for the results requiring more sophisticated topological methods as well.

In Theorem 26 we have only obtained polynomial size extended Frege proofs. Similarly, in Theorems 29, 33 we have only obtained polynomial-size Frege proofs when the number of agents is fixed. We leave open the issue of improving these results. On the other hand, our results have only touched on the most basic topics on the proof complexity of statements in computational social choice. There has been significant progress in this area (see e.g. [22]) and we believe that our framework may be applicable to some of this work (e.g. to the Preservation Theorem of [21]). It would be interesting to see if this is really the case.

References

1 James Aisenberg, Maria Luisa Bonet, and Sam Buss. Quasipolynomial size Frege proofs of Frankl's theorem on the trace of sets. *Journal of Symbolic Logic*, 81(2):687–710, 2016.

2 James Aisenberg, Maria Luisa Bonet, Sam Buss, Adrian Crăciun, and Gabriel Istrate. Short proofs of the Kneser–Lovász coloring principle. *Information and Computation*, 261:296–310, 2018.

3 E. Babson and D.N. Kozlov. Proof of the Lovász conjecture. *Annals of Mathematics*, 165:965–1007, 2007.

4 Yuliy M Baryshnikov. Topological and discrete social choice: in a search of a theory. In *Topological social choice*, pages 53–63. Springer, 1997.

5 Olaf Beyersdorff, Nicola Galesi, Massimo Lauria, and Alexander A Razborov. Parameterized bounded-depth Frege is not optimal. *ACM Transactions on Computation Theory (TOCT)*, 4(3):1–16, 2012.

6 Maria Luisa Bonet, Samuel R Buss, and Toniann Pitassi. Are there hard examples for Frege systems? In *Feasible Mathematics II*, pages 30–56. Birkhäuser Boston, 1995.

7 Felix Brandt, Vincent Conitzer, Ulle Endriss, Jérôme Lang, and Ariel D Procaccia. *Handbook of computational social choice*. Cambridge University Press, 2016.

8 S. Buss. Polynomial size proofs of the propositional pigeonhole principle. *Journal of Symbolic Logic*, 52(4):916–927, 1987.

9 Sam Buss. Quasipolynomial size proofs of the propositional pigeonhole principle. *Theoretical Computer Science*, 576:77–84, 2015.

10 Sam Buss and Jakob Nordström. Proof complexity and SAT solving. *Chapter to appear in the 2nd edition of Handbook of Satisfiability*, 2021. Draft version available at `https://www.math.ucsd.edu/~sbuss/ResearchWeb/ProofComplexitySAT`.

11 Lorenzo Carlucci, Nicola Galesi, and Massimo Lauria. Paris-Harrington tautologies. In *2011 IEEE 26th Annual Conference on Computational Complexity*, pages 93–103, 2011.

12 Graciela Chichilnisky. Social choice and the topology of spaces of preferences. *Advances in Mathematics*, 37(2):165–176, 1980.

13 Graciela Chichilnisky. The topological equivalence of the Pareto condition and the existence of a dictator. *Journal of Mathematical Economics*, 9(3):223–233, 1982.

14 Giovanni Ciná and Ulle Endriss. A syntactic proof of Arrow's theorem in a modal logic of social choice functions. In *Proceedings of the 2015 International Conference on Autonomous Agents and Multiagent Systems*, pages 1009–1017, 2015.

15 Stephen A Cook and Robert A Reckhow. The relative efficiency of propositional proof systems. *The Journal of Symbolic Logic*, 44(1):36–50, 1979.

16 Stefan Dantchev, Barnaby Martin, and Stefan Szeider. Parameterized proof complexity. *Computational Complexity*, 20(1):51, 2011.

17 M. de Longueville. *A Course in Topological Combinatorics*. Springer, 2012.

18 Rodney G Downey and Michael R Fellows. *Fundamentals of parameterized complexity*. Springer, 2013.

19 Fedor V Fomin, Daniel Lokshtanov, Saket Saurabh, and Meirav Zehavi. *Kernelization: theory of parameterized preprocessing*. Cambridge University Press, 2019.

20 Fedor V Fomin, Daniel Lokshtanov, Saket Saurabh, and Meirav Zehavi. Parameterized algorithms. *Beyond the Worst-Case Analysis of Algorithms*, page 27, 2020.

21 Christian Geist and Ulrich Endriss. Automated search for impossibility theorems in social choice theory: Ranking sets of objects. *Journal of Artificial Intelligence Research*, 40:143–174, 2011.

22 Christian Geist and Dominik Peters. Computer-aided methods for social choice theory. *Trends in Computational Social Choice*, pages 249–267, 2017.

23 Umberto Grandi and Ulle Endriss. First-order logic formalisation of Arrow's theorem. In *International Workshop on Logic, Rationality and Interaction*, pages 133–146. Springer, 2009.

24 Pavel Hrubes and Iddo Tzameret. Short proofs for the determinant identities. *SIAM Journal on Computing*, 44(2):340–383, 2015.

25 G. Istrate, A. Percus, and S. Boettcher. Spines of random constraint satisfaction problems: Definition and connection with computational complexity. *Annals of Mathematics and Artificial Intelligence*, 44(4):353–372, 2005.

26 Gabriel Istrate, Cosmin Bonchis, and Adrian Craciun. Kernelization, proof complexity and social choice, 2021. `arXiv:2104.13681`.

27 Gabriel Istrate and Adrian Craciun. Proof complexity and the kneser-lovász theorem. In *International Conference on Theory and Applications of Satisfiability Testing*, pages 138–153. Springer, 2014.

28 Leszek Aleksander Kołodziejczyk, Phuong Nguyen, and Neil Thapen. The provably total NP search problems of weak second order bounded arithmetic. *Annals of Pure and Applied Logic*, 162(6):419–446, 2011.

29 D. Kozlov. *Combinatorial Algebraic Topology*. Springer Verlag, 2008.

30 Jan Krajíček. *Proof complexity*, volume 170. Cambridge University Press, 2019.

31 Massimo Lauria, Pavel Pudlák, Vojtěch Rödl, and Neil Thapen. The complexity of proving that a graph is Ramsey. *Combinatorica*, 37(2):253–268, 2017.

32 L. Lovász. Kneser's conjecture, chromatic number, and homotopy. *Journal of Combinatorial Theory, Series A*, 25:319–324, 1978.

33 C. Moore, G. Istrate, D. Demopoulos, and M. Vardi. A continuous-discontinuous second-order transition in the satisfiability of a class of Horn formulas. *Random Structures and Algorithms*, 31(2):173–185, 2007.

34 Akihiro Nozaki, Toshiyasu Arai, Noriko H Arai, et al. Polynomal-size Frege proofs of Bollobás' theorem on the trace of sets. *Proceedings of the Japan Academy, Series A, Mathematical Sciences*, 84(8):159–161, 2008.

35 A. Schrijver. Vertex-critical subgraphs of Kneser graphs. *Nieuw Arch. Wiskd., III. Ser.*, 26:454–461, 1978.

36 Michael Soltys and Stephen Cook. The proof complexity of linear algebra. *Annals of Pure and Applied Logic*, 130(1-3):277–323, 2004.

37 Pingzhong Tang and Fangzhen Lin. A computer-aided proof to Gibbard–Satterthwaite theorem. Technical report, Technical report, 2008. `http://iiis.tsinghua.edu.cn/~kenshin`, 2008.

38 Pingzhong Tang and Fangzhen Lin. Computer-aided proofs of Arrow's and other impossibility theorems. *Artificial Intelligence*, 173(11):1041–1053, 2009.

Quantum Relational Hoare Logic with Expectations

Yangjia Li
University of Tartu, Estonia
SKLCS, Institute of Software, CAS, Beijing, China

Dominique Unruh
University of Tartu, Estonia

Abstract

We present a variant of the quantum relational Hoare logic from (Unruh, POPL 2019) that allows us to use "expectations" in pre- and postconditions. That is, when reasoning about pairs of programs, our logic allows us to quantitatively reason about how much certain pre-/postconditions are satisfied that refer to the relationship between the programs inputs/outputs.

2012 ACM Subject Classification Theory of computation → Quantum computation theory; Theory of computation → Hoare logic; Theory of computation → Program verification

Keywords and phrases Quantum cryptography, Hoare logics, formal verification

Digital Object Identifier 10.4230/LIPIcs.ICALP.2021.136

Category Track B: Automata, Logic, Semantics, and Theory of Programming

Related Version *Full Version*: https://arxiv.org/abs/1903.08357

Funding By the Air Force Office of Scientific Research through the project "Verification of quantum cryptography" (AOARD Grant FA2386-17-1-4022). By the project "Research and preparation of an ERC grant application on Certified Quantum Security" (MOBERC12). By the ERC consolidator grant CerQuS (819317). By the PRG team grant "Secure Quantum Technology" (PRG946) from the Estonian Research Council.
Yangjia Li: By the National Natural Science Foundation of China (Grant No: 61872342).
Dominique Unruh: By institutional research funding IUT2-1 of the Estonian Ministry of Education and Research. By the Estonian Centre of Exellence in IT (EXCITE) funded by the ERDF.

Acknowledgements We thank Gilles Barthe, Tore Vincent Carstens, and Justin Hsu for valuable discussions.

1 Introduction

Relational Hoare logics (RHL) are logics that allow us to reason about the relationship between two programs. Roughly speaking, they can express facts like "if the variable $\mathbf{x}$ in program $\mathfrak{c}$ is equal to $\mathbf{x}$ in program $\mathfrak{d}$, then after executing $\mathfrak{c}$ and $\mathfrak{d}$, respectively, the content of variable $\mathbf{y}$ in program $\mathfrak{c}$ is greater than that of $\mathbf{y}$ in $\mathfrak{d}$." RHL was introduced in the deterministic case by [6], and generalized to probabilistic programs by [4] (pRHL) and to quantum programs by [17] (qRHL). RHLs have proven especially useful in the context of verification of cryptographic schemes. For example, the CertiCrypt tool [4, 3] and its successor EasyCrypt [2, 1] use pRHL to create formally verified cryptographic proofs. And [16] implements a tool for verifying quantum cryptographic proofs based on qRHL.

On the other hand, "normal" (i.e., not relational) quantum Hoare logics have been developed in the quantum setting, starting with the predicate transformers from [10], see [11, 18, 7, 12]. Out of these, [10, 11, 18] use "expectations" instead of "predicates" for the pre- and postconditions of the Hoare judgments. To understand the difference, consider the case of classical probabilistic programs. Here, a predicate is (logically equivalent to) a set of program states (and a program state is a function from variables to values). In contrast, an

expectation is a function from program states to real numbers, basically assigning a value to each program state. Probabilistic Hoare logic with expectations, implicit in [13], uses expectations as the pre- and postconditions of a Hoare judgment. Then, roughly speaking, the preexpectation tells us what the expected value of the postexpectation is after running the program. This can be used to express much more fine-grained properties of probabilistic programs, giving quantitative guarantees about their probabilistic behavior, instead of just qualitative (a certain final state can or cannot occur). As [10] showed, the same approach can be used for quantum programs. Here, an expectation is modeled by a self-adjoint operator A on the space of all program states. The "value" of a given program state ρ is then computed as the trace $\operatorname{tr}\mathsf{A}\rho$. While at the first glance not as obvious as the meaning of classical expectations, this formalism has nice mathematical properties and is also equivalent to taking the expectation value of the outcome of a real-valued measurement. By using this approach, [10, 11, 18] can express more fine-grained judgments about quantum programs, by not just expressing which final states are possible, but also with what probabilities.

Yet, qRHL [17] did not follow this approach (only mentioning it as possible future work). As a consequence, qRHL does not enable as fine-grained reasoning about probabilities as the non-relational quantum Hoare logics. On the other hand, the non-relational quantum Hoare logics do not allow us to reason about the relationship between programs.

In this work, we combine the best of two worlds. We present a variant of qRHL, expectation-qRHL, that reasons about pairs of programs, and at the same time supports expectations as the pre- and postconditions, thus being as expressive as the calculi from [10, 11, 18] when it comes to the probabilistic behavior of the programs.

Related work. The relevant prior work has already been discussed above. Concurrently and independently, [5] presented a different formalization of expectation-qRHL. (The first versions on arXiv appeared within two months of each other.) The biggest difference is the definition of couplings which in our setting are separable quantum states, and in their setting nonseparable quantum states. Therefore, the soundness proofs are totally different in [5] and in the present paper, even for the same rules. As a consequence, we can avoid having to reason about judgments with side-conditions, making compositional reasoning about more complex programs much easier.

Organization. In Section 2 we introduce notation and preliminaries, including the concept of expectations. In Section 3 we give syntax and semantics of the imperative quantum programming language that we study. In Section 4 we give the definition of expectation-qRHL. In Section 5, we present sound rules for reasoning about expectation-qRHL judgments. And in Section 6, we analyze the quantum Zeno effect as an example of using our logic.

In the full version, we give a detailed comparison of our logic with [5] and full proofs of our results.

2 Preliminaries: Variables, Memories, and Predicates

In this section, we introduce some fundamental concepts and notations needed for this paper, and recap some of the needed quantum background as we go along. When introducing some notation X, the place of definition is marked like this: X. For further mathematical background we recommend [8, 9], and for an introduction to quantum mechanics [15].

Variables. Before we introduce the syntax and semantics of programs, we first need to introduce some basic concepts. A *variable* is described by a variable name $\mathbf{x}, \mathbf{y}, \mathbf{z}$ that identifies the variable, and a nonempty type T. The *type* of $\mathbf{x}$ is simply the nonempty set of all (classical) values the variable can take. E.g., a variable might have type $\{0,1\}$, or $\mathbb{N}$.[1] Lists or sets of variables will be denoted $\mathbf{X}, \mathbf{Y}, \mathbf{Z}$. Given a list $\mathbf{X} = \mathbf{x}_1 \dots \mathbf{x}_n$ of variables, we say its *type* is $T_1 \times \cdots \times T_n$ if T_i is the type of $\mathbf{x}_i$. We write $\mathbf{XY}$ for the concatenation/disjoint union of lists/sets of variables $\mathbf{X}, \mathbf{Y}$.

Memories and quantum states. An *assignment* assigns to each variable a classical value. Formally, for a set $\mathbf{X}$, the *assignments over* $\mathbf{X}$ are all functions m with domain $\mathbf{X}$ such that: for all $\mathbf{x} \in \mathbf{X}$ with type $T_{\mathbf{x}}$, $\mathsf{m}(\mathbf{x}) \in T_{\mathbf{x}}$. That is, assignments can represent the content of classical memories.

To model quantum memories, we simply consider superpositions of assignments: A *(pure) quantum memory* is a superposition of assignments. Formally, $\ell^2[\mathbf{X}]$, the set of all quantum memories over $\mathbf{X}$, is the Hilbert space with basis[2] $\{|\mathsf{m}\rangle\}_{\mathsf{m}}$ where m ranges over all assignments over $\mathbf{X}$. Here $|\mathsf{m}\rangle$ simply denotes the basis vector labeled m. We often write $|\mathsf{m}\rangle_{\mathbf{X}}$ to stress which space we are talking about. We call a quantum memory ψ *normalized* iff $\|\psi\| = 1$. Intuitively, a normalized quantum memory over $\mathbf{X}$ represents a state a quantum computer with variables $\mathbf{X}$ could be in. We also consider quantum states over arbitrary sets X (as opposed to sets of assignments). Namely, $\ell^2(X)$ denotes the Hilbert space with orthonormal basis $\{|x\rangle\}_{x\in X}$. (In that notation, $\ell^2[\mathbf{X}]$ is simply $\ell^2(A)$ where A is the set of all assignments on $\mathbf{X}$.) Normalized elements of $\ell^2[\mathbf{X}]$ represent quantum states.

We often treat elements of $\ell^2(T)$ and $\ell^2[\mathbf{X}]$ interchangeably if T is the type of $\mathbf{X}$ since there is a natural isomorphism between those spaces.

In many situations, we additionally need an additional symbol $\bot$ that denotes that a memory is not available because the program did not terminate. A *quantum $\bot$-memory over* $\mathbf{X}$ is an element of $\ell^2[\mathbf{X}]^\bot := \ell^2(A \cup \{\bot\})$ where A is the set of all assignments on $\mathbf{X}$. That is, a quantum $\bot$-memory is a superposition between a quantum memory and $|\bot\rangle$.

The tensor product $\otimes$ combines two quantum states $\psi \in \ell^2(X), \phi \in \ell^2(Y)$ into a joint system $\psi \otimes \phi \in \ell^2(X \times Y)$. In the case of quantum memories ψ, ϕ over $\mathbf{X}, \mathbf{Y}$, respectively, $\psi \otimes \phi \in \ell^2[\mathbf{XY}]$. (And in this case, $\psi \otimes \phi = \phi \otimes \psi$ since we are composing "named" systems.)

We will need to consider entangled pairs of memories. Specifically, a *quantum bimemory* over $\mathbf{X}_1, \mathbf{X}_2$ is an element of $\ell^2[\mathbf{X}_1] \otimes \ell^2[\mathbf{X}_2] = \ell^2[\mathbf{X}_1\mathbf{X}_2]$. Similarly, a *quantum $\bot$-bimemory* is an element of $\ell^2[\mathbf{X}_1]^\bot \otimes \ell^2[\mathbf{X}_2]^\bot$, i.e., a tensor product of two quantum $\bot$-memories. (Note: "one-sided-$\bot$" states such as $|\mathsf{m}\rangle \otimes |\bot\rangle$ are included in this.) For clarity, we often write $|\bot_1\rangle, |\bot_2\rangle$ instead of $\bot$ to emphasize whether we are talking about elements of $\ell^2[\mathbf{X}_1]^\bot$ or $\ell^2[\mathbf{X}_2]^\bot$.

For a vector (or operator) a, we write a^* for its adjoint. (In the finite dimensional case, the adjoint is simply the conjugate transpose of a vector/matrix. The literature also knows the notation $a^\dagger$.) The adjoint of a vector $|x\rangle$ is also written as $\langle x|$. We abbreviate $\mathsf{proj}(\psi) := \psi\psi^*$. This is the projector onto ψ when $\|\psi\| = 1$.

[1] We stress that we do not assume that the type is a finite or even a countable set. Consequently, the Hilbert spaces considered in this paper are not necessarily finite dimensional or even separable. However, all results can be informally understood by thinking of all sets as finite and hence of all Hilbert spaces as $\mathbb{C}^N$ for suitable $N \in \mathbb{N}$.

[2] When we say "basis", we always mean an orthonormal Hilbert-space basis.

Mixed quantum memories. In many situations, we need to model probabilistic quantum states (e.g., a quantum state that is $|0\rangle$ with probability $\frac{1}{2}$ and $|1\rangle$ with probability $\frac{1}{2}$). This is modeled using *mixed states* (a.k.a. *density operators*). Having normalized state ψ_i with probability p_i is represented by the operator $\rho := \sum_i p_i \mathsf{proj}(\psi_i)$.[3] In particular, $\mathsf{proj}(\psi)$ is the density operator of a pure quantum state ψ. Then ρ encodes all observable information about the distribution of the quantum state (that is, two distributions of quantum states have the same ρ iff they cannot be distinguished by any physical process). And $\operatorname{tr}\rho$ is the total probability $\sum_i p_i$. Note that we do not formally impose the condition $\operatorname{tr}\rho = 1$ or $\operatorname{tr}\rho \leq 1$ unless explicitly specified. We call a mixed state *normalized* iff $\operatorname{tr}\rho = 1$. We will often need to consider mixed states of quantum memories (i.e., mixed states with underlying Hilbert space $\ell^2[\mathbf{X}]$). We call them *mixed (quantum) memories* over $\mathbf{X}$. Analogously, we define *mixed bimemories* and *mixed $\bot$-bimemories* as mixed states of quantum ($\bot$-)bimemories.

Let $P_\bot := \mathsf{proj}(|\bot\rangle)$ and $P_{\not\bot} := \mathsf{id} - P_\bot$. We can easily access the terminating and non-terminating part of a mixed $\bot$-memory: $\downarrow_\bot(\rho) := P_\bot \rho P_\bot$ and $\downarrow_{\not\bot}(\rho) := P_{\not\bot}\rho P_{\not\bot}$ are the memory ρ after measuring that we have termination or do not have termination, respectively.

For a mixed ($\bot$-)bimemory ρ over $\mathbf{X}_1\mathbf{X}_2$ the *partial trace* $\operatorname{tr}_i \rho$ ($i = 1, 2$) is the result of throwing away the left/right memory (i.e., it is a mixed memory over $\mathbf{X}_i$). Formally, tr_i is defined as the continuous linear function satisfying $\operatorname{tr}_1(\sigma \otimes \tau) := \tau \cdot \operatorname{tr}\sigma$, $\operatorname{tr}_2(\sigma \otimes \tau) := \sigma \cdot \operatorname{tr}\tau$.

A mixed memory ρ is $(\mathbf{X}, \mathbf{Y})$-*separable* (i.e., not entangled between $\mathbf{X}$ and $\mathbf{Y}$) iff it can be written as $\rho = \sum_i \rho_i \otimes \rho'_i$ for mixed memories ρ_i, ρ'_i over $\mathbf{X}, \mathbf{Y}$, respectively. (Potentially infinite sum.) When $\mathbf{X}, \mathbf{Y}$ are clear from the context, we simply say *separable*.

In this paper, when we write infinite sums of operators, convergence is always with respect to the trace norm: $\|A\|_{\operatorname{tr}} := \operatorname{tr}\sqrt{AA^\dagger}$. (In the finite-dimensional case, the choice of norm is irrelevant since all norms are equivalent then.)

Operations on quantum states. An operation in a closed quantum system is modeled by an isometry U on $\ell^2(X)$.[4] If we apply such an operation on a mixed state ρ, the result is $U\rho U^*$. In particular, denote by id the identity operation, i.e. $\mathsf{id}\psi = \psi$ for all pure states ψ in this space.

An operator A on $\ell^2[\mathbf{X}]$ can be interpreted as an operator on $\ell^2[\mathbf{X}]^\bot$ by setting $A|\bot\rangle := 0$. To avoid confusion, we often write $A \oplus 0_\bot$ for the operator on $\ell^2[\mathbf{X}]^\bot$. Similarly, an operator A on $\ell^2[\mathbf{X}_1] \otimes \ell^2[\mathbf{X}_2]$ can be seen as an operator on $\ell^2[\mathbf{X}_1]^\bot \otimes \ell^2[\mathbf{X}_2]^\bot$, we write $A \oplus 0_{\bot\bot}$ for the operator on $\ell^2[\mathbf{X}_1]^\bot \otimes \ell^2[\mathbf{X}_2]^\bot$.

Most often, isometries will occur in the context of operations that are performed on a single variable or list of variables, i.e., an isometry U on $\ell^2[\mathbf{X}]$. Then U can also be applied to $\ell^2[\mathbf{Y}]$ with $\mathbf{Y} \supseteq \mathbf{X}$: we identify U with $U \otimes \mathsf{id}_{\mathbf{Y}\setminus\mathbf{X}}$. Furthermore, if $\mathbf{X}$ has type T, then an isometry U on $\ell^2(T)$ can be seen as an isometry on $\ell^2[\mathbf{X}]$ since we identify $\ell^2(T)$ and $\ell^2[\mathbf{X}]$. If we want to make $\mathbf{X}$ explicit, we write $U\,\mathbf{on}\,\mathbf{X}$ for the isometry U on $\ell^2[\mathbf{Y}]$. For example, if U is a 2×2-matrix and $\mathbf{x}$ has type bit, then $U\,\mathbf{on}\,\mathbf{x}$ can be applied to quantum memories over $\mathbf{xy}$, acting on $\mathbf{x}$ only. This notation is not limited to isometries, of course, but applies to other operators, too. (By *operator* we always mean a bounded linear operator in this paper.)

---

[3] Mathematically, density operators are positive Hermitian trace-class operators on $\ell^2(X)$. The requirement "trace-class" ensures that the trace exists and can be ignored in the finite-dimensional case.

[4] That is, a norm-preserving linear operation. Often, one models quantum operations as unitaries instead because in the finite-dimensional case an isometry is automatically unitary. However, in the infinite-dimensional case, unitaries are unnecessarily restrictive. Consider, e.g., the isometry $|i\rangle \mapsto |i+1\rangle$ with $i \in \mathbb{N}$ which is a perfectly valid quantum operation but not a unitary.

In slight overloading of notation, we also write $U \mathbf{on} \mathbf{X}$ for U acting on $\bot$-memories, where $(U \mathbf{on} \mathbf{X})|\bot\rangle = 0$. (That is, $U \mathbf{on} \mathbf{X}$ is short for the more precise $(U \mathbf{on} \mathbf{X}) \oplus 0_\bot$.) We also write $U \mathbf{on} \mathbf{X}_1$ for U acting on $\bot$-bimemories. In this case, we simply have $U \mathbf{on} \mathbf{X}_1 := (U \mathbf{on} \mathbf{X}_1) \otimes \mathsf{id}$. In particular, $(U \mathbf{on} \mathbf{X}_1)(|\mathsf{m}\rangle \otimes |\bot\rangle) = (U \mathbf{on} \mathbf{X}_1)|\mathsf{m}\rangle \otimes |\bot\rangle$ but $(U \mathbf{on} \mathbf{X}_1)(|\bot\rangle \otimes |\mathsf{m}\rangle) = 0$. Analogously for $U \mathbf{on} \mathbf{X}_2$.

We will use only binary measurements in this paper. A *binary measurement* M on $\ell^2[\mathbf{X}]$ has outcomes $\mathsf{true}, \mathsf{false}$ and is described by two bounded operators M_{true}, M_{false} on $\ell^2[\mathbf{X}]$ that satisfy $M_{\mathsf{true}}^* M_{\mathsf{true}} + M_{\mathsf{false}}^* M_{\mathsf{false}} = \mathsf{id}$, its *Krauss operators*. Given a mixed memory ρ, the probability of measurement outcome t is $p_t := \operatorname{tr} M_t \rho M_t^*$, and the post-measurement state is $M_t \rho M_t^* / p_t$.

Expectations. In this work, we will use expectations as pre- and postconditions in Hoare judgments. The idea of using expectations originated in [13] for reasoning about (classical) probabilistic programs. Intuitively, an expectation is a quantitative predicate, that is for any memory (or bimemory, in our case), it does not tell us whether the memory satisfies the predicate but *how much* it satisfies the predicate. Thus, classically, an expectation is simply a function from assignments to reals. By analogy, in the quantum setting, one might want to define expectations, e.g., as functions f from quantum bimemories to reals (i.e., an expectation would be a function $\ell^2[\mathbf{X}] \to \mathbb{R}_{\geq 0}$). However, such expectations might behave badly, for example, it is not clear how to compute the expected value $f(\psi)$ for a random ψ if the distribution of ψ is given in terms of a density operator. A better approach was introduced by [10]. Following their approach, we define an *expectation* as a positive operator A on quantum bimemories.[5] (We use letters $\mathsf{A}, \mathsf{B}, \mathsf{C}, \ldots$ for expectations in this paper.) This expectation then assigns the value $\psi^* \mathsf{A} \psi$ to the quantum memory ψ (equivalently, $\operatorname{tr} \mathsf{A}\, \mathsf{proj}(\psi)$). To understand this, it is best to first look at the special case where A is a projector. Then $\psi^* \mathsf{A} \psi = 1$ iff ψ is in the image of A, and $\psi^* \mathsf{A} \psi = 0$ iff ψ is orthogonal to the image of A. Such an A is basically a predicate (by outputting 1 for states that satisfy the predicate). Of course, states that neither satisfy the predicate nor are orthogonal to it will output a value between 0 and 1. Any expectation A can be written as $\sum_i p_i \mathsf{A}_i$ with projectors A_i. Thus, A would give p_i "points" for satisfying the predicate A_i. In this respect, expectations in the quantum setting are similar to classical ones: classical expectations give a certain amount of "points" for each possible classical input.

The nice thing about this formalism is that, given a density operator $\rho = \sum p_i \mathsf{proj}(\psi_i)$, we can easily compute the expected value of the expectation A. More precisely, the expected value of $\psi^* \mathsf{A} \psi = \operatorname{tr} \mathsf{A} \mathsf{proj}(\psi)$ with $\psi := \psi_i$ with probability p_i. That expected value is $\sum p_i \operatorname{tr} \mathsf{A} \mathsf{proj}(\psi_i) = \operatorname{tr} \mathsf{A} (\sum p_i \mathsf{proj}(\psi_i)) = \operatorname{tr} \mathsf{A} \rho$. This shows that we can evaluate how much a density operator satisfies the expectation A by just computing $\operatorname{tr} \mathsf{A} \rho$. This formula will be the basis for our definitions!

Analogously, we define $\bot$-*expectations* on quantum $\bot$-bimemories. However, we add one restriction: The value of a $\bot$-expectation should not change if we measure whether the $\bot$-bimemory is in $|\bot\rangle$ or not. Formally, a $\bot$-expectation is a positive operator on quantum $\bot$-bimemories that is invariant under $\mathcal{E}_\bot \otimes \mathcal{E}_\bot$ where $\mathcal{E}_\bot(\rho) := P_\bot \rho P_\bot + P_\not\bot \rho P_\not\bot$. ($\mathcal{E}_\bot$ corresponds to measuring and forgetting whether a given mixed $\bot$-memory is $|\bot\rangle$ or not.) Note that for an expectation A, the operator $\mathsf{A} \oplus 0_{\bot\bot}$ is a $\bot$-expectation. We can thus see expectations as special cases of $\bot$-expectations.

[5] Recall from page 4 that operators are always bounded in our context. This means that A is bounded, too. This means that the values that an expectation A can assign to states are between 0 and B for some finite B.

A very simple example of an expectation would be the matrix $\mathsf{A} := \begin{pmatrix} 1 & \\ & \frac{1}{2} \end{pmatrix}$ that assigns 1 to $|0\rangle$, and $\frac{1}{2}$ to $|1\rangle$. And given the density operator $\rho = \frac{1}{2}\mathsf{id}$ (representing a uniform qubit), $\operatorname{tr} \mathsf{A}\rho = \frac{3}{4}$ are intuitively expected.

Quantum equality. In [17], a specific predicate $\mathbf{X}_1 \equiv_{\mathfrak{q}} \mathbf{X}_2$ was introduced to describe the fact that two quantum variables (or list of quantum variables) have the same state. Formally, $\mathbf{X}_1 \equiv_{\mathfrak{q}} \mathbf{X}_2$ is the subspace consisting of all quantum memories in $\ell^2[\mathbf{X}_1\mathbf{X}_2]$ that are invariant under $\mathsf{SWAP}_{\mathbf{X}_1\mathbf{X}_2}$, the unitary that swaps variables $\mathbf{X}_1$ and $\mathbf{X}_2$.[6] Or equivalently, $\mathbf{X}_1 \equiv_{\mathfrak{q}} \mathbf{X}_2$ denotes the subspace spanned by all quantum memories of the form $\phi \otimes \phi$ with $\phi \in \ell^2[\mathbf{X}_1] = \ell^2[\mathbf{X}_2]$. We write $\mathsf{EQUAL}\,\mathbf{on}\,\mathbf{X}_1\mathbf{X}_2$ for the projector onto $\mathbf{X}_1 \equiv_{\mathfrak{q}} \mathbf{X}_2$.

3 Quantum programs

Syntax. We will now define a small imperative quantum language.[7] The set of all programs is described by the following syntax:

$$\mathfrak{c},\mathfrak{d} ::= \underline{\mathbf{apply}}\ U\ \underline{\mathbf{to}}\ \mathbf{X} \mid \mathbf{X} \leftarrow \psi \mid \underline{\mathbf{if}}\ M[\mathbf{X}]\ \underline{\mathbf{then}}\ \mathfrak{c}\ \underline{\mathbf{else}}\ \mathfrak{d} \mid \underline{\mathbf{while}}\ M[\mathbf{X}]\ \underline{\mathbf{do}}\ \mathfrak{c} \mid \mathfrak{c};\mathfrak{d} \mid \underline{\mathbf{skip}} \mid \underline{\mathbf{abort}}$$

Here $\mathbf{X}$ is a list of variables and U an isometry on $\ell^2[\mathbf{X}]$, $\psi \in \ell^2[\mathbf{X}]$ a normalized state, and M is a binary measurement on $\ell^2[\mathbf{X}]$. (There are no fixed sets of allowed U and ψ, any isometry/state that we can describe can be used here).[8]

Intuitively, $\underline{\mathbf{apply}}\ U\ \underline{\mathbf{to}}\ \mathbf{X}$ means that the operation U is applied to the quantum variables $\mathbf{X}$. E.g., $\underline{\mathbf{apply}}\ H\ \underline{\mathbf{to}}\ \mathbf{x}$ would apply the Hadamard gate to the variable $\mathbf{x}$ (we assume that H denote the Hadamard matrix). It is important that we can apply U to several variables $\mathbf{X}$ simultaneously, otherwise no entanglement between variables can ever be produced.

The program $\mathbf{X} \leftarrow \psi$ initializes the variables $\mathbf{X}$ with the quantum state ψ. The program $\underline{\mathbf{if}}\ M[\mathbf{X}]\ \underline{\mathbf{then}}\ \mathfrak{c}\ \underline{\mathbf{else}}\ \mathfrak{d}$ will measure the variables $\mathbf{X}$ with the measurement M, and, if the outcome is true, execute $\mathfrak{c}$, otherwise execute $\mathfrak{d}$.

The program $\underline{\mathbf{while}}\ M[\mathbf{X}]\ \underline{\mathbf{do}}\ \mathfrak{c}$ measures $\mathbf{X}$, and if the outcome is true, it executes $\mathfrak{c}$. This is repeated until the outcome is false.

Finally, $\mathfrak{c};\mathfrak{d}$ executes $\mathfrak{c}$ and then $\mathfrak{d}$. And $\underline{\mathbf{skip}}$ does nothing. We will always implicitly treat ";" as associative and $\underline{\mathbf{skip}}$ as its neutral element. $\underline{\mathbf{abort}}$ never terminates.

Semantics. The *denotational semantics* of our programs $\mathfrak{c}$ are represented as completely positive trace-reducing maps $[\![\mathfrak{c}]\!]$ on the mixed memories over $\mathbf{X}^{\mathbf{all}}$, defined by recursion on the structure of the programs. Here $\mathbf{X}^{\mathbf{all}}$ is a fixed set of program variables, and we will assume that all variables under consideration are contained in this set. The obvious cases are $[\![\underline{\mathbf{skip}}]\!] := \mathsf{id}$ and $[\![\mathfrak{c};\mathfrak{d}]\!] := [\![\mathfrak{d}]\!] \circ [\![\mathfrak{c}]\!]$ and $[\![\underline{\mathbf{abort}}]\!](\rho) := 0$. And application of an isometry U is also fairly straightforward given the syntactic sugar introduced above: $[\![\underline{\mathbf{apply}}\ U\ \underline{\mathbf{to}}\ \mathbf{X}]\!](\rho) := (U\,\mathbf{on}\,\mathbf{X})\rho(U\,\mathbf{on}\,\mathbf{X})^*$. (The notation $U\,\mathbf{on}\,\mathbf{X}$ was introduced on page 4.)

---

[6] That is, $\mathsf{SWAP}_{\mathbf{X}_1\mathbf{X}_2}(\psi \otimes \phi) = \phi \otimes \psi$ for $\psi \in \ell^2[\mathbf{X}_1]$, $\phi \in \ell^2[\mathbf{X}_2]$.

[7] Very similar to [18], except that we replace their case-statement by an if-statement and allow initialization with arbitrary states instead of just $|0\rangle$.

[8] We will assume throughout the paper that all programs satisfy those well-typedness constraints. In particular, rules may implicitly impose type constraints on the variables and constants occurring in them by this assumption.

Initialization of quantum variables is slightly more complicated: $\mathbf{X} \leftarrow \psi$ initializes the variables $\mathbf{X}$ with ψ, which is the same as removing $\mathbf{X}$, and then creating a new variable $\mathbf{X}$ with content ψ. Removing $\mathbf{X}$ is done by the operation $\mathrm{tr}_{\mathbf{X}}$ (partial trace, see page 4). And creating new variables $\mathbf{X}$ in state ψ is done by the operation $\otimes\mathsf{proj}(\psi)$. Thus we define $[\![\mathbf{X} \leftarrow \psi]\!](\rho) := \mathrm{tr}_{\mathbf{X}}\, \rho \otimes \mathsf{proj}(\psi)$.

The if-command first performs a measurement and then branches depending on the outcome. We then have that the state after measurement (without renormalization) is $(M_t \,\mathbf{on}\, \mathbf{X})\rho(M_t \,\mathbf{on}\, \mathbf{X})^*$ for outcome $t = \mathsf{true}, \mathsf{false}$. Then $\mathfrak{c}$ or $\mathfrak{d}$ is applied to that state and the resulting states are added together to get the final mixed state. Altogether:

$$[\![\underline{\mathbf{if}}\ M[\mathbf{X}]\ \underline{\mathbf{then}}\ \mathfrak{c}\ \underline{\mathbf{else}}\ \mathfrak{d}]\!](\rho) := [\![\mathfrak{c}]\!]\big(\downarrow_{\mathsf{true}}(\rho)\big) + [\![\mathfrak{d}]\!]\big(\downarrow_{\mathsf{false}}(\rho)\big)$$
$$\text{where} \qquad \downarrow_t(\rho) := (M_t \,\mathbf{on}\, \mathbf{X})\rho(M_t \,\mathbf{on}\, \mathbf{X})^*$$

While-commands are modeled similarly: In an execution of a while statement, we have $n \geq 0$ iterations of "measure with outcome true and run $\mathfrak{c}$" (which applies $[\![\mathfrak{c}]\!] \circ \downarrow_{\mathsf{true}}$ to the state), followed by "measure with outcome false" (which applies $\downarrow_{\mathsf{false}}$ to the state). Adding all those branches up, we get the definition:

$$[\![\underline{\mathbf{while}}\ M[\mathbf{X}]\ \underline{\mathbf{do}}\ \mathfrak{c}]\!](\rho) := \sum_{n=0}^{\infty} \downarrow_{\mathsf{false}}\big(([\![\mathfrak{c}]\!] \circ \downarrow_{\mathsf{true}})^n(\rho)\big)$$

We call a program $\mathfrak{c}$ *terminating* iff $\mathrm{tr}[\![\mathfrak{c}]\!](\rho) = \mathrm{tr}\,\rho$ for all ρ.

Semantics with explicit non-termination. $[\![\mathfrak{c}]\!]$ is not trace-preserving if $\mathfrak{c}$ is not terminating. For technical reasons, we will need a variant of this function that is trace-preserving. This can be achieved by outputting a mixed state that has an explicit state $\mathsf{proj}(|\bot\rangle)$ to denote non-termination. This semantic function $[\![\mathfrak{c}]\!]^\bot$ takes mixed $\bot$-memories to mixed $\bot$-memories and can be easily derived from $[\![\mathfrak{c}]\!]$:

$$[\![\mathfrak{c}]\!]^\bot(\rho) := [\![\mathfrak{c}]\!](\downarrow_{\not\bot}(\rho)) + \big(\mathrm{tr}\,\rho - \mathrm{tr}[\![\mathfrak{c}]\!](\downarrow_{\not\bot}(\rho))\big)\mathsf{proj}(|\bot\rangle).$$

($P_\bot, P_{\not\bot}$ are defined on page 4.) Operationally, $[\![\mathfrak{c}]\!]^\bot$ first measures if the state is $\bot$. If so, nothing happens. Otherwise, $\mathfrak{c}$ is applied. If $\mathfrak{c}$ does not terminate, the final output memory is set to be $\bot$. $[\![\mathfrak{c}]\!]^\bot$ is easily seen to be trace-preserving. Moreover, we have the composition property $[\![\mathfrak{c};\mathfrak{d}]\!]^\bot = [\![\mathfrak{d}]\!]^\bot \circ [\![\mathfrak{c}]\!]^\bot$, since

$$\begin{aligned}
&[\![\mathfrak{d}]\!]^\bot([\![\mathfrak{c}]\!]^\bot(\rho)) = [\![\mathfrak{d}]\!]^\bot\,[[\![\mathfrak{c}]\!](\rho_{\not\bot}) + (\mathrm{tr}\,\rho - \mathrm{tr}[\![\mathfrak{c}]\!](\rho_{\not\bot}))\mathsf{proj}(|\bot\rangle)] \\
&= [\![\mathfrak{d}]\!]([\![\mathfrak{c}]\!](\rho_{\not\bot})) + \big[\mathrm{tr}[\![\mathfrak{c}]\!]^\bot(\rho) - \mathrm{tr}[\![\mathfrak{d}]\!]([\![\mathfrak{c}]\!](\rho_{\not\bot}))\big]\,\mathsf{proj}(|\bot\rangle) \\
&= [\![\mathfrak{c};\mathfrak{d}]\!](\rho_{\not\bot}) + (\mathrm{tr}\,\rho - \mathrm{tr}[\![\mathfrak{c};\mathfrak{d}]\!](\rho_{\not\bot}))\mathsf{proj}(|\bot\rangle) = [\![\mathfrak{c};\mathfrak{d}]\!]^\bot(\rho).
\end{aligned}$$

4 qRHL with expectations

Defining the logic. We now present our definition of expectation-qRHL. We follow the approach from [17] to use separable couplings to describe the relationship between programs. A *coupling* between two mixed states ρ_1 and ρ_2 is a mixed state ρ that has ρ_1 and ρ_2 as marginals. (That is, $\mathrm{tr}_{\mathbf{X}_2}\,\rho = \rho_1$ and $\mathrm{tr}_{\mathbf{X}_1}\,\rho = \rho_2$ if ρ_1, ρ_2 are over $\mathbf{X}_1, \mathbf{X}_2$, respectively.) This is analogous to probabilistic couplings: a coupling of distributions μ_1, μ_2 is a distribution μ with marginals μ_1, μ_2. Note that couplings trivially always exist if ρ_1 and ρ_2 have the same trace (namely, $\rho := \rho_1 \otimes \rho_2 / \mathrm{tr}\,\rho_1$). Couplings become interesting when we put additional

constraints on the state ρ. For example, if we require the support of ρ to be in the subspace $C := \mathrm{span}\{|00\rangle, |11\rangle\}$, then $\rho_1 = \mathsf{proj}(|0\rangle)$ and $\rho_2 = \mathsf{proj}(|0\rangle)$ have a coupling (namely, $\rho = \mathsf{proj}(|00\rangle)$), as do $\rho_1 = \mathsf{proj}(|1\rangle)$ and $\rho_2 = \mathsf{proj}(|1\rangle)$ (namely, $\rho = \mathsf{proj}(|11\rangle)$), but not $\rho_1 = \mathsf{proj}(|0\rangle)$ and $\rho_2 = \mathsf{proj}(|1\rangle)$. Things become particularly interesting when ρ_1, ρ_2 are not pure states. E.g., $\rho_1 = \frac{1}{2}\mathsf{proj}(|0\rangle) + \frac{1}{2}\mathsf{proj}(|1\rangle)$ and $\rho_2 = \frac{1}{2}\mathsf{proj}(|0\rangle) + \frac{1}{2}\mathsf{proj}(|1\rangle)$ have such a coupling as well (namely, $\rho = \frac{1}{2}\mathsf{proj}(|00\rangle) + \frac{1}{2}\mathsf{proj}(|11\rangle)$ but $\rho := \rho_1 \otimes \rho_2$ is not a coupling with support in C).

Thus, a subspace such as C can be seen as a predicate describing the relationship of ρ_1, ρ_2. The states ρ_1, ρ_2 satisfy C iff there is a coupling with support in C. This idea leads to the following tentative definition of qRHL:

▶ **Definition 1** (qRHL, tentative, without expectations). *For subspaces A, B (i.e., spaces of quantum memories over $\mathbf{X}_1^{\mathbf{all}}\mathbf{X}_2^{\mathbf{all}}$), $\{A\}\,\mathfrak{c} \sim \mathfrak{d}\,\{B\}$ holds iff for any ρ_1, ρ_2 that have a coupling with support in A, the final states $[\![\mathfrak{c}]\!](\rho_1), [\![\mathfrak{d}]\!](\rho_2)$ have a coupling with support in B.*

However, it was noticed in [17] that the definition becomes easier to handle if we impose another condition on the couplings. Namely, the coupling should be separable, i.e., there should be no entanglement between the two systems corresponding to ρ_1, ρ_2. That is, the definition of qRHL used in [17] is Definition 1 with "coupling" replaced by "separable coupling". We will also adopt the separability condition in our definition of expectation-qRHL.[9]

So far, we have basically recapped the definition from [17]. However, that definition only allows us to express Hoare judgments that do not involve expectations since A and B in Definition 1 are subspaces (predicates), not expectations. To define expectation-qRHL, we follow the same idea, but instead of quantifying over only the initial states satisfying the precondition, we quantify over all initial states, and merely require that (the coupling of) the final states satisfies the postexpectation at least as much as (the coupling of the) initial states satisfy the preexpectation. That is:

▶ **Definition 2** (Expectation-qRHL (eqRHL), first attempt). *For expectations A, B, $\{A\}\,\mathfrak{c} \sim \mathfrak{d}\,\{B\}$ holds iff for any ρ_1, ρ_2 with separable coupling ρ, the final states $[\![\mathfrak{c}]\!](\rho_1), [\![\mathfrak{d}]\!](\rho_2)$ have a separable coupling ρ' such that $\mathrm{tr}\,\mathsf{A}\rho \leq \mathrm{tr}\,\mathsf{B}\rho'$. (Recall that $\mathrm{tr}\,\mathsf{A}\rho$ indicates how much ρ satisfies A, and analogously $\mathrm{tr}\,\mathsf{B}\rho'$, cf. Section 2.)*

For terminating programs $\mathfrak{c}, \mathfrak{d}$, this definition already works well. Unfortunately, if $\mathfrak{c}, \mathfrak{d}$ do not terminate with probability 1, we have some undesired effects: For example, assume that $\mathfrak{c} = \underline{\mathbf{skip}}$, and $\mathfrak{d}$ is a program that with probability $1-\varepsilon$ does nothing ($\underline{\mathbf{skip}}$), and but with probability ε does not terminate ($\underline{\mathbf{abort}}$). Then $\{\mathsf{A}\}\,\mathfrak{c} \sim \mathfrak{d}\,\{\mathsf{B}\}$ does not hold *for any* A, B. Why? The final states $[\![\mathfrak{c}]\!](\rho_1), [\![\mathfrak{d}]\!](\rho_2)$ have trace 1 and $1-\varepsilon$, respectively. Therefore there exists no coupling ρ' of $[\![\mathfrak{c}]\!](\rho_1), [\![\mathfrak{d}]\!](\rho_2)$. (It follows from the definition of couplings that the coupling must have the same trace as its marginals.) Hence $\{\mathsf{A}\}\,\mathfrak{c} \sim \mathfrak{d}\,\{\mathsf{B}\}$ does not hold. Similarly, $\{\mathsf{A}\}\,\mathfrak{c} \sim \mathfrak{d}\,\{\mathsf{B}\}$ does not hold whenever there are two input states ρ_1, ρ_2 such that $\mathfrak{c}, \mathfrak{d}$ terminate with different probabilities. (Even if this nontermination only occurs for input states for which A evaluates to 0!) This makes it near impossible to reason about non-terminating programs.[10]

[9] [17] was not able to prove the Frame rule without adding this separability condition. Our reasons for adopting the separability condition are slightly different: we do not have a Frame rule anyway, but for other elementary rules such as the rule Equal in [17] with quantum expectations and quantum variables, it is unclear how to prove them without the separability condition.

[10] Even if we are interested in a Hoare logic with total correctness, this behavior is undesired. Instead, we want that a nontermination with small probability simply introduces some small penalty in the expectations. For example, in the case of non-relational Hoare with total correctness, $\{(1-\varepsilon)\mathsf{id}\}\,\mathsf{A}\,\{\mathsf{id}\}$ means that A is nonterminating with probability $\leq \varepsilon$.

There are a number of approaches how one can circumvent this problem. E.g., one could allow ρ' to be a "subcoupling" instead of a coupling (i.e., its marginals do not have to equal $[\![\mathfrak{c}]\!](\rho_1), [\![\mathfrak{d}]\!](\rho_2)$ but only lower bound them);[11] the subcoupling always exists, even if the traces are not equal. However, we find that adding such "hacks" to the definition makes it more difficult to understand what the definition exactly does in case of non-terminating programs.

Instead, we choose an approach that makes non-termination explicit. That is, when a program does not terminate, we assign a specific state $|\bot\rangle$ to its output, and we allow expectation to explicitly refer to it (e.g., an expectation could assign value 1 to nontermination, and value 0 to termination). The denotation $[\![\mathfrak{c}]\!]^\bot$ defined on page 7 does exactly that. And expectations that live on a space that has an explicit nontermination-state $|\bot\rangle$ were introduced as $\bot$-expectations on page 5. This leads to the following definition:

▶ **Definition 3** (Expectation-qRHL, informal). *For $\bot$-expectations A, B, $\{A\}\,\mathfrak{c} \sim \mathfrak{d}\,\{B\}$ holds iff for any ρ_1, ρ_2 with separable coupling ρ, the final states $[\![\mathfrak{c}]\!]^\bot(\rho_1), [\![\mathfrak{d}]\!]^\bot(\rho_2)$ have a separable coupling ρ' such that $\operatorname{tr} \mathsf{A}\rho \le \operatorname{tr} \mathsf{B}\rho'$.*

Note that a coupling of $[\![\mathfrak{c}]\!]^\bot(\rho_1), [\![\mathfrak{d}]\!]^\bot(\rho_2)$ always exists since $[\![\cdot]\!]^\bot$ is trace-preserving. (Below, we will derive certain specific variants of eqRHL such as eqRHL with total correctness as specific cases of this definition. Also, we will see that subcouling-based definitions can be recovered as special cases in Lemma 9.) By plugging in the definition of couplings into Definition 3, we get the following precise definition:

▶ **Definition 4** (Expectation-qRHL, generic). *Let A, B be $\bot$-expectations and $\mathfrak{c}$, $\mathfrak{d}$ programs. Then $\{\mathsf{A}\}\,\mathfrak{c} \overset{\text{gen}}{\approx} \mathfrak{d}\,\{\mathsf{B}\}$ holds iff for any separable mixed $\bot$-bimemory ρ over $\mathbf{X}_1^{\mathbf{all}}, \mathbf{X}_2^{\mathbf{all}}$, there is a separable mixed $\bot$-bimemory ρ' over $\mathbf{X}_1^{\mathbf{all}}, \mathbf{X}_2^{\mathbf{all}}$ such that*

- $\operatorname{tr}_2 \rho' = [\![\mathfrak{c}]\!]^\bot(\operatorname{tr}_2 \rho)$.
- $\operatorname{tr}_1 \rho' = [\![\mathfrak{d}]\!]^\bot(\operatorname{tr}_1 \rho)$.
- $\operatorname{tr} \mathsf{A}\rho \le \operatorname{tr} \mathsf{B}\rho'$.

In this definition, $\mathbf{X}_1^{\mathbf{all}}, \mathbf{X}_2^{\mathbf{all}}$ are isomorphic copies of the set $\mathbf{X}^{\mathbf{all}}$ of variables. That is, while strictly speaking, $[\![\mathfrak{c}]\!]^\bot$ maps mixed $\bot$-memories over $\mathbf{X}^{\mathbf{all}}$ to mixed $\bot$-memories over $\mathbf{X}^{\mathbf{all}}$, we can also see it as mapping mixed $\bot$-memories over $\mathbf{X}_1^{\mathbf{all}}$ to mixed $\bot$-memories over $\mathbf{X}_1^{\mathbf{all}}$. Analogously for $\mathfrak{d}$ and $\mathbf{X}_2^{\mathbf{all}}$. We make use of this in the preceding definition when we apply $[\![\mathfrak{c}]\!]^\bot, [\![\mathfrak{d}]\!]^\bot$ to ρ_1, ρ_2, respectively. In the rest of the paper, we simply call a mixed state (ρ_1, ρ_2)-*coupling* if it is separable and has marginals ρ_1 and ρ_2.

Note that we defined $\bot$-expectations to be invariant under $\mathcal{E}_\bot \otimes \mathcal{E}_\bot$ (page 5), i.e., that they implicitly measure first whether the quantum memories are $|\bot\rangle$. Otherwise, we would not even have $\{\mathsf{A}\}\,\underline{\mathbf{skip}} \overset{\text{gen}}{\approx} \underline{\mathbf{skip}}\,\{\mathsf{A}\}$ (rule Skip below), for example if $\mathsf{A} := \mathsf{proj}(\frac{1}{\sqrt{2}}|0\rangle + \frac{1}{\sqrt{2}}|\bot\rangle)$. This is because even the program $\underline{\mathbf{skip}}$ measures whether the memory is $|\bot\rangle$ (by definition of $[\![\cdot]\!]^\bot$), so it may change the state if the memory is in a superposition between $|\bot\rangle$ and something else.

Partial/total correctness. The generic definition of eqRHL (Definition 4) allows us to explicitly express in our pre-/postexpectations how nontermination should be treated. While this allows for maximal generality, in practice it might be cumbersome to always have to specify explicitly how the expectations behave on $|\bot\rangle$. Instead, we define below three special cases of eqRHL that hardcode the treatment of $|\bot\rangle$.

[11] This is explored further in Section 4 for special cases of our definition.

The simplest case is eqRHL with total correctness: Here, nontermination is "forbidden", i.e., we assign value 0 to it. Recall from page 4 that for an expectation A, $\mathsf{A} \oplus 0_{\bot\bot}$ is the corresponding $\bot$-expectation. It assigns 0 to a state that has $|\bot\rangle$ in the left or right memory. Hence, eqRHL with total correctness simply means that all pre/postconditions are of the form $\mathsf{A} \oplus 0_{\bot\bot}$. The following definition specifies convenient syntax for this special case:

▶ **Definition 5** (Expectation-qRHL, total)**.** *Let* A, B *be expectations and* $\mathfrak{c}$, $\mathfrak{d}$ *programs. Then* $\{\mathsf{A}\}\,\mathfrak{c} \overset{\text{tot}}{\sim} \mathfrak{d}\,\{\mathsf{B}\}$ *iff* $\{\mathsf{A} \oplus 0_{\bot\bot}\}\,\mathfrak{c} \overset{\text{gen}}{\sim} \mathfrak{d}\,\{\mathsf{B} \oplus 0_{\bot\bot}\}$.

A second common variant of Hoare logic is "partial correctness". Here we allow nontermination, i.e., if a program does not terminate, we assign the value 1. That is, we use pre/postexpectations of the form $(\mathsf{A} \oplus 0_{\bot\bot}) + T$ where T assigns value 1 when the left or right memory is in state $|\bot\rangle$:

▶ **Definition 6** (Expectation-qRHL, partial)**.** *Let* A, B *be expectations and* $\mathfrak{c}$, $\mathfrak{d}$ *programs. Then* $\{\mathsf{A}\}\,\mathfrak{c} \overset{\text{par}}{\sim} \mathfrak{d}\,\{\mathsf{B}\}$ *iff* $\{(\mathsf{A} \oplus 0_{\bot\bot}) + T\}\,\mathfrak{c} \overset{\text{gen}}{\sim} \mathfrak{d}\,\{(\mathsf{B} \oplus 0_{\bot\bot}) + T\}$ *where* $T := \big(\mathsf{proj}(|\bot_1\rangle) \otimes P_{\not\downarrow}\big) + \big(P_{\not\downarrow} \otimes \mathsf{proj}(|\bot_2\rangle)\big) + \mathsf{proj}(|\bot_1\rangle \otimes |\bot_2\rangle)$.

Unfortunately, this definition does not necessarily behave as we would like. E.g., if both $\mathfrak{c}$ and $\mathfrak{d}$ terminate with probability $\leq \frac{1}{2}$ on all inputs, then $\{\mathsf{A}\}\,\mathfrak{c} \overset{\text{par}}{\sim} \mathfrak{d}\,\{\mathsf{B}\}$ holds for any $\mathsf{A} \leq \mathsf{id}$, B. That is, any property holds with probability $\frac{1}{2}$ for those programs which is not what we would expect! Why does this happen? Since $[\![\mathfrak{c}]\!]^{\bot}(\rho_1), [\![\mathfrak{d}]\!]^{\bot}(\rho_2)$ are 50% nontermination, we can "match up" the nonterminating part of $[\![\mathfrak{c}]\!]^{\bot}(\rho_1)$ with the terminating part of $[\![\mathfrak{d}]\!]^{\bot}(\rho_2)$ and vice versa in the coupling ρ' of the output states. Then $\operatorname{tr} T\rho' = 1$ and thus $\{\mathsf{A}\}\,\mathfrak{c} \overset{\text{par}}{\sim} \mathfrak{d}\,\{\mathsf{B}\}$ holds. The problem here is that we treat nontermination as a "wildcard" that is allowed to match any behavior of the other program. While there may be valid use cases for such a notation, we believe that in most cases this is not what we want.

Instead, we define a notion we call "semipartial". In this eqRHL-variant, we allow nontermination, but only when it occurs in the two programs "in sync". I.e., we consider pre/postexpectations that assign 1 to $|\bot\rangle \otimes |\bot\rangle$, but 0 to a state where one program has nonterminated and the other has terminated. Formally:

▶ **Definition 7** (Expectation-qRHL, semipartial)**.** *Let* A, B *be expectations and* $\mathfrak{c}$, $\mathfrak{d}$ *programs. Then* $\{\mathsf{A}\}\,\mathfrak{c} \overset{\text{semi}}{\sim} \mathfrak{d}\,\{\mathsf{B}\}$ *iff*

$$\{(\mathsf{A} \oplus 0_{\bot\bot}) + \mathsf{proj}(|\bot_1\rangle \otimes |\bot_2\rangle)\}\,\mathfrak{c} \overset{\text{gen}}{\sim} \mathfrak{d}\,\{(\mathsf{B} \oplus 0_{\bot\bot}) + \mathsf{proj}(|\bot_1\rangle \otimes |\bot_2\rangle)\}.$$

Pure initial states. In many cases, it is much easier to work with the definition of eqRHL correctness if one can assume that the initial states of $\mathfrak{c}, \mathfrak{d}$ are pure states, and that the initial coupling is the tensor product of those states. (No nontrivial correlations.) The following lemma shows that we can do so without loss of generality:

▶ **Lemma 8.** *Let* A, B *be* $\bot$*-expectations and* $\mathfrak{c}$, $\mathfrak{d}$ *programs. Then* $\{\mathsf{A}\}\,\mathfrak{c} \overset{\text{gen}}{\sim} \mathfrak{d}\,\{\mathsf{B}\}$ *holds iff*

- *for all unit quantum memories* ψ_1, ψ_2 *over* $\mathbf{X}_1$, $\mathbf{X}_2$, *respectively, there is a separable* $\bot$*-mixed bimemory* ρ' *over* $\mathbf{X}_1, \mathbf{X}_2$ *such that*
  - $\operatorname{tr}_2 \rho' = [\![\mathfrak{c}]\!]^{\bot}(\mathsf{proj}(\psi_1))$.
  - $\operatorname{tr}_1 \rho' = [\![\mathfrak{d}]\!]^{\bot}(\mathsf{proj}(\psi_2))$.
  - $\operatorname{tr} \mathsf{A}\,\mathsf{proj}(\psi_1 \otimes \psi_2) \leq \operatorname{tr} \mathsf{B}\rho'$.[12]

[12] Or equivalently, $\|\sqrt{\mathsf{A}}(\psi_1 \otimes \psi_2)\| \leq \operatorname{tr} \mathsf{B}\rho'$. Or $(\psi_1 \otimes \psi_2)^* \mathsf{A}(\psi_1 \otimes \psi_2) \leq \operatorname{tr} \mathsf{B}\rho'$.

- *for all unit quantum memories* ψ_1 *over* $\mathbf{X}_1$*, we have* $\operatorname{tr} \mathsf{A} \operatorname{proj}(\psi_1 \otimes |\bot_2\rangle) \leq \operatorname{tr} \mathsf{B}\big([\![\mathfrak{c}]\!]^\bot(\operatorname{proj}(\psi_1)) \otimes \operatorname{proj}(|\bot_2\rangle)\big)$.
- *for all unit quantum memories* ψ_2 *over* $\mathbf{X}_2$*, we have* $\operatorname{tr} \mathsf{A} \operatorname{proj}(|\bot_1\rangle \otimes \psi_2) \leq \operatorname{tr} \mathsf{B}\big(\operatorname{proj}(|\bot_1\rangle) \otimes [\![\mathfrak{d}]\!]^\bot(\operatorname{proj}(\psi_2))\big)$.
- $\operatorname{tr} \mathsf{A} \operatorname{proj}(|\bot_1\rangle \otimes |\bot_2\rangle) \leq \operatorname{tr} \mathsf{B} \operatorname{proj}(|\bot_1\rangle \otimes |\bot_2\rangle)$.

Equivalent reformulations. As discussed after Definition 2, an alternative means of trying to circumvent the problem that Definition 2 does handle nonterminating programs well is to use subcouplings instead of couplings.

Here, we show that the notions of eqRHL with partial and total correctness can be equivalently restated in terms of subcouplings (instead of the extended semantics $[\![\cdot]\!]^\bot$ over $\bot$-memories). However, we do not know such an equivalent reformulation for semipartial correctness.

▶ **Lemma 9.** *Let* A, B *be expectations and* $\mathfrak{c}$, $\mathfrak{d}$ *programs. Then* $\{\mathsf{A}\}\,\mathfrak{c} \overset{\text{tot}}{\sim} \mathfrak{d}\,\{\mathsf{B}\}$ *iff for any separable mixed bimemory* ρ *over* $\mathbf{X}_1^{\mathbf{all}}, \mathbf{X}_2^{\mathbf{all}}$*, there is a separable mixed bimemory* ρ' *over* $\mathbf{X}_1^{\mathbf{all}}, \mathbf{X}_2^{\mathbf{all}}$ *such that*

- $\operatorname{tr}_2 \rho' \leq [\![\mathfrak{c}]\!](\operatorname{tr}_2 \rho)$.
- $\operatorname{tr}_1 \rho' \leq [\![\mathfrak{d}]\!](\operatorname{tr}_1 \rho)$.
- $\operatorname{tr} \mathsf{A}\rho \leq \operatorname{tr} \mathsf{B}\rho'$.

▶ **Lemma 10.** *Let* A, B *be expectations and* $\mathfrak{c}$, $\mathfrak{d}$ *programs. Then* $\{\mathsf{A}\}\,\mathfrak{c} \overset{\text{par}}{\sim} \mathfrak{d}\,\{\mathsf{B}\}$ *holds iff for any separable mixed bimemory* ρ *over* $\mathbf{X}_1^{\mathbf{all}}, \mathbf{X}_2^{\mathbf{all}}$*, there is a separable mixed bimemory* ρ' *over* $\mathbf{X}_1^{\mathbf{all}}, \mathbf{X}_2^{\mathbf{all}}$ *such that*

- $\operatorname{tr}_2 \rho' \leq [\![\mathfrak{c}]\!](\operatorname{tr}_2 \rho)$.
- $\operatorname{tr}_1 \rho' \leq [\![\mathfrak{d}]\!](\operatorname{tr}_1 \rho)$.
- $\operatorname{tr} \rho' \geq \operatorname{tr}[\![\mathfrak{c}]\!](\operatorname{tr}_2 \rho) + \operatorname{tr}[\![\mathfrak{d}]\!](\operatorname{tr}_1 \rho) - \operatorname{tr} \rho$.
- $\operatorname{tr} \mathsf{A}\rho \leq \operatorname{tr} \mathsf{B}\rho' + \operatorname{tr} \rho - \operatorname{tr} \rho'$.

In this definition, $\operatorname{tr} \rho - \operatorname{tr} \rho' \geq 0$ is describe the nonterminating probability. Lemma 9 and Lemma 10 mean that total and partial correctness can be alternatively defined using the concept of subcouplings, without considering the $\bot$-extension of the expectations and programs.

▶ **Lemma 11.** *Let* A, B *be expectations and* $\mathfrak{c}$, $\mathfrak{d}$ *programs. Then* $\{\mathsf{A}\}\,\mathfrak{c} \overset{\text{tot}}{\sim} \mathfrak{d}\,\{\mathsf{B}\}$ *(resp.* $\{\mathsf{A}\}\,\mathfrak{c} \overset{\text{par}}{\sim} \mathfrak{d}\,\{\mathsf{B}\}$*) holds iff for all unit quantum memories* ψ_1*,* ψ_2 *over* $\mathbf{X}_1^{\mathbf{all}}$*,* $\mathbf{X}_2^{\mathbf{all}}$*, respectively, there is a separable mixed bimemory* ρ *over* $\mathbf{X}_1\mathbf{X}_2$ *such that*

- $\operatorname{tr}_2 \rho \leq [\![\mathfrak{c}]\!](\operatorname{proj}(\psi_1))$.
- $\operatorname{tr}_1 \rho \leq [\![\mathfrak{d}]\!](\operatorname{proj}(\psi_2))$.
- $\operatorname{tr} \mathsf{A}\operatorname{proj}(\psi_1 \otimes \psi_2) \leq \operatorname{tr} \mathsf{B}\rho$.[13]

  (resp. $\operatorname{tr} \mathsf{A}\operatorname{proj}(\psi_1 \otimes \psi_2) \leq \operatorname{tr} \mathsf{B}\rho + 1 - \operatorname{tr} \rho$ *and* $1 + \operatorname{tr} \rho \geq \operatorname{tr}[\![\mathfrak{c}]\!](\operatorname{proj}(\psi_1)) + \operatorname{tr}[\![\mathfrak{d}]\!](\operatorname{proj}(\psi_2))$*.)*

5 Description of the rules

We describe the rules of our logic one by one here. Recall that we essentially have four different logics: partial, semipartial, total, and the general case from which the former three are derived. To keep things readable, we only describe the rules for the partial, semipartial, and total case here. (Listed in Figure 1.) In the full version, we state and prove the rules

[13] Or equivalently, $\|\sqrt{\mathsf{A}}(\psi_1 \otimes \psi_2)\| \leq \operatorname{tr} \mathsf{B}\rho$. Or $(\psi_1 \otimes \psi_2)^* \mathsf{A}(\psi_1 \otimes \psi_2) \leq \operatorname{tr} \mathsf{B}\rho$.

Skip
$$\{\mathsf{A}\}\,\underline{\mathbf{skip}} \overset{\text{any}}{\simeq} \underline{\mathbf{skip}}\,\{\mathsf{A}\}$$

Apply1
$$\big\{(U\,\mathbf{on}\,\mathbf{X}_1)^*\mathsf{A}(U\,\mathbf{on}\,\mathbf{X}_1)\big\}\,\underline{\mathbf{apply}}\ U\ \underline{\mathbf{to}}\ \mathbf{X} \overset{\text{any}}{\simeq} \underline{\mathbf{skip}}\,\{\mathsf{A}\}$$

ExFalso
$$\{0\}\,\mathfrak{c} \overset{\text{any}}{\simeq} \mathfrak{d}\,\{\mathsf{B}\}$$

Init1
$$\big\{\mathsf{id}_{\mathbf{X}_1}\otimes(\psi^*\otimes\mathsf{id}_{\neg\mathbf{X}_1})\mathsf{A}(\psi\otimes\mathsf{id}_{\neg\mathbf{X}_1})\big\}\,\mathbf{X}\leftarrow\psi \overset{\text{any}}{\simeq} \underline{\mathbf{skip}}\,\{\mathsf{A}\}$$

Seq
$$\frac{\{\mathsf{A}\}\,\mathfrak{c}_1 \overset{\text{any}}{\simeq} \mathfrak{d}_1\,\{\mathsf{B}\}\qquad \{\mathsf{B}\}\,\mathfrak{c}_2 \overset{\text{any}}{\simeq} \mathfrak{d}_2\,\{\mathsf{C}\}}{\{\mathsf{A}\}\,\mathfrak{c}_1;\mathfrak{c}_2 \overset{\text{any}}{\simeq} \mathfrak{d}_1;\mathfrak{d}_2\,\{\mathsf{C}\}}$$

Conseq
$$\frac{\mathsf{A}'\le\mathsf{A}\qquad \{\mathsf{A}\}\,\mathfrak{c} \overset{\text{any}}{\simeq} \mathfrak{d}\,\{\mathsf{B}\}\qquad \mathsf{B}\le\mathsf{B}'}{\{\mathsf{A}'\}\,\mathfrak{c} \overset{\text{any}}{\simeq} \mathfrak{d}\,\{\mathsf{B}'\}}$$

Sym
$$\frac{\{\mathsf{A}\}\,\mathfrak{c} \overset{\text{any}}{\simeq} \mathfrak{d}\,\{\mathsf{B}\}}{\{\mathsf{SWAP}^*\cdot\mathsf{A}\cdot\mathsf{SWAP}\}\,\mathfrak{d} \overset{\text{any}}{\simeq} \mathfrak{c}\,\{\mathsf{SWAP}^*\cdot\mathsf{B}\cdot\mathsf{SWAP}\}}$$

Scale
$$\frac{\{\mathsf{A}\}\,\mathfrak{c} \overset{\text{any}}{\simeq} \mathfrak{d}\,\{\mathsf{B}\}\qquad \lambda\in[0,1]}{\{\lambda\mathsf{A}\}\,\mathfrak{c} \overset{\text{any}}{\simeq} \mathfrak{d}\,\{\lambda\mathsf{B}\}}$$

If1
$$\frac{\{\mathsf{A}_T\}\,\mathfrak{c}_T \overset{\text{any}}{\simeq} \mathfrak{d}\,\{\mathsf{B}\}\qquad \{\mathsf{A}_F\}\,\mathfrak{c}_F \overset{\text{any}}{\simeq} \mathfrak{d}\,\{\mathsf{B}\}}{\{\downarrow^*_{\mathsf{true}}(\mathsf{A}_T)+\downarrow^*_{\mathsf{false}}(\mathsf{A}_F)\}\,\underline{\mathbf{if}}\ M[\mathbf{X}]\ \underline{\mathbf{then}}\ \mathfrak{c}_T\ \underline{\mathbf{else}}\ \mathfrak{c}_F \overset{\text{any}}{\simeq} \mathfrak{d}\,\{\mathsf{B}\}}$$

JointIf9
$$\frac{\{\mathsf{A}_{t,u}\}\,\mathfrak{c}_t \overset{\text{any}}{\simeq} \mathfrak{d}_u\,\{\mathsf{B}\}\text{ for }t,u\in\{\mathsf{true},\mathsf{false}\}}{\Big\{\sum_{t,u\in\{\mathsf{true},\mathsf{false}\}}\downarrow^*_{t,u}(\mathsf{A}_{t,u})\Big\}\,\underline{\mathbf{if}}\ M[\mathbf{X}]\ \underline{\mathbf{then}}\ \mathfrak{c}_{\mathsf{true}}\ \underline{\mathbf{else}}\ \mathfrak{c}_{\mathsf{false}} \overset{\text{any}}{\simeq} \underline{\mathbf{if}}\ N[\mathbf{Y}]\ \underline{\mathbf{then}}\ \mathfrak{d}_{\mathsf{true}}\ \underline{\mathbf{else}}\ \mathfrak{d}_{\mathsf{false}}\,\{\mathsf{B}\}}$$

JointIf
$$\frac{\{\mathsf{A}_{\mathsf{true}}\}\,\mathfrak{c}_{\mathsf{true}} \overset{\text{any}}{\simeq} \mathfrak{d}_{\mathsf{true}}\,\{\mathsf{B}\}\qquad \{\mathsf{A}_{\mathsf{false}}\}\,\mathfrak{c}_{\mathsf{false}} \overset{\text{any}}{\simeq} \mathfrak{d}_{\mathsf{false}}\,\{\mathsf{B}\}}{\Big\{\downarrow^*_{\mathsf{true},\mathsf{true}}(\mathsf{A}_{\mathsf{true}})+\downarrow^*_{\mathsf{false},\mathsf{false}}(\mathsf{A}_{\mathsf{false}})\Big\}\,\underline{\mathbf{if}}\ M[\mathbf{X}]\ \underline{\mathbf{then}}\ \mathfrak{c}_{\mathsf{true}}\ \underline{\mathbf{else}}\ \mathfrak{c}_{\mathsf{false}} \overset{\text{any}}{\simeq} \underline{\mathbf{if}}\ N[\mathbf{Y}]\ \underline{\mathbf{then}}\ \mathfrak{d}_{\mathsf{true}}\ \underline{\mathbf{else}}\ \mathfrak{d}_{\mathsf{false}}\,\{\mathsf{B}\}}$$

While1
$$\frac{\big\{\mathsf{A}\big\}\,\mathfrak{c} \overset{\text{any}}{\simeq} \underline{\mathbf{skip}}\,\big\{\downarrow^*_{\mathsf{true}}(\mathsf{A})+\downarrow^*_{\mathsf{false}}(\mathsf{B})\big\}\qquad \underline{\mathbf{while}}\ M[\mathbf{X}]\ \underline{\mathbf{do}}\ \mathfrak{c}\text{ is terminating}}{\big\{\downarrow^*_{\mathsf{true}}(\mathsf{A})+\downarrow^*_{\mathsf{false}}(\mathsf{B})\big\}\,\underline{\mathbf{while}}\ M[\mathbf{X}]\ \underline{\mathbf{do}}\ \mathfrak{c} \overset{\text{any}}{\simeq} \underline{\mathbf{skip}}\,\{\mathsf{B}\}}$$

JointWhile
$$\frac{\begin{array}{c}\{\mathsf{A}\}\,\mathfrak{c} \overset{\text{any}}{\simeq} \mathfrak{d}\,\{\downarrow^*_{\mathsf{true},\mathsf{true}}(\mathsf{A})+\downarrow^*_{\mathsf{false},\mathsf{false}}(\mathsf{B})\}\\ \underline{\mathbf{while}}\ M[\mathbf{X}]\ \underline{\mathbf{do}}\ \mathfrak{c}\text{ or }\underline{\mathbf{while}}\ N[\mathbf{Y}]\ \underline{\mathbf{do}}\ \mathfrak{d}\text{ is terminating}\end{array}}{\{\downarrow^*_{\mathsf{true},\mathsf{true}}(\mathsf{A})+\downarrow^*_{\mathsf{false},\mathsf{false}}(\mathsf{B})\}\,\underline{\mathbf{while}}\ M[\mathbf{X}]\ \underline{\mathbf{do}}\ \mathfrak{c} \overset{\text{any}}{\simeq} \underline{\mathbf{while}}\ N[\mathbf{Y}]\ \underline{\mathbf{do}}\ \mathfrak{d}\,\{\mathsf{B}\}}$$

Figure 1 Rules for total/semipartial/partial eqRHL. In these rules, "any" can be any of "tot", "semi", "par". For any = par, the termination condition in While1 can be replaced by $\mathsf{A}\le\mathsf{id}$, and for any = par, semi the termination condition in JointWhile can be replaced by $\mathsf{A}\le\mathsf{id}$. We refer to the symmetric rules of Apply1, Init1, If1, and While1 (obtained by applying Sym) as Apply2, Init2, If2, and While2.

in the general case. The rules in Figure 1 are then simple consequences of the rules in the general case. The sole exception are rules related to while-loops: here not all of the partial, semipartial, total case follow directly from the general while rule. Those cases that do not follow are proved separately.

5.1 Structural rules

First, we mention the "structural" rules, i.e., rules that do not related to a specific language construct. There is Sym for exchanging the two programs. (In this rule, $\mathsf{SWAP} : \ell^2[\mathbf{X}_2^{\mathbf{all}}] \otimes \ell^2[\mathbf{X}_1^{\mathbf{all}}] \mapsto \ell^2[\mathbf{X}_1^{\mathbf{all}}] \otimes \ell^2[\mathbf{X}_2^{\mathbf{all}}]$ is the unitary operator $\mathsf{SWAP}_\perp : (\psi \otimes \phi) = \phi \otimes \psi.$) ExFalso allows us to show anything from an impossible preexpectation. Seq allows us to analyze the sequential composition of programs. Conseq allows us to weaken a judgment. (The preexpectation can be replaced by a smaller preexpectation, and the postexpectation can be replaced by a larger preexpectation. $\leq$ is the Loewner order). And finally, Scale allows us to scale pre- and postexpectations by a scalar factor.

5.2 One-sided rules

Conceptually simplest are the one-sided rules, i.e., rules that have **skip** on the right (or left) hand side. By combining them with Seq, we can prove facts about pairs of programs one statement at the time. Here, we only describe the rules with **skip** on the right side, the other case is analogous.

Apply: First, consider the Apply1 rule. It is stated (like all our rules), in a backward reasoning style, i.e., for any postexpectation A, the tells us the corresponding preexpectation, here $(U \mathbf{on} \mathbf{X}_1)^* \mathsf{A} (U \mathbf{on} \mathbf{X}_1)$. (Recall that $U \mathbf{on} \mathbf{X}_1$ denotes U applied to $\mathbf{X}_1$.) This is quite intuitive: the left program applies $U \mathbf{on} \mathbf{X}_1$, so the preexpectation corresponding to the postexpectation A is what we get if we apply $U \mathbf{on} \mathbf{X}_1$ and then compute the preexpectation, i.e., $(U \mathbf{on} \mathbf{X}_1)^* \mathsf{A} (U \mathbf{on} \mathbf{X}_1)$. (And it is $(U \mathbf{on} \mathbf{X}_1)^* \mathsf{A} (U \mathbf{on} \mathbf{X}_1)$ and not $\mathsf{A}(U \mathbf{on} \mathbf{X}_1)$ because the latter is not Hermitian and thus not a valid expectation.)

A toy example how to apply this rule: we want to what $\mathbf{x}$ has to be so that it will be $|0\rangle$ after applying a Hadamard H. Thus our postexpectation is $\mathsf{proj}(|0\rangle) \mathbf{on} \mathbf{x}_1$. Applying rule Apply1, we get that $\{\mathsf{B}\}\, \underline{\mathbf{apply}}\ H\ \underline{\mathbf{to}}\ \mathbf{x} \overset{\text{any}}{\sim} \underline{\mathbf{skip}}\, \{\mathsf{proj}(|0\rangle) \mathbf{on} \mathbf{x}_1\}$ for $\mathsf{B} := (H \mathbf{on} \mathbf{x}_1)^* (\mathsf{proj}(|0\rangle) \mathbf{on} \mathbf{x}_1)(H \mathbf{on} \mathbf{x}_1)$. (Here $\overset{\text{any}}{\sim}$ can be any of $\overset{\text{tot}}{\sim}$, $\overset{\text{semi}}{\sim}$, $\overset{\text{par}}{\sim}$.) A simple calculation reveals: $\mathsf{B} = \big(H^* \mathsf{proj}(|0\rangle) H\big) \mathbf{on} \mathbf{x}_1 = \mathsf{proj}(|+\rangle) \mathbf{on} \mathbf{x}_1$. Thus we learned (unsurprisingly) that to get $|0\rangle$, we need to start out with $|+\rangle$.

Init: The rule Init1 rule stated in a similar backwards reasoning way as Apply1, but the preexpectation is somewhat less intuitive. We will illustrate it with a toy example. Assume we want to know what the probability is to measure $|0\rangle$ after initializing a variable $\mathbf{x}$ with $|+\rangle$. That is, our postexpectation is $\mathsf{A} := \mathsf{proj}(|0\rangle) \mathbf{on} \mathbf{x}_1$ and our left program is $\mathbf{x} \leftarrow |+\rangle$. We ask for a suitable preexpectation B in $\{\mathsf{B}\}\, \mathbf{x} \leftarrow |+\rangle \overset{\text{any}}{\sim} \underline{\mathbf{skip}}\, \{\mathsf{A}\}$. The Init1 rule gives us $\mathsf{B} = \mathsf{id}_{\mathbf{x}_1} \otimes \big(\langle +| \otimes \mathsf{id}_{\neg \mathbf{x}_1}\big) \mathsf{A} \big(|+\rangle \otimes \mathsf{id}_{\neg \mathbf{x}_1}\big)$. (Here, $\neg \mathbf{X}_1 := \mathbf{X}_1^{\mathbf{all}} \mathbf{X}_2^{\mathbf{all}} \setminus \mathbf{X}_1$.) By definition of A, we have that $\big(\langle +| \otimes \mathsf{id}_{\neg \mathbf{x}_1}\big) \mathsf{A} \big(|+\rangle \otimes \mathsf{id}_{\neg \mathbf{x}_1}\big) = \langle +|\mathsf{proj}(|0\rangle)|+\rangle \otimes \mathsf{id}_{\neg \mathbf{x}_1} = \frac{1}{2} \mathsf{id}_{\neg \mathbf{x}_1}$. Note that this is not an expectation in our sense because it is an operator on all variable *but* $\mathbf{x}_1$. But by tensoring with $\mathsf{id}_{\mathbf{x}_1}$, we get the expectation $\mathsf{B} = \frac{1}{2}\mathsf{id}$. Thus the preexpectation is $\frac{1}{2}\mathsf{id}$ which intuitively means that, no matter what the initial state, the probability of measuring $|0\rangle$ will be $\frac{1}{2}$, as we would expect.

If: The rule rule If1 allows us to prove a judgment about an if-statement from judgments about the then- and the else-branch. If the preexpectations from the then- and else-branch are A_T and A_F, then the preexpectations for the if-statement is $\downarrow^*_{\mathsf{true}}(\mathsf{A}_T) + \downarrow^*_{\mathsf{false}}(\mathsf{A}_F)$. (Here

$\downarrow_t^*(A) := (M_t \,\mathbf{on}\, \mathbf{X}_1)^* A (M_t \,\mathbf{on}\, \mathbf{X}_1)$.) This is natural since $\downarrow_{\mathsf{true}}^*(\mathsf{A}_T)$ is A_T restricted to the case where the conditional holds, and $\downarrow_{\mathsf{false}}^*(\mathsf{A}_F)$ is A_F restricted to the case where the conditional does not hold.

A toy example: We want to show $\{\mathsf{id}\}$ **if** $M[\mathbf{x}]$ **then apply** X **to** $\mathbf{x}$ **else skip** $\overset{\text{any}}{\sim}$ **skip** $\{\mathsf{B}\}$ with $\mathsf{B} := \mathsf{proj}(|0\rangle) \,\mathbf{on}\, \mathbf{x}_1$. Here M is a computational basis measurement ($M_{\mathsf{true}} = \mathsf{proj}(|1\rangle)$, $M_{\mathsf{false}} = \mathsf{proj}(|0\rangle)$), and X is the pauli-X matrix (quantum bit flip). That is, with probability 1 (preexpectation is id), if we measure $\mathbf{x}$ in the computational basis, and, in case of outcome 1 flip it, we get $|0\rangle$ (postexpectation B). We derive easily (using rules Apply1 and Skip) that $\{\mathsf{proj}(|1\rangle) \,\mathbf{on}\, \mathbf{x}_1\}$ **apply** X **to** $\mathbf{x}$ $\overset{\text{any}}{\sim}$ **skip** $\{\mathsf{B}\}$ and $\{\mathsf{proj}(|0\rangle) \,\mathbf{on}\, \mathbf{x}_1\}$ **skip** $\overset{\text{any}}{\sim}$ **skip** $\{\mathsf{B}\}$. From the If1 rule, we then get $\{\mathsf{A}\}$ **if** $M[\mathbf{x}]$ **then apply** X **to** $\mathbf{x}$ **else skip** $\overset{\text{any}}{\sim}$ **skip** $\{\mathsf{B}\}$ with $\mathsf{A} = \downarrow_{\mathsf{true}}^*(\mathsf{proj}(|1\rangle) \,\mathbf{on}\, \mathbf{x}_1) + \downarrow_{\mathsf{false}}^*(\mathsf{proj}(|0\rangle) \,\mathbf{on}\, \mathbf{x}_1)$. Thus $\mathsf{A} = \mathsf{proj}(|1\rangle) \,\mathbf{on}\, \mathbf{x}_1 + \mathsf{proj}(|0\rangle) \,\mathbf{on}\, \mathbf{x}_1 = \mathsf{id}$, as desired.

While: The While1 rule is similar to the If1 rule. (The preexpectation in the conclusion has the same form.) The main difference is that we need to guess the invariant A because the postexpectation in the premise contains A. The rule also requires us to prove first that the loop is terminating (except in the case of partial correctness). Since this is a statement about a single program (non-relational), it can be shown using existing approaches (e.g., [14]) and is outside the scope of this paper.

5.3 Two-sided rules

The one-sided rules discussed in the previous section allow us to analyze two programs one statement at a time. However, they are not sufficient if want to analyze the relationship of two programs that go in lockstop. (E.g., two while loops that always take the same decision whether to terminate.) For handling if- and while-statements that are in sync, our logic provides the two-sided rules JointIf9, JointIf, and JointWhile. Notice that there are not two-sided analogues to Apply1 and Init1. This is because the resulting rule would be no different from using the one-sided rule twice. However, when random choices happen, two-sided rules are useful. In our case, this happens in if- and while-statements because the measurement of the loop-condition introduces randomness.

If: The JointIf9 rule allows us to compute the preexpectation of two if-statements. It is analogous to the If1 rule, except that the resulting preexpectation is of the form $\sum_{t,u\in\{\mathsf{true},\mathsf{false}\}} \downarrow_{t,u}^*(\mathsf{A}_{t,u})$. (Here $\downarrow_{t,u}^*(\mathsf{A}) := \big((M_t \,\mathbf{on}\, \mathbf{X}_1) \otimes (N_t \,\mathbf{on}\, \mathbf{Y}_2)\big)^* \mathsf{A} \big((M_t \,\mathbf{on}\, \mathbf{X}_1) \otimes (N_t \,\mathbf{on}\, \mathbf{Y}_2)\big)$.) That is, the preexpectations are restricted to all four combinations of true/false for the two if-conditions. And, consequently, we have a premise for each of those four cases. (The rule is called JointIf9 because, in the general case, there are nine cases, due to explicit treatment of non-terminating cases.) For convenience, we additionally state rule JointIf which considers only the cases where the two if-statements are in sync (true/true or false/false).

While: Similar to JointIf, the JointWhile rule allows us to reason about while loops that are in sync. Like with While1, in contrast to JointIf, we need to guess the invariant A. For an example, see Section 6. One difference with the While1 rule is that While1 requires us to prove termination in the semipartial and total case (not in the partical case), while JointWhile requires us to prove termination only in the total case. (Intuitively, this is because in the semipartial case, termination is not required, it is only required that both programs terminate with the same probability.)

6 Example: Quantum Zeno effect

Motivation. In this section, we study (one specific incarnation of) the quantum Zeno effect as an example of application of our logic. The Zeno effect implies that the following processes have the same effect:

- Start with a qubit in state $|0\rangle$. Apply a continuous rotation (with angular velocity ω) to it. (Thus, after time t, the state will have rotated by angle ωt.)
- Start with a qubit in state $|0\rangle$. Continuously observe the state. Namely, at time t, measure whether the qubit has rotated by angle ωt.

The quantum Zeno effect implies that in both processes, the state evolves in the same way (and that the measurement in the second situation always gives answer "yes"). Notice that this means that the measurements can be used to rotate the state.

In our formalization, we will consider the discrete version of this phenomenon: The rotation is split into n rotations by a small angle, and the continuous measurement consists of n measurements. In the limit $n \to \infty$, both processes yield the same state, but if we consider the situation for a concrete value of n, the result of the processes will be slightly different. (And the difference can be quantified in terms of n.) This makes this example a prime candidate for our logic: We want to compare two processes (hence we need relational Hoare logic), but the processes are not exactly equivalent (hence we cannot use qRHL from [17]) but only close to equivalent (and the "amount of equivalence" can be expressed using expectations).

Formalizing the processes. We now formalize the two processes as programs in our language. Let $n \geq 1$ be an integer.

In the first process, we have a continuous rotation, broken down into n small rotations. For simplicity, we will rotate by the angle $\pi/2$ within n steps, thus each small rotation rotates by angle $\frac{\pi}{2n}$. This is described by the rotation matrix $R := \begin{pmatrix} \cos\frac{\pi}{2n} & -\sin\frac{\pi}{2n} \\ \sin\frac{\pi}{2n} & \cos\frac{\pi}{2n} \end{pmatrix}$. Let $\mathbf{y}$ be a variable of type $\{0,1\}$ (i.e., the qubit that is rotated). In order to apply the rotation n times, we will need a counter $\mathbf{x}$ for the while loop. Let $\mathbf{x}$ be a variable of type $\mathbb{Z}$. We will have a loop that continues while (informally speaking) $\mathbf{x} < n$. This is formalized by the projector $P_{<n}$ onto states $|i\rangle$ with $i < n$. I.e., $P_{<n} := \sum_{-\infty<i<n} \mathsf{proj}(|i\rangle)$. In slight abuse of notation, we also write $P_{<n}$ for the binary measurement with Kraus operators $\{P_{<n}, \mathsf{id} - P_{<n}\}$. Furthermore, we need to increase the counter. For this let INCR be the unitary on $\ell^2(\mathbb{Z})$ with $\mathsf{INCR}|i\rangle \mapsto |i+1\rangle$ Then the program that initializes $\mathbf{y}$ with $|0\rangle$ and then applies the rotation R n times can be written as:

$$\mathbf{c} := \mathbf{x} \leftarrow |0\rangle;\ \mathbf{y} \leftarrow |0\rangle;\ \underline{\textbf{while}}\ P_{<n}[\mathbf{x}]\ \underline{\textbf{do}}\ (\underline{\textbf{apply}}\ \mathsf{INCR}\ \underline{\textbf{to}}\ \mathbf{x};\ \underline{\textbf{apply}}\ R\ \underline{\textbf{to}}\ \mathbf{y}) \tag{1}$$

In the second process, instead of applying R, we measure the state in each iteration of the loop. In the first iteration, we expect the original state $\phi_0 := |0\rangle$, and after the i-th iteration, we expect the state $\phi_i := R\phi_{i-1}$ for $i \geq 1$. This can be done using the program $\underline{\textbf{if}}\ \mathsf{proj}(\phi_i)[\mathbf{y}]\ \underline{\textbf{then}}\ \underline{\textbf{skip}}\ \underline{\textbf{else}}\ \underline{\textbf{skip}}$ where we again write in slight abuse of notation $\mathsf{proj}(\phi_i)$ for the corresponding binary measurement. Since the if-statement first measures $\mathbf{y}$ and then executes one of the $\underline{\textbf{skip}}$-branches, this is effectively just a measurement. We abbreviate this as $\underline{\textbf{if}}\ \mathsf{proj}(\phi_i)[\mathbf{y}]$.

However, we cannot simply write $\underline{\textbf{if}}\ \mathsf{proj}(\phi_i)[\mathbf{y}]$ in our loop body, because i should be the value of $\mathbf{x}$. So we need to define the projector that projects onto ϕ_i when $\mathbf{x} = |i\rangle$. This is done by the following projector on $\ell^2[\mathbf{xy}]$: $P_\phi := \sum_i \mathsf{proj}(|i\rangle \otimes \phi_i)$. Then $\underline{\textbf{if}}\ P_\phi[\mathbf{y}]$ will measure whether $\mathbf{y}$ contains ϕ_i whenever $\mathbf{x}$ contains $|i\rangle$.

Armed with that notation, we can now formulate the second process as a program:

$$\mathfrak{d} := \mathbf{x} \leftarrow |0\rangle;\ \mathbf{y} \leftarrow |0\rangle;\ \underline{\textbf{while}}\ P_{<n}[\mathbf{x}]\ \underline{\textbf{do}}\ (\underline{\textbf{apply}}\ \mathsf{INCR}\ \underline{\textbf{to}}\ \mathbf{x};\ \underline{\textbf{if}}\ P_\phi[\mathbf{xy}]) \tag{2}$$

Equivalence of the programs. We claim that the two processes, i.e., the programs $\mathfrak{c}, \mathfrak{d}$ have approximately the same final state in $\mathbf{y}$. Having the same state can be expressed using the "quantum equality" described in Section 2. Specifically, the postexpectation $\mathsf{EQUAL}\,\mathbf{on}\,\mathbf{y}_1\mathbf{y}_2$ corresponds to $\mathbf{y}_1$ and $\mathbf{y}_2$ having the same state. For example, $\{\mathsf{id}\}\,\mathfrak{c} \overset{\text{tot}}{\sim} \mathfrak{d}\,\{\mathsf{EQUAL}\,\mathbf{on}\,\mathbf{y}_1\mathbf{y}_2\}$ implies that the final state of $\mathfrak{c}$ and $\mathfrak{d}$ is the same (if we trace out all variables except $\mathbf{y}_1, \mathbf{y}_2$).[14] The fact that the final states are approximately equal can be expressed by multiplying the preexpectation with a real number close to 1. Specifically, in our case we claim that

$$\{\varepsilon^n \cdot \mathsf{id}\}\,\mathfrak{c} \overset{\text{tot}}{\sim} \mathfrak{d}\,\{\mathsf{EQUAL}\,\mathbf{on}\,\mathbf{y}_1\mathbf{y}_2\} \tag{3}$$

Here $\varepsilon := (\cos\frac{\pi}{2n})^2$. This indeed means that the final states of $\mathfrak{c}$ and $\mathfrak{d}$ are the same asymptotically since $\varepsilon^n = (\cos\frac{\pi}{2n})^{2n} \xrightarrow{n\to\infty} 1$.

Warm up. Before we prove (3), we investigate a simpler case as a warm up. We investigate the special where $n = 3$, and instead of a while-loop, we simply repeat the loop body three times.

$$\begin{aligned}\mathfrak{c}' &:= \mathbf{y} \leftarrow |0\rangle;\underline{\textbf{apply}}\ R\ \underline{\textbf{to}}\ \mathbf{y};\underline{\textbf{apply}}\ R\ \underline{\textbf{to}}\ \mathbf{y};\underline{\textbf{apply}}\ R\ \underline{\textbf{to}}\ \mathbf{y}\\ \mathfrak{d}' &:= \mathbf{y} \leftarrow |0\rangle;\underline{\textbf{if}}\ \mathsf{proj}(\phi_1)[\mathbf{y}];\ \underline{\textbf{if}}\ \mathsf{proj}(\phi_2)[\mathbf{y}];\ \underline{\textbf{if}}\ \mathsf{proj}(\phi_3)[\mathbf{y}]\end{aligned} \tag{4}$$

We claim:

$$\{\varepsilon^3 \cdot \mathsf{id}\}\,\mathfrak{c}' \overset{\text{tot}}{\sim} \mathfrak{d}'\,\{\mathsf{EQUAL}\,\mathbf{on}\,\mathbf{y}_1\mathbf{y}_2\} \tag{5}$$

First, we strengthen the postcondition. Let $\mathsf{A}_3 := (\mathsf{proj}(\phi_3 \otimes \phi_3)\,\mathbf{on}\,\mathbf{y}_1\mathbf{y}_2)$. (This postcondition is intuitively what we expect to (approximately) hold at the end of the execution. It means that $\mathbf{y}_1$ and $\mathbf{y}_2$ are both in state ϕ_3, the result by rotating three times using R. Since $\phi_3 \otimes \phi_3$ is in the image of the projector EQUAL, it follows that $\mathsf{A}_3 \le (\mathsf{EQUAL}\,\mathbf{on}\,\mathbf{y}_1\mathbf{y}_2)$. By rule Conseq it is thus sufficient to show $\{\varepsilon^3 \cdot \mathsf{id}\}\,\mathfrak{c}' \overset{\text{tot}}{\sim} \mathfrak{d}'\,\{\mathsf{A}_3\}$. And by rule Seq, we can show that by the following sequence of Hoare judgments for some $\mathsf{A}_0, \mathsf{A}_1, \mathsf{A}_2$:

$$\left\{\varepsilon^3 \cdot \mathsf{id}\right\} \underset{\text{tot}}{\sim} \left\{\mathsf{A}_0\right\} \underset{\text{tot}}{\sim} \left\{\mathsf{A}_1\right\} \underset{\text{tot}}{\sim} \left\{\mathsf{A}_2\right\} \underset{\text{tot}}{\sim} \left\{\mathsf{A}_3\right\} \tag{6}$$

$$\begin{array}{l}\mathbf{y} \leftarrow |0\rangle\\ \mathbf{y} \leftarrow |0\rangle\end{array}\qquad \begin{array}{l}\underline{\textbf{apply}}\ R\ \underline{\textbf{to}}\ \mathbf{y}\\ \underline{\textbf{if}}\ \mathsf{proj}(\phi_1)[\mathbf{y}]\end{array}\qquad \begin{array}{l}\underline{\textbf{apply}}\ R\ \underline{\textbf{to}}\ \mathbf{y}\\ \underline{\textbf{if}}\ \mathsf{proj}(\phi_2)[\mathbf{y}]\end{array}\qquad \begin{array}{l}\underline{\textbf{apply}}\ R\ \underline{\textbf{to}}\ \mathbf{y}\\ \underline{\textbf{if}}\ \mathsf{proj}(\phi_3)[\mathbf{y}]\end{array}$$

(These are four judgments, we just use a more compact notation to put them in one line.) We will derive suitable values $\mathsf{A}_0, \mathsf{A}_1, \mathsf{A}_2$ by applying our rules backwards from the postcondition.

By applying rule Apply1, we get $\left\{\mathsf{A}_3'\right\}\underline{\textbf{apply}}\ R\ \underline{\textbf{to}}\ \mathbf{y} \overset{\text{tot}}{\sim} \underline{\textbf{skip}}\left\{\mathsf{A}_3\right\}$ where $\mathsf{A}_3' := (R^\dagger\,\mathbf{on}\,\mathbf{y}_1) \circ \mathsf{A}_3$ and where we use $A \circ B$ as an abbreviation for $ABA^\dagger$. And by rule If2 (using rule Skip for its premises), we get

$$\left\{(\mathsf{proj}(\phi_3)\,\mathbf{on}\,\mathbf{y}_2) \circ \mathsf{A}_3' + (1 - \mathsf{proj}(\phi_3)\,\mathbf{on}\,\mathbf{y}_2) \circ \mathsf{A}_3'\right\}\underline{\textbf{skip}} \overset{\text{tot}}{\sim} \underline{\textbf{if}}\ \mathsf{proj}(\phi_3)[\mathbf{y}]\left\{\mathsf{A}_3'\right\}.$$

[14] This is seen as follows: The judgment implies that the finals states are marginals of a state that is invariant under the projector $\mathsf{EQUAL}\,\mathbf{on}\,\mathbf{y}_1\mathbf{y}_2$, i.e., a state with support in the space $\mathbf{Y}_1 \equiv_{\mathsf{q}} \mathbf{Y}_2$. That means that this state is invariant under swapping $\mathbf{Y}_1, \mathbf{Y}_2$, and thus the marginals corresponding to $\mathbf{Y}_1$ and $\mathbf{Y}_2$ are equal.

The precondition is lower bounded by $A_2 := (\mathsf{proj}(\phi_3)\,\mathbf{on}\,\mathbf{y}_2) \circ \mathsf{A}'_3$. (The second term corresponds to the measurement failing to measure ϕ_3, in this case all is lost anyway, so we remove that term.) Hence (with rules Seq and Conseq), $\{\mathsf{A}_2\}$ **apply** R **to** $\mathbf{y} \overset{\text{tot}}{\sim}$ **if** $\mathsf{proj}(\phi_3)[\mathbf{y}]\,\{\mathsf{A}_3\}$ as desired in (6).

Analogously, we can instantiate

$$\mathsf{A}_1 := (\mathsf{proj}(\phi_2)\,\mathbf{on}\,\mathbf{y}_2) \circ (R^*\,\mathbf{on}\,\mathbf{y}_1) \circ \mathsf{A}_2 \quad \text{and} \quad \mathsf{A}_0 := (\mathsf{proj}(\phi_1)\,\mathbf{on}\,\mathbf{y}_2) \circ (R^*\,\mathbf{on}\,\mathbf{y}_1) \circ \mathsf{A}_1$$

in (6). We can simplify the expressions for $\mathsf{A}_0, \mathsf{A}_1, \mathsf{A}_2$ some more. We have

$$\begin{aligned}\mathsf{A}_2 &= (\mathsf{proj}\big(\phi_3\big)\,\mathbf{on}\,\mathbf{y}_2) \circ (R^*\,\mathbf{on}\,\mathbf{y}_1) \circ \big(\mathsf{proj}(\phi_3 \otimes \phi_3)\,\mathbf{on}\,\mathbf{y}_1\mathbf{y}_2\big) \\ &= \mathsf{proj}\big(R^*\phi_3 \otimes \mathsf{proj}(\phi_3)\phi_3\big)\,\mathbf{on}\,\mathbf{y}_1\mathbf{y}_2 = \mathsf{proj}(\phi_2 \otimes \phi_3)\,\mathbf{on}\,\mathbf{y}_1\mathbf{y}_2\end{aligned}$$

And

$$\begin{aligned}\mathsf{A}_1 &= (\mathsf{proj}\big(\phi_2\big)\,\mathbf{on}\,\mathbf{y}_2) \circ (R^*\,\mathbf{on}\,\mathbf{y}_1) \circ \big(\mathsf{proj}(\phi_2 \otimes \phi_3)\,\mathbf{on}\,\mathbf{y}_1\mathbf{y}_2\big) \\ &= \mathsf{proj}\big(R^*\phi_2 \otimes \mathsf{proj}(\phi_2)\phi_3\big)\,\mathbf{on}\,\mathbf{y}_1\mathbf{y}_2 = \varepsilon\,\mathsf{proj}(\phi_1 \otimes \phi_2)\,\mathbf{on}\,\mathbf{y}_1\mathbf{y}_2.\end{aligned}$$

(Note the slight difference: instead of $\mathsf{proj}(\phi_3)\phi_3$ have $\mathsf{proj}(\phi_2)\phi_3$ here, which simplifies to $\phi_2 \cdot \phi_2^*\phi_3 = \phi_2 \cdot \sqrt{\varepsilon}$.) Analogously

$$\mathsf{A}_0 = \varepsilon^2\,\mathsf{proj}(\phi_0 \otimes \phi_1)\,\mathbf{on}\,\mathbf{y}_1\mathbf{y}_2.$$

It is left to show the first judgment in (6), namely $\{\varepsilon^3 \cdot \mathsf{id}\}\,\mathbf{y} \leftarrow |0\rangle \overset{\text{tot}}{\sim} \mathbf{y} \leftarrow |0\rangle\,\{\mathsf{A}_0\}$. By rules Init1 and Init2 (starting from the right), we have

$$\begin{aligned}\Big\{\varepsilon^3 \cdot \mathsf{id}\Big\} &\overset{(**)}{=} \Big\{\mathsf{id}_{\mathbf{y}_2} \otimes \big(\langle 0|_{\mathbf{y}_2} \otimes \mathsf{id}_{\neg\mathbf{y}_2}\big) \circ \varepsilon^2(\mathsf{proj}(\phi_1)\,\mathbf{on}\,\mathbf{y}_2)\Big\}\ \underline{\mathbf{skip}}\ \overset{\text{tot}}{\sim} \mathbf{y} \leftarrow |0\rangle \\ &\Big\{\varepsilon^2(\mathsf{proj}(\phi_1)\,\mathbf{on}\,\mathbf{y}_2)\Big\} \overset{(*)}{=} \Big\{\mathsf{id}_{\mathbf{y}_1} \otimes \big(\langle 0|_{\mathbf{y}_1} \otimes \mathsf{id}_{\neg\mathbf{y}_1}\big) \circ \mathsf{A}_0\Big\}\,\mathbf{y} \leftarrow |0\rangle \overset{\text{tot}}{\sim} \underline{\mathbf{skip}}\,\Big\{\mathsf{A}_0\Big\}. \quad (7)\end{aligned}$$

Here $(*)$ uses that $\phi_0 = |0\rangle$ and thus $\langle 0|\mathsf{proj}(\phi_0)|0\rangle = 1$, and $(**)$ uses that $\phi_1^*\phi_0 = \sqrt{\varepsilon}$ and thus $\langle 0|\mathsf{proj}(\phi_1)|0\rangle = \varepsilon$.

The first judgment in (6) then follows by rule Seq.

This completes the analysis, we have shown (5).

Analysis of the while-programs. Given the experiences from the analysis of the special case (the programs from (4)), we now can solve the original problem, namely analyzing the programs $\mathfrak{c}, \mathfrak{d}$ from (1),(2).

As before, we can replace the postcondition in (3) by the stronger postcondition $\mathsf{B} := (\mathsf{proj}(|n\rangle) \otimes |n\rangle \otimes \phi_n \otimes \phi_n)\,\mathbf{on}\,\mathbf{x}_1\mathbf{x}_2\mathbf{y}_1\mathbf{y}_2)$. By rule Conseq, it is sufficient to show $\{\varepsilon^n \cdot \mathsf{id}\}\,\mathfrak{c} \overset{\text{tot}}{\sim} \mathfrak{d}\,\{\mathsf{B}\}$. By rule Seq, this follows if we can show

$$\Big\{\varepsilon^n \cdot \mathsf{id}\Big\}\ \begin{matrix}\mathbf{x} \leftarrow |0\rangle \\ \overset{\text{tot}}{\sim}\ \mathbf{x} \leftarrow |0\rangle\end{matrix}\ \Big\{\mathsf{D}\Big\}\ \begin{matrix}\mathbf{y} \leftarrow |0\rangle \\ \overset{\text{tot}}{\sim}\ \mathbf{y} \leftarrow |0\rangle\end{matrix}\ \Big\{\mathsf{C}\Big\}\ \begin{matrix}\text{while}_{\mathfrak{c}} \\ \overset{\text{tot}}{\sim}\ \text{while}_{\mathfrak{d}}\end{matrix}\ \Big\{\mathsf{B}\Big\} \quad (8)$$

with

$$\begin{aligned}\text{while}_{\mathfrak{c}} &:= \underline{\mathbf{while}}\ P_{<n}[\mathbf{x}]\ \underline{\mathbf{do}}\ (\underline{\mathbf{apply}}\ \mathsf{INCR}\ \underline{\mathbf{to}}\ \mathbf{x};\ \underline{\mathbf{apply}}\ R\ \underline{\mathbf{to}}\ \mathbf{y}) \\ \text{while}_{\mathfrak{d}} &:= \underline{\mathbf{while}}\ P_{<n}[\mathbf{x}]\ \underline{\mathbf{do}}\ (\underline{\mathbf{apply}}\ \mathsf{INCR}\ \underline{\mathbf{to}}\ \mathbf{x};\ \underline{\mathbf{if}}\ P_\phi[\mathbf{xy}])\end{aligned}$$

for suitably chosen expectations C, D.

To prove the last judgment $\{\mathsf{C}\}\,\text{while}_{\mathfrak{c}} \overset{\text{tot}}{\sim} \text{while}_{\mathfrak{d}}\,\{\mathsf{B}\}$ in (8), we use rule JointWhile. This rule requires us to come up with a loop invariant A. To understand what the right loop invariant is, we draw from our experiences in the special case. There, we had defined the expectations $\mathsf{A}_0, \dots, \mathsf{A}_3$, where A_i described the state of the programs right after the i-th application of $\underline{\textbf{apply}}\ R\ \underline{\textbf{to}}\ \mathbf{y}$ and $\underline{\textbf{if}}\ \mathsf{proj}(\phi_i)[\mathbf{y}]$. We had

$$\mathsf{A}_i = \varepsilon^{2-i}\,\mathsf{proj}(\phi_i \otimes \phi_{i+1})\,\mathbf{on}\,\mathbf{y}_1\mathbf{y}_2 \text{ for } i = 0,1,2 \qquad \text{and} \qquad \mathsf{A}_3 = \mathsf{proj}(\phi_3 \otimes \phi_3)\,\mathbf{on}\,\mathbf{y}_1\mathbf{y}_2$$

One sees easily that this would generalize as

$$\mathsf{A}_i = \varepsilon^{n-i-1}\,\mathsf{proj}(\phi_i \otimes \phi_{i+1})\,\mathbf{on}\,\mathbf{y}_1\mathbf{y}_2 \qquad \text{for } i < n$$
$$\text{and} \qquad \mathsf{A}_n = \mathsf{proj}(\phi_n \otimes \phi_n)\,\mathbf{on}\,\mathbf{y}_1\mathbf{y}_2$$

for values $n \neq 3$. Thus we expect that these expectations A_i also hold in the programs $\text{while}_{\mathfrak{c}}$, $\text{while}_{\mathfrak{d}}$ after the i-th iteration (or before the $(i+1)$-st iteration). Additionally, we keep track of the counter $\mathbf{x}$, which should be $|i\rangle$ after the i-th iteration (or before the $(i+1)$-st iteration). This would be expressed by the expectation $\mathsf{proj}(|i\rangle \otimes |i\rangle)\,\mathbf{on}\,\mathbf{x}_1\mathbf{x}_2$. Thus, for the i-th iteration, we use the "conjunction"

$$\mathsf{A}_i^{\mathbf{x}} := \mathsf{A}_i \cdot (\mathsf{proj}(|i\rangle \otimes |i\rangle)\,\mathbf{on}\,\mathbf{x}_1\mathbf{x}_2)$$
$$= \begin{cases} \varepsilon^{n-i-1}\,\mathsf{proj}(|i\rangle \otimes |i\rangle \otimes \phi_i \otimes \phi_{i+1})\,\mathbf{on}\,\mathbf{x}_1\mathbf{x}_2\mathbf{y}_1\mathbf{y}_2 & (i < n) \\ \mathsf{proj}(|n\rangle \otimes |n\rangle \otimes \phi_n \otimes \phi_n)\,\mathbf{on}\,\mathbf{x}_1\mathbf{x}_2\mathbf{y}_1\mathbf{y}_2 & (i = n) \end{cases}$$

(Note that $\cdot$ is not generally a sensible operation on expectations. But in this case, $\mathsf{fv}(\mathsf{A}_i) = \mathbf{y}_1\mathbf{y}_2$ and $\mathsf{fv}(\mathsf{proj}(|i\rangle \otimes |i\rangle)\,\mathbf{on}\,\mathbf{x}_1\mathbf{x}_2) = \mathbf{x}_1\mathbf{x}_2$, so the expectations commute and their product is again an expectation.)

The final loop invariant A is then the "disjunction" of the $\mathsf{A}_i^{\mathbf{x}}$ for $i = 0, \dots, n-1$, meaning that in every iteration, one of the A_i should hold. (We do not include $\mathsf{A}_i^{\mathbf{x}}$ with $i = n$ here because when applying the JointWhile rule, we only need the invariant to hold when the loop guard was passed.) We define $\mathsf{A} := \sum_{i=0}^{n} \mathsf{A}_i^{\mathbf{x}}$. (In general, summation is not a sensible operation representation of "disjunction", but in the present case, all summands are orthogonal.)

We have now derived a suitable candidate for the invariant A to use in rule JointWhile. We stress that the above argumentation (involving words like "disjunction" and "conjunction" of expectations, and claims that an expectation "holds" at a certain point) was not a formally well-defined argument, merely an explanation how we arrived at our specific choice for A. From the formal point of view, all we will need in the following are the definitions of $\mathsf{A}, \mathsf{A}_i^{\mathbf{x}}$. The rest of the argument above was semi-formal motivation.

We will now show the rightmost judgment in (8), namely $\{\mathsf{C}\}\,\text{while}_{\mathfrak{c}} \overset{\text{tot}}{\sim} \text{while}_{\mathfrak{d}}\,\{\mathsf{B}\}$ (for some suitable C). If we apply rule JointWhile (with A as defined above) to this, we get the premise[15]

$$\Big\{\mathsf{A}\Big\}\ \overbrace{\underbrace{\underline{\textbf{apply}}\ \mathsf{INCR}\ \underline{\textbf{to}}\ \mathbf{x};\ \underline{\textbf{apply}}\ R\ \underline{\textbf{to}}\ \mathbf{y}}_{\overset{\text{tot}}{\sim}\ \underline{\textbf{apply}}\ \mathsf{INCR}\ \underline{\textbf{to}}\ \mathbf{x};\ \underline{\textbf{if}}\ P_\phi[\mathbf{xy}]}}^{=:\text{body}_{\mathfrak{c}}}\ \Big\{\underbrace{P^{\text{both}}_{<n} \circ \mathsf{A} + (P^{\text{none}}_{<n}) \circ \mathsf{B}}_{=:\mathsf{C}'}\Big\} \tag{9}$$
$$=:\text{body}_{\mathfrak{d}}$$

[15] We also additionally get the premise that $\text{while}_{\mathfrak{c}}$ is terminating. This can be shown with techniques from prior work (e.g., [14]) and is quite obvious in the present case. Alternatively, we could have stated this example with respect to partial correctness instead of total correctness. In that case, we do not need to prove termination.

with $P^{\text{both}}_{<n} := P_{<n} \otimes P_{<n}\,\mathbf{on}\,\mathbf{x}_1\mathbf{x}_2$ and $P^{\text{none}}_{<n} := (\mathsf{id} - P_{<n}) \otimes (\mathsf{id} - P_{<n})\,\mathbf{on}\,\mathbf{x}_1\mathbf{x}_2$. (Here we write $A \circ B$ as an abbreviation for $ABA^\dagger$.) By applying rules If2, Apply2, and twice Apply1 (with Seq in between), we get

$$\big\{(\mathsf{INCR}\,\mathbf{on}\,\mathbf{x}_1) \circ (R\,\mathbf{on}\,\mathbf{y}_1) \circ (\mathsf{INCR}\,\mathbf{on}\,\mathbf{x}_2) \circ \mathsf{B}_2\big\}\,\mathrm{body}_{\mathfrak{c}} \overset{\text{tot}}{\sim} \mathrm{body}_{\mathfrak{d}}\,\big\{\mathsf{C}'\big\}$$

where $\mathsf{B}_2 := (P_\phi\,\mathbf{on}\,\mathbf{x}_2\mathbf{y}_2) \circ \mathsf{C}' + (\mathsf{id} - P_\phi\,\mathbf{on}\,\mathbf{x}_2\mathbf{y}_2) \circ \mathsf{C}'$. Since $\mathsf{B}_2 \geq (P_\phi\,\mathbf{on}\,\mathbf{x}_1\mathbf{y}_1) \circ \mathsf{C}'$, by rule Conseq we can weaken this to

$$\big\{\mathsf{A}'\big\}\,\mathrm{body}_{\mathfrak{c}} \overset{\text{tot}}{\sim} \mathrm{body}_{\mathfrak{d}}\,\big\{\mathsf{C}'\big\} \qquad \text{with} \qquad \mathsf{A}' := \underbrace{(\mathsf{INCR}\,\mathbf{on}\,\mathbf{x}_1) \circ (R\,\mathbf{on}\,\mathbf{y}_1) \circ (\mathsf{INCR}\,\mathbf{on}\,\mathbf{x}_2) \circ (P_\phi\,\mathbf{on}\,\mathbf{x}_2\mathbf{y}_2)}_{=:L} \circ \mathsf{C}'$$

If we can show that $\mathsf{A} \leq \mathsf{A}'$ then we have proven (9). By definition of $\mathsf{A}^{\mathbf{x}}_i$, L, R, P_ϕ, INCR, $P^{\text{both}}_{<n}$, we have

$$\begin{aligned} L \circ P^{\text{both}}_{<n} \circ \mathsf{A}^{\mathbf{x}}_i &= \varepsilon^{n-i-1}\,\mathsf{proj}(\mathsf{INCR}^*|i\rangle \otimes \mathsf{INCR}^*|i\rangle \otimes R^*\phi_i \otimes \mathsf{proj}(\phi_i)\phi_{i+1})\,\mathbf{on}\,\mathbf{x}_1\mathbf{x}_2\mathbf{y}_1\mathbf{y}_2 \\ &= \varepsilon^{n-i}\,\mathsf{proj}(|i-1\rangle \otimes |i-1\rangle \otimes \phi_{i-1} \otimes \phi_i)\,\mathbf{on}\,\mathbf{x}_1\mathbf{x}_2\mathbf{y}_1\mathbf{y}_2 = \mathsf{A}^{\mathbf{x}}_{i-1}. \end{aligned}$$

And $L \circ P^{\text{both}}_{<n} \circ \mathsf{A}^{\mathbf{x}}_n = 0$ since $P^{\text{both}}_{<n} \circ \mathsf{A}^{\mathbf{x}}_n = 0$. Thus $L \circ P^{\text{both}}_{<n} \circ \mathsf{A} = \sum_{i=0}^{n-1} A^{\mathbf{x}}_{i-1} \geq \sum_{i=0}^{n-2} A^{\mathbf{x}}_i$. And by definition of B, L, R, P_ϕ, INCR, $P^{\text{none}}_{<n}$, we have

$$\begin{aligned} L \circ P^{\text{none}}_{<n} \circ \mathsf{B} &= \mathsf{proj}\big(\mathsf{INCR}^*|n\rangle \otimes \mathsf{INCR}^*|n\rangle \otimes R^*\phi_n \otimes \mathsf{proj}(\phi_n)\phi_{n+1}\big)\,\mathbf{on}\,\mathbf{x}_1\mathbf{x}_2\mathbf{y}_1\mathbf{y}_2 \\ &= \mathsf{proj}\big(|n-1\rangle \otimes |n-1\rangle \otimes \phi_{n-1} \otimes \phi_n\big)\,\mathbf{on}\,\mathbf{x}_1\mathbf{x}_2\mathbf{y}_1\mathbf{y}_2 = \mathsf{A}^{\mathbf{x}}_{n-1}. \end{aligned}$$

Thus $\mathsf{A}' = L \circ \mathsf{C}' \geq \sum_{i=0}^{n-2} A^{\mathbf{x}}_i + A^{\mathbf{x}}_{n-1} = \mathsf{A}$. Thus we have proven (9). By rule JointWhile, this implies $\{\mathsf{C}'\}\,\mathrm{while}_{\mathfrak{c}} \overset{\text{tot}}{\sim} \mathrm{while}_{\mathfrak{d}}\,\{\mathsf{B}\}$ with C' as defined in (9). With $\mathsf{C} := A^{\mathbf{x}}_0 \leq \mathsf{C}'$, $\{\mathsf{C}\}\,\mathrm{while}_{\mathfrak{c}} \overset{\text{tot}}{\sim} \mathrm{while}_{\mathfrak{d}}\,\{\mathsf{B}\}$ follows by rule Conseq. This is the rightmost judgment in (8).

Using rules Init1, Init2, and Seq, we get $\{\mathsf{D}\}\,\mathbf{y} \leftarrow |0\rangle \overset{\text{tot}}{\sim} \mathbf{y} \leftarrow |0\rangle\,\{\mathsf{C}\}$ with $\mathsf{D} := \varepsilon^n \cdot (\mathsf{proj}(|0\rangle \otimes |0\rangle)\,\mathbf{on}\,\mathbf{x}_1\mathbf{x}_2)$. (This is done very similarly to (7).) This shows the middle judgment in (8).

Also using rules Init1, Init2, and Seq, we get $\{\varepsilon^n \cdot \mathsf{id}\}\,\mathbf{x} \leftarrow |0\rangle \overset{\text{tot}}{\sim} \mathbf{x} \leftarrow |0\rangle\,\{\mathsf{D}\}$. This shows the leftmost judgment in (8).

Thus we have shown the three judgments in (8). By rule Seq, it follows that $\{\varepsilon^n \cdot \mathsf{id}\}\,\mathfrak{c} \overset{\text{tot}}{\sim} \mathfrak{d}\,\{\mathsf{B}\}$. Since $\mathsf{B} \leq (\mathsf{EQUAL}\,\mathbf{on}\,\mathbf{y}_1\mathbf{y}_2)$, by rule Conseq, we get (3).

References

1. Gilles Barthe, François Dupressoir, Benjamin Grégoire, César Kunz, Benedikt Schmidt, and Pierre-Yves Strub. Easycrypt: A tutorial. In *FOSAD 2012/2013 Tutorial Lectures*, pages 146–166. Springer, 2014. doi:10.1007/978-3-319-10082-1_6.
2. Gilles Barthe, Benjamin Grégoire, Sylvain Heraud, and Santiago Zanella Béguelin. Computer-aided security proofs for the working cryptographer. In *Crypto 2011*, volume 6841 of *LNCS*, pages 71–90. Springer, 2011.
3. Gilles Barthe, Benjamin Grégoire, Federico Olmedo, and Santiago Zanella Béguelin. CertiCrypt: Computer-aided cryptographic proofs in Coq. http://certicrypt.gforge.inria.fr/. Accessed 2018-10-24.
4. Gilles Barthe, Benjamin Grégoire, and Santiago Zanella Béguelin. Formal certification of code-based cryptographic proofs. In *POPL 2009*, pages 90–101. ACM, 2009. doi:10.1145/1480881.1480894.

5 Gilles Barthe, Justin Hsu, Mingsheng Ying, Nengkun Yu, and Li Zhou. Relational proofs for quantum programs. *Proc. ACM Program. Lang.*, 4:21:1–21:29, 2019. Proceedings of POPL 2020. Full version is arXiv:1901.05184v2. doi:10.1145/3371089.

6 Nick Benton. Simple relational correctness proofs for static analyses and program transformations. In *POPL '04*, pages 14–25. ACM, 2004. doi:10.1145/964001.964003.

7 Rohit Chadha, Paulo Mateus, and Amílcar Sernadas. Reasoning about imperative quantum programs. *ENTCS*, 158:19–39, 2006. doi:10.1016/j.entcs.2006.04.003.

8 John B. Conway. *A course in functional analysis*. Number 96 in Graduate texts in mathematics. Springer, 2nd ed edition, 1997.

9 John B. Conway. *A course in operator theory*. Number 21 in Graduate studies in mathematics. American Mathematical Society, Providence, RI, 2000.

10 Ellie D'Hondt and Prakash Panangaden. Quantum weakest preconditions. *Mathematical. Structures in Comp. Sci.*, 16(3):429–451, 2006. doi:10.1017/S0960129506005251.

11 Yuan Feng, Runyao Duan, Zhengfeng Ji, and Mingsheng Ying. Proof rules for the correctness of quantum programs. *Theoretical Computer Science*, 386(1):151–166, 2007. doi:10.1016/j.tcs.2007.06.011.

12 Yoshihiko Kakutani. A logic for formal verification of quantum programs. In Anupam Datta, editor, *ASIAN 2009*, pages 79–93, Berlin, Heidelberg, 2009. Springer.

13 Dexter Kozen. A probabilistic PDL. In *STOC '83*, pages 291–297, New York, NY, USA, 1983. ACM. doi:10.1145/800061.808758.

14 Yangjia Li and Mingsheng Ying. Algorithmic analysis of termination problems for quantum programs. *Proc. ACM Program. Lang.*, 2:35:1–35:29, 2018. Proceedings of POPL 2019. doi:10.1145/3158123.

15 Michael A. Nielsen and Isaac L. Chuang. *Quantum Computation and Quantum Information*. Cambridge University Press, Cambridge, 10th anniversary edition, 2010.

16 Dominique Unruh. dominique-unruh/qrhl-tool: Prototype proof assistant for qRHL. GitHub, 2018. URL: https://github.com/dominique-unruh/qrhl-tool.

17 Dominique Unruh. Quantum relational Hoare logic. *Proc. ACM Program. Lang.*, 3:33:1–33:31, January 2019. Proceedings of POPL 2019. Full version is arXiv:1802.03188 [quant-ph]. doi:10.1145/3290346.

18 Mingsheng Ying. Floyd–Hoare logic for quantum programs. *ACM Trans. Program. Lang. Syst.*, 33(6):19:1–19:49, 2012. doi:10.1145/2049706.2049708.

Playing Stochastically in Weighted Timed Games to Emulate Memory

Benjamin Monmege
Aix Marseille Univ, Université de Toulon, CNRS, LIS, France

Julie Parreaux
Aix Marseille Univ, Université de Toulon, CNRS, LIS, France

Pierre-Alain Reynier
Aix Marseille Univ, Université de Toulon, CNRS, LIS, France

Abstract

Weighted timed games are two-player zero-sum games played in a timed automaton equipped with integer weights. We consider optimal reachability objectives, in which one of the players, that we call Min, wants to reach a target location while minimising the cumulated weight. While knowing if Min has a strategy to guarantee a value lower than a given threshold is known to be undecidable (with two or more clocks), several conditions, one of them being the divergence, have been given to recover decidability. In such weighted timed games (like in untimed weighted games in the presence of negative weights), Min may need finite memory to play (close to) optimally. This is thus tempting to try to emulate this finite memory with other strategic capabilities. In this work, we allow the players to use stochastic decisions, both in the choice of transitions and of timing delays. We give for the first time a definition of the expected value in weighted timed games, overcoming several theoretical challenges. We then show that, in divergent weighted timed games, the stochastic value is indeed equal to the classical (deterministic) value, thus proving that Min can guarantee the same value while only using stochastic choices, and no memory.

2012 ACM Subject Classification Software and its engineering → Formal software verification; Theory of computation → Algorithmic game theory

Keywords and phrases Weighted timed games, Algorithmic game theory, Randomisation

Digital Object Identifier 10.4230/LIPIcs.ICALP.2021.137

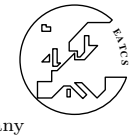
Category Track B: Automata, Logic, Semantics, and Theory of Programming

Related Version *Full Version*: https://arxiv.org/abs/2105.00984

Funding This work was partially funded by ANR project Ticktac (ANR-18-CE40-0015).

1 Introduction

Real-time aspects are often inherent in the behaviour of critical software systems. *Timed automata* [2] extend finite-state automata with timing constraints, providing an automata-theoretic framework to model and verify real-time systems. While this has lead to the development of mature verification tools, the design of programs verifying some real-time specifications remains a notoriously difficult problem. One way to avoid the need to a posteriori debugging is to automatise the process as much as possible. To do so, the situation is modelled into a *timed game*, played by a *controller* and an antagonistic *environment*: they act, in a turn-based fashion, over a timed automaton. A simple, yet realistic, objective for the controller is to reach a target location. We are thus looking for a *strategy* of the controller, that is a recipe dictating how to play so that the target is reached no matter how the environment plays. Reachability timed games are decidable [4], and EXPTIME-complete [19].



For many applications, this qualitative setting is often too coarse to model faithfully the system. This motivated a shift to a quantitative setting, based on weighted extensions of the models considered so far. Weighted extensions of these timed automata and games have thus been considered in order to measure the quality of the winning strategy for the controller [11, 1]: when the controller has several winning strategies, the quantitative version of the game helps choosing a good one with respect to some metrics. More precisely, the controller, which we now call player Min, wants to reach the target while *minimising* the cumulated weight. The model we consider, called *weighted timed game* (WTG for short), is defined as follows: the game takes place over a weighted (or priced) timed automaton [5, 3], where locations are split among the two players, transitions are equipped with weights, and locations with rates of weights (the cost is then proportional to the time spent in this location, with the rate as proportional coefficient). In this setting, the possibility to use negative weights on transitions and locations is crucial when one wants to model energy or other resources that can grow or decrease during the execution of the system under study.

While solving the optimal reachability problem on weighted timed automata has been shown to be PSPACE-complete [8] (i.e. the same complexity as the non-weighted version), WTGs are known to be undecidable [13]. Many restrictions have then been considered in order to regain decidability, the first and most interesting one being the class of strictly non-Zeno cost with only non-negative weights (in transitions and locations) [11]: this hypothesis requires that every execution of the timed automaton that follows a cycle of the region automaton has a weight far from 0 (in interval $[1, +\infty)$, for instance). This setting has been extended in the presence of negative weights in transitions and locations [16]: in the so-called *divergent WTGs*, each execution that follows a cycle of the region automaton has a weight in $(-\infty, -1] \cup [1, +\infty)$. A triply-exponential-time algorithm allows one to compute the values and almost-optimal strategies, while deciding the divergence of a WTG is PSPACE-complete.

When studying optimal reachability objectives with both positive and negative weights, it is known that strategies of player Min require memory to play optimally (see [15] for the case of finite games). More precisely, the memory needed is pseudo-polynomial (i.e. polynomial if constants are encoded in unary). For WTGs, the memory needed even becomes exponential. An important challenge is thus to find ways to avoid using such complex strategies, e.g. by proposing alternative classes of strategies that are more easily amenable to implementation.

Strategies considered so far are deterministic. Though the game has no stochastic edges, it is possible to allow players to use stochastic strategies. This approach has been recently studied in the setting of finite games [20], where it is shown that memory may indeed be emulated using randomness in finite reachability games with integer weights. More precisely, the minimal value Min can achieve using memoryless stochastic strategies is the same as the value achievable using deterministic strategies. In the present work, we lift the results obtained in [20] for finite games to the timed setting.

A first important challenge is to analyse how to play stochastically in WTGs. To our knowledge, this has not been studied before. Starting from a notion of stochastic behaviours in a timed automaton considered in [7] (for the one-player setting), we propose a new class of stochastic strategies. Compared with [7], our class is larger in the sense that we allow Dirac distributions for delays, which subsumes the setting of deterministic strategies. However, in order to ensure that strategies yield a well-defined probability distribution on sets of executions, we need measurability properties stronger than the one considered in [7] (we actually provide an example showing that their hypothesis was not strong enough).

Then, we turn our attention towards the expected cumulated weight of the set of plays conforming to a pair of stochastic strategies. We first prove that under the previous measurability hypotheses, this expectation is well-defined when restricting to the set of plays

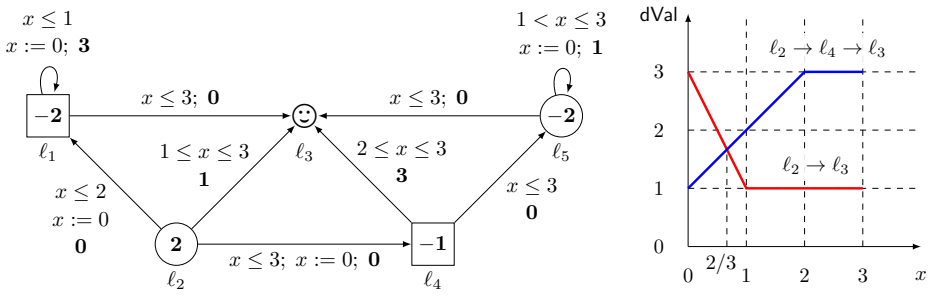


Figure 1 On the left, a weighted timed game. Locations belonging to Min (resp. Max) are depicted by circles (resp. squares). The target location is ℓ_3. Location ℓ_1 (resp. ℓ_5) has (deterministic) value $+\infty$ (resp. $-\infty$). As a consequence, the value in ℓ_4 is determined by the edge to ℓ_3, and depicted in blue on the right. In location ℓ_2, the value associated with the transition to ℓ_3 is depicted in red, and the deterministic value in ℓ_2 is obtained as the minimum of these two curves.

following a finite sequence of transitions. In order to have the convergence of the global expectation, we identify another property of strategies of Min, which intuitively ensures that the set of target locations is reached quickly enough. This allows us to define a notion of stochastic value (resp. memoryless stochastic value) of the game, i.e. the best value Min can achieve using stochastic strategies (resp. memoryless stochastic strategies), when Max uses stochastic strategies (resp. memoryless stochastic strategies) too.

In a second step, we aim at adapting the proof techniques of [20] from finite to infinite games. It is well-known that the classical region abstraction of timed automata is not suited to analyse WTGs (there are cases in which one has to split regions). In order to obtain positive results, we focus on the class of divergent WTGs. We prove that the notion of optimal deterministic switching strategy, which was central in the approach of [20], can be adapted to divergent WTGs. Our main result is then to show that for these games, the two versions of stochastic values are equal to the deterministic value. In other terms, we show that Min can emulate memory using randomisation, and vice versa. Moreover, combining memory and randomisation does not increase Min's capabilities. Due to the lack of space, detailed proofs of all results can be found in the long version [21].

2 Weighted timed games

We let $\mathcal{C}$ be a finite set of variables called clocks. A valuation is a mapping $\nu \colon \mathcal{C} \to \mathbb{R}_{\geq 0}$. For a valuation ν, a delay $t \in \mathbb{R}_{\geq 0}$ and a subset $Y \subseteq \mathcal{C}$ of clocks, we define the valuation $\nu + t$ as $(\nu + t)(x) = \nu(x) + t$, for all $x \in \mathcal{C}$, and the valuation $\nu[Y := 0]$ as $(\nu[Y := 0])(x) = 0$ if $x \in Y$, and $(\nu[Y := 0])(x) = \nu(x)$ otherwise. The valuation $\mathbf{0}$ assigns 0 to every clock. A (non-diagonal) guard on clocks of $\mathcal{C}$ is a conjunction of atomic constraints of the form $x \bowtie c$, where $\bowtie \in \{\leq, <, =, >, \geq\}$ and $c \in \mathbb{N}$. A valuation $\nu \colon \mathcal{C} \to \mathbb{R}_{\geq 0}$ satisfies an atomic constraint $x \bowtie c$ if $\nu(x) \bowtie c$. The satisfaction relation is extended to all guards g naturally, and denoted by $\nu \models g$. We let $\mathsf{Guards}(\mathcal{C})$ denote the set of guards over $\mathcal{C}$.

▶ **Definition 1.** *A weighted timed game (WTG) is a tuple* $\mathcal{G} = \langle L_{\mathsf{Min}}, L_{\mathsf{Max}}, L_T, \Delta, \mathsf{wt} \rangle$ *where* L_{Min}, L_{Max}, L_T *are finite disjoint subsets of* Min *locations,* Max *locations, and target locations, respectively (we let* $L = L_{\mathsf{Min}} \uplus L_{\mathsf{Max}} \uplus L_T$*),* $\Delta \subseteq L \times \mathsf{Guards}(\mathcal{C}) \times 2^{\mathcal{C}} \times L$ *is a finite set of transitions,* $\mathsf{wt} \colon \Delta \uplus L \to \mathbb{Z}$ *is the weight function.*

Without loss of generality, we suppose the absence of deadlocks except on target locations, i.e. for each location $\ell \in L \backslash L_T$ and valuation ν, there exists $(\ell, g, Y, \ell') \in \Delta$ such that $\nu \models g$, and no transitions start in L_T. The semantics of a WTG $\mathcal{G}$ is defined in terms of a game played on an infinite transition system whose vertices are configurations of the WTG. A configuration is a pair (ℓ, ν) with a location and a valuation of the clocks. Configurations are split into players according to the location. A configuration is final if its location is a target location of L_T. The alphabet of the transition system is given by $\Delta \times \mathbb{R}_{\geq 0}$: a pair (δ, t) encodes the delay t that a player wants to spend in the current location, before firing transition δ. For every delay $t \in \mathbb{R}_{\geq 0}$, transition $\delta = (\ell, g, Y, \ell') \in \Delta$ and valuation ν, there is an edge $(\ell, \nu) \xrightarrow{\delta, t} (\ell', \nu')$ if $\nu + t \models g$ and $\nu' = (\nu + t)[Y := 0]$. The weight of such an edge e is given by $t \times \mathsf{wt}(\ell) + \mathsf{wt}(\delta)$. An example is depicted on Figure 1.

A *finite play* is a finite sequence of consecutive edges $\rho = (\ell_0, \nu_0) \xrightarrow{\delta_0, t_0} (\ell_1, \nu_1) \xrightarrow{\delta_1, t_1} \cdots (\ell_k, \nu_k)$. We sometimes denote such a play $(\ell_0, \nu_0) \xrightarrow{(\delta_0, t_0) \cdots (\delta_{k-1}, t_{k-1})}$, since intermediate locations and valuations are uniquely defined by the initial configuration and the sequence of transitions and delays. We denote by $|\rho|$ the length k of ρ. The concatenation of two finite plays ρ_1 and ρ_2, such that ρ_1 ends in the same configuration as ρ_2 starts, is denoted by $\rho_1 \rho_2$. We denote by $I(\rho, \delta)$ the interval of delays t such that the play ρ can be extended with the edge $\xrightarrow{\delta, t}$. We let FPlays be the set of all finite plays, whereas $\mathsf{FPlays}_{\mathsf{Min}}$ (resp. $\mathsf{FPlays}_{\mathsf{Max}}$) denote the finite plays that end in a configuration of Min (resp. Max). A *play* is then a maximal sequence of consecutive edges (it is either infinite or it reaches L_T).

We call *path* a finite or infinite sequence π of transitions of $\mathcal{G}$. Each play ρ of $\mathcal{G}$ is associated with a unique path π (by projecting away everything but the transitions): we say that ρ *follows* the path π. A *target path* is a finite path ending in the target set L_T. We denote by TPaths the set of target paths. We let TPaths_ρ (resp. TPaths^n_ρ) the subset of target paths that start from the last location of the finite play ρ (resp. containing n transitions). A path is said to be *maximal* if it is infinite or if it is a target path.

A *deterministic strategy* for Min (resp. Max) is a mapping $\sigma \colon \mathsf{FPlays}_{\mathsf{Min}} \to \Delta \times \mathbb{R}_{\geq 0}$ (resp. $\tau \colon \mathsf{FPlays}_{\mathsf{Max}} \to \Delta \times \mathbb{R}_{\geq 0}$) such that for all finite plays $\rho \in \mathsf{FPlays}_{\mathsf{Min}}$ (resp. $\rho \in \mathsf{FPlays}_{\mathsf{Max}}$) ending in non-target configuration (ℓ, ν), there exists an edge $(\ell, \nu) \xrightarrow{\sigma(\rho)} (\ell', \nu')$. We let $\mathsf{dStrat}_{\mathsf{Min}}$ and $\mathsf{dStrat}_{\mathsf{Max}}$ denote the set of deterministic strategies in $\mathcal{G}$ for players Min and Max, respectively. A play or finite play $\rho = (\ell_0, \nu_0) \xrightarrow{\delta_0, t_0} (\ell_1, \nu_1) \xrightarrow{\delta_1, t_1} \cdots$ conforms to a deterministic strategy σ of Min (resp. Max) if for all k such that (ℓ_k, ν_k) belongs to Min (resp. Max), we have that $(\delta_k, t_k) = \sigma((\ell_0, \nu_0) \xrightarrow{\delta_0, t_0} \cdots (\ell_k, \nu_k))$. For all deterministic strategies σ and τ of players Min and Max, respectively, and for all configurations (ℓ_0, ν_0), we let $\mathsf{Play}((\ell_0, \nu_0), \sigma, \tau)$ be the outcome of σ and τ, defined as the unique maximal play conforming to σ and τ and starting in (ℓ_0, ν_0).

The objective of Min is to reach a target configuration, while minimising the cumulated weight up to the target. Hence, we associate to every finite play $\rho = (\ell_0, \nu_0) \xrightarrow{\delta_0, t_0} (\ell_1, \nu_1) \xrightarrow{\delta_1, t_1} \cdots (\ell_k, \nu_k)$ its cumulated weight, taking into account both discrete and continuous costs: $\mathsf{wt}_\Sigma(\rho) = \sum_{i=0}^{k-1} [t_i \times \mathsf{wt}(\ell_i) + \mathsf{wt}(\delta_i)]$. Then, the weight of a maximal play ρ, denoted by $\mathsf{wt}(\rho)$, is defined by $+\infty$ if ρ is infinite (does not reach L_T), and $\mathsf{wt}_\Sigma(\rho)$ if it ends in (ℓ_T, ν) with $\ell_T \in L_T$.

A deterministic strategy $\sigma \in \mathsf{dStrat}_{\mathsf{Min}}$ guarantees a certain value, against all possible strategies of the opponent: for all locations ℓ and valuations ν, we let $\mathsf{dVal}^\sigma_{\ell,\nu} = \sup_{\tau \in \mathsf{dStrat}_{\mathsf{Max},\mathcal{G}}} \mathsf{wt}(\mathsf{Play}((\ell, \nu), \sigma, \tau))$. Then, for all locations ℓ and valuations ν, we let $\mathsf{dVal}_{\ell,\nu}$ be the *deterministic value* of $\mathcal{G}$ in (ℓ, ν), defined as $\mathsf{dVal}_{\ell,\nu} = \inf_{\sigma \in \mathsf{dStrat}_{\mathsf{Min}}} \mathsf{dVal}^\sigma_{\ell,\nu}$. We say that a deterministic strategy σ of Min is *ε-optimal wrt the deterministic value* if $\mathsf{dVal}^\sigma_{\ell,\nu} \leq \mathsf{dVal}_{\ell,\nu} + \varepsilon$ for all (ℓ, ν). It is said optimal if this holds for $\varepsilon = 0$.

As usual in related work [1, 11, 12, 16], we assume that all clocks are *bounded* by a constant $M \in \mathbb{N}$, i.e. every transition of the WTG is equipped with a guard g such that $\nu \models g$ implies $\nu(x) \leq M$ for all clocks $x \in \mathcal{C}$. We denote by $w^L_{\max}$ (resp. $w^\Delta_{\max}$, $w^e_{\max}$) the maximal weight in absolute values of locations (resp. of transitions, edges) of $\mathcal{G}$, i.e. $w^L_{\max} = \max_{\ell \in L} |\mathsf{wt}(\ell)|$ (resp. $w^\Delta_{\max} = \max_{\delta \in \Delta} |\mathsf{wt}(\delta)|$, $w^e_{\max} = M w^L_{\max} + w^\Delta_{\max}$).

In the following, we rely on the crucial notion of regions, as introduced in the seminal work on timed automata [2]. A game $\mathcal{G}$ can be populated with the region information, without loss of generality, as described formally in [16], e.g. The *region automaton, or region game*, $\mathcal{R}(\mathcal{G})$ is thus the WTG with locations $S = L \times \mathsf{Reg}(\mathcal{C}, M)$ and all transitions $((\ell, r), g'', Y, (\ell', r'))$ with $(\ell, g, Y, \ell') \in \Delta$ such that the model of guard g'' (i.e. all valuations ν such that $\nu \models g''$) is a region r'', time successor of r such that r'' satisfies the guard g, and r' is the region obtained from r'' by resetting all clocks of Y. Distribution of locations to players, final locations, and weights are inherited from $\mathcal{G}$. We call *region path* a finite or infinite sequence of transitions in this automaton, and we again denote by π such paths. A play ρ in $\mathcal{G}$ is projected on a region path π, with a similar definition as the projection on paths: we again say that ρ *follows* the region path π. It is important to notice that, even if π is a *cycle* (i.e. starts and ends in the same location of the region game), there may exist plays following it in $\mathcal{G}$ that are not cycles, due to the fact that regions are sets of valuations.

As shown in previous work [11, 16], knowing whether $\mathsf{dVal}_{\ell,\nu} = +\infty$ for a certain configuration is a purely qualitative problem that can be decided easily by using the region game: indeed, $\mathsf{dVal}_{\ell,\nu} = +\infty$ if and only if Min has no strategies that guarantee reaching the target L_T. This is thus a reachability objective, where weights are useless. Moreover, Max has a strategy that guarantees that no plays reach the target L_T from any configuration (ℓ, ν) such that $\mathsf{dVal}_{\ell,\nu} = +\infty$. In this situation, considering stochastic choices is not interesting. **We thus rule out this case by supposing in the following that no configurations of $\mathcal{G}$ have a value $+\infty$**: such configurations can be removed in the region game by strengthening the guard on transitions.

3 Playing stochastically in WTGs

Our first contribution consists in allowing both players to use stochastic choices in their strategies. From a game theory point of view, this seems natural. From a controller synthesis point of view, we claim that the question is natural too, especially because player Min may require exponential memory to play optimally in WTGs. This is already the case even without clocks (such games are then sometimes called *shortest-path games*) where it has been shown in [20] that the memory required by Min could be traded for stochastic choices instead (and vice versa). We aim at extending this result in the context of weighted timed games. Before doing so, we must introduce stochastic strategies in the context of weighted timed games, which has never been explored until now, as far as we are aware of. We will however strongly rely on a recent line of works aiming at studying stochastic timed automata [7, 9, 6, 10], thus extending the results in the context of two-player games (instead of model-checking) and with weights, which indeed represents the main challenge in order to give a meaning to the expected payoff.

Naturally, deterministic strategies for Min are extended to more general stochastic strategies as mappings $\eta \colon \mathsf{FPlays}_{\mathsf{Min}} \to \mathsf{Dist}(\Delta \times \mathbb{R}_{\geq 0})$ where each finite play is associated to a probability distribution over the set of pairs of transition and delay. Here, we let $\mathsf{Dist}(S)$ the set of all probability distributions over a set S (equipped with an underlying σ-algebra). Since Δ is a finite set, this is equivalent to letting first Min choose a transition

via $\eta_\Delta : \mathsf{FPlays}_{\mathsf{Min}} \to \mathsf{Dist}(\Delta)$, and then, knowing the chosen transition, choose a delay via $\eta_{\mathbb{R}^+} : \mathsf{FPlays}_{\mathsf{Min}} \times \Delta \to \mathsf{Dist}(\mathbb{R}_{\geq 0})$, the support of the distribution $\eta_{\mathbb{R}^+}(\rho, \delta)$ being included in the interval $I(\rho, \delta)$ of valid delays. We can then recombine η_Δ and $\eta_{\mathbb{R}^+}$ to obtain the distribution $\eta(\rho)$. Similar definitions hold for Max whose general strategies are denoted by θ.

Notice that deterministic strategies are a special case of strategies, where the distributions are chosen to be Dirac distributions. Another useful restriction over strategies is the non-use of memory: a strategy η is *memoryless* if for all finite plays ρ, ρ' ending in the same configuration, we have that $\eta(\rho) = \eta(\rho')$. A similar definition holds for Max.

Probability measure on plays. We fix two strategies η and θ for both players, and an initial configuration (ℓ_0, ν_0). Our goal is to define a probability measure on plays. To do so, and following the contribution of [7] for stochastic timed automata, the set of plays of a WTG $\mathcal{G}$ starting from (ℓ_0, ν_0) and conforming to η and θ can naturally be equipped with a structure of σ-algebra whose generators are all subsets of plays that start with a finite prefix following the same finite path π (remember that paths are sequences of transitions, with no information on the delayed time) with some Borel-measurable constraints on the delays taken along π. The a priori idea is thus to define a probability measure $\mathbb{P}^{\eta,\theta}_{\ell_0,\nu_0}$ on such generators which extends uniquely as a probability measure over the whole σ-algebra, by Carathéodory's extension theorem.

Consider thus a finite path π, starting in location ℓ, and a play ρ ending in the same location ℓ. We define the probability $\mathbb{P}^{\eta,\theta}_\rho(\pi)$ taking into account all possible plays that start with ρ and continue according to π (we leave the Borel-measurable constraints on the delays for now, but discuss them later). It is defined by induction on the length of π by $\mathbb{P}^{\eta,\theta}_\rho(\varepsilon) = 1$, and for all transitions $\delta = (\ell, g, Y, \ell') \in \Delta$, $\mathbb{P}^{\eta,\theta}_\rho(\delta\pi) = \int_{I(\rho,\delta)} \eta_\Delta(\rho)(\delta) \times \mathbb{P}^{\eta,\theta}_{\rho \xrightarrow{\delta,t}}(\pi)\, \mathrm{d}\eta_{\mathbb{R}^+}(\rho, \delta)(t)$. This definition is very similar to the one in [7] except that we choose to decouple the distribution on pairs of $\Delta \times \mathbb{R}_{\geq 0}$ by first selecting a transition and then delay, whereas authors of [7] consider independent choices, the one on transitions being described by some weights on transitions (depending on the current region).

For modelling purposes, authors of [7] enforce that probability distributions on delays do not forbid any delays of the interval $I(\rho, \delta)$ of possible delays, thus ruling out singular distributions like Dirac ones that would consider taking a single possible delay (like deterministic strategies do). More formally, they require $\eta_{\mathbb{R}^+}(\rho, \delta)$ to be absolutely continuous (i.e. equivalent to the Lebesgue measure) on interval $I(\rho, \delta)$. We claim that even with this assumption, the previous definition of the probability may not be well-founded, as demonstrated by the example given in [21, Appendix A]. From this example, we see the importance to moreover enforce that the distributions $\eta_\Delta(\rho)$ and $\eta_{\mathbb{R}^+}(\rho, \delta)$ are "measurable wrt the sequence of delays along the play ρ". This is easy to define for the transition part. For delays, since we want deterministic strategies to be a subset of stochastic strategies, we must be able to choose delays by using Dirac distributions, and by extension discrete distributions (that are not absolutely continuous, as [7] requires). This results in the following hypothesis:

▶ **Hypothesis 1.** *A strategy η satisfies this hypothesis if*

1. *for all transitions $\delta_0, \dots, \delta_k, \delta$, the mapping*

$$(t_0, \dots, t_{k-1}) \mapsto \eta_\Delta\big((\ell_0, \nu_0) \xrightarrow{(\delta_0,t_0)\cdots(\delta_{k-1},t_{k-1})} \big)(\delta)$$

is measurable; and

2. *for all plays ρ and transition δ, the probability distribution $\eta_{\mathbb{R}^+}(\rho,\delta)$ (of the random variable t) is described by a cumulative distribution function (CDF) that is the sum of an absolutely continuous function $G(\rho,\delta)$ and Heaviside functions*[1] *$t \mapsto \sum_i \alpha_i(\rho,\delta)H(t - a_i(\rho,\delta))$. Moreover, for all transitions $\delta_0,\dots,\delta_{k-1},\delta$, the mappings $(t_0,\dots,t_{k-1},t) \mapsto G(\rho,\delta)(t)$, $(t_0,\dots,t_{k-1}) \mapsto \alpha_i(\rho,\delta)$, and $(t_0,\dots,t_{k-1}) \mapsto a_i(\rho,\delta)$ must be measurable (where we use notation ρ to denote the play $(\ell_0,\nu_0) \xrightarrow{(\delta_0,t_0)\cdots(\delta_{k-1},t_{k-1})}$).*

This hypothesis allows us to obtain :

▶ **Lemma 2.** *If η and θ are strategies satisfying Hypothesis 1, the probabilities $\mathbb{P}^{\eta,\theta}_\rho(\pi)$ of following a path π after the play ρ are well defined. It can be extended into a probability distribution over maximal paths π starting in the last location of ρ.*

Apart from the well-definition that is new, the rest of the proof is very close to the one of [7]. The probability measure easily extends to unions of maximal paths: in particular, $\mathbb{P}^{\eta,\theta}_{\ell_0,\nu_0}(\mathsf{TPaths}_{\ell_0,\nu_0})$ is set as the sum $\sum_{\pi\in\mathsf{TPaths}_{\ell_0,\nu_0}} \mathbb{P}^{\eta,\theta}_{\ell_0,\nu_0}(\pi)$ of probabilities of all paths reaching L_T from ℓ_0. Authors of [7] go one step further, by using Carathéodory's theorem to extend the probability measure on paths ($\mathbb{P}^{\eta,\theta}_\rho(\pi)$) to a measure on plays ($\mathbb{P}^{\eta,\theta}_\rho$), whose σ-algebra is generated by maximal plays with Borel-measurable constraints on the delays. We do not formally need this further extension and will only use such extension to give an intuitive introduction of the expected payoff below. In the following, we let $\mathsf{Strat}_{\mathsf{Min}}$ and $\mathsf{Strat}_{\mathsf{Max}}$ be the sets of (stochastic) strategies satisfying Hypothesis 1, for both players. We let $\mathsf{mStrat}_{\mathsf{Min}}$ and $\mathsf{mStrat}_{\mathsf{Max}}$ be the respective subsets of memoryless strategies.

Expected payoff of plays. As explained before, by Carathéodory's theorem, the set of plays can be equipped with a probability distribution, and we are interested in the expectation of the random variable $\mathsf{wt}(\rho)$ (where ρ conforms with two fixed strategies η and θ). This only makes sense if the probability to reach a target location is equal to 1, since otherwise, the expected weight will intuitively be $+\infty$ (there is a non-zero probability to not reach the target location, the weight of all such plays being $+\infty$). We thus now require that $\mathbb{P}^{\eta,\theta}_{\ell_0,\nu_0}(\mathsf{TPaths}_{\ell_0,\nu_0}) = 1$ (i.e. the probability to follow an infinite path is 0). We will see afterwards that this is not a sufficient condition to ensure that the expected weight is finite.

We would like to define the expected weight to reach the target as (we write $\mathbb{E}^{\eta,\theta}_{\ell_0,\nu_0}$ instead of $\mathbb{E}^{\eta,\theta}_{\ell_0,\nu_0}(\mathsf{wt})$, since we only consider the expectation of the weight wt): $\mathbb{E}^{\eta,\theta}_{\ell_0,\nu_0} = \int_\rho \mathsf{wt}(\rho)\,\mathrm{d}\mathbb{P}^{\eta,\theta}_{\ell_0,\nu_0}(\rho)$ where the integral is over all plays ρ that start in (ℓ_0,ν_0) and reach the target L_T (such restriction is again justified by the fact that the probability mass of all other plays is 0). This is problematic a priori (and we will see below an example where this indeed would be a problem) since the cumulated weight is not known to be a measurable function of the play, wrt the measure $\mathbb{P}^{\eta,\theta}_{\ell_0,\nu_0}$.

To overcome this challenge, we follow a different approach, consisting in mimicking the construction of the probability before: first define the expected payoff of all plays following a given path, and then sum over all possible paths.

▶ **Definition 3.** *We define the expected weight $\mathbb{E}^{\eta,\theta}_\rho(\pi)$ of plays that can extend ρ (the weight of ρ is thus not counted in the expectation) and that follow the path π. It is defined by induction on the length of π by $\mathbb{E}^{\eta,\theta}_\rho(\varepsilon) = 0$ and for all transitions $\delta = (\ell,g,Y,\ell')$:*

$$\mathbb{E}^{\eta,\theta}_\rho(\delta\pi) = \int_{I(\rho,\delta)} \eta_\Delta(\rho)(\delta)\Big[\big(t\,\mathsf{wt}(\ell) + \mathsf{wt}(\delta)\big)\mathbb{P}^{\eta,\theta}_{\rho\xrightarrow{\delta,t}}(\pi) + \mathbb{E}^{\eta,\theta}_{\rho\xrightarrow{\delta,t}}(\pi)\Big]\,\mathrm{d}\eta_{\mathbb{R}^+}(\rho,\delta)(t)$$

We then define the expected weight $\mathbb{E}^{\eta,\theta}_\rho = \sum_{\pi\in\mathsf{TPaths}_\rho} \mathbb{E}^{\eta,\theta}_\rho(\pi)$, when this sum converges.

[1] We let H denote the mapping from $\mathbb{R}$ to $[0,1]$ such that $H(t) = 0$ if $t < 0$ and $H(t) = 1$ otherwise. Recall that it is the CDF of the Dirac distribution choosing $t = 0$.

Hypothesis 1 is sufficient to show the well-definition of all expectations $\mathbb{E}^{\eta,\theta}_{\rho}(\pi)$:

▶ **Lemma 4.** *If $\eta \in \mathsf{Strat}_{\mathsf{Min}}$ and $\theta \in \mathsf{Strat}_{\mathsf{Max}}$, $\mathbb{E}^{\eta,\theta}_{\rho}(\pi)$ is well-defined for all ρ and π.*

However, the infinite sum in $\mathbb{E}^{\eta,\theta}_{\rho}$ can be problematic. We thus need a stronger hypothesis to ensure its convergence. We adopt here an asymmetrical point of view, relying only on hypothesis on the strategy η of Min. Our choice is grounded in our controller synthesis view, Min being the controller desiring to reach a target location with minimum expected payoff, while Max is an uncontrollable environment.

▶ **Definition 5.** *A strategy $\eta \in \mathsf{Strat}_{\mathsf{Min}}$ of Min is said* proper *if for all finite plays ρ and strategies $\theta \in \mathsf{Strat}_{\mathsf{Max}}$, $\mathbb{P}^{\eta,\theta}_{\rho}(\mathsf{TPaths}_{\rho}) = 1$ and the infinite sum $\sum_{\pi \in \mathsf{TPaths}_{\rho}} \mathbb{E}^{\eta,\theta}_{\rho}(\pi)$ converges.*

We let $\mathsf{Strat}^{\mathsf{P}}_{\mathsf{Min}}$ be the set of proper strategies of Min, $\mathsf{mStrat}^{\mathsf{P}}_{\mathsf{Min}}$ the subset of memoryless proper strategies. Notice that a deterministic strategy of Min is proper as soon as it guarantees to reach the target set of locations (remember that we have ruled out configurations with a deterministic value $\mathsf{dVal}(\ell,\nu) = +\infty$ where Min cannot deterministically guarantee to reach the target L_T): this shows that proper strategies exist (even without using memory). For stochastic strategies, we have seen above that reaching the target set of locations with probability 1 is a necessary but not sufficient condition to be proper. Not only Max must reach the target almost surely, but he must do it *quickly enough* so that the expectation converges. We now give a sufficient condition for a strategy to be proper, that we will use in the rest of this article.

▶ **Hypothesis 2.** *A strategy $\eta \in \mathsf{Strat}_{\mathsf{Min}}$ of Min satisfies this hypothesis if there exist $m \in \mathbb{N}$ and $\alpha \in (0,1]$ such that for all finite plays ρ and strategies $\theta \in \mathsf{Strat}_{\mathsf{Max}}$, $\mathbb{P}^{\eta,\theta}_{\rho}(\bigcup_{n \leq m} \mathsf{TPaths}^{n}_{\rho}) \geq \alpha$.*

This hypothesis is indeed a sufficient condition for a strategy to be proper:

▶ **Lemma 6.** *All strategies of Min satisfying Hypothesis 2 are proper.*

Sketch of proof. The idea is to decompose $\mathbb{E}^{\eta,\theta}_{\rho}$ for all ρ as $\mathbb{E}^{\eta,\theta}_{\rho} = \sum_{n=0}^{\infty} \sum_{\pi \in \mathsf{TPaths}^{n}_{\rho}} \mathbb{E}^{\eta,\theta}_{\rho}(\pi)$. Since $\mathsf{TPaths}^{n}_{\rho}$ is a finite set, only the first sum must be shown to be converging. It is done by noticing that the weight of plays of length n grows linearly wrt n, while the probability $\sum_{\pi \in \mathsf{TPaths}^{n}_{\rho}} \mathbb{P}^{\eta,\theta}_{\rho}(\pi)$ decreases exponentially wrt n (thanks to Hypothesis 2). More precisely, we show for all $n \in \mathbb{N}$ and all $\rho \in \mathsf{FPlays}$, that

1. for all $\pi \in \mathsf{TPaths}^{n}_{\rho}$, $|\mathbb{E}^{\eta,\theta}_{\rho}(\pi)| \leq \mathbb{P}^{\eta,\theta}_{\rho}(\pi)\, n\, w^{e}_{\max}$; and
2. $\sum_{\pi \in \mathsf{TPaths}^{n}_{\rho}} \mathbb{P}^{\eta,\theta}_{\rho}(\pi) \leq (1-\alpha)^{\lfloor n/m \rfloor}$.

This allows us to show that $\mathbb{P}^{\eta,\theta}_{\rho}(\mathsf{TPaths}_{\rho}) = 1$ and that the sum $\mathbb{E}^{\eta,\theta}_{\rho} = \sum_{\pi \in \mathsf{TPaths}_{\rho}} \mathbb{E}^{\eta,\theta}_{\rho}(\pi)$ converges. ◀

Now that we have associated an expected payoff to each convenient pair of strategies, we are able to mimic the classical definition of *value* to stochastic strategies. Let ℓ be a location and ν be a valuation. For all $\eta \in \mathsf{Strat}^{\mathsf{P}}_{\mathsf{Min}}$ and $\theta \in \mathsf{Strat}_{\mathsf{Max}}$, we let $\mathsf{Val}^{\eta}_{\ell,\nu} = \sup_{\theta \in \mathsf{Strat}_{\mathsf{Max}}} \mathbb{E}^{\eta,\theta}_{\ell,\nu}$. Then, we let $\mathsf{Val}_{\ell,\nu}$ be the value of $\mathcal{G}$ in (ℓ,ν), defined as the best expected payoff Min can hope for: $\mathsf{Val}_{\ell,\nu} = \inf_{\eta \in \mathsf{Strat}^{\mathsf{P}}_{\mathsf{Min}}} \mathsf{Val}^{\eta}_{\ell,\nu}$. Both definitions can be generalised by replacing configurations (ℓ,ν) by finite plays ρ: we let $\mathsf{Val}^{\eta}_{\rho}$ and Val_{ρ} be the generalised versions. We also define the *memoryless values* mVal^{η} and mVal, where all strategies are taken memoryless.

Our main contribution, presented in details in Section 5, is to compare the memoryless (stochastic) value, the deterministic value and the stochastic value, showing their equality for a fragment of WTGs. Along the way, we will need the following result showing that when Min plays with a proper strategy, Max always has a *best response* strategy that is deterministic:

▶ **Lemma 7.** *Let $\eta \in \mathsf{Strat}^{\mathsf{p}}_{\mathsf{Min}}$ and $\varepsilon > 0$. There exists a deterministic strategy $\tau \in \mathsf{dStrat}_{\mathsf{Max}}$ such that for all finite plays ρ, $\mathbb{E}^{\eta,\tau}_{\rho} \geq \mathsf{Val}^{\eta}_{\rho} - \varepsilon$. If $\eta \in \mathsf{mStrat}^{\mathsf{p}}_{\mathsf{Min}}$ is memoryless, then there exists a deterministic strategy $\tau \in \mathsf{dStrat}_{\mathsf{Max}}$ such that for all finite plays ρ, $\mathbb{E}^{\eta,\tau}_{\rho} \geq \mathsf{mVal}^{\eta}_{\rho} - \varepsilon$.*

4 Divergent weighted timed games

As we have already seen in the introduction, interesting fragments of WTGs have been designed, in order to regain decidability of the problem of determining whether the value of a WTG is below a certain threshold. One such fragment is obtained by enforcing a semantical property of divergence (originally called *strictly non-Zeno cost* when only dealing with non-negative weights [11]): it asks that every play following a cycle in the region automaton has weight far from 0. We will consider this restriction in the following, since it allows for a large class of decidable WTGs, with no limitations on the number of clocks. Formally, a cyclic region path π of $\mathcal{R}(\mathcal{G})$ is said to be a positive cycle (resp. a negative cycle) if every finite play ρ following π satisfies $\mathsf{wt}_{\Sigma}(\rho) \geq 1$ (resp. $\mathsf{wt}_{\Sigma}(\rho) \leq -1$).

▶ **Definition 8** ([16])**.** *A WTG is divergent if every cyclic region path is positive or negative.*

In [16], it is shown that this definition is equivalent to requiring that for all strongly connected components (SCC) S of the graph of $\mathcal{R}(\mathcal{G})$, either every cycle π inside S is positive (we say that the SCC is positive), or every cycle π inside S is negative (we say that the SCC is negative). The best computability result in this setting is:

▶ **Theorem 9** ([16])**.** *The deterministic value of a divergent WTG can be computed in triply-exponential-time.*

We explain how to recover from Theorem 9 the needed shape of ε-optimal strategies, since this is one of the new technical ingredient we need afterwards.

Switching strategies for Min. Theorem 9 is obtained in [16] by using a *value iteration algorithm* (originally described in [1] for acyclic timed automata). If V represents a value function, i.e. a mapping $L \times \mathbb{R}^{\mathcal{C}}_{\geq 0} \to \mathbb{R}_{\infty}$, we denote by $V_{\ell,\nu}$ the image $V(\ell,\nu)$, for better readability. One step of the game is summarised in the following operator $\mathcal{F}$ mapping each value function V to the value function defined for all $(\ell,\nu) \in L \times \mathbb{R}^{\mathcal{C}}_{\geq 0}$ by $\mathcal{F}(V)_{\ell,\nu} = 0$ if $\ell \in L_T$, $\mathcal{F}(V)_{\ell,\nu} = \sup_{(\ell,\nu) \xrightarrow{\delta,t} (\ell',\nu')} \left[t \times \mathsf{wt}(\ell) + \mathsf{wt}(\delta) + V_{\ell',\nu'}\right]$ if $\ell \in L_{\mathsf{Max}}$, and $\mathcal{F}(V)_{\ell,\nu} = \inf_{(\ell,\nu) \xrightarrow{\delta,t} (\ell',\nu')} \left[t \times \mathsf{wt}(\ell) + \mathsf{wt}(\delta) + V_{\ell',\nu'}\right]$ if $\ell \in L_{\mathsf{Min}}$, where $(\ell,\nu) \xrightarrow{\delta,t} (\ell',\nu')$ ranges over valid edges in $\mathcal{G}$. Then, starting from V^0 mapping every configuration to $+\infty$, except for the targets mapped to 0, we let $V^i = \mathcal{F}(V^{i-1})$ for all $i > 0$. The value function V^i is intuitively what Min can guarantee when forced to reach the target in at most i steps.

The value computation of Theorem 9 is then obtained in two steps. First, configurations (ℓ,ν) of value $\mathsf{dVal}_{\ell,\nu} = -\infty$ are found by using a decomposition of the region game $\mathcal{R}(\mathcal{G})$ into strongly-connected components (SCC). Indeed, in divergent WTGs, configurations of value $-\infty$ are all the ones from which Min has a strategy to visit infinitely many times configurations of a single location (ℓ,r) of $\mathcal{R}(\mathcal{G})$ contained in a negative SCC. This is thus a Büchi objective on the region game, that can easily be solved with some attractor computations. Notice that if a configuration (ℓ,ν) has value $-\infty$, this implies that all configurations (ℓ,ν') with ν' in the same region as ν have value $-\infty$. As we explained at the end of Section 2 for the values $+\infty$, we can then remove configurations of value $-\infty$ by strengthening the guards on transitions, while letting unchanged other finite values.

Then, the (finite) value dVal is obtained as an iterate V^H of the previous operator, with H polynomial in the size of the region game and the maximal weights of $\mathcal{G}$. This means that playing for only a bounded number of steps is equivalent to the original game. In particular, at horizon H, we have that $\mathcal{F}(V^H) = V^{H+1} = \mathsf{dVal}$ so that dVal is a fixpoint of $\mathcal{F}$. As a side effect, this allows one to decompose the clock space $\mathbb{R}^{\mathcal{C}}_{\geq 0}$ into a finite number α of *cells* (a refinement of the classical regions) such that dVal is affine on each cell.

Based on this, we can construct good strategies for Min that have a special form, the so-called *switching strategies* (introduced in [15] in the untimed setting, further extended in the timed setting with only one-clock in [14]).

▶ **Definition 10.** *A switching strategy σ is described by two deterministic memoryless strategies σ^1 and σ^2, as well as a switching threshold K. The strategy σ then consists in playing strategy σ^1 until either we reach a target location, or the finite play has length at least K, in which case we switch to strategy σ^2.*

Our new contribution is as follows:

▶ **Theorem 11.** *In a divergent WTG, for all $\varepsilon > 0$ and $N \in \mathbb{N}$, there exists a switching strategy σ for Min, for which the two components σ^1 and σ^2 satisfy Hypothesis 1, such that for all configurations (ℓ, ν), $\mathsf{dVal}^{\sigma}_{\ell,\nu} \leq \max(-N, \mathsf{dVal}_{\ell,\nu}) + \varepsilon$.*

In particular, if all configurations have a finite deterministic value, **there exists an ε-optimal switching strategy wrt the deterministic value**. In the presence of a configuration with a deterministic value $-\infty$, we build from Theorem 11 a family of switching strategies (indexed by the parameter N) whose value tends to $-\infty$.

The proof of Theorem 11 requires to build both strategies σ^1 and σ^2, as well as a switching threshold K. The second strategy σ^2 only consists in reaching the target and is thus obtained as a deterministic memoryless strategy from a classical attractor computation in the region game $\mathcal{R}(\mathcal{G})$. It is easy to choose σ^2 smooth enough so that it fulfils Hypothesis 1. In contrast, the first strategy σ^1 requires more care. We build it so that it fulfils two properties, that we summarise in saying that σ^1 is *fake-ε-optimal* wrt the deterministic value:

1. each finite play conforming to σ^1 from (ℓ, ν) and reaching the target has a cumulated weight at most $\mathsf{dVal}_{\ell,\nu} + |\rho|\,\varepsilon$ (in particular, if $\mathsf{dVal}_{\ell,\nu} = -\infty$, no such plays should exist);
2. each finite play conforming to σ^1 following a *long enough* cycle in the region game $\mathcal{R}(\mathcal{G})$ has a cumulated weight at most -1.

Here, "fake" means that σ^1 is not obliged to guarantee reaching the target, but if it does so, it must do it with a cumulated weight close to $\mathsf{dVal}_{\ell,\nu}$, the error factor depending linearly on the size of the play. The second property ensures that playing long enough σ^1 without reaching the target results in diminishing the cumulated weight. Then, if the switch happens at horizon K big enough, ($K = (w^e_{\max}|\mathcal{R}(\mathcal{G})|(|L|\alpha + 2) + N)(|\mathcal{R}(\mathcal{G})|(|L|\alpha + 1) + 1)$ suffices for instance), Min is sure that the cumulated weight so far is low enough so that the rest of the play to reach a target location (following σ^2 only) will not make the weight increase too much. In the absence of values $-\infty$ in dVal, the first property allows one to obtain a $K\varepsilon$-optimal strategy even in the case where the switch does not occur (because we reach the target prematurely). The construction of a fake-ε/K-optimal strategy σ^1 (the linear dependency on the length of the play in the first property of fake-optimality is thus taken care by a division by K here) relies on the fact that $\mathcal{F}(\mathsf{dVal}) = \mathsf{dVal}$ to play almost-optimally at horizon 1. More formally:

- For all configurations of value $-\infty$, σ^1 is built as a winning strategy for the Büchi objective "visit infinitely often configurations of a location (ℓ, r) of $\mathcal{R}(\mathcal{G})$ contained in a negative SCC". By definition, all cyclic paths following σ^1 will be inside a negative

SCC, and thus be of cumulated weight at most -1, by divergence of the WTG. Moreover, no plays conforming to σ^1 from such a configuration of value $-\infty$ will reach a target location, since the chosen negative SCC is a trap controlled by Min. It is easy to choose σ^1 smooth enough so that it fulfils Hypothesis 1.

- For the remaining configurations of finite value, we rely upon operator $\mathcal{F}$, letting σ^1 choose a decision that minimises the value at horizon 1. However, because of the guards on clocks, infimum/supremum operators in $\mathcal{F}$ are not necessarily minima/maxima, and we thus need to allow for a small error at each step of the strategy: this is the main difference with the untimed setting, which by the way explains why our definition of switching strategy needed to be adapted. We will use the $\operatorname{arginf}^{\varepsilon}$ operator defined for all mappings $f\colon A \to \mathbb{R}$ and $B \subseteq A$ by $\operatorname{arginf}^{\varepsilon}_B f = \{a \in B \mid f(a) \leq \inf_B f + \varepsilon\}$. Then, for all configurations $(\ell, \nu) \in L_{\mathsf{Min}} \times \mathbb{R}^{\mathcal{C}}_{\geq 0}$, we choose $\sigma^1(\ell, \nu)$ as a pair (δ, t) in

$$\operatorname{arginf}^{\varepsilon/K}_{(\ell,\nu) \xrightarrow{\delta,t} (\ell',\nu')} (t\,\mathsf{wt}(\ell) + \mathsf{wt}(\delta) + \mathsf{dVal}_{\ell',\nu'})$$

  This set is non empty since dVal is a fixpoint of operator $\mathcal{F}$ in this case. Moreover, knowing that the mapping dVal_ℓ is piecewise affine by the results shown in [16], it is possible to choose σ^1 so that it fulfils the measurability (even piecewise continuity) conditions of Hypothesis 1. More precisely, we can consider it to take the same kind of decision for all configurations of a same cell: same transition, and either no delay or a delay jumping to the same border of cell.

The strategy σ^1 thus built makes a small error wrt the optimal at each step. But once again strongly relying on the divergence of the WTG, we can nevertheless show that σ^1 is fake-ε/K-optimal wrt the deterministic value.

Memoryless strategies for Max. WTGs are known to be determined [14], i.e. the deterministic value is also equal to $\mathsf{dVal}_{\ell,\nu} = \sup_{\tau \in \mathsf{dStrat}_{\mathsf{Max}}} \inf_{\sigma \in \mathsf{dStrat}_{\mathsf{Min}}} \mathsf{wt}(\mathsf{Play}((\ell,\nu),\sigma,\tau))$. In this setting, we can turn our study to the point of view of Max, looking for good strategies for this other player. A deterministic strategy τ of Max has an associated value: $\mathsf{dVal}^{\tau}_{\ell,\nu} = \inf_{\sigma \in \mathsf{dStrat}_{\mathsf{Min}}} \mathsf{wt}(\mathsf{Play}((\ell,\nu),\sigma,\tau))$. It is ε-optimal wrt the deterministic value if $\mathsf{dVal}^{\tau}_{\ell,\nu} \geq \mathsf{dVal}_{\ell,\nu} - \varepsilon$ for all (ℓ,ν).

As Max does not wish to go to the target, we show that no switch is necessary to play ε-optimally: memoryless strategies are sufficient to guarantee a value as close as wanted to the deterministic value. For a configuration with a value equal to $-\infty$, all the deterministic strategies for Max are equivalent where they are all equally bad. Without loss of generality, we can therefore suppose that there are no configurations in $\mathcal{G}$ with a value equal to $-\infty$. Then, it is shown in [16] that remaining values are bounded in absolute value by $w^{e}_{\max}|\mathcal{R}(\mathcal{G})|$, since *optimal plays* have no cycles. We use that fact to build a memoryless deterministic strategy τ analogous to strategy σ^1 before:

▶ **Theorem 12.** *In a divergent WTG, there exists a memoryless ε-optimal strategy for player* Max *wrt the deterministic value (that moreover satisfies Hypothesis 1).*

5 Emulate memory with randomness, and vice versa

The main contribution of this article, apart from defining a notion of expected value in weighted timed games, is to relate the different notions of values. In divergent WTGs, memory can thus be fully emulated with stochastic choices, and combining memory and stochastic choices does not bring more power to players, which we summarise by:

▶ **Theorem 13.** *In all divergent WTGs, for all* (ℓ, ν), $\mathsf{dVal}_{\ell,\nu} = \mathsf{Val}_{\ell,\nu} = \mathsf{mVal}_{\ell,\nu}$.

The proof of this result is decomposed into several inequalities on these values. One is easier, and holds for all WTGs: the stochastic value is at most equal to the deterministic value, using the inclusion of deterministic strategies into stochastic ones, and Lemma 7.

▶ **Lemma 14.** *In all WTGs* $\mathcal{G}$*, for all configurations* (ℓ, ν), $\mathsf{Val}_{\ell,\nu} \leq \mathsf{dVal}_{\ell,\nu}$.

We show in the rest of this section the inequalities comparing the deterministic value with the other values: first we show that memoryless stochastic strategies can emulate deterministic ones ($\mathsf{mVal}_{\ell,\nu} \leq \mathsf{dVal}_{\ell,\nu}$); then we show that deterministic strategies can emulate stochastic ones ($\mathsf{dVal}_{\ell,\nu} \leq \mathsf{Val}_{\ell,\nu}$ and $\mathsf{dVal}_{\ell,\nu} \leq \mathsf{mVal}_{\ell,\nu}$).

Simulating deterministic strategies with memoryless strategies. We focus here on showing that, for all configurations (ℓ, ν), $\mathsf{mVal}_{\ell,\nu} \leq \mathsf{dVal}_{\ell,\nu}$. We build a memoryless strategy of Min at least as good as a deterministic strategy. By Theorem 11, we can start from a switching strategy for Min. For $N \in \mathbb{N}$ and $\varepsilon > 0$, we thus consider a switching strategy $\sigma = (\sigma^1, \sigma^2, K)$ of value $\mathsf{dVal}^{\sigma}_{\ell,\nu} \leq \max(-N, \mathsf{dVal}_{\ell,\nu}) + \varepsilon$, and simulate it with a memoryless strategy for Min, denoted η^p, with a probability parameter $p \in (0,1)$. This new strategy is a probabilistic superposition of the two memoryless deterministic strategies σ^1 and σ^2.

Formally, we define $\eta^p(\ell, \nu)$, with $\ell \in L_{\mathsf{Min}}$, depending on the sign of the SCC containing the location (ℓ, r), with r the region of ν, of the region game $\mathcal{R}(\mathcal{G})$.

In a positive SCC, Min always chooses to play σ^1, thus looking for a negative cycle in the next SCCs (in topological order) if any. Formally, letting $(\delta_1, t_1) = \sigma^1(\ell, \nu)$, we define $\eta^p_{\Delta}(\ell, \nu) = \mathsf{Dirac}_{\delta_1}$ and $\eta^p_{\mathbb{R}^+}((\ell, \nu), \delta_1) = \mathsf{Dirac}_{t_1}$.

In a negative SCC, we let $\eta^p(\ell, \nu)$ be the distribution picking $\sigma^1(\ell, \nu)$ with probability p, and $\sigma^2(\ell, \nu)$ with probability $1-p$. Formally, η^p_{Δ} chooses the transition given by $\sigma^1(\ell, \nu)$ with probability p and of σ^2, with probability $1-p$, except if those transitions are equal, in which case η^p_{Δ} chooses it with probability 1. If transitions chosen by σ^1 and σ^2 are distinct, $\eta^p_{\mathbb{R}^+}$ is a Dirac distribution over the corresponding delay. Otherwise, $\eta^p_{\mathbb{R}^+}$ chooses with probability p the delay given by σ^1, and with probability $1-p$ the one given by σ^2.

Theorem 11 ensuring that strategies σ^1 and σ^2 satisfy Hypothesis 1, the superposition η^p also satisfies these hypotheses. Moreover, we use the sufficient condition in Hypothesis 2 to show that η^p is also proper:

▶ **Lemma 15.** *For all* $p \in (0,1)$*, the strategy* η^p *satisfies Hypothesis 2.*

To show the expected result, we prove that $\mathsf{mVal}^{\eta^p}_{\ell,\nu} \leq \max(-N, \mathsf{dVal}_{\ell,\nu}) + 3\varepsilon$ for all (ℓ, ν), for p *close enough to* 1: we conclude that $\mathsf{mVal}_{\ell,\nu} \leq \mathsf{dVal}_{\ell,\nu}$ by taking the limit when N tends to $+\infty$ and ε tends to 0. We get that inequality by showing the following result, paired with the fact that $\mathsf{dVal}^{\sigma}_{\ell,\nu} \leq \max(-N, \mathsf{dVal}_{\ell,\nu}) + \varepsilon$.

▶ **Proposition 16.** *For all configurations* (ℓ, ν) *and* $\varepsilon > 0$ *small enough, there exists* $\tilde{p} \in (0,1)$ *so that for all* $p \in [\tilde{p}, 1)$, $\mathsf{mVal}^{\eta^p}_{\ell,\nu} \leq \mathsf{dVal}^{\sigma}_{\ell,\nu} + 2\varepsilon$.

Proof. Lemma 7 allows us to limit ourselves to deterministic strategies for Max. For all deterministic strategies τ of Max, we compute a lower bound on p independent of τ such that $\mathbb{E}^{\eta^p,\tau}_{\ell,\nu} \leq \mathsf{dVal}^{\sigma}_{\ell,\nu} + 3\varepsilon/2$. By Lemma 7 (with $\varepsilon/2$), we obtain the desired $\mathsf{mVal}^{\eta^p}_{\ell,\nu} \leq \mathsf{dVal}^{\sigma}_{\ell,\nu} + 2\varepsilon$.

The case where the whole region game only contains positive SCCs is easy, since then η^p chooses the transition and delay given by σ^1 with probability 1. By divergence, $\mathcal{G}$ then contains no negative cycles. A play conforming to η^p is also conforming to the deterministic

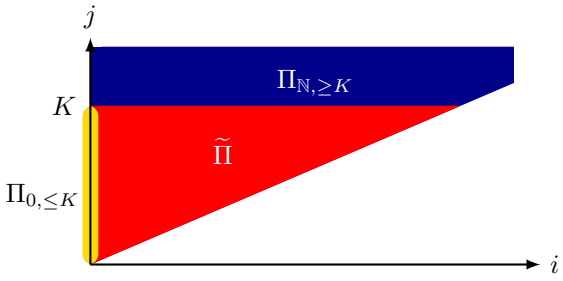


Figure 2 Partition of paths TPaths.

strategy σ^1, so it must be acyclic. In particular, there exists only one play ρ conforming to η^p and τ. This one is also conforming to σ and thus reaches the target with a cumulated weight $\mathsf{wt}_\Sigma(\rho) = \mathbb{E}^{\eta^p,\tau}_{\ell,\nu} \leq \mathsf{dVal}^\sigma_{\ell,\nu}$ as expected.

Now, suppose that the region graph contains at least a negative SCC. Thus, we let $c > 0$ be the maximal size of an elementary cycle of the region game (that visits a pair (ℓ, r) at most once) and $w^- > 0$ be the opposite of the maximal cumulated weight of an elementary negative cycle in $\mathcal{R}(\mathcal{G})$ (necessarily bounded by $w^e_{\max}|\mathcal{R}(\mathcal{G})|$).

We partition the set $\mathsf{FPlays}^{\eta^p,\tau}_{\ell,\nu}$ into subsets $\Pi_{i,j}$ according to the number i of choices of probability $1-p$ along the play (the probability as described previously with the product of the probabilities given by η^p_Δ and $\eta^p_{\mathbb{R}_+}$), and their length j (we always have $i \leq j$). The partition is depicted in Figure 2:

- $\Pi_{\mathbb{N},\geq K}$, depicted in blue, contains all plays with a length greater than K (the switching threshold)
- $\Pi_{0,\leq K}$, depicted in yellow, contains all plays without any probability $1-p$, with a length at most K;
- $\widetilde{\Pi}$, depicted in red, contains the rest of the plays.

We can use the particular shape of the memoryless strategy η^p for Min, and the fact that we fixed a deterministic strategy τ for Max, to decompose the expectation $\mathbb{E}^{\eta^p,\tau}_\rho$ on the partition. Indeed, notice that the set of plays conforming to η^p and τ, from a particular configuration (ℓ, ν), is countable. Moreover, we can associate a probability to each *play* (instead of a probability to a *path*). For a finite play $\rho = (\ell_0, \nu_0) \xrightarrow{\delta_0,t_0} \cdots (\ell_{k-1}, \nu_{k-1})$ conforming to η^p and τ, we let

$$\mathbb{P}^{\eta^p,\tau}_{\ell,\nu}(\rho) = \prod_{i=0}^{k-1} p_i$$

where, for all $i \in \{0, \ldots, k-1\}$

$$p_i = \begin{cases} 1 & \text{if } \ell_i \in L_{\mathsf{Max}} \\ \eta^p_\Delta(\ell_i, \nu_i)(\delta_i) \times \eta^p_{\mathbb{R}_+}((\ell_i, \nu_i), \delta_i)(t_i) & \text{if } \ell_i \in L_{\mathsf{Min}} \end{cases}$$

This definition allows us to recover the probability of a path π using the probability of all plays following π. Then, we obtain easily

$$\mathbb{E}^{\eta^p,\tau}_{\ell,\nu} = \underbrace{\sum_{\rho\in\Pi_{0,\leq K}} \mathsf{wt}(\rho)\mathbb{P}^{\eta^p,\tau}_{\ell,\nu}(\rho)}_{\gamma_{0,\leq K}} + \underbrace{\sum_{\rho\in\Pi_{\mathbb{N},\geq K}} \mathsf{wt}(\rho)\mathbb{P}^{\eta^p,\tau}_{\ell,\nu}(\rho)}_{\gamma_{\mathbb{N},\geq K}} + \underbrace{\sum_{\rho\in\widetilde{\Pi}} \mathsf{wt}(\rho)\mathbb{P}^{\eta^p,\tau}_{\ell,\nu}(\rho)}_{\widetilde{\gamma}} \tag{1}$$

We now compute and bound the three expectations $\gamma_{0,\leq K}$, $\gamma_{\mathbb{N},\geq K}$ and $\widetilde{\gamma}$. In the following, for a set Π of plays, we let $\mathbb{P}^{\eta^p,\tau}_{\ell,\nu}(\Pi) = \sum_{\rho\in\Pi}\mathbb{P}^{\eta^p,\tau}_{\ell,\nu}(\rho)$.

Red zone is such that $\widetilde{\gamma} \leq \varepsilon/4$. All plays in $\widetilde{\Pi}$ have a length at most K: so the cumulated weight of all such play is at most $Kw^e_{\max}$. So, we have

$$\widetilde{\gamma} = \sum_{\rho\in\widetilde{\Pi}} \mathsf{wt}(\rho)\mathbb{P}^{\eta^p,\tau}_{\ell,\nu}(\rho) \leq \sum_{\rho\in\widetilde{\Pi}} Kw^e_{\max}\mathbb{P}^{\eta^p,\tau}_{\ell,\nu}(\rho) = Kw^e_{\max}\mathbb{P}^{\eta^p,\tau}_{\ell,\nu}(\widetilde{\Pi})$$

But, all plays $\rho \in \bigcup_{j\leq K}\Pi_{i,j}$ with $i \leq K$ take i transitions of probability $1-p$. In particular, by bounding all other probabilities by 1, and since there are at most 2^K plays in $\bigcup_{j\leq K}\Pi_{i,j}$, we obtain (using that $1-(1-p)^K \leq 1$)

$$\mathbb{P}^{\eta^p,\tau}_{\ell,\nu}(\widetilde{\Pi}) \leq 2^K \sum_{i=1}^{K-1}(1-p)^i = 2^K\frac{(1-p)(1-(1-p)^K)}{p} \leq 2^K\frac{1-p}{p} \longrightarrow_{p\to 1} 0 \tag{2}$$

If we suppose that

$$p \geq \frac{1}{1+\varepsilon/(4\times 2^K K w^e_{\max})}$$

we obtain as desired

$$\widetilde{\gamma} \leq 2^K K w^e_{\max}\frac{1-p}{p} \leq \frac{\varepsilon}{4}$$

Yellow and blue zones are such that $\gamma_{0,\leq K} + \gamma_{\mathbb{N},\geq K} \leq \mathsf{dVal}^\sigma_{\ell,\nu} + 5\varepsilon/4$. All plays in $\Pi_{0,\leq K}$ reach the target without taking any probability $1-p$ from η^p_Δ, so they are conforming to σ^1. In the case where $\mathsf{dVal}_{\ell,\nu} = -\infty$, $\Pi_{0,\leq K} = \emptyset$ and $\gamma_{0,\leq K} = 0$, since no play conforming to σ^1 from (ℓ,ν) reaches the target. In this case, Min can stay in a cycle with a negative cumulated weight as long as he wants. Now, if $\mathsf{dVal}_{\ell,\nu}$ is finite, Theorem 11 (see [21, Lemma 19]) allows us to show that the cumulated weight of a play in $\Pi_{0,\leq K}$ is at most $\mathsf{dVal}_{\ell,\nu} + K\varepsilon/K = \mathsf{dVal}_{\ell,\nu} + \varepsilon$, as $\mathsf{dVal}_{\ell,\nu} = \inf_{\sigma\in\mathsf{dStrat}_{\mathsf{Min}}}\mathsf{dVal}^\sigma_{\ell,\nu} \leq \mathsf{dVal}^\sigma_{\ell,\nu}$. Therefore, in both cases, we can write

$$\gamma_{0,\leq K} \leq \left(\mathsf{dVal}^\sigma_{\ell,\nu} + \varepsilon\right)\mathbb{P}^{\eta^p,\tau}_{\ell,\nu}(\Pi_{0,\leq K})$$

Let ρ be a play in $\Pi_{i,j}$ with $0 \leq i$ and $j \geq K$. Since η^p only allows cycles in negative SCCs, all region cycles in ρ have a cumulated weight at most -1. By definition of K and the proof of Theorem 11, $\mathsf{wt}(\rho) \leq \mathsf{dVal}^\sigma_{\ell,\nu} \leq \mathsf{dVal}^\sigma_{\ell,\nu} + \varepsilon$.

By summing up the contribution of yellow and blue zones, we get

$$\gamma_{0,\leq K} + \gamma_{\mathbb{N},\geq K} \leq \left(\mathsf{dVal}^\sigma_{\ell,\nu} + \varepsilon\right)\mathbb{P}^{\eta^p,\tau}_{\ell,\nu}(\Pi_{0,\leq K}\cup\Pi_{\mathbb{N},\geq K}) \tag{3}$$

We distinguish two cases.

- If $\mathsf{dVal}^\sigma_{\ell,\nu} \geq -5\varepsilon/4$, having $\mathbb{P}^{\eta^p,\tau}_{\ell,\nu}(\Pi_{0,\leq K}\cup\Pi_{\mathbb{N},\geq K}) \leq 1$, we get

$$\gamma_{0,\leq K} + \gamma_{\mathbb{N},\geq K} \leq \left(\mathsf{dVal}^\sigma_{\ell,\nu} + \frac{5\varepsilon}{4}\right)\mathbb{P}^{\eta^p,\tau}_{\ell,\nu}(\Pi_{0,\leq K}\cup\Pi_{\mathbb{N},\geq K}) \leq \mathsf{dVal}_{\ell,\nu} + \frac{5\varepsilon}{4}$$

- If $\mathsf{dVal}^\sigma_{\ell,\nu} < -5\varepsilon/4$, then by (2),

$$\mathbb{P}^{\eta^p,\tau}_{\ell,\nu}(\Pi_{0,\leq K}\cup\Pi_{\mathbb{N},\geq K}) = 1 - \mathbb{P}^{\eta^p,\tau}_{\ell,\nu}(\widetilde{\Pi}) \geq 1 - 2^K\frac{1-p}{p} \longrightarrow_{p\to 1} 1$$

If we suppose that

$$p \geq \frac{2^K}{2^K + 1 - \frac{\mathsf{dVal}^\sigma_{\ell,\nu} + 5\varepsilon/4}{\mathsf{dVal}^\sigma_{\ell,\nu} + \varepsilon}} \in (0,1)$$

then

$$\mathbb{P}^{\eta^p,\tau}_{\ell,\nu}(\Pi_{0,\leq K} \cup \Pi_{\mathbb{N},\geq K}) \geq \frac{\mathsf{dVal}^\sigma_{\ell,\nu} + 5\varepsilon/4}{\mathsf{dVal}^\sigma_{\ell,\nu} + \varepsilon}$$

and, by negativity of $\mathsf{dVal}^\sigma_{\ell,\nu} + \varepsilon$, we can rewrite (3) as

$$\gamma_{0,\leq K} + \gamma_{\mathbb{N},\geq K} \leq \left(\mathsf{dVal}^\sigma_{\ell,\nu} + \varepsilon\right) \frac{\mathsf{dVal}^\sigma_{\ell,\nu} + 5\varepsilon/4}{\mathsf{dVal}^\sigma_{\ell,\nu} + \varepsilon} = \mathsf{dVal}^\sigma_{\ell,\nu} + \frac{5\varepsilon}{4}$$

In all cases, we have $\gamma_{0,\leq K} + \gamma_{\mathbb{N},\geq K} \leq \mathsf{dVal}^\sigma_{\ell,\nu} + 5\varepsilon/4$.

Gathering all constraints on p. We gather all the lower bounds over p that we need in the proof:

$$\tilde{p} = \begin{cases} \max\left(\frac{1}{1+\varepsilon/(4\times 2^K K w^e_{\max})}, \frac{1}{2}\right) & \text{if } \mathsf{dVal}^\sigma_{\ell,\nu} \geq -5\varepsilon/4 \\ \max\left(\frac{1}{1+\varepsilon/(4\times 2^K K w^e_{\max})}, \frac{1}{2}, \frac{2^K}{2^K+1-\frac{\mathsf{dVal}^\sigma_{\ell,\nu}+5\varepsilon/4}{\mathsf{dVal}^\sigma_{\ell,\nu}+\varepsilon}}\right) & \text{otherwise} \end{cases}$$

Then, for $p \in (\tilde{p}, 1)$, we have $\mathbb{E}^{\eta^p,\tau}_{\ell,\nu} \leq \mathsf{dVal}^\sigma_{\ell,\nu} + 3\varepsilon/2$. Since $\tilde{p}$ does not depend on τ, we conclude that for $p \in (\tilde{p}, 1)$, we have $\mathsf{mVal}^{\eta^p}_{\ell,\nu} \leq \mathsf{dVal}^\sigma_{\ell,\nu} + 2\varepsilon$. ◀

Simulating (stochastic) strategies with deterministic strategies. Finally, we show that, for all configurations (ℓ,ν), $\mathsf{dVal}_{\ell,\nu} \leq \mathsf{Val}_{\ell,\nu}$ and $\mathsf{dVal}_{\ell,\nu} \leq \mathsf{mVal}_{\ell,\nu}$. To do so, we consider a proper strategy $\eta \in \mathsf{Strat}^{\mathsf{p}}_{\mathsf{Min}}$ of Min and an initial configuration (ℓ,ν). We build a deterministic strategy σ such that $\mathsf{dVal}^\sigma_{\ell,\nu} \leq \mathsf{Val}^\eta_{\ell,\nu} + \varepsilon$. Thus, in the case where $\mathsf{Val}_{\ell,\nu} \neq -\infty$, we can consider η to be an ε-optimal strategy so that $\mathsf{dVal}^\sigma_{\ell,\nu} \leq \mathsf{Val}_{\ell,\nu} + 2\varepsilon$ which allows us to conclude. In the case where $\mathsf{Val}_{\ell,\nu} = -\infty$, then, for all $N \in \mathbb{N}$, we can consider η to be such that $\mathsf{Val}^\eta_{\ell,\nu} \leq -N$. Then, $\mathsf{dVal}^\sigma_{\ell,\nu} \leq -N + \varepsilon$, which implies that $\mathsf{dVal}_{\ell,\nu} = -\infty$, by taking the limit when N tends to $+\infty$.

The deterministic strategy σ uses the same kind of *memory* as η (in particular, it will be memoryless if η is memoryless). However, we want this strategy to be relatively simple to define, independent of the memory of η. Intuitively, we want to build a switching strategy (as in Section 4) on a game induced by the memory of η, i.e. a deterministic strategy σ^1 that uses the memory capabilities of η, a memoryless deterministic strategy σ^2 obtained by an attractor in the region game, and a threshold K. Strategy σ then consists in playing σ^1 for at most K steps, before switching to strategy σ^2. The construction of σ^1 is done in a similar way as in the deterministic case, Min always choosing the best possible candidate according to the choices of η, thus trying to minimise the immediate reward obtained in one turn. Under this condition, we verify that σ^1 satisfies some properties similar to the fake-ε-optimality encountered in Section 4. Then, by mimicking the techniques of Theorem 11, we obtain that the switching strategy η obtained from η^1 satisfies the desired inequality $\mathsf{dVal}^\sigma_{\ell,\nu} \leq \mathsf{Val}^\eta_{\ell,\nu} + \varepsilon$.

6 Conclusion

We have introduced stochastic strategies for WTGs, showing that, in divergent games, Min can use randomisation to emulate memory, and vice versa. We aim at extending our study to more general WTGs. As a first step, we may consider the class of almost-divergent WTGs (adding the possibility for an execution following a region cycle to have weight *exactly 0*), used in [12, 17] to obtain an approximation schema of the optimal value. We wonder if similar ε-optimal switching strategies may exist also in this context, one of the crucial argument in order to extend our emulation result. Another question concerns the implementability of the randomised strategies: even if they use no memory, they still need to know the precise current clock valuation. In (non-weighted) timed games, previous work [18] aimed at removing this need for precision, by using stochastic strategies where the delays are chosen with probability distributions that do not require exact knowledge of the clocks measurements. In our setting, we aim at further studying the implementability of the randomised strategies of Min in WTGs, e.g. by requiring them to be robust against small imprecisions.

References

1 Rajeev Alur, Mikhail Bernadsky, and P. Madhusudan. Optimal reachability for weighted timed games. In *Proceedings of the 31st International Colloquium on Automata, Languages and Programming (ICALP'04)*, volume 3142 of *LNCS*, pages 122–133. Springer, 2004.

2 Rajeev Alur and David L. Dill. A theory of timed automata. *Theoretical Computer Science*, 126(2):183–235, 1994.

3 Rajeev Alur, Salvatore La Torre, and George J. Pappas. Optimal paths in weighted timed automata. *Theoretical Computer Science*, 318(3):297–322, 2004.

4 Eugene Asarin and Oded Maler. As soon as possible: Time optimal control for timed automata. In *Hybrid Systems: Computation and Control*, volume 1569 of *LNCS*, pages 19–30. Springer, 1999.

5 Gerd Behrmann, Ansgar Fehnker, Thomas Hune, Kim Larsen, Paul Pettersson, Judi Romijn, and Frits Vaandrager. Minimum-cost reachability for priced time automata. In *International Workshop on Hybrid Systems: Computation and Control*, pages 147–161. Springer, 2001.

6 Nathalie Bertrand, Patricia Bouyer, Thomas Brihaye, and Pierre P. Carlier. When are stochastic transition systems tameable? *Journal of Logical and Algebraic Methods in Programming*, 99:41–96, 2018.

7 Nathalie Bertrand, Patricia Bouyer, Thomas Brihaye, Quentin Menet, Christel Baier, Marcus Größer, and Marcin Jurdzinski. Stochastic timed automata. *Log. Methods Comput. Sci.*, 10(4), 2014.

8 Patricia Bouyer, Thomas Brihaye, Véronique Bruyère, and Jean-François Raskin. On the optimal reachability problem of weighted timed automata. *Formal Methods in System Design*, 31(2):135–175, 2007.

9 Patricia Bouyer, Thomas Brihaye, Pierre Carlier, and Quentin Menet. Compositional design of stochastic timed automata. In Alexander S. Kulikov and Gerhard J. Woeginger, editors, *Computer Science - Theory and Applications - 11th International Computer Science Symposium in Russia, CSR 2016, St. Petersburg, Russia, June 9-13, 2016, Proceedings*, volume 9691 of *Lecture Notes in Computer Science*, pages 117–130. Springer, 2016.

10 Patricia Bouyer, Thomas Brihaye, Mickael Randour, Cédric Rivière, and Pierre Vandenhove. Decisiveness of stochastic systems and its application to hybrid models. In Davide Bresolin and Jean-François Raskin, editors, *Proceedings of the 11th International Symposium on Games, Automata, Logics, and Formal Verification (GandALF'20)*, volume 326 of *Electronic Proceedings in Theoretical Computer Science*, 2020.

11 Patricia Bouyer, Franck Cassez, Emmanuel Fleury, and Kim G. Larsen. Optimal strategies in priced timed game automata. In Kamal Lodaya and Meena Mahajan, editors, *FSTTCS 2004: Foundations of Software Technology and Theoretical Computer Science*, pages 148–160, Berlin, Heidelberg, 2005. Springer Berlin Heidelberg.

12 Patricia Bouyer, Samy Jaziri, and Nicolas Markey. On the value problem in weighted timed games. In *Proceedings of the 26th International Conference on Concurrency Theory (CONCUR'15)*, volume 42 of *Leibniz International Proceedings in Informatics*, pages 311–324. Leibniz-Zentrum für Informatik, 2015.

13 Thomas Brihaye, Véronique Bruyère, and Jean-François Raskin. On optimal timed strategies. In Paul Pettersson and Wang Yi, editors, *Formal Modeling and Analysis of Timed Systems*, pages 49–64, Berlin, Heidelberg, 2005. Springer Berlin Heidelberg.

14 Thomas Brihaye, Gilles Geeraerts, Axel Haddad, Engel Lefaucheux, and Benjamin Monmege. Simple priced timed games are not that simple. In *Proceedings of the 35th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS'15)*, volume 45 of *LIPIcs*, pages 278–292. Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 2015.

15 Thomas Brihaye, Gilles Geeraerts, Axel Haddad, and Benjamin Monmege. Pseudopolynomial iterative algorithm to solve total-payoff games and min-cost reachability games. *Acta Informatica*, 54(1):85–125, 2017.

16 Damien Busatto-Gaston, Benjamin Monmege, and Pierre-Alain Reynier. Optimal reachability in divergent weighted timed games. In *Foundations of Software Science and Computation Structures - 20th International Conference, FOSSACS 2017, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2017, Uppsala, Sweden, April 22-29, 2017, Proceedings*, pages 162–178, 2017.

17 Damien Busatto-Gaston, Benjamin Monmege, and Pierre-Alain Reynier. Symbolic approximation of weighted timed games. In Sumit Ganguly and Paritosh Pandya, editors, *Proceedings of the 38th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS'18)*, volume 122 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 28:1–28:16. Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 2018.

18 Krishnendu Chatterjee, Thomas A. Henzinger, and Vinayak S. Prabhu. Trading infinite memory for uniform randomness in timed games. In *Hybrid Systems: Computation and Control, 11th International Workshop, HSCC 2008, St. Louis, MO, USA, April 22-24, 2008. Proceedings*, pages 87–100, 2008.

19 Marcin Jurdziński and Ashutosh Trivedi. Reachability-time games on timed automata. In *Proceedings of the 34th International Colloquium on Automata, Languages and Programming (ICALP'07)*, volume 4596 of *LNCS*, pages 838–849. Springer, 2007.

20 Benjamin Monmege, Julie Parreaux, and Pierre-Alain Reynier. Reaching your goal optimally by playing at random with no memory. In Igor Konnov and Laura Kovács, editors, *Proceedings of the 31st International Conference on Concurrency Theory (CONCUR 2020)*, volume 171 of *LIPIcs*, pages 26:1–26:21. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020.

21 Benjamin Monmege, Julie Parreaux, and Pierre-Alain Reynier. Playing stochastically in weighted timed games to emulate memory. Technical Report 2105.00984, arXiv, 2021. `arXiv:2105.00984`.

Smooth Approximations and Relational Width Collapses

Antoine Mottet
Department of Algebra, Faculty of Mathematics and Physics,
Charles University, Prague, Czech Republic

Tomáš Nagy
Institut für Diskrete Mathematik und Geometrie, Technische Universität Wien, Austria
Department of Algebra, Faculty of Mathematics and Physics,
Charles University, Prague, Czech Republic

Michael Pinsker
Institut für Diskrete Mathematik und Geometrie, Technische Universität Wien, Austria
Department of Algebra, Faculty of Mathematics and Physics,
Charles University, Prague, Czech Republic

Michał Wrona
Theoretical Computer Science Department, Jagiellonian University, Kraków, Poland

Abstract

We prove that relational structures admitting specific polymorphisms (namely, canonical pseudo-WNU operations of all arities $n \geq 3$) have low relational width. This implies a collapse of the bounded width hierarchy for numerous classes of infinite-domain CSPs studied in the literature. Moreover, we obtain a characterization of bounded width for first-order reducts of unary structures and a characterization of MMSNP sentences that are equivalent to a Datalog program, answering a question posed by Bienvenu *et al.*. In particular, the bounded width hierarchy collapses in those cases as well.

2012 ACM Subject Classification Theory of computation → Logic

Keywords and phrases local consistency, bounded width, constraint satisfaction problems, polymorphisms, smooth approximations

Digital Object Identifier 10.4230/LIPIcs.ICALP.2021.138

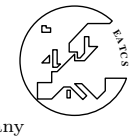
Category Track B: Automata, Logic, Semantics, and Theory of Programming

Related Version *Full Version*: `https://arxiv.org/abs/2102.07531` [44]

Funding *Antoine Mottet*: this author has received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreement No 771005).
Tomáš Nagy: this author has received funding from the Austrian Science Fund (FWF) through project No P32337 and through Lise Meitner Grant No M 2555-N35 and from the Czech Science Foundation (grant No 18-20123S).
Michael Pinsker: this author has received funding from the Austrian Science Fund (FWF) through project No P32337 and from the Czech Science Foundation (grant No 18-20123S).
Michał Wrona: this author is partially supported by National Science Centre, Poland grant number 2020/37/B/ST6/01179.

48th International Colloquium on Automata, Languages, and Programming (ICALP 2021).
Editors: Nikhil Bansal, Emanuela Merelli, and James Worrell; Article No. 138; pp. 138:1–138:20

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany



1 Introduction

Local consistency checking is an algorithmic technique that is central in computer science. Intuitively speaking, it consists in propagating local information through a structure so as to infer global information (consider, e.g., computing the transitive closure of a relation as deriving global information from local one). Local consistency checking has a prominent role in the area of constraint satisfaction, where one is given a set of variables V and constraints and one has to find a satisfying assignment $h\colon V \to D$ for the constraints. In this setting, the local consistency algorithm can be used to decrease the size of the search space efficiently or even to correctly solve some constraint satisfaction problems in polynomial time (for example, 2-SAT or Horn-SAT). However, the use of local consistency methods is not limited to constraint satisfaction. Indeed, local consistency checking is also used for such problems as the graph isomorphism problem, where it is is known as the Weisfeiler-Leman algorithm. Again, the technique can be used to derive implied constraints that an isomorphism between two graphs has to satisfy so as to narrow down the search space, but local consistency is in fact powerful enough to solve the graph isomorphism problem over any non-trivial minor-closed class of graphs [36]. Notably, the best algorithm for graph isomorphism to date also uses local consistency as a subroutine [3]. Finally, local consistency can be used to solve games involved in formal verification such as parity games and mean-payoff games [16].

One of the reasons for the ubiquity of local consistency is that its underlying principles can be described in many different languages, such as the language of category theory [1], in the language of finite model theory (by Spoiler-Duplicator games [38] or by homomorphism duality [2]), and logical definability (in Datalog, or infinitary logics with bounded number of variables). For constraint satisfaction problems over a finite template, the power of local consistency checking can additionally be characterised algebraically. More precisely, there are conditions on the set of *polymorphisms* of a template $\mathbb{A}$ such that local consistency correctly solves its constraint satisfaction problem $\mathrm{CSP}(\mathbb{A})$ if, and only if, the polymorphisms of $\mathbb{A}$ satisfy these conditions. Moreover, whenever local consistency correctly solves $\mathrm{CSP}(\mathbb{A})$, where $\mathbb{A}$ is finite, then in fact only a very restricted form of local consistency checking is needed [4]. This fact is known as the *collapse of the bounded width hierarchy*, and it has strong consequences both for complexity and logic. On the one hand, the collapse gives efficient algorithms that are able to solve *all* the CSPs that are solvable by local consistency methods, and in fact this gives a polynomial-time algorithm solving instances of the *uniform CSP*. On the other hand, this collapse induces collapses in all the areas mentioned at the beginning of this paragraph.

Many natural problems from computer science can only be phrased as CSPs where the template is infinite. This is the case for linear programming, some reasoning problems in artificial intelligence such as ontology-mediated data access, or even problems as simple to formulate as the digraph acyclicity problem. In order to understand the power of local consistency in more generality it is thus necessary to consider its use for infinite-domain CSPs. Infinite-domain CSPs with an ω-categorical template form a very general class of problems for which the algebraic approach from the finite case can be extended, and numerous results in the recent years have shown the power of this approach. An algebraic characterisation of local consistency checking for infinite-domain CSPs is, however, missing. In fact, the negative results of [19], refined in [35], show that no purely algebraic description of local consistency is possible for CSPs with ω-categorical templates; this is even the case for temporal CSPs [20]. These negative results are to be compared with the recent result by Mottet and Pinsker [45] that did provide an algebraic description of local consistency for several subclasses of ω-categorical templates.

In the finite, the algebraic characterisation of local consistency relies on a set of algebraic tools whose development eventually led to the solutions of the Feder-Vardi dichotomy conjecture. Bulatov's proof of the Feder-Vardi conjecture [30] builds on his theory of edge-colored algebras, that were also used in his characterisation of bounded width [29]; Zhuk's proof [50, 51] relies on the concept of absorption, which was developed by Barto and Kozik in their effort to prove the bounded width conjecture [5, 7]. Comparable algebraic tools, or a general theory, are at the moment missing in the theory of infinite-domain CSPs, even with an ω-categorical template. The most general results obtained so far use *canonical operations*, which behave like operations on finite sets, and for which it is sometimes possible to mimic the universal-algebraic approach to finite-domain CSPs. Canonical operations alone do not seem to be sufficient in full generality and a characterisation of their applicability is also missing, but on the positive side their applicability covers a vast majority of the results that were proved in the area. The application of canonical operations to approach the question of local consistency for infinite-domain CSPs has only been started recently [18, 45, 49].

1.1 Results

In the present paper, we focus on applying the theory of canonical functions to study the power of local consistency checking for constraint satisfaction problems over ω-categorical templates. Our objective is two-fold: on the one hand, we wish to obtain generic sufficient conditions that imply that local consistency solves a given CSP, and on the other hand we wish to understand the amount of locality needed for local consistency to solve the CSP, as measured by the so-called *relational width*. The definitions of all concepts mentioned in this section can be found in the preliminaries.

In order to solve the first objective, we build on recent work by Mottet and Pinsker [45] and expand the use of their *smooth approximations* to fully suit *equational (non-)affineness*, which is roughly the algebraic situation imposed by local consistency solvability. The main technical contribution is a new *loop lemma* that exploits deep algebraic tools from the finite [6] and, assuming the use of canonical functions is unfruitful, allows to obtain the existence of polymorphisms of every arity $n \geq 2$ and satisfying certain strong symmetry conditions. Using this loop lemma, we are able to obtain a characterisation of bounded width for particular classes of templates, namely for first-order reducts of unary structures and for certain structures related to the logic MMSNP. In particular, we obtain a decidable necessary and sufficient condition for an MMSNP sentence to be equivalent to a Datalog program, solving an open problem from [11, 34].

▶ **Theorem 1.** *The Datalog-rewritability problem for MMSNP is decidable, and is 2NExpTime-complete.*

In order to solve the second objective, we prove that sufficiently locally consistent instances of a given CSP can be turned into locally consistent instances of a finite-domain CSP. If the finite-domain CSP has bounded width then it has relational width $(2, 3)$ by [4], which allows us to obtain a collapse of bounded width for structures whose clone of canonical polymorphisms satisfy suitable identities, thus obtaining a similar collapse as in the finite case. In particular, it turns out that the relational width of a structure then only depends on certain simple parameters of the structure whose automorphism group is considered in the notion of canonicity.

▶ **Theorem 2.** *Let $k, \ell \geq 1$, and let $\mathbb{A}$ be a first-order reduct of a k-homogeneous ℓ-bounded ω-categorical structure $\mathbb{B}$.*

- *If the clone of $\mathrm{Aut}(\mathbb{B})$-canonical polymorphisms of $\mathbb{A}$ contains pseudo-WNUs modulo $\overline{\mathrm{Aut}(\mathbb{B})}$ of all arities $n \geq 3$, then $\mathbb{A}$ has relational width $(2k, \max(3k, \ell))$.*
- *If the clone of $\mathrm{Aut}(\mathbb{B})$-canonical polymorphisms of $\mathbb{A}$ contains pseudo-totally symmetric operations modulo $\overline{\mathrm{Aut}(\mathbb{B})}$ of all arities, then $\mathbb{A}$ has relational width $(k, \max(k+1, \ell))$.*

Note that every finite structure $\mathbb{A}$ with domain $\{a_1, \ldots, a_n\}$ is a first-order reduct of the structure $(\{a_1, \ldots, a_n\}; \{a_1\}, \ldots, \{a_n\})$, which is easily seen to be 1-homogeneous and 2-bounded. Thus the width obtained in Theorem 2 coincides with the width given by Barto's collapse result from [4].

As a corollary of Theorem 2, we obtain a collapse of the bounded width hierarchy for first-order reducts of the unary structures mentioned above, as well as of numerous other structures studied in the literature [22, 18, 17, 39].

▶ **Corollary 3.** *Let $\mathbb{A}$ be a structure that has bounded width. If $\mathbb{A}$ is a first-order reduct of:*

- *the universal homogeneous graph $\mathbb{G}$ or tournament $\mathbb{T}$, or of a unary structure, then $\mathbb{A}$ has relational width at most $(4, 6)$;*
- *the universal homogeneous K_n-free graph $\mathbb{H}_n$, where $n \geq 3$, then at most $(2, n)$;*
- *$(\mathbb{N}; =)$, the countably infinite equivalence relation with infinitely many equivalence classes $\mathbb{C}^\omega_\omega$, or the universal homogeneous partial order $\mathbb{P}$, then at most $(2, 3)$.*

Proof. A first-order reduct of $\mathbb{G}$ or $\mathbb{T}$ has bounded width if and only if the algebraic condition in the first item of Theorem 2 is satisfied [45]. Since both $\mathbb{G}$ and $\mathbb{T}$ are 2-homogeneous and 3-bounded our claim follows. First-order reducts of $\mathbb{H}_n$, $(\mathbb{N}; =)$ or $\mathbb{C}^\omega_\omega$ have bounded width if and only if the condition in the second item of Theorem 2 is satisfied, by [17], [14] and [26], respectively. Since $\mathbb{H}_n$ is 2-homogeneous and n-bounded, and since both $(\mathbb{N}; =)$ and $\mathbb{C}^\omega_\omega$ are 2-homogeneous and 3-bounded, the claimed bound follows.

By appeal to Theorems 2 and 23 in the present paper our claim holds for first-order reducts of unary structures.

Finally, a first-order reduct of $\mathbb{P}$ with bounded width is either homomorphically equivalent to a first-order reduct of $(\mathbb{Q}; <)$ or it satisfies the algebraic condition in the second item of Theorem 2 [39]. In the latter case we are done by Theorem 2, in the former we appeal to the syntactical characterization of first-order reducts of $(\mathbb{Q}; <)$. Indeed, such a structure has bounded width iff it is definable by a conjunction of so-called Ord-Horn clauses [20]. It then follows by [28] that a first-order reduct of $(\mathbb{Q}; <)$ with bounded width has relational width $(2, 3)$. The result for $\mathbb{P}$ follows. ◀

The following example shows that for some of the structures under consideration, the bounds on relational width provided by Corollary 3 are tight.

▶ **Example 4.** To show the tightness of the first bound in the case of the universal homogeneous graph, we exhibit a first-order reduct $\mathbb{A}$ such that for all $i \leq j$ with $1 \leq j < 4$ or $1 \leq i < 6$ there exists a non-trivial, (i, j)-minimal instance of $\mathrm{CSP}(\mathbb{A})$ that has no solution. Let $\mathbb{B} := (A; E)$ be the universal homogeneous graph with edge relation E, and let $N := (A^2 \setminus E) \cap \neq$. Consider the first-order reduct $\mathbb{A} := (A; R_=, R_\neq)$ of $\mathbb{B}$, where $R_= := \{(a, b, c, d) \in A^4 \mid E(a, b) \wedge E(c, d)$ or $N(a, b) \wedge N(c, d)\}$ and $R_\neq := \{(a, b, c, d) \in A^4 \mid E(a, b) \wedge N(c, d)$ or $N(a, b) \wedge E(c, d)\}$.

It can be seen that $\mathbb{A}$ has bounded width, so that Theorem 2 implies that $\mathbb{A}$ has relational width $(4, 6)$. It is easy to see that the instance

$$\Phi = R_=(v_1, v_2, v_3, v_4) \wedge R_\neq(v_1, v_2, v_3, v_4)$$

is non-trivial, (i,j)-minimal for all $i \leq j$ with $1 \leq i \leq 3$, and has no solution. Moreover, the $(4,5)$-minimal instance equivalent to the instance

$$\Psi = R_{=}(v_1, v_2, v_3, v_4) \wedge R_{\neq}(v_3, v_4, v_5, v_6) \wedge R_{\neq}(v_1, v_2, v_5, v_6)$$

is non-trivial and has no solution. It follows that the exact relational width of $\mathbb{A}$ is $(4,6)$.

The bounds on relational width provided by the second and third item of Corollary 3 are easily seen to be tight as well. Indeed, let $n \geq 3$, let $\mathbb{B} := (A; E)$ be the universal homogeneous K_n-free graph, let $N := (A^2 \setminus E) \cap \neq$ and let $\mathbb{A} := (A; E, N)$. $\mathbb{A}$ is preserved by canonical pseudo-totally symmetric operations modulo $\overline{\mathrm{Aut}(\mathbb{B})}$ of all arities and has therefore relational width $(2,n)$ by Theorem 2. But the non-trivial, $(2, n-1)$-minimal instance

$$\Phi = \bigwedge_{1 \leq i \neq j \leq n} E(v_i, v_j)$$

has no solution; moreover, the instance $\Psi = E(v_1, v_2) \wedge N(v_1, v_2)$ is non-trivial, $(1,j)$-minimal for every $j \geq 1$ and has no solution either.

For the other structures from Corollary 3, the tightness of the bound can be shown similarly.

1.2 Related results

Local consistency for ω-categorical structures was studied for the first time in [13] where basic notions were introduced and some basic results provided. First-order reducts of certain k-homogeneous ℓ-bounded structures with bounded width were characterized in [45, 20].

A structure $\mathbb{A}$ has *bounded strict width* [33] if not only $\mathrm{CSP}(\mathbb{A})$ is solvable by local consistency, but moreover every partial solution of a locally consistent instance can be extended to a total solution. The articles [49] and [48] give the upper bound $(2,\ell)$ on the relational width for some classes of 2-homogeneous, ℓ-bounded structures under the stronger assumption of bounded strict width; it also follows from [49] that first-order reducts of $\mathbb{H}_n$ with bounded width have relational width at most $(2,n)$.

1.3 Organisation of the present article

In Section 2 we provide the basic notions and definitions. The reduction to the finite using canonical functions which leads to the collapse of the bounded width hierarchy is given in Section 3. We then extend the algebraic theory of smooth approximations in Section 4 before applying it to first-order reducts of unary structures and MMSNP in Section 5. For lack of space, some proofs are omitted and can be found in the full version of the paper.

2 Preliminaries

2.1 Structures and model-theoretic notions

For sets B, I, the *orbit* of a tuple $b \in B^I$ under the action of a permutation group $\mathcal{G}$ on B is the set $\{\alpha(b) \mid \alpha \in \mathcal{G}\}$. A countable structure $\mathbb{B}$ is *ω-categorical* if its automorphism group $\mathrm{Aut}(\mathbb{B})$ is *oligomorphic*, i.e., for all $n \geq 1$, the number of orbits of the action of $\mathrm{Aut}(\mathbb{B})$ on n-tuples is finite. For $\ell \geq 1$, we say that $\mathbb{B}$ is *ℓ-bounded* if for every finite $\mathbb{X}$, if all substructures $\mathbb{Y}$ of $\mathbb{X}$ of size at most ℓ embed in $\mathbb{B}$, then $\mathbb{X}$ embeds in $\mathbb{B}$. For $k \geq 1$, we say that $\mathbb{B}$ is *k-homogeneous* if for all tuples a, b of arbitrary finite length, if all k-subtuples of a

and b are in the same orbit under $\mathrm{Aut}(\mathbb{B})$, then a and b are in the same orbit under $\mathrm{Aut}(\mathbb{B})$. We say that $\mathbb{B}$ is *unary* if all relations in its signature are unary. A *first-order reduct* of a structure $\mathbb{B}$ is a structure on the same domain whose relations have a first-order definition in $\mathbb{B}$.

2.2 Polymorphisms, clones and identities

A *polymorphism* of a relational structure $\mathbb{A}$ is a homomorphism from some finite power of $\mathbb{A}$ to $\mathbb{A}$. The set of all polymorphisms of a structure $\mathbb{A}$ is denoted by $\mathrm{Pol}(\mathbb{A})$; it is a *function clone*, i.e., a set of finitary operations on a fixed set which contains all projections and which is closed under arbitrary compositions.

If $\mathcal{C}$ is a function clone, then we denote the domain of its functions by C; we say that $\mathcal{C}$ *acts on* C. The clone $\mathcal{C}$ also naturally acts (componentwise) on C^l for any $l \geq 1$, on any invariant subset S of C (by restriction), and on the classes of any invariant equivalence relation $\sim$ on an invariant subset S of C (by its action on representatives of the classes). We write $\mathcal{C} \curvearrowright C^l$, $\mathcal{C} \curvearrowright S$ and $\mathcal{C} \curvearrowright S/\!\sim$ for these actions. Any action $\mathcal{C} \curvearrowright S/\!\sim$ is called a *subfactor* of $\mathcal{C}$, and we also call the pair $(S, \sim)$ a subfactor. A subfactor $(S, \sim)$ is *minimal* if $\sim$ has at least two classes and no proper subset of S intersecting at least two $\sim$-classes is invariant under $\mathcal{C}$. For a clone $\mathcal{C}$ acting on a set X and $Y \subseteq X$ we write $\langle Y \rangle_{\mathcal{C}}$ for the smallest $\mathcal{C}$-invariant subset of X containing Y.

For $n \geq 1$, a k-ary operation f defined on the domain C of a permutation group $\mathcal{G}$ is *n-canonical* with respect to $\mathcal{G}$ if for all $a_1, \ldots, a_k \in C^n$ and all $\alpha_1, \ldots, \alpha_k \in \mathcal{G}$ there exists $\beta \in \mathcal{G}$ such that $f(a_1, \ldots, a_k) = \beta \circ f(\alpha_1(a_1), \ldots, \alpha_k(a_k))$. In particular, f induces an operation on the set $C^n/\mathcal{G}$ of $\mathcal{G}$-orbits of n-tuples. If all functions of a function clone $\mathcal{C}$ are n-canonical with respect to $\mathcal{G}$, then $\mathcal{C}$ acts on $C^n/\mathcal{G}$ and we write $\mathcal{C}^n/\mathcal{G}$ for this action; if $\mathcal{G}$ is oligomorphic then $\mathcal{C}^n/\mathcal{G}$ is a function clone on a finite set. A function is *canonical* with respect to a permutation group $\mathcal{G}$ if it is n-canonical with respect to $\mathcal{G}$ for all $n \geq 1$. We say that it is *diagonally canonical* if it satisfies the definition of canonicity in case $\alpha_1 = \cdots = \alpha_k$.

We write $\mathcal{G}_{\mathcal{C}}$ to denote the largest permutation group contained in a function clone $\mathcal{C}$, and say that $\mathcal{C}$ is oligomorphic if $\mathcal{G}_{\mathcal{C}}$ is oligomorphic. For $n \geq 1$, the *n-canonical (canonical) part* of $\mathcal{C}$ is the clone of those functions of $\mathcal{C}$ which are n-canonical (canonical) with respect to $\mathcal{G}_{\mathcal{C}}$. We write $\mathcal{C}^{\mathrm{can}}_n$ and $\mathcal{C}^{\mathrm{can}}$ for these sets which form themselves function clones.

For a set of functions $\mathcal{F}$ over the same fixed set C we write $\overline{\mathcal{F}}$ for the set of those functions g such that for all finite subsets F of C, there exists a function in $\mathcal{F}$ which agrees with g on F. We say that f *locally interpolates* g modulo $\mathcal{G}$, where f, g are k-ary functions and $\mathcal{G}$ is a permutation group all of which act on the same domain, if $g \in \overline{\{\beta \circ f(\alpha_1, \ldots, \alpha_k) \mid \beta, \alpha_1, \ldots, \alpha_k \in \mathcal{G}\}}$. Similarly, we say that f *diagonally interpolates* g modulo $\mathcal{G}$ if f locally interpolates g with $\alpha_1 = \cdots = \alpha_k$. If $\mathcal{G}$ is the automorphism group of a *Ramsey structure* in the sense of [12], then every function on its domain locally (diagonally) interpolates a canonical (diagonally canonical) function modulo $\mathcal{G}$ [25, 21]. We say that a clone $\mathcal{D}$ locally interpolates a clone $\mathcal{C}$ modulo a permutation group $\mathcal{G}$ if for every $g \in \mathcal{D}$ there exists $f \in \mathcal{C}$ such that g locally interpolates f modulo $\mathcal{G}$. A clone $\mathcal{C}$ is a *model-complete core* if its unary functions are equal to $\overline{\mathcal{G}_{\mathcal{C}}}$. A structure $\mathbb{A}$ is called a model- complete core if its polymorphism clone is.

A function f is *idempotent* if $f(x, \ldots, x) = x$ for all values x of its domain; a function clone is idempotent if all of its functions are. A k-ary operation w is called a *weak near-unanimity (WNU)* operation if it satisfies the set of identities containing an equation for each pair of terms in $\{w(x, \ldots, x, y), w(x, y, \ldots, x), \ldots, w(y, x, \ldots, x)\}$. It is called *totally symmetric* if $w(x_1, \ldots, x_k) = w(y_1, \ldots, y_k)$ whenever $\{x_1, \ldots, x_k\} = \{y_1, \ldots, y_k\}$ (where $1 \leq |\{x_1, \ldots, x_k\}| \leq k$). Each set of identities also has a *pseudo*-variant obtained by composing each term appearing in the identities with a distinct unary function symbol. For example,

a ternary pseudo-WNU operation f satisfies the identities: $e_1 \circ f(y,x,x) = e_2 \circ f(x,y,x)$, $e_3 \circ f(y,x,x) = e_4 \circ f(x,x,y)$ and $e_5 \circ f(x,y,x) = e_6 \circ f(x,x,y)$. If $\mathcal{C}$ is a function clone and $\mathcal{U} \subseteq \mathcal{C}$ is a set of unary functions, then $\mathcal{C}$ *satisfies a set of pseudo-identities modulo* $\mathcal{U}$ if it satisfies the identities in such a way that the unary function symbols are assigned values in $\mathcal{U}$.

An arity-preserving map $\xi\colon \mathcal{C} \to \mathcal{D}$ between function clones is called a *clone homomorphism* if it preserves projections, i.e., maps every projection in $\mathcal{C}$ to the corresponding projection in $\mathcal{D}$, and compositions, i.e., it satisfies $\xi(f \circ (g_1,\dots,g_n)) = \xi(f) \circ (\xi(g_1),\dots,\xi(g_n))$ for all $n,m \geq 1$ and all n-ary $f \in \mathcal{C}$ and m-ary $g_1,\dots,g_n \in \mathcal{C}$. An arity-preserving map ξ is a *minion homomorphism* if it preserves compositions with projections, i.e., compositions where $g_1,\dots,g_n$ are projections. We say that a function clone $\mathcal{C}$ is *equationally trivial* if it has a clone homomorphism to the clone $\mathcal{P}$ of projections over the two-element domain, and *equationally non-trivial* otherwise. We also say that $\mathcal{C}$ is *equationally affine* if it has a clone homomorphism to an *affine clone*, i.e., a clone of affine maps over a finite module. It is known that a finite idempotent clone is either equationally affine or it contains WNU operations of all arities $n \geq 3$ ([43], this stronger version is attributed to E. Kiss in [40, Theorem 2.8]). Similarly, if $\mathbb{A}$ is an ω-categorical model-complete core, then $\mathrm{Pol}(\mathbb{A})^{\mathrm{can}}$ is either equationally affine, or it contains pseudo-WNU operations modulo $\overline{\mathrm{Aut}(\mathbb{A})}$ of all arities $n \geq 3$ (see [24, 45] for the lift of the corresponding result from the finite).

If $\mathcal{C}$, $\mathcal{D}$ are function clones and $\mathcal{D}$ has a finite domain, then a clone (or minion) homomorphism $\xi\colon \mathcal{C} \to \mathcal{D}$ is *uniformly continuous* if for all $n \geq 1$ there exists a finite subset F of C^n such that $\xi(f) = \xi(g)$ for all n-ary $f,g \in \mathcal{C}$ which agree on F.

A first-order formula is called a *primitive-positive* (pp-)formula if it is built exclusively from atomic formulae, existential quantifiers, and conjunction. A relation is pp-definable in a structure $\mathbb{B}$ if it is first-order definable by a pp-formula; in that case, it is invariant under $\mathrm{Pol}(\mathbb{B})$. Any ω-categorical model-complete core pp-defines all orbits of n-tuples with respect to its own automorphism group, for all $n \geq 1$.

2.3 CSP, Relational Width, Minimality

A *CSP instance* over a set A is a pair $\mathcal{I} = (\mathcal{V}, \mathcal{C})$, where $\mathcal{V}$ is a finite set of variables, and $\mathcal{C}$ is a set of *constraints*; each constraint $C \in \mathcal{C}$ is a subset of A^U for some non-empty $U \subseteq \mathcal{V}$ (U is the *scope* of C). We say that $\mathcal{I}$ is an *instance of* $\mathrm{CSP}(\mathbb{A})$ if for every $C \in \mathcal{C}$ with scope U, there exists an enumeration $u_1,\dots,u_k$ of the elements of U and a k-ary relation R of $\mathbb{A}$ such that for all $f\colon U \to A$ we have $f \in C \Leftrightarrow (f(u_1),\dots,f(u_k)) \in R$. Given a constraint $C \subseteq A^U$ and $K \subseteq U$, the *projection of* C *onto* K is defined by $C|_K := \{f|_K \colon f \in C\}$.

▶ **Definition 5.** *Let $1 \leq k \leq \ell$. We say that an instance $\mathcal{I}$ over $\mathcal{V}$ of* $\mathrm{CSP}(\mathbb{A})$ *is* (k,ℓ)-minimal *if both of the following hold:*

- *every non-empty subset of at most ℓ variables in $\mathcal{V}$ is the scope of some constraint in $\mathcal{I}$;*
- *for every at most k-element subset of variables $K \subseteq \mathcal{V}$ and any two constraints $C_1, C_2 \in \mathcal{I}$ whose scopes contain K, the projections of C_1 and C_2 onto K coincide.*

We say that an instance $\mathcal{I}$ of the CSP is *non-trivial* if it does not contain any empty constraint. Otherwise, $\mathcal{I}$ is *trivial*.

Let $1 \leq k \leq \ell$. Clearly not every instance $\mathcal{I}$ over variables $\mathcal{V}$ of $\mathrm{CSP}(\mathbb{A})$ is (k,ℓ)-minimal. However, every instance $\mathcal{I}$ is *equivalent* to a (k,ℓ)-minimal instance $\mathcal{I}'$ in the sense that $\mathcal{I}$ and $\mathcal{I}'$ have the same set of solutions. In particular we have that if $\mathcal{I}'$ is trivial, then $\mathcal{I}$ has no solutions. Moreover, if $\mathbb{A}$ is ω-categorical, then $\mathcal{I}'$ can be computed in time polynomial in the size of $\mathcal{I}$. Indeed, it is enough to introduce a new constraint A^L for every set $L \subseteq \mathcal{V}$ with

at most ℓ elements to satisfy the first condition. Then the algorithm removes tuples (in fact, orbits of tuples with respect to $\mathrm{Aut}(\mathbb{A})$) from the constraints in the instance as long as the second condition is not satisfied. Since $\mathbb{A}$ is ω-categorical and thus every relation in $\mathcal{I}$ is a union of a finite number of orbits of tuples with respect to $\mathrm{Aut}(\mathbb{A})$, the algorithm terminates.

▶ **Definition 6.** *Let $1 \leq k \leq \ell$. A relational structure $\mathbb{A}$ has* relational width (k, ℓ) *if every non-trivial (k, ℓ)-minimal instance of $\mathbb{A}$ has a solution. $\mathbb{A}$ has* bounded width *if it has relational width (k, ℓ) for some natural numbers $k \leq \ell$.*

▶ **Theorem 7** ([7])**.** *Let $\mathcal{I}$ be a non-trivial $(2, 3)$-minimal CSP instance over a finite set. Suppose that the constraints of $\mathcal{I}$ are preserved by WNUs of all arities $m \geq 3$. Then $\mathcal{I}$ has a solution.*

▶ **Theorem 8** ([32, 33])**.** *Let $\mathcal{I}$ be a non-trivial $(1, 1)$-minimal CSP instance over a finite set. Suppose that the constraints of $\mathcal{I}$ are preserved by totally symmetric polymorphisms of all arities. Then $\mathcal{I}$ has a solution.*

2.4 Smooth Approximations

We are going to apply the fundamental theorem of smooth approximations [45] to lift an action of a function clone to a larger clone.

▶ **Definition 9.** (Smooth approximations) *Let A be a set, $n \geq 1$, and let $\sim$ be an equivalence relation on a subset S of A^n. We say that an equivalence relation η on some set S' with $S \subseteq S'$* approximates *$\sim$ if the restriction of η to S is a (possibly non-proper) refinement of $\sim$. We call η an* approximation *of $\sim$.*

For a permutation group $\mathcal{G}$ acting on A and leaving η as well as the $\sim$-classes invariant, we say that the approximation η is smooth *if each equivalence class C of $\sim$ intersects some equivalence class C' of η such that $C \cap C'$ contains a $\mathcal{G}$-orbit.*

▶ **Theorem 10** (The fundamental theorem of smooth approximations [45])**.** *Let $\mathcal{C} \subseteq \mathcal{D}$ be function clones on a set A, and let $\mathcal{G}$ be a permutation group on A such that $\mathcal{D}$ locally interpolates $\mathcal{C}$ modulo $\mathcal{G}$. Let $\sim$ be a $\mathcal{C}$-invariant equivalence relation on $S \subseteq A$ with $\mathcal{G}$-invariant classes and finite index, and η be a $\mathcal{D}$-invariant smooth approximation of $\sim$ with respect to $\mathcal{G}$. Then there exists a uniformly continuous minion homomorphism from $\mathcal{D}$ to $\mathcal{C} \curvearrowright S/\sim$.*

3 Collapses in the Relational Width Hierarchy

▶ **Definition 11.** *Let $\mathcal{I} = (\mathcal{V}, \mathcal{C})$ be a CSP instance over A. Let $\mathcal{G}$ be a permutation group on A, let $k \geq 1$, and let $\mathcal{O}$ be the set of orbits of k-tuples under $\mathcal{G}$. Let $\mathcal{I}_{\mathcal{G},k}$ be the following instance over $\mathcal{O}$:*

- *The variable set of $\mathcal{I}_{\mathcal{G},k}$ is the set $\binom{\mathcal{V}}{k}$ of k-element subsets of $\mathcal{V}$. Thus, every variable K of $\mathcal{I}_{\mathcal{G},k}$ is meant to take a value in $\mathcal{O}$, and we consider that the values for K are K-orbits, i.e., orbits of maps $f\colon K \to A$ under the natural action of $\mathcal{G}$.*
- *For every constraint $C \subseteq A^U$ in $\mathcal{I}$, $\mathcal{I}_{\mathcal{G},k}$ contains the constraint $C_{\mathcal{G},k} \subseteq \mathcal{O}^{\binom{U}{k}}$ defined by*

$$C_{\mathcal{G},k} = \left\{ g\colon \binom{U}{k} \to \mathcal{O} \mid \exists f \in C \ \ \forall K \in \binom{U}{k} \ \ (f|_K \in g(K)) \right\}.$$

Note that the notation $f|_K \in g(K)$ makes sense precisely because $g(K)$ is a K-orbit.

Observe that if $\mathcal{I}$ is non-trivial, then so is $\mathcal{I}_{\mathcal{G},k}$.

▶ **Lemma 12.** *Let $1 \leq a \leq b$. If $\mathcal{I}$ is (ak, bk)-minimal, then $\mathcal{I}_{\mathcal{G},k}$ is (a, b)-minimal.*

Note that for every solution h of $\mathcal{I}$, the map $\chi_h \colon \binom{\mathcal{V}}{k} \to \mathcal{O}$ defined by $K \mapsto \{\alpha h|_K \mid \alpha \in \mathcal{G}\}$ defines a solution to $\mathcal{I}_{\mathcal{G},k}$. The next lemma proves that every solution to $\mathcal{I}_{\mathcal{G},k}$ is of the form χ_h for some solution h of $\mathcal{I}$, provided that $\mathcal{I}$ is (k, ℓ)-minimal and that $\mathcal{G} = \mathrm{Aut}(\mathbb{B})$ for some ℓ-bounded k-homogeneous structure $\mathbb{B}$.

▶ **Lemma 13.** *Let $1 \leq k < \ell$. Let $\mathbb{B}$ be ℓ-bounded and k-homogeneous, let $\mathbb{A}$ be a first-order reduct of $\mathbb{B}$, and let $\mathcal{I}$ be a (k, ℓ)-minimal instance of $\mathrm{CSP}(\mathbb{A})$. Then every solution to $\mathcal{I}_{\mathrm{Aut}(\mathbb{B}),k}$ lifts to a solution of $\mathcal{I}$.*

Proof. Let $h \colon \binom{\mathcal{V}}{k} \to \mathcal{O}$ be a solution to $\mathcal{I}_{\mathrm{Aut}(\mathbb{B}),k}$. Recall that for any $K \in \binom{\mathcal{V}}{k}$, we view $h(K)$ as a K-orbit, and one can therefore restrict $h(K)$ to any $L \subseteq K$ by setting $h(K)|_L := \{f|_L \mid f \in h(K)\}$. Note that since $\mathcal{I}$ is (k, k)-minimal, we have $h(K)|_{K \cap K'} = h(K')|_{K \cap K'}$ for all $K, K' \in \binom{\mathcal{V}}{k}$.

We now define an equivalence relation $\sim$ on $\mathcal{V}$. Suppose first that $k = 1$. Then every orbit of $\mathbb{B}$ must be a singleton (for any orbit with two elements a, b, the pairs (a, a) and (a, b) are not in the same orbit but their entries are, so that $\mathbb{B}$ is not 1-homogeneous). In that case, we identify $\mathcal{O}$ with the domain B itself, and set $x \sim y$ if and only if $h(\{x\}) = h(\{y\})$; that is, $\sim$ is essentially the kernel of h.

Suppose next that $k \geq 2$, and set $x \sim y$ iff there is $K \in \binom{\mathcal{V}}{k}$ containing x, y such that $h(K)|_{\{x,y\}}$ consists of constant maps. It can be seen that one could equivalently ask that this holds for *all* K containing x, y by 2-minimality, and that this is indeed an equivalence relation by $(2, 3)$-minimality of $\mathcal{I}$. Moreover, h descends to $\binom{\mathcal{V}/\sim}{k}$: if $K' = \{[v_1]_\sim, \ldots, [v_k]_\sim\}$ is a k-element set, define $\tilde{h}(K') := h(\{v_1, \ldots, v_k\})$. The definition of $\tilde{h}$ does not depend on the choice of representatives, by the very definition of $\sim$.

Define a finite structure $\mathbb{C}$ with domain $\mathcal{V}/\sim$ in the signature of $\mathbb{B}$ as follows. Let $K = \{[v_1]_\sim, \ldots, [v_k]_\sim\}$. The orbit $\tilde{h}(K)$ describes an atomic type on the elements of K; one defines $\mathbb{C}$ such that its substructure induced by K has the same atomic type. This is a well-defined construction by the previous paragraphs.

Finally, note that all substructures of $\mathbb{C}$ of size at most ℓ embed into $\mathbb{B}$. Indeed, let $\mathbb{L}$ be an ℓ-element substructure of $\mathbb{C}$, and let $L' \subseteq V$ be an ℓ-element set containing one representative for each element of $\mathbb{L}$. By (k, ℓ)-minimality of $\mathcal{I}$, there exists $C \subseteq A^{L'}$ in $\mathcal{I}$, and a corresponding $C_{\mathrm{Aut}(\mathbb{B}),k} \subseteq \mathcal{I}_{\mathrm{Aut}(\mathbb{B}),k}$. Thus, $h|_{\binom{L'}{k}} \in C_{\mathrm{Aut}(\mathbb{B}),k}$, so that there exists $g \in C$ such that for all $K \in \binom{L'}{k}$, $g|_K \in h(K)$. Thus g corresponds to an embedding of every k-element substructure of $\mathbb{L}$ into $\mathbb{B}$, and since $\mathbb{B}$ is k-homogeneous, g is an embedding of $\mathbb{L}$ into $\mathbb{B}$. Finally, since $\mathbb{B}$ is ℓ-bounded, it follows that there exists an embedding e of $\mathbb{C}$ into $\mathbb{B}$.

It remains to check that the composition of e with the canonical projection $\mathcal{V} \to \mathcal{V}/\sim$ is a solution to $\mathcal{I}$, which is trivial since the relations of $\mathbb{A}$ are definable in $\mathbb{B}$. ◀

Every operation f that is canonical with respect to a group $\mathcal{G}$ induces an operation on the set orbits of k-tuples under $\mathcal{G}$, by definition. We denote this operation by $f_{\mathcal{G},k}$.

▶ **Lemma 14.** *Let f be a polymorphism of $\mathbb{A}$ that is canonical with respect to $\mathcal{G}$. Every constraint $C_{\mathcal{G},k}$ in $\mathcal{I}_{\mathcal{G},k}$ is preserved under $f_{\mathcal{G},k}$.*

Finally, this allows us to prove Theorem 2 from the introduction.

Proof of Theorem 2. Suppose that the assumption of the first item of Theorem 2 is satisfied. Let $\mathcal{I}$ be a non-trivial $(2k, \max(3k, \ell))$-minimal instance of $\mathrm{CSP}(\mathbb{A})$, and let $\mathcal{I}_{\mathrm{Aut}(\mathbb{B}),k}$ be the associated instance of Definition 11. By Lemma 12, $\mathcal{I}_{\mathrm{Aut}(\mathbb{B}),k}$ is a $(2, 3)$-minimal instance,

and it is non-trivial by definition. The constraints of $\mathcal{I}_{\mathrm{Aut}(\mathbb{B}),k}$ are preserved by WNUs of all arities $m \geq 3$ (Lemma 14). By Theorem 7, $\mathcal{I}_{\mathrm{Aut}(\mathbb{B}),k}$ admits a solution and since $\mathcal{I}$ is $(k, \max(3k, \ell))$-minimal, this solution lifts to a solution of $\mathcal{I}$ by Lemma 13. Thus, $\mathbb{A}$ has width $(2k, \max(3k, \ell))$.

Suppose now that the assumption in the second item is satisfied. By the same reasoning but using Theorem 8 instead of Theorem 7, given a $(k, \max(k+1, \ell))$-minimal instance $\mathcal{I}$, the associated instance $\mathcal{I}_{\mathrm{Aut}(\mathbb{B}),k}$ is $(1,1)$-minimal and therefore has a solution. Since $\mathcal{I}$ is $(k, \max(k+1, \ell))$-minimal, this solution lifts to a solution of $\mathcal{I}$. ◀

4 A New Loop Lemma for Smooth Approximations

We refine the algebraic theory of smooth approximations from [45]. Building on deep algebraic results from [6] on finite idempotent algebras that are equationally non-trivial, we lift some of the theory from binary symmetric relations to cyclic relations of arbitrary arity.

4.1 The loop lemma

▶ **Definition 15.** *The* linkedness congruence *of a binary relation $R \subseteq A \times B$ is the equivalence relation λ_R on $\mathrm{proj}_{(2)}(R)$ defined by $(b, b') \in \lambda_R$ iff there are $k \geq 0$ and $a_0, \ldots, a_{k-1} \in A$ and $b = b_0, \ldots, b_k = b' \in B$ such that $(a_i, b_i) \in R$ and $(a_i, b_{i+1}) \in R$ for all $i \in \{0, \ldots, k-1\}$. We say that R is* linked *if it is non-empty and λ_R relates any two elements of $\mathrm{proj}_{(2)}(R)$.*

If A is a set and $m \geq 2$, then we call a relation $R \subseteq A^m$ cyclic *if it is invariant under cyclic permutations of the components of its tuples. The* support *of R is its projection on any argument. We apply the same terminology as above to any cyclic R, viewing R as a binary relation between $\mathrm{proj}_{(1,\ldots,m-1)}(R)$ and $\mathrm{proj}_{(m)}(R)$.*

If R is invariant under an oligomorphic group action on $A \times B$, then there is an upper bound on the length k to witness $(b, b') \in \lambda_R$, and therefore λ_R is pp-definable from R; in particular, it is invariant under any function clone acting on $A \times B$ and preserving R.

▶ **Definition 16.** *Let $\mathcal{G}$ be a permutation group on a set A. A* pseudo-loop with respect to $\mathcal{G}$ *is a tuple of elements of A all of whose components belong to the same $\mathcal{G}$-orbit [46, 9, 10]. If $\mathcal{G}$ contains only the identity function, then a pseudo-loop is called a* loop.

▶ **Theorem 17** (Consequence of the proof of Theorem 4.2 in [6])**.** *Let $\mathcal{C}$ be an idempotent function clone on a finite domain that is equationally non-trivial. Then any $\mathcal{C}$-invariant cyclic linked relation on its domain contains a loop.*

The following is a generalization of [45, Theorem 10] from binary symmetric relations to arbitrary cyclic relations.

▶ **Proposition 18.** *Let $n \geq 1$, and let $\mathcal{D}$ be an oligomorphic function clone on a set A which is a model-complete core. Let $\mathcal{C} \subseteq \mathcal{D}^{\mathrm{can}}_n$ be such that $\mathcal{C}^n/\mathcal{G}_\mathcal{D}$ is equationally non-trivial. Let $(S, \sim)$ be a minimal subfactor of the action $\mathcal{C}^n$ with $\mathcal{G}_\mathcal{D}$-invariant $\sim$-classes. Then for every $\mathcal{D}$-invariant cyclic relation R with support $\langle S \rangle_\mathcal{D}$ one of the following holds:*

1. *The linkedness congruence of R is a $\mathcal{D}$-invariant approximation of $\sim$.*
2. *R contains a pseudo-loop with respect to $\mathcal{G}_\mathcal{D}$.*

Proof. Let R be given, and denote its arity by m. Assuming that (1) does not hold, we prove (2).

Denote by $\mathcal{O}$ the set of orbits of n-tuples under the action of $\mathcal{G}_\mathcal{D}$ thereon. Let R' be the relation obtained by considering R as a relation on $\mathcal{O}$, i.e.,

$$R' := \{(O_1, \dots, O_m) \in \mathcal{O}^m \mid R \cap (O_1 \times \cdots \times O_m) \neq \emptyset\}.$$

Thus, R' is an m-ary cyclic relation with support $S' \subseteq \mathcal{O}$, and R' contains a loop if and only if R satisfies (2).

By assumption, the action $\mathcal{C}^n/\mathcal{G}_\mathcal{D}$ is equationally non-trivial; moreover, it is idempotent since $\mathcal{D}$ is a model-complete core. Note also that R', and in particular S', are preserved by this action. It is therefore sufficient to show that R' is linked and apply Theorem 17.

Recall that we consider R also as a binary relation between $\mathrm{proj}_{m-1}(R)$ and $\langle S\rangle_\mathcal{D}$; similary, we consider R' as a binary relation between $\mathrm{proj}_{m-1}(R')$ and S'. By the oligomorphicity of $\mathcal{D}$, the linkedness congruence λ_R of R is invariant under $\mathcal{D}$.

By our assumption that (1) does not hold, there exist $c, d \in S$ which are not $\sim$-equivalent and such that $\lambda_R(c, d)$ holds; otherwise, λ_R would be an approximation of $\sim$. This implies that the orbits O_c, O_d of c, d are related via $\lambda_{R'}$. By the minimality of $(S, \sim)$, we have that $\langle S\rangle_\mathcal{D} = \langle\{c, d\}\rangle_\mathcal{D}$. Since $\mathcal{D}$ is a model-complete core, it preserves the $\mathcal{G}_\mathcal{D}$-orbits, and it follows that any tuple in $\langle S\rangle_\mathcal{D} = \langle\{c, d\}\rangle_\mathcal{D}$ is λ_R-related to a tuple in the orbit of c. Hence, $\lambda_{R'} = (S')^2$, and thus R' is linked. Theorem 17 therefore implies that R' contains a loop, and hence R contains a pseudo-loop with respect to $\mathcal{G}_\mathcal{D}$, which is what we had to show. ◀

The following is a generalization of Lemma 12 in [45] from binary relations and functions to relations and functions of higher arity.

▶ **Lemma 19.** *Let $n \geq 1$, and let $\mathcal{D}$ be an oligomorphic polymorphism clone on a set A that is a model-complete core. Let $\sim$ be an equivalence relation on a set $S \subseteq A^n$ with $\mathcal{G}_\mathcal{D}$-invariant classes. Let $m \geq 1$, and let P be an m-ary relation on $\langle S\rangle_\mathcal{D}$. Suppose that every m-ary $\mathcal{D}$-invariant cyclic relation R on $\langle S\rangle_\mathcal{D}$ which contains a tuple in P with components in at least two $\sim$-classes contains a pseudo-loop with respect to $\mathcal{G}_\mathcal{D}$.*

Then there exists an m-ary $f \in \mathcal{D}$ such that for all $a_1, \dots, a_m \in A^n$ we have that if the tuple $(f(a_1, \dots, a_m), f(a_2, \dots, a_m, a_1), \dots, f(a_m, a_1, \dots, a_{m-1}))$ is in P, then it intersects at most one $\sim$-class.

5 Applications: Collapses of the bounded width hierachies for some classes of infinite structures

We now apply the algebraic results of Section 4 and the theory of smooth approximations to obtain a characterisation of bounded width for CSPs of first-order reducts of unary structures ($k = 2, \ell = 2$) and for CSPs in MMSNP (where k and ℓ are arbitrarily large). Moreover, the results of Section 3 then imply a collapse of the bounded width hierarchy for such CSPs.

5.1 Unary Structures

▶ **Lemma 20** (Proposition 6.5 in [18]). *Let $\mathbb{A}$ be a first-order expansion of a stabilized partition $(\mathbb{N}; V_1, \dots, V_r)$. For every $f \in \mathrm{Pol}(\mathbb{A})$ there exists $g \in \mathrm{Pol}(\mathbb{A})^{\mathrm{can}}$ which is locally interpolated by f modulo $\mathrm{Aut}(\mathbb{A})$.*

▶ **Proposition 21** (Proposition 6.6 in [18]). *Let $\mathbb{A}$ be a first-order expansion of a stabilized partition $(\mathbb{N}; V_1, \dots, V_r)$, and assume it is a model-complete core. Suppose that $\mathrm{Pol}(\mathbb{A})$ contains a binary operation whose restriction to V_i is injective for all $1 \leq i \leq r$. Then the following are equivalent:*

- *$\mathrm{Pol}(\mathbb{A})^{\mathrm{can}}$ is equationally affine;*
- *$\mathrm{Pol}(\mathbb{A})^{\mathrm{can}} \curvearrowright \mathbb{N}/\mathrm{Aut}(\mathbb{A})$ is equationally affine.*

▶ **Lemma 22** (Subset of the proof of Proposition 6.6 [18]). *Let $\mathbb{A}$ be a first-order expansion of a stabilized partition $(\mathbb{N}; V_1, \dots, V_r)$, and assume it is a model-complete core. If $\mathrm{Pol}(\mathbb{A})$ has no continuous clone homomorphism to $\mathcal{P}$, then it contains operations of all arities whose restrictions to V_i are injective for all $1 \leq i \leq r$.*

▶ **Theorem 23.** *Let $\mathbb{A}$ be a first-order reduct of a unary structure, and assume that $\mathbb{A}$ is a model-complete core. Then one of the following holds:*

- $\mathrm{Pol}(\mathbb{A})^{\mathrm{can}}$ *is not equationally affine, or equivalently, it contains pseudo-WNUs modulo* $\mathrm{Aut}(\mathbb{A})$ *of all arities $n \geq 3$;*
- $\mathrm{Pol}(\mathbb{A})$ *has a uniformly continuous minion homomorphism to an affine clone.*

In the first case, $\mathbb{A}$ has relational width $(4, 6)$ by Theorem 2, and in the second case it does not have bounded width by results from [23, 41]. Theorem 23 gives a characterization of bounded width for *all* first-order reducts of unary structures, since this class is closed under taking model-complete cores by Lemma 6.7 in [18].

The two items of Theorem 23 are invariant under expansions of $\mathbb{A}$ by a finite number of constants. Thus, by Proposition 6.8 in [18], one can assume that $\mathbb{A}$ is a first-order expansion of $(\mathbb{N}; V_1, \dots, V_r)$ where $V_1, \dots, V_r$ form a partition of $\mathbb{N}$ in which every set is either a singleton or infinite. Such partitions were called *stabilized partitions* in [18], and we shall also call the structure $(\mathbb{N}; V_1, \dots, V_r)$ a stabilized partition.

Proof of Theorem 23. Let $\mathbb{A}$ as in Theorem 23 be given; by the remark preceding this proof, we may without loss of generality assume that $\mathbb{A}$ is a first-order expansion of a stabilized partition $(\mathbb{N}; V_1, \dots, V_r)$. Assume henceforth that $\mathrm{Pol}(\mathbb{A})^{\mathrm{can}}$ is equationally affine; we show that $\mathrm{Pol}(\mathbb{A})$ has a uniformly continuous minion homomorphism to an affine clone.

If $\mathrm{Pol}(\mathbb{A})$ has a continuous clone homomorphism to $\mathcal{P}$, then we are done. Assume therefore the contrary; then by Lemma 22, $\mathrm{Pol}(\mathbb{A})$ contains for all $k \geq 2$ a k-ary operation whose restriction to V_i is injective for all $1 \leq i \leq r$. In particular, Proposition 21 applies, and thus $\mathrm{Pol}(\mathbb{A})^{\mathrm{can}} \curvearrowright \mathbb{N}/\mathrm{Aut}(\mathbb{A})$ is equationally affine. Let $(S, \sim)$ be a minimal subfactor of $\mathrm{Pol}(\mathbb{A})^{\mathrm{can}}$ such that $\mathrm{Pol}(\mathbb{A})^{\mathrm{can}}$ acts on the $\sim$-classes as an affine clone; the fact that this exists is well-known (see, e.g., Proposition 3.1 in [47]).

Let R be any $\mathrm{Pol}(\mathbb{A})$-invariant cyclic relation with support $\langle S \rangle_{\mathrm{Pol}(\mathbb{A})}$, containing a tuple with components in pairwise distinct $\mathrm{Aut}(\mathbb{A})$-orbits and which intersects at least two $\sim$-classes. By Proposition 18, R either gives rise to a $\mathrm{Pol}(\mathbb{A})$-invariant approximation of $\sim$, or it contains a pseudo-loop with respect to $\mathrm{Aut}(\mathbb{A})$. In the first case, the presence of the tuple required above implies smoothness of the approximation: if $t \in R$ is such a tuple, $c \in S$ appears in t, and $d \in S$ belongs to the same $\mathrm{Aut}(\mathbb{A})$-orbit as c, then there exists an element of $\mathrm{Aut}(\mathbb{A})$ which sends c to d and fixes all other elements of t. Hence, c and d are linked in R, and the entire $\mathrm{Aut}(\mathbb{A})$-orbit of c is contained in a class of the linkedness relation of R. Thus, $\mathrm{Pol}(\mathbb{A})$ admits a uniformly continuous minion homomorphism to an affine clone by Theorem 10.

Hence we may assume that for any R as above the second case holds. We are now going to show that this leads to a contradiction, finishing the proof of Theorem 23. By Lemma 19 applied with any $m \geq 2$ and P the set of m-tuples with entries in pairwise distinct $\mathrm{Aut}(\mathbb{A})$-orbits within $\langle S \rangle_{\mathrm{Pol}(\mathbb{A})}$, we obtain an m-ary function $f \in \mathrm{Pol}(\mathbb{A})$ with the property that the tuple $(f(a_0, \dots, a_{m-1}), \dots, f(a_1, \dots, a_{m-1}, a_0))$ intersects at most one $\sim$-class whenever it has entries in pairwise distinct $\mathrm{Aut}(\mathbb{A})$-orbits, for all $a_0, \dots, a_{m-1} \in S$. Let $(\mathbb{A}, <)$ be the expansion of $\mathbb{A}$ by a linear order that is convex with respect to the partition $V_1, \dots, V_r$ and dense and without endpoints on every infinite set of the partition. The structure $(\mathbb{A}, <)$ can be seen to be a *Ramsey structure*, since $\mathrm{Aut}(\mathbb{A}, <)$ is isomorphic as a permutation group to

the action of the product $\prod_{i=1}^{r} \mathrm{Aut}(V_i;<)$, and each of the groups of the product is either trivial or the automorphism group of a Ramsey structure [37]. By diagonal interpolation we may assume that f is diagonally canonical with respect to $\mathrm{Aut}(\mathbb{A},<)$. Let $a, a' \in A^m$ be so that a_i, a'_i belong to the same orbit with respect to $\mathrm{Aut}(\mathbb{A})$ for all $1 \leq i \leq m$. Then there exists $\alpha \in \mathrm{Aut}(\mathbb{A},<)$ such that $\alpha(a) = \alpha(a')$, and hence $f(a)$ and $f(a')$ lie in the same $\mathrm{Aut}(\mathbb{A})$-orbit by diagonal canonicity; hence f is 1-canonical with respect to $\mathrm{Aut}(\mathbb{A})$. Applying Lemma 20, we obtain a canonical function $g \in \mathrm{Pol}(\mathbb{A})^{\mathrm{can}}$ which acts like f on $\mathbb{N}/\mathrm{Aut}(\mathbb{A})$. The property of f stated above then implies for g that $g(a_0,\dots,a_{m-1}) \sim g(a_1,\dots,a_{m-1},a_0)$ for all $a_0,\dots,a_{m-1} \in S$ such that the values $g(a_0,\dots,a_{m-1}),\dots,g(a_{m-1},a_0,\dots,a_{m-2})$ lie in pairwise distinct $\mathrm{Aut}(\mathbb{A})$-orbits.

By the choice of $(S,\sim)$ we have that $\mathrm{Pol}(\mathbb{A})^{\mathrm{can}}$ acts on $S/\!\sim$ by affine functions over a finite module. We use the symbols $+,\cdot$ for the addition and multiplication in the corresponding ring, and also $+$ for the addition in the module and $\cdot$ for multiplication of elements of the module with elements of the ring. We denote by 1 the multiplicative identity of the ring, by -1 its additive inverse, and identify their powers in the additive group with the non-zero integers. The domain of the module is $S/\!\sim$, and we denote the identity element of its additive group by $[a_0]_\sim$. Pick an arbitrary element $[a_1]_\sim \neq [a_0]_\sim$ from $S/\!\sim$, and let $m \geq 2$ be its order in the additive group of the module, i.e., the minimal positive number such that $m \cdot [a_1]_\sim = [a_0]_\sim$. Let $g \in \mathrm{Pol}(\mathbb{A})^{\mathrm{can}}$ be the m-ary operation obtained in the preceding paragraph. If the values $g(a_0,\dots,a_{m-1}),\dots,g(a_{m-1},a_0,\dots,a_{m-2})$ lie in pairwise distinct $\mathrm{Aut}(\mathbb{A})$-orbits, then computing indices modulo m we have that $g([a_0]_\sim,\dots,[a_{m-1}]_\sim),\dots,g([a_{m-1}]_\sim,\dots,[a_{m+m-1}]_\sim)$ are all equal. If on the other hand they do not, then $g([a_k]_\sim,\dots,[a_{k+m-1}]_\sim) = g([a_{k+j}]_\sim,\dots,[a_{k+j+m-1}]_\sim)$ for some $0 \leq k < m$ and $1 \leq j < m$. Hence, in either case we may assume the latter equation holds. By assumption, g acts on $S/\!\sim$ as an affine map, i.e., as a map of the form $(x_0,\dots,x_{m-1}) \mapsto \sum_{i=0}^{m-1} c_i \cdot x_i$, where $c_0,\dots,c_{m-1}$ are elements of the ring which sum up to 1. We compute (with indices to be read modulo m)

$$\begin{aligned}
[a_0]_\sim &= g([a_{k+j}]_\sim,\dots,[a_{k+j+m-1}]_\sim) \;+\; (-1)\cdot g([a_k]_\sim,\dots,[a_{k+m-1}]_\sim)\\
&= \sum_{i=0}^{m-1} c_i \cdot [a_{k+j+i}]_\sim \;+\; (-1)\cdot \sum_{i=0}^{m-1} c_i \cdot [a_{k+i}]_\sim\\
&= \sum_{i=0}^{m-1} c_i \cdot (k+i+j) \cdot [a_1]_\sim \;+\; (-1)\cdot \sum_{i=0}^{m-1} c_i \cdot (k+i) \cdot [a_1]_\sim\\
&= \left(\sum_{i=0}^{m-1} c_i\right) \cdot j \cdot [a_1]_\sim = j \cdot [a_1]_\sim.
\end{aligned}$$

But $j \cdot [a_1]_\sim \neq [a_0]_\sim$ since the order of $[a_1]_\sim$ equals $m > j$, a contradiction. ◀

5.2 MMSNP

MMSNP is a fragment of existential second order logic that was discovered by Feder and Vardi in their seminal paper [33]. We prefer not to define the syntax of MMSNP, and rather define it using a correspondence between MMSNP sentences and certain coloring problems. We refer to [15] for a precise definition of all the terms employed here.

Let τ be a relational signature, let σ be a unary signature whose relations are called the *colors*, and let $\mathcal{F}$ be a finite set of finite connected $(\tau \cup \sigma)$-structures whose vertices have exactly one color. We call $\mathcal{F}$ a *colored obstruction set* in the following. The problem $\mathrm{FPP}(\mathcal{F})$ takes as input a τ-structure $\mathbb{G}$ and asks whether there exists a $(\tau\cup\sigma)$-expansion $\mathbb{G}^*$ of $\mathbb{G}$ whose

vertices are all colored with exactly one color and such that for every $\mathbb{F} \in \mathcal{F}$, there exists no homomorphism from $\mathbb{F}$ to $\mathbb{G}^*$. The connection between MMSNP and FPP is shown in [42, Corollary 3.7]: every MMSNP sentence Φ is equivalent to a union $\mathrm{FPP}(\mathcal{F}_1) \cup \cdots \cup \mathrm{FPP}(\mathcal{F}_p)$, in the sense that a τ-structure $\mathbb{G}$ satisfies Φ iff it is a yes-instance for one of the problems $\mathrm{FPP}(\mathcal{F}_i)$. We say Φ is *connected* if it is equivalent to a single $\mathrm{FPP}(\mathcal{F})$.

Every set $\mathcal{F}$ as above has a *strong normal form* $\mathcal{G}$ such that $\mathrm{FPP}(\mathcal{F}) = \mathrm{FPP}(\mathcal{G})$. We say $\mathcal{F}$ is *precolored* if for every symbol $M \in \sigma$, there is an associated unary symbol $P_M \in \tau$, and moreover if $\mathcal{F}$ contains for every $M \neq M'$ a 1-element structure whose vertex belongs to P_M and M'. Every $\mathcal{F}$ has a *standard precoloration*, obtained by enlarging τ with the necessary symbols and enlarging $\mathcal{F}$ with the associated obstructions.

It was shown in [15, Definition 4.3] that for every set $\mathcal{F}$ in strong normal form, there exists an ω-categorical τ-structure $\mathbb{A}_\mathcal{F}$ such that for any finite τ-structure $\mathbb{B}$, $\mathbb{B}$ is a yes-instance of $\mathrm{FPP}(\mathcal{F})$ iff there exists an injective homomorphism from $\mathbb{B}$ to $\mathbb{A}_\mathcal{F}$, and such that:

- If $\mathcal{F}$ is precolored, then the orbits of the elements of $\mathbb{A}_\mathcal{F}$ under $\mathrm{Aut}(\mathbb{A}_\mathcal{F})$ correspond to the colors of $\mathcal{F}$ and to the corresponding predicates in τ. In particular, the action of $\mathrm{Pol}(\mathbb{A}_\mathcal{F})^{\mathrm{can}}_1$ on $\mathrm{Aut}(\mathbb{A}_\mathcal{F})$-orbits of elements is idempotent [15, Proposition 7.1].
- Every $f \in \mathrm{Pol}(\mathbb{A}_\mathcal{F})$ locally interpolates an operation $g \in \mathrm{Pol}(\mathbb{A}_\mathcal{F})^{\mathrm{can}}_1$, and there exists a linear order $<$ on $\mathbb{A}_\mathcal{F}$ such that every f diagonally interpolates an operation f' that is diagonally canonical with respect to $\mathrm{Aut}(\mathbb{A}_\mathcal{F}, <)$.

We finally solve the Datalog-rewritability problem for MMSNP and prove that a connected sentence Φ is equivalent to a Datalog-program iff the action of $\mathrm{Pol}(\mathbb{A}_\mathcal{F})^{\mathrm{can}}_1$ on $\mathrm{Aut}(\mathbb{A}_\mathcal{F})$-orbits of elements is not equationally affine, where $\mathcal{F}$ is any strong normal form for Φ.

The following proposition is proved in [15] in the case where $m = 2$. Only small alterations to the proof are needed to prove the more general version, so we omit it.

▶ **Proposition 24.** *Let $\mathcal{F}$ be a precolored obstruction set and in normal form. Let $\mathbb{B}$ have a homomorphism to $\mathbb{A}_\mathcal{F}$ and let $m \geq 1$. There exists an embedding e of $\{1, \ldots, m\} \times \mathbb{B}$, the disjoint union of m copies of $\mathbb{B}$, into $\mathbb{A}_\mathcal{F}$ such that $(e(i_1, a_1), \ldots, e(i_m, a_m))$ and $(e(j_1, b_1), \ldots, e(j_m, b_m))$ are in the same orbit under $\mathrm{Aut}(\mathbb{A}_\mathcal{F}, <)$ provided that:*

- *a_k and b_k are in the same color for all $k \in \{1, \ldots, m\}$*
- *a_k and a_ℓ are in distinct colors for all $k \neq \ell$,*
- *$\{i_1, \ldots, i_m\} = \{j_1, \ldots, j_m\} = \{1, \ldots, m\}$.*

The following proposition shows that for the question of Datalog-rewritability, one can reduce to the precolored case without loss of generality. The same proposition was shown in [15] for the P/NP-complete dichotomy, with $\mathcal{P}$ replacing affine clones in the statement.

▶ **Proposition 25.** *Let $\mathcal{F}$ be a colored obstruction set in strong normal form and let $\mathcal{G}$ be its standard precoloration. There is a uniformly continuous minion homomorphism from $\mathrm{Pol}(\mathbb{A}_\mathcal{G})$ to an affine clone if and only if there is a uniformly continuous minion homomorphism from $\mathrm{Pol}(\mathbb{A}_\mathcal{F})$ to an affine clone.*

Proof. It is shown in [15] that $\mathrm{Pol}(\mathbb{A}_\mathcal{G})$ has a uniformly continuous minion homomorphism to $\mathrm{Pol}(\mathbb{A}_\mathcal{F})$ and that $\mathrm{Pol}(\mathbb{A}_\mathcal{F}, \neq)$ has a uniformly continuous minion homomorphism to $\mathrm{Pol}(\mathbb{A}_\mathcal{G})$. Thus, it suffices to show that if $\mathrm{Pol}(\mathbb{A}_\mathcal{F}, \neq)$ has a uniformly continuous minion homomorphism to an affine clone, then so does $\mathrm{Pol}(\mathbb{A}_\mathcal{F})$.

Let $p \geq 2$ be prime and let R_0 and R_1 be the relations defined by $\{(x, y, z) \in \mathbb{Z}_p \mid x + y + z = i \bmod p\}$ for $i \in \{0, 1\}$. For an arbitrary ω-categorical structure $\mathbb{B}$, it is known that the existence of a uniformly continuous minion homomorphism $\mathrm{Pol}(\mathbb{B})$ to an affine clone is equivalent to the existence of a p such that the relational structure $(\mathbb{Z}_p; R_0, R_1)$ has a *pp-construction* in $\mathbb{B}$.

Suppose that $(\mathbb{Z}_p; R_0, R_1)$ has a pp-construction in $(\mathbb{A}_\mathcal{F}, \neq)$. Thus, there is $n \geq 1$ and pp-formulas $\phi_0(\mathbf{x}, \mathbf{y}, \mathbf{z}), \phi_1(\mathbf{x}, \mathbf{y}, \mathbf{z})$ defining relations S_0, S_1 such that $(A^n; S_0, S_1)$ and $(\mathbb{Z}_p; R_0, R_1)$ are homomorphically equivalent; we take n to be minimal with the property that such pp-formulas exist. Since R_0 and R_1 are totally symmetric relations (i.e., the order of the entries in a tuple does not affect its membership into any of R_0 or R_1), we can assume that S_0 and S_1 are, too, and that the formulas pp-defining them are syntactically invariant under permutation of the block of variables $\mathbf{x}$, $\mathbf{y}$, and $\mathbf{z}$.

We first claim that ϕ_i does not contain any equality atom or any inequality atom $x_j \neq y_j$ for $j \in \{1, \dots, n\}$ (so that by symmetry, also $y_j \neq z_j$ and $x_j \neq z_j$ do not appear). Let $h\colon (\mathbb{Z}_p; R_0, R_1) \to (A^n; S_0, S_1)$ be a homomorphism. Since $(0,0,0) \in R_0$, we have that $(h(0), h(0), h(0))$ satisfies ϕ_0, and therefore the listed inequality atoms cannot appear. The same holds for ϕ_1, by considering $(h(0), h(0), h(1))$ and its permutations.

In order to rule out equalities, we proceed as in [15]. Suppose that ϕ_0 contains $x_i = x_j$ for $i \neq j$. Then the entries i and j of $h(q)$ are equal, for any $q \in \{0, \dots, p-1\}$, since every q belongs to the support of R_0. Thus, one can also add $x_i = x_j$ to ϕ_1, since the structure defined by the modified formula still admits a homomorphism from $(\mathbb{Z}_p; R_0, R_1)$. By existentially quantifying x_j, y_j, z_j in ϕ_0 and ϕ_1, one obtains a pp-construction of some $(A^{n-1}; S_0', S_1')$ that is still homomorphically equivalent to $(\mathbb{Z}_p; R_0, R_1)$, a contradiction to the minimality of n. If ϕ_0 contains $x_i = y_j$ for $j \neq i$, then it also contains $y_j = z_i$ and $z_i = x_j$ since we enforced that ϕ_0 is syntactically symmetric. By transitivity, we obtain that $x_i = x_j$ is implied by ϕ_0 and we are back in the first case. Suppose now that ϕ_0 contains $x_i = y_i$. Then the ith entry of $h(0)$ and $h(q)$ are equal, for all $q \in \{0, \dots, p-1\}$, since for all q there exists r such that $(0, q, r) \in R_0$. Thus we can again reduce n by fixing the ith coordinate.

Let ψ_i be the formula obtained from ϕ_i by removing the possible inequality literals, and let T_i be defined by ψ_i in $\mathbb{A}_\mathcal{F}$. We claim that $(A^n; T_0, T_1)$ and $(A^n; S_0, S_1)$ are homomorphically equivalent, which concludes the proof. Since ϕ_i implies ψ_i, we have that $(A^n; S_0, S_1)$ is a (non-induced) substructure of $(A^n; T_0, T_1)$, and therefore it homomorphically maps to $(A^n; T_0, T_1)$ by the identity map. For the other direction, we prove the result by compactness and show that every finite substructure $\mathbb{B}$ of $(A^n; T_0, T_1)$ has a homomorphism to $(A^n; S_0, S_1)$. Let $\mathbf{b}^1, \dots, \mathbf{b}^m$ be the elements of $\mathbb{B}$. Let $\mathbb{C}$ be the τ-structure over precisely $n \cdot m$ elements $\{c^i_j \mid i, j\}$ corresponding to the entries of $\mathbf{b}^i_j$, whose relations are pulled back from $\mathbb{A}_\mathcal{F}$ under the map $\pi\colon c^i_j \mapsto b^i_j$. Note that no structure from $\mathcal{F}$ has a homomorphism to $\mathbb{C}$ (otherwise, we would obtain a homomorphism to $\mathbb{A}_\mathcal{F}$ by composition with π), and thus $\mathbb{C}$ admits an injective homomorphism g to $\mathbb{A}_\mathcal{F}$. We claim that if $(\mathbf{b}^i, \mathbf{b}^j, \mathbf{b}^k) \in T_0$ then $(g(\mathbf{c}^i), g(\mathbf{c}^j), g(\mathbf{c}^k)) \in S_0$. Indeed, suppose that $(\mathbf{b}^i, \mathbf{b}^j, \mathbf{b}^k)$ satisfies ψ_0. Then by construction $(g(\mathbf{c}^i), g(\mathbf{c}^j), g(\mathbf{c}^k))$ satisfies ψ_0. Moreover, by injectivity of g, we have $g(c^i_r) \neq g(c^j_s)$ as long as $i \neq j$ or $r \neq s$. Consider any inequality atom in ϕ_0. By our first claim, it is not of the form $x_r \neq y_r$, and therefore it is satisfied by $(g(\mathbf{c}^i), g(\mathbf{c}^j), g(\mathbf{c}^k))$. Thus, $(g(\mathbf{c}^i), g(\mathbf{c}^j), g(\mathbf{c}^k))$ satisfies ϕ_0. The same reasoning for ϕ_1 shows that g induces a homomorphism $\mathbb{B} \to (A^n; S_0, S_1)$ by mapping $\mathbf{b}^i$ to $g(\mathbf{c}^i)$. ◀

▶ **Lemma 26.** *Let $(S, \sim)$ be a subfactor of $\mathrm{Pol}(\mathbb{A}_\mathcal{F})^{(\mathrm{can})}_1$ with $\mathrm{Aut}(\mathbb{A}_\mathcal{F})$-invariant $\sim$-classes. Let $m \geq 2$, and let $f \in \mathrm{Pol}(\mathbb{A}_\mathcal{F})$ be as in Lemma 19: for all $a_1, \dots, a_m \in A_\mathcal{F}$ we have that if the entries of the tuple $(f(a_1, \dots, a_m), f(a_2, \dots, a_m, a_1), \dots, f(a_m, a_1, \dots, a_{m-1}))$ all belong to different colors, then it intersects at most one $\sim$-class. Let $O_0, \dots, O_{m-1} \in S$ be pairwise distinct orbits under $\mathrm{Aut}(\mathbb{A}_\mathcal{F})$. There exists $g \in \mathrm{Pol}(\mathbb{A}_\mathcal{F})^{(\mathrm{can})}_1$ that is locally interpolated by f and that satisfies*

$$g(O_k, \dots, O_{k+m-1}) \sim g(O_{j+k}, \dots, O_{j+k+m-1}) \tag{$\star$}$$

for some $0 \leq k < m$ and $1 \leq j < m$.

Proof. Recall that the expansion of $\mathbb{A}_\mathcal{F}$ by a generic linear order is a Ramsey structure [15]. Thus, f diagonally interpolates a function $g \in \mathrm{Pol}(\mathbb{A}_\mathcal{F})$ with the same properties and which is diagonally canonical with respect to $\mathrm{Aut}(\mathbb{A}_\mathcal{F}, <)$, and without loss of generality we can therefore assume that f is itself diagonally canonical.

Let $\mathbb{B} := \{0, \dots, m-1\} \times \mathbb{A}_\mathcal{F}$ be the disjoint union of m copies of $\mathbb{A}_\mathcal{F}$ and let e be an embedding of $\{0, \dots, m-1\} \times \mathbb{B}$ into $\mathbb{A}_\Phi$ with the properties stated in Proposition 24. Let $e_i(x) := e(i, x)$, which is a self-embedding of $\mathbb{A}_\mathcal{F}$. Consider $f'(x_0, \dots, x_{m-1}) := f(e_0 x_0, \dots, e_{m-1} x_{m-1})$, and note that f' is 1-canonical when restricted to m-tuples where all entries are in pairwise distinct orbits. Let g be obtained by canonising f' with respect to $\mathrm{Aut}(\mathbb{A}_\mathcal{F}, <)$. In particular $g \in \mathrm{Pol}(\mathbb{A}_\mathcal{F})^{\mathrm{can}}_1$ and $g(O_k, \dots, O_{k+m-1})$ and $f'(O_k, \dots, O_{k+m-1})$ are in S and $\sim$-equivalent for all k.

As in the proof of Theorem 23, there are suitable $0 \leq k < m$ and $1 \leq j < m$ such that

$$f(e_k O_k, \dots, e_{k+m-1} O_{k+m-1}) \sim f(e_{k+j} O_{k+j}, \dots, e_{k+j+m-1} O_{j+k+m-1})$$

holds, where indices are computed modulo m. Then

$$\begin{aligned} g(O_k, \dots, O_{k+m-1}) &\sim f(e_0 O_k, \dots, e_{m-1} O_{k+m-1}) \\ &\sim f(e_k O_k, \dots, e_{k+m-1} O_{k+m-1}) && (\star) \\ &\sim f(e_{k+j} O_{k+j}, \dots, e_{k+j+m-1} O_{k+j+m-1}) \\ &\sim f(e_0 O_{k+j}, \dots, e_{m-1} O_{k+j+m-1}) && (\star) \\ &\sim g(O_{k+j}, \dots, O_{k+j+m-1}), \end{aligned}$$

where the equivalences marked $(\star)$ hold by the fact that f is diagonally canonical with respect to $\mathrm{Aut}(\mathbb{A}_\mathcal{F}, <)$ and by Proposition 24. ◀

The following theorem gives a characterization of Datalog-rewritability in terms of precolored normal forms. The proof is similar to that of Theorem 23.

▶ **Theorem 27.** *Let Φ be a connected MMSNP τ-sentence, let $\mathcal{F}$ be an equivalent colored obstruction set and suppose that $\mathcal{F}$ is precolored and in strong normal form. The following are equivalent:*

1. *$\neg\Phi$ is equivalent to a Datalog program;*
2. *$\mathrm{Pol}(\mathbb{A}_\mathcal{F})$ does not have a uniformly continuous minion homomorphism to an affine clone;*
3. *The action of $\mathrm{Pol}(\mathbb{A}_\mathcal{F})^{\mathrm{can}}_1$ on $\mathrm{Aut}(\mathbb{A}_\mathcal{F})$-orbits of elements is not equationally affine;*
4. *$\mathbb{A}_\mathcal{F}$ has relational width $(k, \max(k+1, \ell))$, where k and ℓ are such that $\mathbb{A}_\mathcal{F}$ is k-homogeneous ℓ-bounded.*

Proof. (1) implies (2) by general principles [41, 8].

(2) implies (3). We do the proof by contraposition. The proof is essentially the same as in the case of reducts of unary structures (Theorem 23). Suppose that $\mathrm{Pol}(\mathbb{A}_\mathcal{F})^{\mathrm{can}}_1 \curvearrowright \mathbb{A}_\mathcal{F}/\mathrm{Aut}(\mathbb{A}_\mathcal{F})$ is equationally affine and let $(S, \sim)$ be a minimal module for this action.

Let $m \geq 2$ and let R be an m-ary cyclic relation invariant under $\mathrm{Pol}(\mathbb{A}_\mathcal{F})$ and containing a tuple $(a_1, \dots, a_m)$ whose entries are pairwise distinct. By Proposition 18, either the linkedness congruence of R defines an approximation of $\sim$, or R contains a pseudoloop modulo $\mathrm{Aut}(\mathbb{A}_\mathcal{F})$. In the first case, the approximation is smooth and we obtain a uniformly continuous minion homomorphism from $\mathrm{Pol}(\mathbb{A}_\mathcal{F})$ to a clone of affine maps. Any such clone admits a uniformly continuous minion homomorphism to $\mathbb{Z}_p$ for some p, and by composition this gives us a uniformly continuous minion homomorphism $\mathrm{Pol}(\mathbb{A}_\mathcal{F}) \to \mathbb{Z}_p$.

So let us assume that for all $m \geq 2$, every such relation R contains a pseudoloop. By applying Lemma 19, we obtain a polymorphism f such that for all $a_1, \dots, a_m$, if $f(a_1, \dots, a_m), \dots, f(a_m, a_0, \dots, a_{m-1})$ are pairwise distinct, then they intersect at most one $\sim$-class. As in the proof of Theorem 23, pick an arbitrary $a_1 \in S$ such that $[a_1]_\sim$ is not the zero element of the module $S/\sim$. Let $m \geq 2$ be its order, and let O_i be the orbit of $i \cdot [a_1]_\sim$, for $i \in \{0, 1, \dots, m-1\}$. By Lemma 26, we obtain $g \in \mathrm{Pol}(\mathbb{A}_\mathcal{F})^{\mathrm{can}}_1$ such that

$$g(O_k, \dots, O_{k+m-1}) \sim g(O_{j+k}, \dots, O_{j+k+m-1})$$

for some $k \in \{0, \dots, m-1\}$ and $j \in \{1, \dots, m-1\}$. The same computation as in Theorem 23 then gives a contradiction and concludes the proof.

(3) implies (4). First, note that $\mathbb{A}_\mathcal{F}$ is infinite, and therefore $k \geq 2$. Let $\mathcal{I}$ be a non-trivial $(k, \max(k+1, \ell))$-minimal instance of $\mathbb{A}_\mathcal{F}$. Let $\mathcal{G}$ be $\mathrm{Aut}(\mathbb{A}_\mathcal{F})$. Consider the instance $\mathcal{I}_{\mathcal{G},1}$ as in Definition 11. Thus, the variables of $\mathcal{I}_{\mathcal{G},1}$ are the same as the variables of $\mathcal{I}$ (up to the natural bijection between $\mathcal{V}$ and $\binom{\mathcal{V}}{1}$) and the values for the variables are taken from the set of colors of $\mathcal{F}$. By Lemma 12, $\mathcal{I}_{\mathcal{G},1}$ is $(2,3)$-minimal, and from (3) and Lemma 14 we obtain that it has a solution h. Note that we cannot use Lemma 13 to obtain a solution to $\mathcal{I}$, since we only considered $\mathcal{I}_{\mathcal{G},1}$. Let $\mathbb{B}$ be the τ-structure described by $\mathcal{I}$ (i.e., $\mathbb{B}$ is the *canonical database* of $\mathcal{I}$). Let $\mathbb{B}^*$ be the $(\tau \cup \sigma)$-expansion of $\mathbb{B}$ obtained by coloring the vertices of $\mathbb{B}$ according to h. Since $\mathcal{I}$ is (k, ℓ)-minimal, it can be seen that $\mathbb{B}^*$ does not contain any homomorphic copy of $\mathbb{F} \in \mathcal{F}$, so that $\mathbb{B}$ admits a homomorphism to $\mathbb{A}_\mathcal{F}$, i.e., $\mathcal{I}$ has a solution in $\mathbb{A}_\mathcal{F}$.

(4) implies (1). Trivial. ◀

Combining Proposition 25, Theorem 27, and known facts about MMSNP and normal forms [15], this allows us to obtain Theorem 1 from the introduction.

▶ **Theorem 1.** *The Datalog-rewritability problem for MMSNP is decidable, and is 2NExpTime-complete.*

Proof. Let Φ be an MMSNP sentence, which is equivalent to a disjunction $\Phi_1 \vee \dots \vee \Phi_p$ of connected MMSNP sentences [15, Proposition 3.2]. Moreover, if p is minimal then $\neg\Phi$ is equivalent to a Datalog program iff every $\neg\Phi_i$ is equivalent to a Datalog program (see, e.g., Proposition 3.3 in [15], for a proof of a similar fact).

By Theorem 4.3 in [15], one can compute for every Φ_i a coloured obstruction set $\mathcal{F}_i$ that is in strong normal form. Let $\mathcal{G}_i$ be the standard precoloration of $\mathcal{F}_i$. By Proposition 25, one has a uniformly continuous minion homomorphism from $\mathrm{Pol}(\mathbb{A}_{\mathcal{G}_i})$ to an affine clone iff one has one from $\mathrm{Pol}(\mathbb{A}_{\mathcal{F}_i})$ to an affine clone. Then, by Theorem 27, we get that deciding Datalog-rewritability for $\mathcal{G}_i$ is equivalent to deciding whether $\mathrm{Pol}(\mathbb{A}_{\mathcal{G}_i})^{\mathrm{can}}_1 \curvearrowright \mathbb{A}_{\mathcal{G}_i}/\mathrm{Aut}(\mathbb{A}_{\mathcal{G}_i})$ is equationally non-affine, which is known to be decidable in polynomial time since $\mathrm{Pol}(\mathbb{A}_{\mathcal{G}_i})^{\mathrm{can}}_1 \curvearrowright \mathbb{A}_{\mathcal{G}_i}/\mathrm{Aut}(\mathbb{A}_{\mathcal{G}_i})$ is idempotent.

The computation of a strong normal form is costly and can be performed in 2-ExpSpace. In order to obtain a 2NExpTime-algorithm, we rather compute a normal form $\mathcal{F}_i$ for Φ_i (by Lemma 3.1 in [15]), which can be done in doubly exponential-time. The consequence of not working with a strong normal form is that the clone $\mathrm{Pol}(\mathbb{A}_{\mathcal{F}_i})^{\mathrm{can}}_1 \curvearrowright \mathbb{A}_{\mathcal{F}_i}/\mathrm{Aut}(\mathbb{A}_{\mathcal{F}_i})$ is not a core; its core is the action considered for the strong normal form. Deciding whether such a clone admits a minion homomorphism to an affine clone is in NP [31, Corollary 6.8]. We obtain overall a 2NExpTime algorithm. The complexity lower bound is Theorem 18 in [27]. ◀

References

1 Samson Abramsky, Anuj Dawar, and Pengming Wang. The pebbling comonad in finite model theory. In *32nd Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2017, Reykjavik, Iceland, June 20-23, 2017*, pages 1–12. IEEE Computer Society, 2017. doi:10.1109/LICS.2017.8005129.

2 Albert Atserias, Andrei A. Bulatov, and Víctor Dalmau. On the power of k -consistency. In Lars Arge, Christian Cachin, Tomasz Jurdzinski, and Andrzej Tarlecki, editors, *Automata, Languages and Programming, 34th International Colloquium, ICALP 2007, Wroclaw, Poland, July 9-13, 2007, Proceedings*, volume 4596 of *Lecture Notes in Computer Science*, pages 279–290. Springer, 2007. doi:10.1007/978-3-540-73420-8_26.

3 László Babai. Graph isomorphism in quasipolynomial time [extended abstract]. In Daniel Wichs and Yishay Mansour, editors, *Proceedings of the 48th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2016, Cambridge, MA, USA, June 18-21, 2016*, pages 684–697. ACM, 2016. doi:10.1145/2897518.2897542.

4 Libor Barto. The collapse of the bounded width hierarchy. *J. Log. Comput.*, 26(3):923–943, 2016. doi:10.1093/logcom/exu070.

5 Libor Barto and Marcin Kozik. Congruence distributivity implies bounded width. *SIAM J. Comput.*, 39(4):1531–1542, 2009. doi:10.1137/080743238.

6 Libor Barto and Marcin Kozik. Absorbing subalgebras, cyclic terms, and the constraint satisfaction problem. *Log. Methods Comput. Sci.*, 8(1), 2012. doi:10.2168/LMCS-8(1:7)2012.

7 Libor Barto and Marcin Kozik. Constraint satisfaction problems solvable by local consistency methods. *J. ACM*, 61(1):3:1–3:19, 2014. doi:10.1145/2556646.

8 Libor Barto, Jakub Opršal, and Michael Pinsker. The wonderland of reflections. *Israel Journal of Mathematics*, 223(1):363–398, 2018.

9 Libor Barto and Michael Pinsker. The algebraic dichotomy conjecture for infinite domain constraint satisfaction problems. In *Proceedings of the 31th Annual IEEE Symposium on Logic in Computer Science – LICS'16*, pages 615–622, 2016. Preprint arXiv:1602.04353.

10 Libor Barto and Michael Pinsker. Topology is irrelevant. *SIAM Journal on Computing*, 49(2):365–393, 2020.

11 Meghyn Bienvenu, Balder ten Cate, Carsten Lutz, and Frank Wolter. Ontology-based data access: A study through disjunctive datalog, csp, and MMSNP. *ACM Trans. Database Syst.*, 39(4):33:1–33:44, 2014. doi:10.1145/2661643.

12 Manuel Bodirsky. Ramsey classes: Examples and constructions. In *Surveys in Combinatorics. London Mathematical Society Lecture Note Series 424*. Cambridge University Press, 2015. Invited survey article for the British Combinatorial Conference; arXiv:1502.05146.

13 Manuel Bodirsky and Víctor Dalmau. Datalog and constraint satisfaction with infinite templates. *J. Comput. Syst. Sci.*, 79(1):79–100, 2013. doi:10.1016/j.jcss.2012.05.012.

14 Manuel Bodirsky and Jan Kára. The complexity of equality constraint languages. *Theory of Computing Systems*, 3(2):136–158, 2008. A conference version appeared in the proceedings of Computer Science Russia (CSR'06).

15 Manuel Bodirsky, Florent R. Madelaine, and Antoine Mottet. A universal-algebraic proof of the complexity dichotomy for monotone monadic SNP. In Anuj Dawar and Erich Grädel, editors, *Proceedings of the 33rd Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2018, Oxford, UK, July 09-12, 2018*, pages 105–114. ACM, 2018. doi:10.1145/3209108.3209156.

16 Manuel Bodirsky, Barnaby Martin, and Antoine Mottet. Discrete temporal constraint satisfaction problems. *J. ACM*, 65(2):9:1–9:41, 2018. doi:10.1145/3154832.

17 Manuel Bodirsky, Barnaby Martin, Michael Pinsker, and András Pongrácz. Constraint satisfaction problems for reducts of homogeneous graphs. *SIAM J. Comput.*, 48(4):1224–1264, 2019. A conference version appeared in the Proceedings of the 43rd International Colloquium on Automata, Languages, and Programming, ICALP 2016, pages 119:1-119:14. doi:10.1137/16M1082974.

18 Manuel Bodirsky and Antoine Mottet. A dichotomy for first-order reducts of unary structures. *Log. Methods Comput. Sci.*, 14(2), 2018. doi:10.23638/LMCS-14(2:13)2018.

19 Manuel Bodirsky, Antoine Mottet, Miroslav Olšák, Jakub Opršal, Michael Pinsker, and Ross Willard. Topology is relevant (in a dichotomy conjecture for infinite-domain constraint satisfaction problems). In *34th Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2019, Vancouver, BC, Canada, June 24-27, 2019*, pages 1–12. IEEE, 2019. doi:10.1109/LICS.2019.8785883.

20 Manuel Bodirsky, Wied Pakusa, and Jakub Rydval. Temporal constraint satisfaction problems in fixed-point logic. In Holger Hermanns, Lijun Zhang, Naoki Kobayashi, and Dale Miller, editors, *LICS '20: 35th Annual ACM/IEEE Symposium on Logic in Computer Science, Saarbrücken, Germany, July 8-11, 2020*, pages 237–251. ACM, 2020. doi:10.1145/3373718.3394750.

21 Manuel Bodirsky and Michael Pinsker. Canonical Functions: a Proof via Topological Dynamics. To appear. Preprint arXiv:1610.09660.

22 Manuel Bodirsky and Michael Pinsker. Schaefer's theorem for graphs. *J. ACM*, 62(3):19:1–19:52, 2015. doi:10.1145/2764899.

23 Manuel Bodirsky and Michael Pinsker. Topological Birkhoff. *Transactions of the American Mathematical Society*, 367:2527–2549, 2015.

24 Manuel Bodirsky, Michael Pinsker, and András Pongrácz. Projective clone homomorphisms. *Journal of Symbolic Logic*, 2019. doi:10.1017/jsl.2019.23.

25 Manuel Bodirsky, Michael Pinsker, and Todor Tsankov. Decidability of definability. *Journal of Symbolic Logic*, 78(4):1036–1054, 2013. A conference version appeared in the Proceedings of the Twenty-Sixth Annual IEEE Symposium on Logic in Computer Science (LICS) 2011, pages 321–328.

26 Manuel Bodirsky and Michał Wrona. Equivalence constraint satisfaction problems. In *Computer Science Logic (CSL'12) - 26th International Workshop/21st Annual Conference of the EACSL, CSL 2012, September 3-6, 2012, Fontainebleau, France*, pages 122–136, 2012. doi:10.4230/LIPIcs.CSL.2012.122.

27 Pierre Bourhis and Carsten Lutz. Containment in monadic disjunctive datalog, mmsnp, and expressive description logics. In Chitta Baral, James P. Delgrande, and Frank Wolter, editors, *Principles of Knowledge Representation and Reasoning: Proceedings of the Fifteenth International Conference, KR 2016, Cape Town, South Africa, April 25-29, 2016*, pages 207–216. AAAI Press, 2016. URL: http://www.aaai.org/ocs/index.php/KR/KR16/paper/view/12847.

28 Hans-Jürgen Bückert and Bernhard Nebel. Reasoning about temporal relations: a max-imal tractable subclass of allen's interval algebra. *J. ACM*, 42(1):43–66, 1995.

29 Andrei A. Bulatov. Bounded relational width, 2009. URL: https://www2.cs.sfu.ca/~abulatov/papers/relwidth.pdf.

30 Andrei A. Bulatov. A dichotomy theorem for nonuniform csps. In Chris Umans, editor, *58th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2017, Berkeley, CA, USA, October 15-17, 2017*, pages 319–330. IEEE Computer Society, 2017. doi:10.1109/FOCS.2017.37.

31 Hubie Chen and Benoît Larose. Asking the metaquestions in constraint tractability. *ACM Trans. Comput. Theory*, 9(3):11:1–11:27, 2017. doi:10.1145/3134757.

32 Víctor Dalmau and Justin Pearson. Closure functions and width 1 problems. In *Proceedings of the International Conference on Principles and Practice of Constraint Programming (CP)*, pages 159–173, 1999.

33 Tomás Feder and Moshe Y. Vardi. The computational structure of monotone monadic SNP and constraint satisfaction: A study through datalog and group theory. *SIAM J. Comput.*, 28(1):57–104, 1998. doi:10.1137/S0097539794266766.

34 Cristina Feier, Antti Kuusisto, and Carsten Lutz. Rewritability in monadic disjunctive datalog, mmsnp, and expressive description logics. *Log. Methods Comput. Sci.*, 15(2), 2019. doi:10.23638/LMCS-15(2:15)2019.

35 Pierre Gillibert, Julius Jonušas, Michael Kompatscher, Antoine Mottet, and Michael Pinsker. Hrushovski's encoding and ω-categorical CSP monsters. In Artur Czumaj, Anuj Dawar, and Emanuela Merelli, editors, *47th International Colloquium on Automata, Languages, and Programming, ICALP 2020, July 8-11, 2020, Saarbrücken, Germany (Virtual Conference)*, volume 168 of *LIPIcs*, pages 131:1–131:17. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPIcs.ICALP.2020.131.

36 Martin Grohe. Fixed-point definability and polynomial time on graphs with excluded minors. *J. ACM*, 59(5):27:1–27:64, 2012. doi:10.1145/2371656.2371662.

37 Alexander Kechris, Vladimir Pestov, and Stevo Todorčević. Fraïssé limits, Ramsey theory, and topological dynamics of automorphism groups. *Geometric and Functional Analysis*, 15(1):106–189, 2005.

38 Phokion G. Kolaitis and Moshe Y. Vardi. Conjunctive-query containment and constraint satisfaction. *J. Comput. Syst. Sci.*, 61(2):302–332, 2000. doi:10.1006/jcss.2000.1713.

39 Michael Kompatscher and Trung Van Pham. A complexity dichotomy for poset constraint satisfaction. *FLAP*, 5(8):1663–1696, 2018. URL: https://www.collegepublications.co.uk/downloads/ifcolog00028.pdf.

40 Marcin Kozik, Andrei Krokhin, Matt Valeriote, and Ross Willard. Characterizations of several Maltsev conditions. *Algebra universalis*, 73(3-4):205–224, 2015. doi:10.1007/s00012-015-0327-2.

41 Benoit Larose and László Zádori. Bounded width problems and algebras. *Algebra Universalis*, 56(3-4):439–466, 2007.

42 Florent R. Madelaine and Iain A. Stewart. Constraint satisfaction, logic and forbidden patterns. *SIAM J. Comput.*, 37(1):132–163, 2007. doi:10.1137/050634840.

43 Miklós Maróti and Ralph McKenzie. Existence theorems for weakly symmetric operations. *Algebra Universalis*, 59(3), 2008.

44 Antoine Mottet, Tomáš Nagy, Michael Pinsker, and Michał Wrona. Smooth approximations and relational width collapses, 2021. arXiv:2102.07531.

45 Antoine Mottet and Michael Pinsker. Smooth approximations and CSPs over finitely bounded homogeneous structures. *CoRR*, abs/2011.03978, 2020. arXiv:2011.03978.

46 Michael Pinsker, Pierre Gillibert, and Julius Jonušas. Pseudo-loop conditions. *Bulletin of the London Mathematical Society*, 51(5):917–936, 2019.

47 Matthew A. Valeriote. A subalgebra intersection property for congruence distributive varieties. *Canadian Journal of Mathematics*, 61(2):451–464, 2009. doi:10.4153/CJM-2009-023-2.

48 Michał Wrona. On the relational width of first-order expansions of finitely bounded homogeneous binary cores with bounded strict width. In Holger Hermanns, Lijun Zhang, Naoki Kobayashi, and Dale Miller, editors, *LICS '20: 35th Annual ACM/IEEE Symposium on Logic in Computer Science, Saarbrücken, Germany, July 8-11, 2020*, pages 958–971. ACM, 2020. doi:10.1145/3373718.3394781.

49 Michał Wrona. Relational width of first-order expansions of homogeneous graphs with bounded strict width. In Christophe Paul and Markus Bläser, editors, *37th International Symposium on Theoretical Aspects of Computer Science, STACS 2020, March 10-13, 2020, Montpellier, France*, volume 154 of *LIPIcs*, pages 39:1–39:16. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPIcs.STACS.2020.39.

50 Dmitriy Zhuk. A proof of CSP dichotomy conjecture. In Chris Umans, editor, *58th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2017, Berkeley, CA, USA, October 15-17, 2017*, pages 331–342. IEEE Computer Society, 2017. doi:10.1109/FOCS.2017.38.

51 Dmitriy Zhuk. A proof of the CSP dichotomy conjecture. *J. ACM*, 67(5):30:1–30:78, 2020. doi:10.1145/3402029.

Comparison-Free Polyregular Functions

Lê Thành Dũng (Tito) Nguyễn
Laboratoire d'informatique de Paris Nord, Villetaneuse, France

Camille Noûs
Laboratoire Cogitamus, Université Volante, Sevran, France

Pierre Pradic
Department of Computer Science, University of Oxford, UK

Abstract

This paper introduces a new automata-theoretic class of string-to-string functions with polynomial growth. Several equivalent definitions are provided: a machine model which is a restricted variant of pebble transducers, and a few inductive definitions that close the class of regular functions under certain operations. Our motivation for studying this class comes from another characterization, which we merely mention here but prove elsewhere, based on a λ-calculus with a linear type system.

As their name suggests, these *comparison-free polyregular functions* form a subclass of polyregular functions; we prove that the inclusion is strict. We also show that they are incomparable with HDT0L transductions, closed under usual function composition – but not under a certain "map" combinator – and satisfy a comparison-free version of the pebble minimization theorem.

On the broader topic of polynomial growth transductions, we also consider the recently introduced layered streaming string transducers (SSTs), or equivalently k-marble transducers. We prove that a function can be obtained by composing such transducers together if and only if it is polyregular, and that k-layered SSTs (or k-marble transducers) are closed under "map" and equivalent to a corresponding notion of $(k+1)$-layered HDT0L systems.

2012 ACM Subject Classification Theory of computation → Transducers

Keywords and phrases pebble transducers, HDT0L systems, polyregular functions

Digital Object Identifier 10.4230/LIPIcs.ICALP.2021.139

Category Track B: Automata, Logic, Semantics, and Theory of Programming

Related Version *Full Version with appendices*: `https://hal.archives-ouvertes.fr/hal-02986228`

Funding *Lê Thành Dũng (Tito) Nguyễn*: Partially funded by the French ANR project CoGITARe (ANR-18-CE25-0001).

Acknowledgements Thanks to Mikołaj Bojańczyk and Sandra Kiefer for inspiring discussions, to Gaëtan Douéneau-Tabot and Amina Doumane for explaining some features of their work to us, to Charles Paperman for his help with bibliography and to the reviewers for their feedback.

1 Introduction

The theory of transducers (as described in the surveys [21, 29]) has traditionally dealt with devices that take as input strings of length n and output strings of length $O(n)$. However, several recent works have investigated function classes going beyond linear growth. We review three classes in this landscape below.

- *Polyregular functions* (§2.3) are thus named because they have (at most) polynomial growth and include regular functions (§2.2) (the most expressive of the traditional string-to-string transduction classes). They were defined in 2018 [4] by four equivalent computational models, one of which – the *pebble transducers* – is the specialization to strings of a tree transducer model that existed previously in the literature [28] (this specialization had been investigated earlier in [17, 14]). A subsequent work [8] gave a logical characterization based on Monadic Second-Order logic (MSO). They enjoy two nice properties:

  - *preservation of regular languages (by preimage)*: if $f : \Gamma^* \to \Sigma^*$ is polyregular and $L \subseteq \Sigma^*$ is regular, then $f^{-1}(L) \subseteq \Gamma^*$ is regular;
  - *closure under function composition*: if $f : \Gamma^* \to \Delta^*$ and $g : \Delta^* \to \Sigma^*$ are both polyregular, then so is $g \circ f : \Gamma^* \to \Sigma^*$.
- *HDT0L transductions* (§2.1) form another superclass of regular functions, whose output size may be at most exponential in the input size. They are older than polyregular functions, and we shall discuss their history in Section 2.1; suffice to say for now, they also admit various equivalent characterizations scattered in several papers [20, 22, 13]. These functions preserve regular languages by preimage, but are *not* closed under composition (the growth rate of a composition of HDT0L transductions may be a tower of exponentials).
- Very recently, the polynomially bounded HDT0L transductions (§2.3) have been characterized using two transducer models [13]. One of them, the *k-marble transducers* (where $k \in \mathbb{N}$ depends on the function to be computed), is obtained by putting a syntactic constraint on the model of *(unbounded) marble transducers* [13] which computes HDT0L transductions. But it can also be seen as a restricted variant of pebble transducers; it follows (although this is not explicitly stated in [13]) that a HDT0L transduction has polynomial growth if and only if it is polyregular. Moreover, as claimed in [13, Section 6], the functions computed by k-marble transducers are not closed under composition either, and thus form a strict subclass of polyregular functions.

A new subclass of polyregular functions. In this paper, we start by proving a few results on the above classes (Section 3). For instance, we supply a proof for the aforementioned claim of [13, Section 6], and show that the polyregular functions are exactly those computable by compositions of k-marble transducers. Those complements are not particularly difficult nor surprising and are included mostly for the sake of giving a complete picture.

But our main contribution is the introduction of a new class, giving its title to the paper; as we show, it admits three equivalent definitions:

- two ways to inductively generate the class (Sections 4 and 6 respectively):
  - by closing regular functions under a certain "composition by substitution" operation;
  - by combining regular functions and a certain kind of *squaring* functions (less powerful than the squaring plus underlining functions used to characterize general poyregular functions) with usual function composition;
- a restriction on pebble transducers (Section 5) – we disallow comparing the positions of a transducer's multiple reading heads, hence the name *comparison-free polyregular functions* (henceforth abbreviated as *cfp*).

Properties. By the third definition above, comparison-free polyregular functions are indeed polyregular, while the second one implies that our new class contains the regular functions and is closed under composition. (In fact, in the proof that our first definition is equivalent to the second one, most of the work goes into showing that the former enjoys closure under composition.) We rule out inclusions involving the other classes that we mentioned by proving some *separation results* (Section 8): there exist

- comparison-free polyregular functions that are not HDT0L (we take one example from [13]),
- and polynomially bounded HDT0L transductions which are not comparison-free:
  - one of our examples follows from a precise characterization of cfp functions over unary input alphabets (extending a known result for regular functions with unary inputs [10]), which we give in Section 9;
  - another example shows that unlike (poly)regular functions, cfp functions are *not* closed under a certain counterpart of the "map" operation in functional programming.

We summarize the inclusions and separations between classes that we get in Figure 1.

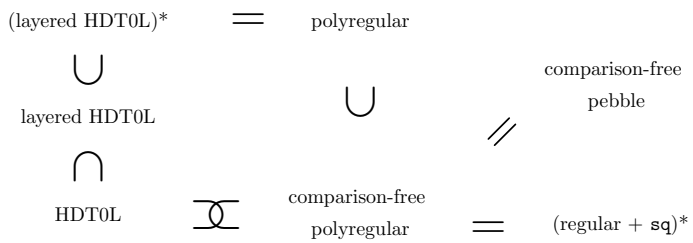


Figure 1 Summary of the known relationships between superlinear transduction classes, taking our results into account. Inclusions $\subset$ are strict, and $\not\subset\not\supset$ means that there is no inclusion either way. Finally C^* denotes the composition closure of the class C.

Finally, we show in Section 7 that the number of pebbles required to compute a function using a comparison-free transducer is related to its growth rate. The analogous result for pebble transducers was proved recently, with a whole paper dedicated to it [25]; we adapt its arguments to our setting, resulting in our longest and most technical proof. There is a similar property for k-marble transducers [13], but it is proved using very different tools.

Motivations. Although this is the first proper paper to introduce comparison-free pebble transducers, we were told that they had already been considered by several colleagues (Mikołaj Bojańczyk, personal communication). But in fact, the starting point in our investigation was a characterization of regular functions using a linear λ-calculus (in the sense of linear logic) that we had previously obtained [30]; this was part of a research programme relating automata and functional programming that we initiated in [31]. As we reported in a previous version of the present paper (version 1 of the full paper archived on HAL), by tweaking a parameter in this characterization, one gets the cfp functions instead; we initially defined the latter using composition by substitution, and only later realized the connection with pebble transducers. One interesting feature of the λ-calculus characterization is that it is trivially closed under composition, and this led us to take inspiration from the category-theoretic machinery that we used in [30] for our standalone composition proof in this paper.

2 Preliminaries

Notations. The set of natural numbers is $\mathbb{N} = \{0, 1, \dots\}$. We write $|w|$ for the length of a string $w \in \Sigma^*$; for $\Pi \subseteq \Sigma$, we write $|s|_\Pi$ for the number of occurrences of letters from Π in w; and for $c \in \Sigma$, we abbreviate $|w|_{\{c\}}$ as $|w|_c$. The i-th letter of $w \in \Sigma^*$ is denoted by either w_i or $w[i]$ (for $i \in \{1, \dots, |w|\}$). Given monoids M and N, $\mathrm{Hom}(M, N)$ is the set of monoid morphisms. We write ε for the empty word and $\underline{\Sigma} = \{\underline{a} \mid a \in \Sigma\}$ for a disjoint copy of the alphabet Σ made of "underlined" letters.

2.1 HDT0L transductions and streaming string transducers

L-systems were originally introduced by Lindenmayer [26] in the 1960s as a way to generate formal languages, with motivations from biology. While this language-centric view is still predominant, the idea of considering variants of L-systems as specifications for string-to-string functions – whose range are the corresponding languages – seems to be old. For instance, in a paper from 1980 [18], one can find (multi-valued) string functions defined by ET0L systems.

More recently, Ferté, Marin and Sénizergues [20] provided alternative characterizations[1] (by catenative recurrent equations and higher-order pushdown transducers of level 2) of the string-to-string functions that *HDT0L systems* can express – what we call here *HDT0L transductions*. Later work by Filiot and Reynier [22] and then by Douéneau-Tabot, Filiot and Gastin [13] – that does not build on [36, 20] – proved the equivalence with, respectively, copyful SSTs (Definition 2.3) and unbounded marble transducers (not presented here).

▶ **Definition 2.1** (following [22]). *A* HDT0L system *consists of:*

- *an input alphabet* Γ*, an output alphabet* Σ*, and a working alphabet* Δ *(all* finite*);*
- *an initial word* $d \in \Delta^*$*;*
- *for each* $c \in \Gamma$*, a* monoid morphism $h_c \in \mathrm{Hom}(\Delta^*, \Delta^*)$*;*
- *a final morphism* $h' \in \mathrm{Hom}(\Delta^*, \Sigma^*)$*.*

It defines the transduction taking $w = w_1 \dots w_n \in \Gamma^*$ *to* $h' \circ h_{w_1} \circ \dots \circ h_{w_n}(d) \in \Sigma^*$*.*

(The definition of HDT0L systems given in [36, 20] makes slightly different choices of presentation[2].) To define the equivalent model of copyful streaming string transducers, we must first introduce the notion of register assignment.

▶ **Definition 2.2.** *Fix a finite alphabet* Σ*. Let* R *and* S *be two finite sets* disjoint from Σ*; we shall consider their elements to be "register variables".*

For any word $\omega \in (\Sigma \cup R)^*$*, we write* $\omega^\dagger : (\Sigma^*)^R \to \Sigma^*$ *for the map that sends* $(u_r)_{r\in R}$ *to* ω *in which every occurrence of a register variable* $r \in R$ *is replaced by* u_r *– formally, we apply to* ω *the morphism* $(\Sigma \cup R)^* \to \Sigma^*$ *that maps* $c \in \Sigma$ *to itself and* $r \in R$ *to* u_r*.*

A register assignment[3] α *from* R *to* S *(over* Σ*) is a map* $\alpha : S \to (\Sigma \cup R)^*$*. It induces the action* $\alpha^\dagger : \vec{u} \in (\Sigma^*)^R \mapsto (\alpha(s)^\dagger(\vec{u}))_{s\in S} \in (\Sigma^*)^S$ *(which indeed goes "from* R *to* S*").*

▶ **Definition 2.3** ([22]). *A (deterministic copyful)* streaming string transducer *(SST) with input alphabet* Γ *and output alphabet* Σ *is a tuple* $\mathcal{T} = (Q, q_0, R, \delta, \vec{u}_I, F)$ *where*

- Q *is a finite set of* states *and* $q_0 \in Q$ *is the* initial state*;*
- R *is a finite set of* register variables*, that we require to be* disjoint from Σ*;*
- $\delta : Q \times \Gamma \to Q \times (R \to (\Sigma \cup R)^*)$ *is the* transition function *– we abbreviate* $\delta_{\mathrm{st}} = \pi_1 \circ \delta$ *and* $\delta_{\mathrm{reg}} = \pi_2 \circ \delta$*, where* π_i *is the projection from* $X_1 \times X_2$ *to its* i*-th component* X_i*;*
- $\vec{u}_I \in (\Sigma^*)^R$ *describes the* initial register values*;*
- $F : Q \to (\Sigma \cup R)^*$ *describes how to recombine the final values of the registers, depending on the final state, to produce the output.*

The function $\Gamma^* \to \Sigma^*$ *computed by* $\mathcal{T}$ *is*

$$w_1 \dots w_n \quad \mapsto \quad F(q_n)^\dagger \circ \delta_{\mathrm{reg}}(q_{n-1}, w_n)^\dagger \circ \dots \circ \delta_{\mathrm{reg}}(q_0, w_1)^\dagger(\vec{u}_I)$$

where the sequence of states $(q_i)_{0\le i\le n}$ *(sometimes called the* run *of the transducer over the input word) is inductively defined, starting from the fixed initial state* q_0*, by* $q_i = \delta_{\mathrm{st}}(q_{i-1}, w_i)$*.*

---

[1] Those characterizations had previously been announced in an invited paper by Sénizergues [36]. Some other results announced in [36] are proved in [9].

[2] The family $(h_c)_{c\in\Gamma}$ is presented as a morphism $H : \Gamma^* \to \mathrm{Hom}(\Delta^*, \Delta^*)$ (whose codomain is indeed a monoid for function composition). And an initial *letter* is used instead of an initial word; this is of no consequence regarding the functions that can be expressed (proof sketch: consider $\Delta' = \Delta \cup \{x\}$ with a new letter $x \notin \Delta$, take x as the initial letter and let $h_c(x) = h_c(w)$, $h'(x) = h'(w)$).

[3] Some papers e.g. [11, 13] call register assignments *substitutions*. We avoid this name since it differs from its meaning in the context of our "composition by substitution" operation.

▶ **Example 2.4.** Let $\Sigma = \Gamma \cup \underline{\Gamma}$. We consider a SST $\mathcal{T}$ with $Q = \{q\}$, $R = \{X, Y\}$ and

$$\vec{u}_I = (\varepsilon)_{r\in R} \qquad F(q) = Y \qquad \forall c \in \Gamma,\ \delta(q, c) = (q,\ (X \mapsto cX,\ Y \mapsto \underline{c}XY))$$

If we write (v, w) for the family $(u_r)_{r\in R}$ with $u_X = v$ and $u_Y = w$, then the action of the register assignments may be described as $(X \mapsto cX,\ Y \mapsto \underline{c}XY)^\dagger(v, w) = (c \cdot v,\ \underline{c} \cdot v \cdot w)$.

Let $\mathtt{1}, \mathtt{2}, \mathtt{3}, \mathtt{4} \in \Gamma$. After reading $\mathtt{1234} \in \Gamma^*$, the values stored in the registers of $\mathcal{T}$ are

$$(X \mapsto \mathtt{4}X,\ Y \mapsto \underline{\mathtt{4}}XY)^\dagger \circ \ldots \circ (X \mapsto \mathtt{1}X,\ Y \mapsto \underline{\mathtt{1}}XY)^\dagger(\varepsilon, \varepsilon) = (\mathtt{4321}, \underline{\mathtt{4}}\mathtt{321}\underline{\mathtt{3}}\mathtt{21}\underline{\mathtt{2}}\mathtt{11})$$

Since $F(q) = Y$, the function defined by $\mathcal{T}$ maps $\mathtt{1234}$ to $\underline{\mathtt{4}}\mathtt{321}\underline{\mathtt{3}}\mathtt{21}\underline{\mathtt{2}}\mathtt{11} \in (\Gamma \cup \underline{\Gamma})^* = \Sigma^*$.

This gives us an example of HDT0L transduction $\Gamma^* \to (\Gamma \cup \underline{\Gamma})^*$, since:

▶ **Theorem 2.5** ([22]). *A function $\Gamma^* \to \Sigma^*$ can be computed by a copyful SST if and only if it can be specified by a HDT0L system.*

▶ Remark 2.6. As observed in [22, Lemma 3.3], there is a natural translation from HDT0L systems to SSTs whose range is composed precisely of the *single-state* SSTs whose transitions and final output function *do not access the letters of their output alphabet* – those are called *simple SSTs* in [13, §5.1]. This involves a kind of reversal: the initial register values correspond to the final morphisms, while the final output function corresponds to the initial word. Thus, Theorem 2.5 is essentially a state elimination result; a direct translation from SSTs to single-state SSTs has also been given by Benedikt et al. [2, Proposition 8]. However, it does *not* preserve the subclass of *copyless* SSTs (this would contradict Proposition 3.7).

The lookahead elimination theorem for macro tree transducers [19, Theorem 4.21] arguably generalizes this to trees. Indeed, while those transducers are generally presented as a top-down model, their formal definition can also be read as bottom-up register tree transducers in the style of [7, §4], and top-down lookahead corresponds to bottom-up states.

2.2 Regular functions

▶ **Definition 2.7** (Alur and Černý [1]). *A register assignment $\alpha : S \to (\Sigma \cup R)^*$ from R to S is said to be* copyless *when each $r \in R$* occurs at most once among all the strings $\alpha(s)$ for $s \in S$, *i.e. it does not occur at least twice in some $\alpha(s)$, nor at least once in $\alpha(s)$ and at least once in $\alpha(s')$ for some $s \neq s'$. (This restriction does not apply to the letters in Σ.)*

A streaming string transducer is copyless *if all the assignments in the image of its transition function are copyless. In this paper, we take computability by copyless SSTs as the definition of* regular functions *(but see Theorem 5.3 for another standard definition).*

▶ Remark 2.8. Thanks to Theorem 2.5, every regular function is a HDT0L transduction.

▶ Remark 2.9. The SST of Example 2.4 is *not* copyless: in a transition $\alpha = \delta_{\mathrm{reg}}(q, c)$, the register X appears twice, once in $\alpha(X) = cX$ and once in $\alpha(Y) = \underline{c}XY$; in other words, its value is *duplicated* by the action $\alpha^\dagger$. In fact, it computes a function whose output size is quadratic in the input size, while regular functions have linearly bounded output.

▶ **Example 2.10** (Iterated reverse [4, p. 1]). The following single-state SST is copyless:

$$\Gamma = \Sigma \text{ with } \# \in \Sigma \qquad Q = \{q\} \qquad R = \{X, Y\} \qquad \vec{u}_I = (\varepsilon)_{r\in R} \qquad F(q) = XY$$

$$\delta(q, \#) = (q,\ (X \mapsto XY\#,\ Y \mapsto \varepsilon)) \qquad \forall c \in \Sigma \setminus \{\#\},\ \delta(q, c) = (q,\ (X \mapsto X,\ Y \mapsto cY))$$

For $u_1, \ldots, u_n \in (\Sigma \setminus \{\#\})^*$, it maps $u_1\# \ldots \#u_n$ to $\mathtt{reverse}(u_1)\# \ldots \#\mathtt{reverse}(u_n)$.

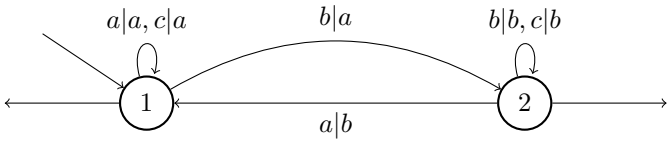


Figure 2 An example of sequential transducer.

The concrete SSTs (copyless or not) that we have seen for now are all single-state. As a source of stateful copyless SSTs, one can consider the translations of *sequential transducers*. These are usual finite automata, whose transitions additionally produce a word catenated to the end of the would-be output function. For instance, the one in Figure 2 computes the function $\{a, b, c\}^* \to \{a, b\}^*$ that replaces each c in its input by the closest non-c letter on its left (or a if no such letter exists). We do not give a detailed definition (which can be found e.g. in [33, Chapter V]) here, but for our purpose, it suffices to observe any sequential transducer can be translated into a copyless SST with the same set of states and a single register.

2.3 Polynomial growth transductions

Next, we recall one way to define Bojańczyk's *polyregular functions* [4].

▶ **Definition 2.11** ([4]). *The class of* polyregular functions *is the smallest class of string-to-string functions closed under composition containing:*

- *the functions computed by sequential transducers (for instance, the one of Figure 2);*
- *the* iterated reverse *function of Example 2.10, over any finite alphabet containing #;*
- *the* squaring with underlining *functions* $\texttt{squaring}_\Gamma : \Gamma^* \to (\Gamma \cup \underline{\Gamma})^*$*, for any finite* Γ*, illustrated by* $\texttt{squaring}_\Gamma(1234) = \underline{1}234\underline{12}341\underline{23}41\underline{234}$.

As mentioned in the introduction, the intersection between the above class and HDT0L transductions has been recently characterized by Douéneau-Tabot et al. [13].

▶ **Theorem 2.12** ([13]). *Let* $f : \Gamma^* \to \Sigma^*$*. The following conditions are equivalent:*

- *f is both a polyregular function and a HDT0L transduction;*
- *f is a HDT0L transduction and has at most polynomial growth: $f(|w|) = |w|^{O(1)}$;*
- *there exists $k \in \mathbb{N}$ such that f is computed by some k-*layered SST*, defined below.*

(Another equivalent model, the *k-marble transducers*, was mentioned in the introduction, but we will not use it in the rest of the paper.) Those k-layered SST propose a compromise between copyful and copyless SSTs: duplication is controlled, but not outright forbidden.

▶ **Definition 2.13** ([13]). *A register assignment $\alpha : R \to (\Sigma \cup R)^*$ is* k-layered *(for $k \in \mathbb{N}$) with respect to a partition $R = R_0 \sqcup \ldots \sqcup R_k$ when for $0 \leq i \leq k$,*

- *for $r \in R_i$, we have $\alpha(r) \in (\Sigma \cup R_0 \cup \ldots \cup R_i)^*$;*
- *each register variable in R_i appears at most once among all the $\alpha(r)$ for $r \in R_i$ (however, those from $R_0 \sqcup \ldots \sqcup R_{i-1}$ may appear an arbitrary number of times).*

A SST is k-layered *if its registers can be partitioned in such a way that all assignments in the transitions of the SST are k-layered.*

Beware: with this definition, the registers of a k-layered SST are actually divided into $k + 1$ layers, not k. In particular, a SST is copyless if and only if it is 0-layered. (We chose this convention for backwards compatibility with [13]; see also Remark 5.4.)

For instance, the transducer of Example 2.4 is 1-layered with $R_0 = \{X\}$ and $R_1 = \{Y\}$. There also exist register assignments that cannot be made k-layered no matter the choice of partition, such as $X \mapsto XX$. Using such assignments, one can indeed build SSTs that compute functions f such that e.g. $|f(w)| = 2^{|w|}$.

▶ Remark 2.14. There is arguably an old precursor to this recent characterization of HDT0L transductions with polynomial growth by a syntactic "layering" condition: Schützenberger's theorem on polynomially bounded $\mathbb{Z}$-rational series, which dates back to the 1960s (see for instance [3, Chapter 9, Section 2] – the preface of the same book describes this theorem as "one of the most difficult results in the area"). Let us give a brief exposition.

A *$\mathbb{Z}$-rational series* $f : \Sigma^* \to \mathbb{Z}$ is a function of the form $f : w \in \Sigma^* \mapsto X^T \cdot \Phi(w) \cdot Y$ where $X, Y \in \mathbb{Z}^R$ and Φ is a morphism from Σ^* to the multiplicative monoid of R-indexed square matrices over $\mathbb{Z}$, where R is a finite set. This data (X, Φ, Y) has a clear interpretation as a "simple SST" (cf. Remark 2.6) with register set R, whose register values are integers rather than strings. Schützenberger's theorem says that any $\mathbb{Z}$-rational series f with polynomial growth (i.e. $|f(w)| = |w|^{O(1)}$ where $|\cdot|$ on the left is the absolute value) can be written as $f : w \mapsto X^T \cdot \Phi(w) \cdot Y$ where

(i) the image of Φ has a block triangular structure;
(ii) the projection of this image on each diagonal block is a finite monoid.

The first item gives us a partition of the register into layers where each layer "depends" only on the ones below them. The finiteness condition in the second item is equivalent to having bounded coefficients, which means that the register assignments within each layer are *bounded-copy*, while in a layered SST, they would be *copyless* instead – but bounded-copy SSTs are known to be equivalent to copyless SSTs (see e.g. [11]). The theorem also states a relationship between the number of blocks and the growth rate; compare this to Remark 7.2.

Via the canonical isomorphism $\{a\}^* \cong \mathbb{N}$, HDT0L transductions with unary output alphabet are the same thing as $\mathbb{N}$-*rational series*. The counterpart of Schützenberger's theorem over $\mathbb{N}$ is thus a corollary of the results of [13] on layered SSTs.

2.4 Transition monoids for streaming string transducers

To wrap up the preliminaries, let us recall some algebraic tools for working with SSTs (this technical section can be safely skipped on a first reading). Let us start by putting a monoid structure on register assignments (Definition 2.2).

▶ **Definition 2.15.** *Let $\mathcal{M}_{R,\Sigma} = R \to (\Sigma \cup R)^*$ for $R \cap \Sigma = \varnothing$. We endow it with the following composition operation, that makes it into a monoid:*

$$\alpha \bullet \beta = \alpha^{\odot} \circ \beta \quad \text{where } \alpha^{\odot} \in \mathrm{Hom}((\Sigma \cup R)^*, (\Sigma \cup R)^*),\ \alpha^{\odot}(x) = \begin{cases} \alpha(x) & \text{for } x \in R \\ x & \text{for } x \in \Sigma \end{cases}$$

The monoid $\mathcal{M}_{R,\Sigma}$ thus defined is isomorphic to a submonoid of $\mathrm{Hom}((\Sigma \cup R)^*, (\Sigma \cup R)^*)$ with function composition. It admits a submonoid of *copyless* assignments.

▶ **Definition 2.16.** *We write $\mathcal{M}^{\mathrm{cl}}_{R,\Sigma}$ for the set of all $\alpha \in \mathcal{M}_{R,\Sigma}$ such that each letter $r \in R$ occurs at most once among all the $\alpha(r')$ for $r' \in R$.*

▶ **Proposition 2.17.** *$\mathcal{M}^{\mathrm{cl}}_{R,\Sigma}$ is a submonoid of $\mathcal{M}_{R,\Sigma}$. In other words, copylessness is preserved by composition (and the identity assignment is copyless).*

The following proposition ensures that this composition does what we expect. Recall from Definition 2.2 that $(-)^\dagger$ sends $\mathcal{M}_{R,\Sigma}$ to $(\Sigma^*)^R \to (\Sigma^*)^R$.

▶ **Proposition 2.18.** *For all $\alpha, \beta \in \mathcal{M}_{R,\Sigma}$, we have $(\alpha \bullet \beta)^\dagger = \beta^\dagger \circ \alpha^\dagger$.*

To incorporate information concerning the states of an SST, we define below a special case of the *wreath product* of transformation monoids.

▶ **Definition 2.19.** *Let M be a monoid whose multiplication is denoted by $m, m' \mapsto m \cdot m'$. We define $M \wr Q$ as the monoid whose set of elements is $Q \to Q \times M$ and whose monoid multiplication is, for $\mu, \mu' : Q \to Q \times M$,*

$$(\mu \bullet \mu') : q \mapsto (\pi_1 \circ \mu' \circ \pi_1 \circ \mu(q),\ (\pi_2 \circ \mu(q)) \cdot (\pi_2 \circ \mu' \circ \pi_1 \circ \mu(q)))$$

where $\pi_1 : Q \times M \to Q$ and $\pi_2 : Q \times M \to M$ are the projections.

For instance, if M is the trivial monoid with one element, $Q \wr M$ is isomorphic to $Q \to Q$ with *reverse* composition as the monoid multiplication: $f \bullet g = g \circ f$.

▶ **Proposition 2.20.** *Let $(Q, q_0, R, \delta, \vec{u}_I, F)$ be an SST that computes $f : \Gamma^* \to \Sigma^*$ (using the notations of Definition 2.3). For all $c \in \Gamma$, we have $\delta(-, c) \in \mathcal{M}_{R,\Sigma} \wr Q$, and the SST is copyless if and only if $\{\delta(-, c) \mid c \in \Gamma\} \subseteq \mathcal{M}^{\mathrm{cl}}_{R,\Sigma} \wr Q$. Furthermore, for all $w_1 \dots w_n \in \Gamma^*$,*

$$f(w_1 \dots w_n) \quad = \quad F(g(q_0))^\dagger(\alpha^\dagger(\vec{v})) \quad \text{where} \quad (g, \alpha) = \delta(-, w_1) \bullet \dots \bullet \delta(-, w_n)$$

Finally, it will sometimes be useful to consider monoids of assignments over an *empty* output alphabet. This allows us to keep track of how the registers are shuffled around by transitions.

▶ **Proposition 2.21.** *Let R and Σ be disjoint finite sets. There is a monoid morphism $\mathcal{M}_{R,\Sigma} \to \mathcal{M}_{R,\varnothing}$, that sends the submonoid $\mathcal{M}^{\mathrm{cl}}_{R,\Sigma}$ to $\mathcal{M}^{\mathrm{cl}}_{R,\varnothing}$. For any Q, this extends to a morphism $\mathcal{M}_{R,\Sigma} \wr Q \to \mathcal{M}_{R,\varnothing} \wr Q$ that sends $\mathcal{M}^{\mathrm{cl}}_{R,\Sigma} \wr Q$ to $\mathcal{M}^{\mathrm{cl}}_{R,\varnothing} \wr Q$. We shall use the name $\mathtt{erase}_\Sigma$ for both morphisms (R and Q being inferred from the context).*

▶ Remark 2.22. Consider an SST with a transition function δ. Let $\varphi_\delta \in \mathrm{Hom}(\Gamma^*, \mathcal{M}^{\mathrm{cl}}_{R,\varnothing} \wr Q)$ be defined by $\varphi_\delta(c) = \mathtt{erase}_\Sigma(\delta(-, c))$ for $c \in \Gamma$. The range $\varphi_\delta(\Gamma^*)$ is precisely the *substitution transition monoid (STM)* defined in [11, Section 3].

▶ **Proposition 2.23.** *For any finite R, the monoid $\mathcal{M}^{\mathrm{cl}}_{R,\varnothing}$ is* finite. *As a consequence, the substitution transition monoid of any* copyless *SST is finite.*

Proof idea. For all $\alpha \in \mathcal{M}^{\mathrm{cl}}_{R,\varnothing}$ and $r \in R$, observe that $|\alpha(r)| \leq |R|$. ◀

3 Complements on HDT0L systems, SSTs and polyregular functions

Before embarking on the study of our new comparison-free polyregular functions, we state some minor results that consolidate our understanding of pre-existing classes.

Layered HDT0L systems. Let us transpose the layering condition from SSTs to HDT0L systems. The hierarchy of models that we get corresponds *with an offset* to layered SSTs.

▶ **Definition 3.1.** *A HDT0L system $(\Gamma, \Sigma, \Delta, d, (h_c)_{c \in \Gamma}, h')$ is* k-layered *if its working alphabet can be partitioned as $\Delta = \Delta_0 \sqcup \dots \sqcup \Delta_k$ such that, for all $c \in \Gamma$ and $i \in \{0, \dots, k\}$:*

- *for $r \in \Delta_i$, we have $h_c(r) \in (\Delta_0 \sqcup \dots \sqcup \Delta_i)^*$;*
- *each letter in Δ_i appears at most once among all the $\alpha(r)$ for $r \in \Delta_i$ (but those in $\Delta_0 \sqcup \dots \sqcup \Delta_{i-1}$ may appear an arbitrary number of times).*

▶ **Theorem 3.2.** *For $k \in \mathbb{N}$, a function can be computed by a k-layered SST if and only if it can be specified by a $(k+1)$-layered HDT0L system.*

In particular, regular functions correspond to 1-layered HDT0L systems.

The obvious translation from HDT0L systems to SSTs preserves 1-layeredness and produces a single-state machine, so one may sacrifice copylessness to eliminate states for SSTs.

▶ **Corollary 3.3.** *Every regular function can be computed by a single-state 1-layered SST.*

The converse to this corollary does not hold: the single-state 1-layered SST of Example 2.4 computes a function which is not regular (cf. Remark 2.9).

Polyregular functions vs layered SSTs. By applying some results from [4], we can state a variant of Definition 2.11 which is a bit more convenient for us.

▶ **Proposition 3.4.** *Polyregular functions are the smallest class closed under composition that contains the regular functions and the squaring with underlining functions* $\texttt{squaring}_\Gamma$.

This allows us to show that composing HDT0L transductions with at most polynomial growth yields the polyregular functions. One direction of this equivalence is proved by encoding $\texttt{squaring}_\Gamma$ as a composition of two SSTs, one of which is Example 2.4. More precisely:

▶ **Theorem 3.5.** *Let $f : \Gamma^* \to \Sigma^*$. The following are equivalent:*

(i) *f is polyregular;*
(ii) *f can be obtained as a composition of layered SSTs;*
(iii) *f can be obtained as a composition of single-state 1-layered SSTs.*

But layered SSTs by themselves are *strictly less expressive* than polyregular functions, as we shall see later in Theorem 8.1. Therefore, as promised in the introduction:

▶ **Corollary 3.6** (claimed in [13, Section 6])**.** *Layered SSTs are* not *closed under composition.*

The importance of being stateful. One interesting aspect of Theorem 3.2 is that 1-layered HDT0L systems can be seen, through Remark 2.6, as a kind of one-way transducer model for regular functions that does not use an explicit control state. This is in contrast with copyless SSTs, whose expressivity critically depends on the states (unlike copyful SSTs).

▶ **Proposition 3.7.** *The sequential (and therefore regular) function defined by the transducer of Figure 2 (Section 2.2)* cannot *be computed by a* single-state copyless SST.

In fact, the knowledgeable reader can verify that this counterexample belongs to the *first-order letter-to-letter sequential functions*, one of the weakest classical transduction classes.

Closure under map. The pattern of Example 2.10 (iterated reverse) can be generalized:

▶ **Definition 3.8.** *Let $f : \Gamma^* \to \Sigma^*$ and suppose that $\# \notin \Gamma \cup \Sigma$. We define the function* $\mathbf{map}(f) : w_1\# \dots \#w_n \in (\Gamma \cup \{\#\})^* \mapsto f(w_1)\# \dots \#f(w_n) \in (\Sigma \cup \{\#\})^*$.

▶ **Proposition 3.9.** *If f is an HDT0L transduction, then so is* $\mathbf{map}(f)$. *For each $k \geq 1$, the functions that can be computed by k-layered HDT0L systems are also closed under* **map**.

As an immediate corollary, closure under **map** holds for both regular and polyregular functions, but this was already known. In fact, $\texttt{map}(f, [x_1, \dots, x_n]) = [f(x_1), \dots, f(x_n)]$ is an essential primitive in the *regular list functions* [6] and *polynomial list functions* [4, §4], two list-processing programming languages that characterize regular and polyregular functions respectively. We will come back to this point in Corollary 8.5 and the subsequent remark.

4 Composition by substitution

At last, we now introduce the class of *comparison-free polyregular functions*. The simplest way to define them is to start from the regular functions.

▶ **Definition 4.1.** *Let $f : \Gamma^* \to I^*$, and for each $i \in I$, let $g_i : \Gamma^* \to \Sigma^*$. The* composition by substitution *of f with the family $(g_i)_{i\in I}$ is the function*

$$\mathrm{CbS}(f, (g_i)_{i\in I}) \quad : \quad w \mapsto g_{i_1}(w) \dots g_{i_k}(w) \quad \text{where } i_1 \dots i_k = f(w)$$

That is, we first apply f to the input, then every letter i in the result of f is substituted by the image of the original input by g_i. Thus, $\mathrm{CbS}(f, (g_i)_{i\in I})$ is a function $\Gamma^* \to \Sigma^*$.

▶ **Definition 4.2.** *The smallest class of string-to-string functions closed under* CbS *and containing all regular functions is called the class of* comparison-free polyregular functions.

▶ **Example 4.3.** The following variant of "squaring with underlining" (cf. Definition 2.11) is comparison-free polyregular: $\mathtt{cfsquaring}_\Gamma : \mathtt{123} \in \Gamma^* \mapsto \underline{1}123\underline{2}123\underline{3}123 \in (\Gamma \cup \underline{\Gamma})^*$.

Indeed, it can be expressed as $\mathtt{cfsquaring}_\Gamma = \mathrm{CbS}(f, (g_i)_{i\in I})$ where $I = \Gamma \cup \{\#\}$, the function $f : w_1 \dots w_n \mapsto w_1 \# \dots w_n \#$ is regular (more than that, a morphism between free monoids) and $g_\# = \mathrm{id}$, $g_c : w \mapsto \underline{c}$ for $c \in \Gamma$ are also regular. Its growth rate is quadratic, while regular functions have at most linear growth. Other examples that also require a single composition by substitution are given in Theorem 8.1.

We can already justify the latter half of the name of our new class. Using the "polynomial list functions" mentioned at the end of the previous section, we prove:

▶ **Theorem 4.4.** *Polyregular functions are closed under composition by substitution.*

▶ **Corollary 4.5.** *Every comparison-free polyregular function is, indeed, polyregular.*

Fundamentally, Definition 4.2 is inductive: it considers the functions generated from the base case of regular functions by applying compositions by substitution. The variant below with more restricted generators is sometimes convenient.

▶ **Definition 4.6.** *A string-to-string function is said to be:*

- *of* rank at most 0 *if it is regular;*
- *of* rank at most $k+1$ (for $k \in \mathbb{N}$) *if it can be written as $\mathrm{CbS}(f, (g_i)_{i\in I})$ where $f : \Gamma^* \to I^*$ is* regular *and each $g_i : \Gamma^* \to \Sigma^*$ is of rank at most k.*

▶ **Proposition 4.7.** *A function f is comparison-free polyregular if and only if there exists some $k \in \mathbb{N}$ such that f has rank at most k. In that case, we write $\mathrm{rk}(f)$ for the least such k and call it the* rank *of f. If $(g_i)_{i\in I}$ is a family of comparison-free polyregular functions,*

$$\mathrm{rk}(\mathrm{CbS}(f, (g_i)_{i\in I})) \leq 1 + \mathrm{rk}(f) + \max_{i\in I} \mathrm{rk}(g_i)$$

A straightforward consequence of this definition is that, just like regular functions, cfp functions are closed under *regular conditionals* and concatenation.

▶ **Proposition 4.8.** *Let $f, g : \Gamma^* \to \Sigma^*$ be comparison-free polyregular functions and $L \subseteq \Gamma^*$ be a regular language. The function that coincides with f on L and with g on $\Gamma^* \setminus L$ is cfp, and so is $w \in \Gamma^* \mapsto f(w) \cdot g(w)$; both have rank at most $\max(\mathrm{rk}(f), \mathrm{rk}(g))$.*

5 Comparison-free pebble transducers

We now characterize our function class by a machine model that will explain our choice of the adjective "comparison-free", as well as the operational meaning of the notion of rank we just defined. It is based on the *pebble transducers* first introduced for trees by Milo, Suciu and Vianu [28] and later investigated in the special case of strings by Engelfriet and Maneth [17, 14]. However, the definition using composition by substitution will remain our tool of choice to prove further properties, so the next sections do not depend on this one.

▶ **Definition 5.1.** *Let $k \in \mathbb{N}$ with $k \geq 1$. Let Γ, Σ be finite alphabets and $\triangleright, \triangleleft \notin \Gamma$.*

*A k-*pebble stack *on an input string $w \in \Gamma^*$ consists of an ordered list of p positions in the string $\triangleright w \triangleleft$ (i.e. of p integers between 1 and $|w|+2$) for some $p \in \{1, \dots, k\}$. We therefore write $\mathrm{Stack}_k = \mathbb{N}^0 \cup \mathbb{N}^1 \cup \cdots \cup \mathbb{N}^k$, keeping in mind that given an input w, we will be interested in "legal" values bounded by $|w|+2$.*

A comparison-free k-pebble transducer (k-CFPT) *consists of a finite set of states Q, an initial state $q_I \in Q$ and a family of transition functions*

$$Q \times (\Gamma \cup \{\triangleright, \triangleleft\})^p \to Q \times (\mathbb{N}^p \to \mathrm{Stack}_k) \times \Sigma^* \quad \text{for } 1 \leq p \leq k$$

where the $\mathbb{N}^p$ on the left is considered as a subset of Stack_k. For a given state and given letters $(c_1, \dots, c_p) \in (\Gamma \cup \{\triangleright, \triangleleft\})^p$, the allowed values for the stack update function $\mathbb{N}^p \to \mathrm{Stack}_k$ returned by the transition function are:

(identity)	$(i_1, \dots, i_p)$	$\mapsto$	$(i_1, \dots, i_p)$	$\in$	$\mathbb{N}^p$
(move left, only allowed when $c_p \neq \triangleright$)	$(i_1, \dots, i_p)$	$\mapsto$	$(i_1, \dots, i_p - 1)$	$\in$	$\mathbb{N}^p$
(move right, only allowed when $c_p \neq \triangleleft$)	$(i_1, \dots, i_p)$	$\mapsto$	$(i_1, \dots, i_p + 1)$	$\in$	$\mathbb{N}^p$
(push, only allowed when $p \leq k-1$)	$(i_1, \dots, i_p)$	$\mapsto$	$(i_1, \dots, i_p, 1)$	$\in$	$\mathbb{N}^{p+1}$
(pop, only allowed when $p \geq 1$)	$(i_1, \dots, i_p)$	$\mapsto$	$(i_1, \dots, i_{p-1})$	$\in$	$\mathbb{N}^{p-1}$

(Note that the codomains of all these functions are indeed subsets of Stack_k.)

The run of a CFPT over an input string $w \in \Gamma^*$ starts in the initial configuration comprising the initial state q_I, the initial k-pebble stack $(1) \in \mathbb{N}^1$, and the empty string as an initial output log. As long as the current stack is non-empty a new configuration is computed by applying the transition function to q and to $((\triangleright w \triangleleft)[i_1], \dots, (\triangleright w \triangleleft)[i_p])$ where $(i_1, \dots, i_p)$ is the current stack; the resulting stack update function is applied to $(i_1, \dots, i_p)$ to get the new stack, and the resulting output string in Σ^* is appended to the right of the current output log. If the CFPT ever terminates by producing an empty stack, the *output associated to w* is the final value of the output log.

This amounts to restricting in two ways[4] the definition of *pebble transducers* from [4, §2]:

- in a general pebble transducer, one can *compare positions*, i.e. given a stack $(i_1, \dots, i_p)$, the choice of transition can take into account whether[5] $i_j \leq i_{j'}$ (for any $1 \leq j, j' \leq p$);
- in a "push", new pebbles are initialized to the leftmost position ($\triangleright$) for a CFPT, instead of starting at the same position as the previous top of the stack (the latter would ensure the equality of two positions at some point; it is therefore an implicit comparison that we must relinquish to be truly "comparison-free").

[4] There is also an inessential difference: the definition given in [4] does not involve end markers and handles the edge case of an empty input string separately. This has no influence on the expressiveness of the transducer model. Our use of end markers follows [15, 25].

[5] One would get the same computational power, with the same stack size, by only testing whether $i_j = i_p$ for $j \leq p-1$ as in [28] (this is also essentially what happens in the nested transducers of [25]).

This limitation is similar to (but goes a bit further than) the "invisibility" of pebbles in a transducer model introduced by Engelfriet et al. [16] (another difference, unrelated to position comparisons, is that their transducers use an unbounded number of invisible pebbles).

▶ Remark 5.2. Our definition guarantees that "out-of-bounds errors" cannot happen during the run of a comparison-free pebble transducer. The sequence of successive configurations is therefore always well-defined. But it may be infinite, that is, it may happen that the final state is never reached. Thus, a CFPT defines a *partial* function.

That said, the set of inputs for which a given pebble tree transducer does not terminate is always a regular language [28, Theorem 4.7]. This applies *a fortiori* to CFPTs. Using this, it is possible[6] to extend any partial function $f : \Gamma^* \rightharpoonup \Sigma^*$ computed by a k-CFPT into a total function $f' : \Gamma^* \to \Sigma^*$ computed by another k-CFPT for the same $k \in \mathbb{N}$, such that $f'(x) = f(x)$ for x in the domain of f and $f'(x) = \varepsilon$ otherwise. This allows us to *only consider CFPTs computing total functions* in the remainder of the paper.

A special case of particular interest is $k = 1$: the transducer has a single reading head, push and pop are always disallowed.

▶ **Theorem 5.3** ([1])**.** *Copyless SSTs and 1-CFPTs – which are more commonly called* two-way (deterministic) finite transducers (2DFTs) *– are equally expressive.*

Since we took copyless SSTs as our reference definition of regular functions, this means that 2DFTs characterize regular functions. But putting it this way is historically backwards: the equivalence between 2DFTs and MSO transductions came first [15] and made this class deserving of the name "regular functions" before the introduction of copyless SSTs.

▶ Remark 5.4. There are two different numbering conventions for pebble transducers. In [4, 25], 2DFTs are 1-pebble transducers, which is consistent with our choice. However, several other papers (e.g. [28, 17, 14, 16, 12]) consider that a 2DFT is a *0-pebble* transducer (likewise, in [13], 2DFTs are 0-marble transducers). This is because they think of a pebble automaton not as a restricted multi-head automaton, but as an enriched 2DFA that can drop stationary markers (called pebbles) on input positions, with a single moving head that is not a pebble.

Let us now show the equivalence with Definition 4.2. The reason for this is similar to the reason why k-pebble transducers are equivalent to the *k-nested transducers*[7] of [25], which is deemed "trivial" and left to the reader in [25, Remark 6]. But in our case, one direction (Theorem 5.6) involves an additional subtlety compared to in [25]; to take care of it, we use the fact that the languages recognized by pebble automata are regular (this is also part of [28, Theorem 4.7]) together with regular conditionals (Proposition 4.8).

▶ **Proposition 5.5.** *If f is computed by a k-CFPT, and the g_i are computed by l-CFPTs, then $\mathrm{CbS}(f, (g_i)_{i\in I})$ is computed by a $(k + l)$-CFPT.*

▶ **Theorem 5.6.** *If $f : \Gamma^* \to \Sigma^*$ is computed by a k-CFPT, for $k \geq 2$, then there exist a finite alphabet I, a regular function $h : \Gamma^* \to I^*$ and a family $(g_i)_{i\in I}$ computed by $(k-1)$-CFPTs such that $f = \mathrm{CbS}(h, (g_i)_{i\in I})$.*

▶ **Corollary 5.7.** *For all $k \in \mathbb{N}$, the functions computed by $(k + 1)$-CFPTs are exactly the comparison-free polyregular functions of rank at most k.*

---

[6] Proof idea: do a first left-to-right pass to determine whether the input leads to non-termination of the original CFPT; if so, terminate immediately with an empty output; otherwise, move the first pebble back to the leftmost position and execute the original CFPT's behavior. This can be implemented by adding finitely many states, including those for a DFA recognizing non-terminating inputs.

[7] Remark: nested transducers should yield a machine-independent definition of polyregular functions as the closure of regular functions under a CbS-like operation that relies on *origin semantics* [29, §5].

6 Composition of basic functions

Another possible definition of cfp functions consists in swapping out $\mathtt{squaring}_\Gamma$ for some other function in Proposition 3.4:

▶ **Theorem 6.1.** *The class of comparison-free polyregular functions is the smallest class closed under usual function composition and containing both all regular functions and the functions* $\mathtt{cfsquaring}_\Gamma$ *(cf. Example 4.3) for all finite alphabets* Γ*.*

The hard part is to show that cfp functions are closed under composition. We exploit the following combinatorial phenomenon, often applied to the study of copyless SSTs: a copyless register assignment, i.e. an element of $\mathcal{M}^{\mathrm{cl}}_{R,\Delta}$ (cf. Section 2.4), can be specified by

- a "shape" described by an element of the *finite* monoid $\mathcal{M}^{\mathrm{cl}}_{R,\varnothing}$ (Proposition 2.23),
- plus finitely many "labels" in Σ^* (where Σ is the output alphabet) describing the constant factors that will be concatenated with the old register contents to give the new ones.

▶ **Proposition 6.2.** *There is a bijection*

$$\mathcal{M}^{\mathrm{cl}}_{R,\Delta} \quad\cong\quad \left\{ \left(\alpha, \vec{\ell}\right) \;\middle|\; \alpha \in \mathcal{M}^{\mathrm{cl}}_{R,\varnothing},\ \vec{\ell} \in \prod_{r\in R} (\Delta^*)^{|\alpha(r)|+1} \right\}$$

through which $\mathtt{erase}_\Delta : \mathcal{M}^{\mathrm{cl}}_{R,\Delta} \to \mathcal{M}^{\mathrm{cl}}_{R,\varnothing}$ *can be seen as simply removing the "labels"* $\vec{\ell}$*.*

Proof idea. Let $\beta \in \mathcal{M}^{\mathrm{cl}}_{R,\Delta}$. For each $r \in R$, one can write $\beta(r) = w_0 r'_1 w_1 \dots r'_n w_n$ with $w_0, \dots, w_n \in \Delta^*$ and $r'_1, \dots, r'_n \in R$ such that $r'_1 \dots r'_n = \mathtt{erase}_\Delta(\beta)(r) \in R^*$. ◀

This provides a clear way to represent a copyless register assignment inside the working memory of an SST: store the shape in the state and the labels in registers. Another important fact for us is that given two assignments $\beta, \beta' \in \mathcal{M}^{\mathrm{cl}}_{R,\Delta}$ the labels of $\beta \bullet \beta'$ can be obtained as a *copyless* recombination of the labels of β and β'.

(There is a subtlety worth mentioning here: while the set of stateful transitions $\mathcal{M}^{\mathrm{cl}}_{R,\Delta} \wr Q$ also admits a "shape + labels" representation, its monoid multiplication does *not* have this copylessness property. This prevents a naive proof of the closure under composition of copyless SSTs from working. Nevertheless, the composition of two regular functions *is* always regular, and we rely on this fact to prove Theorem 6.1.)

The rest of the proof of Theorem 6.1 is relegated to the technical appendix.

7 Rank vs asymptotic growth

Our next result is the comparison-free counterpart to recent work on polyregular functions by Lhote [25], whose proof techniques (in particular the use of Ramsey's theorem) we reuse. Compare item (ii) below to the main theorem of [25] and item (iii) – which provides yet another definition of cfp functions – to [25, Appendix A].

▶ **Theorem 7.1.** *Let* $f : \Gamma^* \to \Sigma^*$ *and* $k \in \mathbb{N}$*. The following are equivalent:*

(i) *f is comparison-free polyregular with rank at most k;*

(ii) *f is comparison-free polyregular and $|f(w)| = O(|w|^{k+1})$;*

(iii) *there exists a regular function $g : (\{0, \dots, k\} \times \Gamma)^* \to \Sigma^*$ such that $f = g \circ \mathtt{cfpow}^{(k+1)}_\Gamma$, with the following inductive definition: $\mathtt{cfpow}^{(0)}_\Gamma : w \in \Gamma^* \mapsto \varepsilon \in (\varnothing \times \Gamma)^*$ and*

$$\mathtt{cfpow}^{(n+1)}_\Gamma \quad : \quad w \mapsto (n, w_1) \cdot \mathtt{cfpow}^{(n)}_\Gamma(w) \cdot \ldots \cdot (n, w_{|w|}) \cdot \mathtt{cfpow}^{(n)}_\Gamma(w)$$

To make (ii) $\implies$ (i) more precise, if f is cfp with $\mathrm{rk}(f) \geq 1$, then it admits a sequence of inputs $w_0, w_1, \ldots \in \Gamma^*$ such that $|w_n| \to +\infty$ and $|f(w_n)| = \Omega(|w_n|^{\mathrm{rk}(f)+1})$.

Note that $\mathtt{cfpow}^{(2)}_\Gamma$ and $\mathtt{cfsquaring}_\Gamma$ are the same up to a bijection $\{0,1\} \times \Gamma \cong \Gamma \cup \underline{\Gamma}$.

▶ Remark 7.2. The growth of an HDT0L transduction is also related, in a very similar way to item (ii) above, to the number of layers required in any SST that computes it [13, §5].

Some proof elements. Let us present a few definitions and lemmas to give an idea of the ingredients that go into the proof. Those technical details take up the rest of this section.

Lhote's paper [25] makes a heavy use of *factorizations* of strings that depend on a morphism to a finite monoid. This is also the case for our proof, but we have found that a slightly different definition of the kind of factorization that we want works better for us.

▶ **Definition 7.3** (similar but not equivalent to [25, Definition 19])**.** *An* r-split *of a string* $s \in \Gamma^*$ *according to a morphism* $\varphi : \Gamma^* \to M$ *is a tuple* $(u, v_1, \dots, v_r, w) \in (\Gamma^*)^{r+2}$ *such that:*

- $s = uv_1 \dots v_n w$ *with* v_i *non-empty for all* $i \in \{1, \dots, r\}$*;*
- $\varphi(u) = \varphi(uv_1) = \cdots = \varphi(uv_1v_r)$*;*
- $\varphi(w) = \varphi(v_r w) = \cdots = \varphi(v_1 \dots v_r w)$*.*

▶ **Proposition 7.4** (immediate from the definition)**.** $(u, v_1, \dots, v_r, w)$ *is an* r*-split if and only if, for all* $i \in \{1, \dots, r\}$*,* $(uv_1 \dots v_{i-1}, v_i, v_{i+1} \dots v_r w)$ *is a 1-split.*

The difference with the $(1, r)$*-factorizations* of [25, Definition 19] is that we have replaced the equality and idempotency requirements on $\varphi(v_1), \dots, \varphi(v_n)$ by the "boundary conditions" involving $\varphi(u)$ and $\varphi(w)$ (actually, $(1, r+2)$-factorizations induce r-splits). This change allows us to establish a subclaim used in the proof of Lemma 7.7 in an elementary way.

The point of r-splits is that given a split of an input string according to the morphism that sends it to the corresponding transition in a SST, we have some control over what happens to the output of the SST if we pump a middle factor in the split. Furthermore, it suffices to consider a quotient of the transition monoid which is finite when the SST is copyless (this is similar to Proposition 2.23). More precisely, we have the key lemma below, which is used pervasively throughout our proof of Theorem 7.1:

▶ **Lemma 7.5.** *Let* $f : \Gamma^* \to \Sigma^*$ *be a regular function. There exist a morphism to a* finite *monoid* $\nu_f : \Gamma^* \to \mathcal{N}(f)$ *and, for each* $c \in \Sigma$*, a set of* producing triples $P(f, c) \subseteq \mathcal{N}(f)^3$ *such that, for any* 1-split *according to* ν_f *composed of* $u, v, w \in \Gamma^*$ *– i.e.* $\nu_f(uv) = \nu_f(u)$ *and* $\nu_f(vw) = \nu_f(w)$ *– we have:*

- *if* $(\nu_f(u), \nu_f(v), \nu_f(w)) \in P(f, c)$*, then* $|f(uvw)|_c > |f(uw)|_c$*;*
- *otherwise (when the triple is not producing),* $|f(uvw)|_c = |f(uw)|_c$*.*

Furthermore, in the producing case, we get as a consequence that $\forall n \in \mathbb{N},\ |f(uv^n w)|_c \geq n$*.*

▶ **Definition 7.6.** *We fix once and for all a choice of* $\mathcal{N}(f)$*,* ν_f *and* $P(f, c)$ *for each* $c \in \Sigma$ *and regular* $f : \Gamma^* \to \Sigma^*$*. We say that a 1-split* (u, v, w) *is* producing with respect to (f, c) *when* $(\nu_f(u), \nu_f(v), \nu_f(w)) \in P(f, c)$*. For* $\Pi \subseteq \Sigma$*, we also set* $P(f, \Pi) = \bigcup_{c \in \Pi} P(f, c)$*.*

Something like Lemma 7.5 (but not exactly) appears in the proof of [25, Lemma 18]. We first apply it to prove the following lemma, which is morally a counterpart to the "$k = 1$ case" of the central Dichotomy Lemma from [25], with r-splits instead of (k, r)-factorizations.

▶ **Lemma 7.7.** *Let* $f : \Gamma^* \to \Sigma^*$ *be regular and* $\varphi : \Gamma^* \to M$ *be a morphism with* M *finite. Suppose that* $\pi \circ \varphi = \nu_f$ *for some other morphism* $\pi : M \to \mathcal{N}(f)$*. Let* $r \geq 1$ *and* $\Pi \subseteq \Sigma$*.*

We define $L(f, \Pi, \varphi, r)$ *to be the set of strings that admit an* r*-split* $s = uv_1 \dots v_r w$ *according to* φ *such that all the triples* $(uv_1 \dots v_{i-1}, v_i, v_{i+1} \dots v_r w)$ *are producing with respect to* (f, Π) *– let us call this a producing* r*-split with respect to* (f, Π, φ)*.*

Then $L(f, \Pi, \varphi, r)$ *is a regular language, and* $\sup\{|f(s)|_\Pi \mid s \in \Gamma^* \setminus L(f, \Pi, \varphi, r)\} < \infty$*.*

Our proof of the above lemma uses the proposition below, analogous to [25, Claim 20]. Its statement is a bit stronger than necessary for this purpose, but it will be reused in the proof of Theorem 8.3; as for its proof, this is where a standard Ramsey argument occurs.

▶ **Proposition 7.8.** *Let Γ be an alphabet, M be a finite monoid and $\varphi : \Gamma^* \to M$ be a morphism. There exists $N \in \mathbb{N}$ such that any string $s = uvw \in \Gamma^*$ such that $|v| \geq N$ admits an r-split $s = u'v'_1 \dots v'_r w'$ according to φ in which u is a prefix of u' and w is a suffix of w'.*

To leverage Lemma 7.7, we combine it with an elementary property of composition by substitution that does not depend on the previous technical development. (Compare the assumptions of the lemma below with the conclusion of Lemma 7.7.)

▶ **Lemma 7.9.** *Let $g : \Gamma^* \to I^*$ be a regular function and, for each $i \in I$, let $h_i : \Gamma^* \to \Sigma^*$ be comparison-free polyregular of rank at most k. Suppose that $\sup_{s \in \Gamma^*} |g(s)|_J < \infty$ where*

$$J = \begin{cases} \{i \in I \mid \mathrm{rk}(h_i) = k\} & \text{when } k \geq 1 \\ \{i \in I \mid |h_i(\Gamma^*)| = \infty\} & \text{when } k = 0 \end{cases}$$

(Morally, regular functions with finite range play the role of "comparison-free polyregular functions of rank -1".) Then $\mathrm{rk}(\mathrm{CbS}(g, (h_i)_{i \in I})) \leq k$.

The above lemma can be compared to [25, Claim 22], but it also seems to be related to the way the "nested transducer" R_{k+1} is defined in the proof of the Dichotomy Lemma in [25]: indeed, R_{k+1} can call either a k-nested subroutine or a $(k-1)$-nested one.

The remainder of the proof of Theorem 7.1 consists mainly of a rather technical induction on the rank, which we present in the appendix.

8 Separation results

Let us now demonstrate that the class of cfp functions is incomparable with the class of HDT0L transductions and is a strict subclass of polyregular functions.

▶ **Theorem 8.1.** *There exist comparison-free polyregular functions which are not HDT0L:*
- **(i)** *the function $a^n \in \{a\}^* \mapsto (a^n b)^{n+1} \in \{a, b\}^*$ for $a \neq b$;*
- **(ii)** *the function $w \in \Sigma^* \mapsto w^{|w|}$ for $|\Sigma| \geq 2$ (a simplification of Example 4.3);*
- **(iii)** (from [13, §6]) *the cfp functions that map $a^n \# w \in \Sigma^*$ to $(w\#)^n$ for $a, \# \in \Sigma$, $a \neq \#$.*

▶ Remark 8.2. The first example in [13, §5] shows that $a^n \mapsto a^{n \times n}$ is HDT0L (via the equivalent model of marble transducers), hence the necessity of $|\Sigma| \geq 2$ above. More generally, Douéneau-Tabot has shown very recently that *every polyregular function with unary output alphabet is HDT0L* [12]. So polyregular functions with unary output coincide with *polynomial growth* $\mathbb{N}$*-rational series* (cf. Remark 2.14), and the latter admit several algebraic characterizations in the literature (see [32] and [3, Chapter 9, Exercise 1.2]).

▶ **Theorem 8.3.** *Some HDT0L transductions are polyregular but not comparison-free:*
- **(i)** *$f : a^n \in \{a\}^* \mapsto ba^{n-1}b \dots baabab$ (with $f(\varepsilon) = \varepsilon$ and $f(a) = b$);*
- **(ii)** $\mathbf{map}(a^n \mapsto a^{n \times n}) : a^{n_1} \# \dots \# a^{n_k} \mapsto a^{n_1 \times n_1} \# \dots \# a^{n_k \times n_k}$ *(cf. Definition 3.8).*

▶ Remark 8.4. The function $a^{n_1} \# \dots \# a^{n_k} \mapsto a^{n_1 \times n_1 + \dots + n_k \times n_k}$ obtained by erasing the #s in the output of $\mathbf{map}(a^n \mapsto a^{n \times n})$ is also *not* comparison-free. This result implies the second item of Theorem 8.3 by composition with the erasing morphism; we do not prove it here, but it appears in Douéneau-Tabot's aforementioned paper [12]. Therefore, according to [12], *not* every polyregular function with unary output is comparison-free.

To see why the first of the two functions in Theorem 8.3 is HDT0L, observe that it is Example 2.4 for $\Gamma = \{a\}$ (taking $b = \underline{a}$). As for the second one, combine Proposition 3.9 and the first observation in Remark 8.2.

The non-membership parts of Theorems 8.1 and 8.3 require more work. For the former, we use pumping arguments on HDT0L systems. Item (ii) of Theorem 8.3 is handled by first appealing to Theorem 7.1 to reduce to showing that $\mathbf{map}(a^n \mapsto a^{n\times n}) \neq \mathrm{CbS}(g, (h_i)_{i\in I})$ when g and all the h_i are *regular* functions; a combination of pumping and of a combinatorial argument then shows that inputs with $|I|$ occurrences of # suffice to discriminate the two sides of the inequality. This result also has the following consequence:

▶ **Corollary 8.5.** *Comparison-free polyregular functions are not closed under* **map**.

▶ Remark 8.6. Contrast with Proposition 3.9. The discussion that follows that proposition lends some significance to the above corollary: the latter rules out the obvious conjectures for a characterization of cfp functions in the style of regular/polynomial list functions.

As for item (i) of Theorem 8.3, it concerns a function whose domain consists of words over a unary alphabet, i.e., up to isomorphism, a *sequence*. This motivates the study of such sequences, which is the subject of the next section.

9 Comparison-free polyregular sequences

From now on, we identify $\mathbb{N}$ with the set of words $\{a\}^*$ and freely speak, for instance, of cfp sequences $\mathbb{N} \to \Gamma^*$ instead of cfp functions $\{a\}^* \to \Gamma^*$. It turns out that cfp sequences admit a characterization as finite combinations of what we call *poly-pumping sequences*.

▶ **Definition 9.1.** *A* poly-pumping sequence *is a function of the form* $[\![e]\!] : \mathbb{N} \to \Sigma^*$ *where*
- *e is a* polynomial word expression *generated by* $e ::= w \mid e \cdot e' \mid e^*$ *where* $w \in \Sigma^*$*;*
- $[\![w]\!](n) = w$, $[\![e \cdot e']\!](n) = [\![e]\!](n)[\![e']\!](n)$ *and* $[\![e^*]\!](n) = ([\![e]\!](n))^n$.

The star-height *of a polynomial word expression is defined in the usual way.*

▶ **Theorem 9.2.** *Let* $s : \mathbb{N} \to \Sigma^*$ *and* $k \in \mathbb{N}$. *The sequence s is comparison-free polyregular with* $\mathrm{rk}(s) \leq k$ *if and only if there exists* $p > 0$ *such that, for any* $m < p$, *there is a polynomial word expression e of star-height at most* $k + 1$ *such that* $\forall n \in \mathbb{N},\ s((n + 1)p + m) = [\![e]\!](n)$.

In short, the cfp sequences are exactly the *ultimately periodic combinations* of poly-pumping sequences. Our proof strategy is an induction on k.

The base case $k = 0$ says that regular sequences are ultimately periodic combinations of *pumping sequences* $n \mapsto u_0(v_1)^n \dots (v_l)^n u_l$. An essentially equivalent result is stated with a proof sketch using 2DFTs in [10, p. 90]; we propose an alternative proof using copyless SSTs. (Non-deterministic two-way transducers (2NFTs) taking unary inputs have also been studied [23]; furthermore, the notion of "k-iterative language" that appears in a pumping lemma for general 2NFTs [35] is related to the shape of the above pumping sequences.)

To make the inductive step go through, it is enough to synchronize the periods of the different poly-pumping sequences involved and to observe that $\mathrm{CbS}([\![e]\!], ([\![e'_i]\!])_{i\in I})$ is realized by an expression obtained by substituting the e'_i for i in e.

Coming back to Theorem 8.3, we show that $a^n \mapsto ba^{n-1}b \dots bab$ is not comparison-free polyregular by proving that its subsequences are *not* poly-pumping: for every poly-pumping sequence $s : \mathbb{N} \to \{a, b\}^*$, there is a uniform bound on the number of distinct contiguous subwords of the shape $baa \dots ab$ occuring in each $s(n)$ for $n \in \mathbb{N}$. Another consequence of Theorem 9.2 that we establish by induction over expressions contrasts with Corollary 8.5:

▶ **Corollary 9.3.** *If* $f : \Gamma^* \to \Sigma^*$ *and* $s : \mathbb{N} \to (\Gamma \cup \{\#\})^*$ *are cfp, so is* $\mathbf{map}(f) \circ s$.

10 Further topics

Functional programming. We mentioned in the introduction a forthcoming characterization of cfp functions using Church-encoded strings in a λ-calculus with linear types, in the vein of our previous results [31, 30]. Meanwhile, Corollary 8.5 could be understood as negative result in the search for another kind of functional programming characterization (cf. Remark 8.6).

It is also worth noting that the copying discipline of layered SSTs is very similar to what happens in the *parsimonious λ-calculus* [27]: a datum of type $!\tau$ cannot be duplicated into two copies of the same type $!\tau$, but it may yield an arbitrary number of copies of type τ without the modality "!". Since the function classes defined following the methodology of [31, 30] are automatically closed under composition, Theorem 3.5 leads us to conjecture that polyregular functions can be characterized in a variant of the parsimonious λ-calculus.

First-order interpretations. As we already said, regular and polyregular functions both admit logical characterizations using Monadic Second-Order Logic [15, 8]. The basic conceit behind these definitions is that a string w may be regarded as a finite model $\mathfrak{M}(w)$ over a signature containing the order relation $\leq$ on positions and predicates encoding their labeling.

The classes obtained by replacing MSO with first-order logic (FO) are to (poly)regular functions what star-free languages are to regular languages, see [11, 4]. We expect that in the same way, replacing regular functions (i.e. MSO transductions) by FO transductions in Definition 4.2 and Theorem 6.1 results in the same class in both cases, which would then be the natural FO counterpart of cfp functions. Furthermore, we believe it can be defined logically. Given a finite model $\mathfrak{U} = (U, R, \ldots)$, we write $\mathfrak{U}^k$ for the k^{th} power $(U^k, R_1, \ldots, R_k, \ldots)$ where $R_i(x_1, \ldots, x_m)$ of arity m is defined as $R(\pi_i(x_1), \ldots, \pi_i(x_m))$ for $1 \leq i \leq k$.

▶ **Conjecture 10.1.** *A function $f : \Gamma^* \to \Sigma^*$ is "FO comparison-free polyregular" if and only if there exists $k \in \mathbb{N}$ and a one-dimensional FO interpretation φ such that for every $w \in \Gamma^*$ with $|w| \geq 2$, there is an isomorphism of structures $\mathfrak{M}(f(w)) \simeq \varphi\left(\mathfrak{M}(w)^k\right)$.*

On an intuitive level, this seems to capture the inability to compare the positions of two heads of comparison-free pebble transducers. However, as mentioned to us by M. Bojańczyk, the naive transposition of this conjecture to MSO fails because the direct product, generalized to Henkin structures, does not preserve *standard* second-order models.

Integer sequences. Recall from Remarks 8.2 and 8.4 that for unary outputs, polyregular and layered HDT0L transductions coincide, but comparison-free polyregular functions form a strictly smaller class (those results come from [12]). If we also restrict to unary inputs – in other words, if we consider sequences $\mathbb{N} \to \mathbb{N}$ – then we are fairly confident at this stage that the three classes collapse to a single one, and that this can be shown by routine methods:

▷ Claim 10.2. The classes of polyregular, comparison-free polyregular and layered HDT0L functions coincide on sequences of natural numbers.

Note that we already have a description of cfp integer sequences by specializing Theorem 9.2.

Membership and equivalence. We presented comparison-free polyregular functions as a strict subclass of polyregular functions. This leads to a natural *membership problem*, for which partial results were recently obtained by Douéneau-Tabot [12]:

▷ Problem 10.3. Is there an algorithm taking as input a (code for a) pebble transducer which decides whether the corresponding function $\Sigma^* \to \Gamma^*$ is comparison-free or not?

There are many similar problems of interest on the frontier between comparison-free and general polyregular functions. We hope that investigating such issues may also lead to machine/syntax-free characterizations of the containment between the two classes.

Finally, a major open problem on polyregular functions is the *equivalence problem*:

▷ Problem 10.4. Is there an algorithm taking as input two pebble transducers which decides whether they compute the same function?

Interestingly, a positive answer is known for HDT0L transductions. There is an short proof using Hilbert's basis theorem [24], which is now understood to be an example of a general approach using polynomial grammars (see e.g. [2, 5]). One could hope that a restriction to comparison-free pebble transducers also puts the equivalence problem within reach of known tools. Unfortunately, the extended polynomial grammars that would serve as the natural target for a reduction from 2-CFPT equivalence already have an undecidable zeroness problem (this was shown recently by Schmude [34]). This does *not* extend, however, to an undecidability proof for the CFPT equivalence problem, so the latter is still open.

References

1 Rajeev Alur and Pavol Černý. Expressiveness of streaming string transducers. In Kamal Lodaya and Meena Mahajan, editors, *IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS 2010, December 15-18, 2010, Chennai, India*, volume 8 of *LIPIcs*, pages 1–12. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2010. doi:10.4230/LIPIcs.FSTTCS.2010.1.

2 Michael Benedikt, Timothy Duff, Aditya Sharad, and James Worrell. Polynomial automata: Zeroness and applications. In *2017 32nd Annual ACM/IEEE Symposium on Logic in Computer Science (LICS)*, pages 1–12, Reykjavik, Iceland, June 2017. IEEE. doi:10.1109/LICS.2017.8005101.

3 Jean Berstel and Christophe Reutenauer. *Noncommutative Rational Series with Applications*, volume 137 of *Encyclopedia of Mathematics and its Applications*. Cambridge University Press, 2010.

4 Mikołaj Bojańczyk. Polyregular functions, 2018. arXiv:1810.08760.

5 Mikołaj Bojańczyk. The Hilbert method for transducer equivalence. *ACM SIGLOG News*, 6(1):5–17, 2019. doi:10.1145/3313909.3313911.

6 Mikołaj Bojańczyk, Laure Daviaud, and Shankara Narayanan Krishna. Regular and First-Order List Functions. In *Proceedings of the 33rd Annual ACM/IEEE Symposium on Logic in Computer Science - LICS '18*, pages 125–134, Oxford, United Kingdom, 2018. ACM Press. doi:10.1145/3209108.3209163.

7 Mikołaj Bojańczyk and Amina Doumane. First-order tree-to-tree functions. In Holger Hermanns, Lijun Zhang, Naoki Kobayashi, and Dale Miller, editors, *LICS '20: 35th Annual ACM/IEEE Symposium on Logic in Computer Science, Saarbrücken, Germany (online conference), July 8-11, 2020*, pages 252–265. ACM, 2020. doi:10.1145/3373718.3394785.

8 Mikołaj Bojańczyk, Sandra Kiefer, and Nathan Lhote. String-to-String Interpretations With Polynomial-Size Output. In Christel Baier, Ioannis Chatzigiannakis, Paola Flocchini, and Stefano Leonardi, editors, *46th International Colloquium on Automata, Languages, and Programming (ICALP 2019)*, volume 132 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 106:1–106:14. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2019. doi:10.4230/LIPIcs.ICALP.2019.106.

9 Michaël Cadilhac, Filip Mazowiecki, Charles Paperman, Michał Pilipczuk, and Géraud Sénizergues. On polynomial recursive sequences. In Artur Czumaj, Anuj Dawar, and Emanuela Merelli, editors, *47th International Colloquium on Automata, Languages, and Programming, ICALP 2020, July 8-11, 2020, Saarbrücken, Germany (Virtual Conference)*, volume 168 of *LIPIcs*, pages 117:1–117:17. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPIcs.ICALP.2020.117.

10 Christian Choffrut. Sequences of words defined by two-way transducers. *Theoretical Computer Science*, 658:85–96, 2017. doi:10.1016/j.tcs.2016.05.004.

11 Luc Dartois, Ismaël Jecker, and Pierre-Alain Reynier. Aperiodic String Transducers. *International Journal of Foundations of Computer Science*, 29(05):801–824, August 2018. doi:10.1142/S0129054118420054.

12 Gaëtan Douéneau-Tabot. Pebble transducers with unary output, 2021. arXiv:2104.14019.

13 Gaëtan Douéneau-Tabot, Emmanuel Filiot, and Paul Gastin. Register Transducers Are Marble Transducers. In Javier Esparza and Daniel Kráľ, editors, *45th International Symposium on Mathematical Foundations of Computer Science (MFCS 2020)*, volume 170 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 29:1–29:14, Dagstuhl, Germany, 2020. Schloss Dagstuhl–Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.MFCS.2020.29.

14 Joost Engelfriet. Two-way pebble transducers for partial functions and their composition. *Acta Informatica*, 52(7-8):559–571, 2015. doi:10.1007/s00236-015-0224-3.

15 Joost Engelfriet and Hendrik Jan Hoogeboom. MSO definable string transductions and two-way finite-state transducers. *ACM Transactions on Computational Logic*, 2(2):216–254, April 2001. doi:10.1145/371316.371512.

16 Joost Engelfriet, Hendrik Jan Hoogeboom, and Bart Samwel. XML navigation and transformation by tree-walking automata and transducers with visible and invisible pebbles. *Theoretical Computer Science*, 850:40–97, January 2021. doi:10.1016/j.tcs.2020.10.030.

17 Joost Engelfriet and Sebastian Maneth. Two-way finite state transducers with nested pebbles. In Krzysztof Diks and Wojciech Rytter, editors, *Mathematical Foundations of Computer Science 2002, 27th International Symposium, MFCS 2002, Warsaw, Poland, August 26-30, 2002, Proceedings*, volume 2420 of *Lecture Notes in Computer Science*, pages 234–244. Springer, 2002. doi:10.1007/3-540-45687-2_19.

18 Joost Engelfriet, Grzegorz Rozenberg, and Giora Slutzki. Tree transducers, L systems, and two-way machines. *Journal of Computer and System Sciences*, 20(2):150–202, 1980. doi:10.1016/0022-0000(80)90058-6.

19 Joost Engelfriet and Heiko Vogler. Macro tree transducers. *Journal of Computer and System Sciences*, 31(1):71–146, 1985. doi:10.1016/0022-0000(85)90066-2.

20 Julien Ferté, Nathalie Marin, and Géraud Sénizergues. Word-Mappings of Level 2. *Theory of Computing Systems*, 54(1):111–148, January 2014. doi:10.1007/s00224-013-9489-5.

21 Emmanuel Filiot and Pierre-Alain Reynier. Transducers, Logic and Algebra for Functions of Finite Words. *ACM SIGLOG News*, 3(3):4–19, August 2016. doi:10.1145/2984450.2984453.

22 Emmanuel Filiot and Pierre-Alain Reynier. Copyful streaming string transducers. *Fundamenta Informaticae*, 178(1-2):59–76, January 2021. doi:10.3233/FI-2021-1998.

23 Bruno Guillon. Input- or output-unary sweeping transducers are weaker than their 2-way counterparts. *RAIRO – Theoretical Informatics and Applications*, 50(4):275–294, 2016. doi:10.1051/ita/2016028.

24 Juha Honkala. A short solution for the HDT0L sequence equivalence problem. *Theoretical Computer Science*, 244(1-2):267–270, 2000. doi:10.1016/S0304-3975(00)00158-4.

25 Nathan Lhote. Pebble minimization of polyregular functions. In Holger Hermanns, Lijun Zhang, Naoki Kobayashi, and Dale Miller, editors, *LICS '20: 35th Annual ACM/IEEE Symposium on Logic in Computer Science, Saarbrücken, Germany, July 8-11, 2020*, pages 703–712. ACM, 2020. doi:10.1145/3373718.3394804.

26 Aristid Lindenmayer. Mathematical models for cellular interactions in development II. Simple and branching filaments with two-sided inputs. *Journal of Theoretical Biology*, 18(3):300–315, March 1968. doi:10.1016/0022-5193(68)90080-5.

27 Damiano Mazza. Simple Parsimonious Types and Logarithmic Space. In *24th EACSL Annual Conference on Computer Science Logic (CSL 2015)*, pages 24–40, 2015. doi:10.4230/LIPIcs.CSL.2015.24.

28 Tova Milo, Dan Suciu, and Victor Vianu. Typechecking for XML transformers. *Journal of Computer and System Sciences*, 66(1):66–97, 2003. Journal version of a PODS 2000 paper. doi:10.1016/S0022-0000(02)00030-2.

29 Anca Muscholl and Gabriele Puppis. The Many Facets of String Transducers. In Rolf Niedermeier and Christophe Paul, editors, *36th International Symposium on Theoretical Aspects of Computer Science (STACS 2019)*, volume 126 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 2:1–2:21. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2019. doi:10.4230/LIPIcs.STACS.2019.2.

30 Lê Thành Dũng Nguyễn, Camille Noûs, and Pierre Pradic. Implicit automata in typed λ-calculi II: streaming transducers vs categorical semantics, 2020. arXiv:2008.01050.

31 Lê Thành Dũng Nguyễn and Pierre Pradic. Implicit automata in typed λ-calculi I: aperiodicity in a non-commutative logic. In Artur Czumaj, Anuj Dawar, and Emanuela Merelli, editors, *47th International Colloquium on Automata, Languages, and Programming, ICALP 2020, July 8-11, 2020, Saarbrücken, Germany (Virtual Conference)*, volume 168 of *LIPIcs*, pages 135:1–135:20. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPIcs.ICALP.2020.135.

32 Christophe Reutenauer. Sur les séries associées à certains systèmes de Lindenmayer. *Theoretical Computer Science*, 9:363–375, 1979. doi:10.1016/0304-3975(79)90036-7.

33 Jacques Sakarovitch. *Elements of Automata Theory*. Cambridge University Press, 2009. Translated by Reuben Thomas. doi:10.1017/CBO9781139195218.

34 Janusz Schmude. On polynomial grammars extended with substitution, 2021. arXiv:2102.08705.

35 Tim Smith. A pumping lemma for two-way finite transducers. In Erzsébet Csuhaj-Varjú, Martin Dietzfelbinger, and Zoltán Ésik, editors, *Mathematical Foundations of Computer Science 2014 - 39th International Symposium, MFCS 2014, Budapest, Hungary, August 25-29, 2014. Proceedings, Part I*, volume 8634 of *Lecture Notes in Computer Science*, pages 523–534. Springer, 2014. doi:10.1007/978-3-662-44522-8_44.

36 Géraud Sénizergues. Sequences of level 1, 2, 3, ..., k, ... In Volker Diekert, Mikhail V. Volkov, and Andrei Voronkov, editors, *Computer Science - Theory and Applications, Second International Symposium on Computer Science in Russia, CSR 2007, Ekaterinburg, Russia, September 3-7, 2007, Proceedings*, volume 4649 of *Lecture Notes in Computer Science*, pages 24–32. Springer, 2007. doi:10.1007/978-3-540-74510-5_6.

Higher-Order Model Checking Step by Step

Paweł Parys
Institute of Informatics, University of Warsaw, Poland

Abstract

We show a new simple algorithm that solves the model-checking problem for recursion schemes: check whether the tree generated by a given higher-order recursion scheme is accepted by a given alternating parity automaton. The algorithm amounts to a procedure that transforms a recursion scheme of order n to a recursion scheme of order $n-1$, preserving acceptance, and increasing the size only exponentially. After repeating the procedure n times, we obtain a recursion scheme of order 0, for which the problem boils down to solving a finite parity game. Since the size grows exponentially at each step, the overall complexity is n-EXPTIME, which is known to be optimal. More precisely, the transformation is linear in the size of the recursion scheme, assuming that the arity of employed nonterminals and the size of the automaton are bounded by a constant; this results in an FPT algorithm for the model-checking problem.

Our transformation is a generalization of a previous transformation of the author (2020), working for reachability automata in place of parity automata. The step-by-step approach can be opposed to previous algorithms solving the considered problem "in one step", being compulsorily more complicated.

2012 ACM Subject Classification Theory of computation → Rewrite systems

Keywords and phrases Higher-order recursion schemes, Parity automata, Model-checking, Transformation, Order reduction

Digital Object Identifier 10.4230/LIPIcs.ICALP.2021.140

Category Track B: Automata, Logic, Semantics, and Theory of Programming

Related Version *Full Version*: http://arxiv.org/abs/2105.01861

Funding Work supported by the National Science Centre, Poland (grant no. 2016/22/E/ST6/00041).

Acknowledgements The author would like to thank the anonymous reviewers for their constructive comments.

1 Introduction

Recursion schemes are faithful and algorithmically manageable abstractions of the control flow of programs involving higher-order functions [19]. Such functions are nowadays widely used not only in functional programming languages such as Haskell and the OCAML family, but also in mainstream languages such as Java, JavaScript, Python, and C++. Simultaneously, the formalism of recursion schemes is equivalent via direct translations to simply-typed λY-calculus [28]. Collapsible pushdown systems [15] and ordered tree-pushdown systems [10] are other equivalent formalisms. Recursion schemes cover some other models such as indexed grammars [1] and ordered multi-pushdown automata [3].

The most celebrated algorithmic result in the analysis of recursion schemes is the decidability of the *model-checking problem* against regular properties of trees: given a recursion scheme $\mathcal{G}$ and a parity tree automaton $\mathcal{A}$, one can decide whether the tree generated by $\mathcal{G}$ is accepted by $\mathcal{A}$ [23]. This fundamental result has been reproved several times, that is, using collapsible higher-order pushdown automata [14], intersection types [20], Krivine machines [26], and it has been extended in diverse directions such as global model checking [7], logical reflection [5], effective selection [9], and a transfer theorem via models of lambda-

calculus [27]. The model-checking problem for recursion schemes of order n is complete for n-fold exponential time [23]. Despite this hardness result, the model-checking problem can be solved efficiently on multiple nontrivial examples, thanks to the development of several recursion-scheme model checkers [13, 21, 29] (including some model checkers that work only for automata models weaker than parity tree automata [17, 18, 6, 22, 25]).

In this paper, we give a new simple algorithm solving the model-checking problem for recursion schemes, mentioned above. The algorithm amounts to a procedure that transforms a recursion scheme of order n to a recursion scheme of order $n-1$, preserving acceptance, and increasing the size only exponentially. After repeating the procedure n times, we obtain a recursion scheme of order 0, for which acceptance boils down to winning a finite parity game. Since the size grows exponentially at each step, we reach the optimal overall complexity of n-fold exponential time. In a more detailed view, the complexity looks even better: the size growth is exponential only in the arity of types appearing in the recursion scheme, and in the size of the parity automaton; if these two parameters are bounded by a constant, the transformation is linear in the size of the recursion scheme. Since solving a finite parity game is FPT in the number of priorities [8], our algorithm for the the model-checking algorithm is FTP in the two parameters.[1]

The main difference between our algorithm and all the others is that we solve the problem step by step, repeatedly reducing the order by one, while most previous algorithms work "in one step", being compulsorily more complicated. The only algorithms that have been reducing the order by one, were algorithms using collapsible pushdown automata [14, 5, 9]. Notice, however, that these algorithms: first, are very technical; second, are contained only in unpublished appendices and in an arXiv paper [4]; third, if we want to use them for recursion schemes, it is necessary to employ a (nontrivial) translation from recursion schemes to collapsible pushdown automata [15, 28, 9]. A reduction of order was also possible for a subclass of recursion schemes, called *safe* recursion schemes [16], but it was not known how to extend it to all recursion schemes.

The transformation presented in this paper generalizes of a previous transformation of the author [24], working for reachability automata in place of parity automata. It has also a close relationship with a transformation given by Asada and Kobayashi [2].

2 Preliminaries

For a number $k \in \mathbb{N}$ we write $[k]$ for $\{1, \dots, k\}$. For any relation $\longrightarrow$ we write $\longrightarrow^*$ for the reflexive transitive closure of $\longrightarrow$.

For a function Z we write $Z[z \mapsto r]$ to denote the function that maps z to r while all other elements of the domain of Z are mapped as in Z. Likewise, we write $Z[z_i \mapsto r_i \mid i \in I]$ to denote the function that maps z_i to r_i for all $i \in I$, while all other elements of the domain of Z are mapped as in Z. We also use this notation without the "Z" part, for a function Z with empty domain.

Recursion schemes. The set of *(simple) types* is constructed from a unique ground type o using a binary operation $\to$; namely o is a type, and if α and β are types, so is $\alpha \to \beta$. By convention, $\to$ associates to the right, that is, $\alpha \to \beta \to \gamma$ is understood as $\alpha \to (\beta \to \gamma)$.

[1] This is not new. Actually, most previous algorithms reduce the model-checking problem to the problem of solving a parity game whose size is polynomial (for a polynomial of a fixed degree, for some algorithms just linear) in the size of the input, assuming that the arity of types appearing in the recursion scheme and the size of the parity automaton are fixed. Thus, only the method introduced by us is new, not the complexity results.

We often abbreviate $\underbrace{\alpha \to \cdots \to \alpha}_{\ell} \to \beta$ as $\alpha^{\ell} \to \beta$. The *order* of a type α, denoted $\mathsf{ord}(\alpha)$, is defined by induction: $\mathsf{ord}(\alpha_1 \to \cdots \to \alpha_k \to \mathsf{o}) = \max(\{0\} \cup \{\mathsf{ord}(\alpha_i) + 1 \mid i \in [k]\})$; for example $\mathsf{ord}(\mathsf{o}) = 0$, $\mathsf{ord}(\mathsf{o} \to \mathsf{o} \to \mathsf{o}) = 1$, and $\mathsf{ord}((\mathsf{o} \to \mathsf{o}) \to \mathsf{o}) = 2$.

Having a set of typed nonterminals $\mathcal{X}$, a set of typed variables $\mathcal{Y}$, and a set of symbols Σ, *terms* over $(\mathcal{X}, \mathcal{Y}, \Sigma)$ are defined by induction:

- nonterminal: every nonterminal $X \in \mathcal{X}$ of type α is a term of type α;
- variable: every variable $y \in \mathcal{Y}$ of type α is a term of type α;
- node constructor: if $K_1, \ldots, K_k$ are terms of type o and $a \in \Sigma$, then $\langle a, K_1, \ldots, K_k \rangle$ is a term of type o;
- application: if K is a term of type $\alpha \to \beta$, and L is a term of type α, then $K\,L$ is a term of type β.

The type of a term K is denoted $\mathsf{tp}(K)$. The order of a term K, written $\mathsf{ord}(K)$, is defined as the order of its type.

A *(higher-order) recursion scheme* is a tuple $\mathcal{G} = (\mathcal{X}, X_0, \Sigma, \mathcal{R})$, where $\mathcal{X}$ is a finite set of typed nonterminals, and $X_0 \in \mathcal{X}$ is a *starting nonterminal* of type o, and Σ is a finite set of symbols (called an *alphabet*), and $\mathcal{R}$ is a function assigning to every nonterminal $X \in \mathcal{X}$ a *rule* of the form $X\,y_1\,\ldots\,y_k \to R$, where $\mathsf{tp}(X) = (\mathsf{tp}(y_1) \to \cdots \to \mathsf{tp}(y_k) \to \mathsf{o})$, and R is a term of type o over $(\mathcal{X}, \{y_1, \ldots, y_k\}, \Sigma)$. The order of a recursion scheme, $\mathsf{ord}(\mathcal{G})$, is defined as the maximum of orders of its nonterminals.

Having a recursion scheme $\mathcal{G} = (\mathcal{X}, X_0, \Sigma, \mathcal{R})$, for every set of variables $\mathcal{Y}$ we define a *reduction relation* $\longrightarrow_{\mathcal{G}}$ between terms over $(\mathcal{X}, \mathcal{Y}, \Sigma)$ as the least relation such that

- $X\,K_1\,\ldots\,K_k \longrightarrow_{\mathcal{G}} R[K_1/y_1, \ldots, K_k/y_k]$ if the rule for X is $X\,y_1\,\ldots\,y_k \to R$, where $R[K_1/y_1, \ldots, K_k/y_k]$ denotes the term obtained from R by substituting K_i for y_i for all $i \in [k]$.

A (potentially infinite) *tree* over an alphabet Σ is defined by coinduction: every tree over Σ is of the form $\langle a, T_1, \ldots, T_k \rangle$, where $a \in \Sigma$ and $T_1, \ldots, T_k$ are again trees over Σ (for an introduction to coinductive definitions and proofs see, e.g., Czajka [12]). We employ the usual notions of nodes, children, branches, etc. Formally, we can define nodes as sequences of natural numbers; then for a tree $T = \langle a, T_1, \ldots, T_k \rangle$, the empty sequence $()$ is a node of T labeled by a, and any longer sequence $(i_1, i_2, \ldots, i_n)$ is a node of T labeled by b if $i_1 \in [k]$ and $(i_2, \ldots, i_n)$ is a node of T_{i_1} labeled by b. For a tree T and its node v, we write $T{\restriction_v}$ for the subtree of T starting at v.

Again by coinduction, we define the tree *generated* by a recursion scheme $\mathcal{G} = (\mathcal{X}, X_0, \Sigma, \mathcal{R})$ from a term M of type o (over $(\mathcal{X}, \varnothing, \Sigma)$), denoted $\mathsf{BT}_{\mathcal{G}}(M)$:

- if $M \longrightarrow_{\mathcal{G}}^{*} \langle a, K_1, \ldots, K_k \rangle$, then $\mathsf{BT}_{\mathcal{G}}(M) = \langle a, \mathsf{BT}_{\mathcal{G}}(K_1), \ldots, \mathsf{BT}_{\mathcal{G}}(K_k) \rangle$;
- otherwise, $\mathsf{BT}_{\mathcal{G}}(M) = \langle \omega \rangle$ for a special symbol $\omega \notin \Sigma$.

The tree generated by $\mathcal{G}$ (without mentioning a term), denoted $\mathsf{BT}(\mathcal{G})$, is defined as $\mathsf{BT}_{\mathcal{G}}(X_0)$.

Parity games. As already said, in the model-checking problem we are given a recursion scheme $\mathcal{G}$ and an alternating parity automaton $\mathcal{A}$, and we are asked whether the tree $T_{\mathcal{G}}$ generated by $\mathcal{G}$ is accepted by $\mathcal{A}$. One can, however, create a product of $\mathcal{G}$ and $\mathcal{A}$, which is a recursion scheme $\mathcal{G}_{\mathcal{A}}$ generating the tree of all possible runs of $\mathcal{A}$ on $T_{\mathcal{G}}$. This tree is a parity game; the game is won by Eve if and only if $\mathcal{A}$ accepts $T_{\mathcal{G}}$ (see Appendix A of the full version for more details). Due to this reduction, it is enough to work with recursion schemes generating parity games, and consider the problem of finding a winner in such games.

For every $d \in \mathbb{N}_+$ we consider the alphabet $\Sigma_d = \{\mathrm{Adam}, \mathrm{Eve}\} \times [d]$. A *parity tree* is a tree over Σ_d where every node has at least one child. A *parity recursion scheme* is a recursion scheme generating a parity tree (in particular the generated tree cannot have nodes

without children, including ω-labeled nodes). For a node labeled by $(\wp, p) \in \Sigma_d$, we say that it *belongs* to the player $\wp$, and that it has *priority* p. For trees and terms we write $\langle \wp, p, K_1, \ldots, K_k \rangle$ instead of $\langle (\wp, p), K_1, \ldots, K_k \rangle$, avoiding excessive brackets.

A branch ξ in a parity tree T is *won by Eve* (*Adam*) if the greatest priority appearing infinitely often on ξ is even (odd, respectively). A *strategy* ρ of a player $\wp \in \{\text{Adam}, \text{Eve}\}$ in a parity tree T is a function that assigns numbers to nodes of T belonging to the player $\wp$; if a node v has k children, we require that $\rho(v) \in [k]$. A branch ξ *agrees* with ρ if for every node v on ξ that belongs to $\wp$, the next node of ξ is the $\rho(v)$-th child of v. A strategy ρ of $\wp$ is *winning* if all branches that agree with ρ are winning for $\wp$. Finally, $\wp$ *wins* in T if $\wp$ has a winning strategy in T; otherwise $\wp$ *loses* in T. It is a standard result that in every parity tree T exactly one of the players wins.

It is useful to consider the following order $\preceq$ on positive natural numbers (priorities): $\cdots \preceq 5 \preceq 3 \preceq 1 \preceq 2 \preceq 4 \preceq 6 \preceq \ldots$ (first we have odd numbers in the reversed order, and then positive even numbers). We use the words *worse* and *better* to say that a priority is, respectively, earlier or later in this order. The intuition is that while playing a parity game, Eve always prefers to see better priorities.

3 Transformation

In this section we present a transformation, called *order-reducing transformation*, resulting in the main theorem of this paper:

▶ **Theorem 3.1.** *For any $n \geq 1$, there exists a transformation from order-n parity recursion schemes to order-$(n-1)$ parity recursion schemes, and a polynomial p_n such that, for any order-n parity recursion scheme $\mathcal{G}$, the winner in the tree generated by the resulting recursion scheme $\mathcal{G}^\dagger$ is the same as in the tree generated by $\mathcal{G}$, and $|\mathcal{G}^\dagger| \leq 2^{p_n(|\mathcal{G}|)}$.*

Intuitions. Let us first present intuitions behind our transformation. While reducing the order, we have to replace, in particular, order-1 functions by order-0 terms. Consider for example a tree T generated from a term $K\,L$ of type o, where K has type $\mathsf{o} \to \mathsf{o}$. Essentially, T consists of a context C_K, generated by K, where the tree T_L generated by L is inserted in some "holes". Instead of playing in T, we propose the following modification of the game. At the beginning, we ask Eve a question: how is she going to reach subtrees T_L while playing in T? She may declare that, according to her winning strategy,

- she is able to ensure that the greatest priority seen before reaching T_L will not be worse than r, for some number r of her choice, or
- she will not reach subtrees T_L at all, which amounts to choosing for r an even number greater than d, say $r = 2d$.

Then, we ask Adam if he believes in this declaration. If so, we simply read the declared worst-case priority r, and we continue playing in T_L (this possibility is unavailable for Adam, if Eve declared that she will not visit T_L). Otherwise, we check the declaration: we start playing in C_K; while reaching a place where T_L should be placed, Eve immediately wins (loses) if her declaration is fulfilled (not fulfilled, respectively).

We can see that such a modification of the game (even applied in infinitely many places of the considered tree) does not change the winner. A subtle point is that, in the modified game, Eve has to make a declaration on the priority r before actually starting the game in the tree generated from $K\,L$, and it is not completely obvious why the need for the declaration introduces no disadvantage for Eve. Nevertheless, for a fixed Eve's winning strategy, the worst greatest priority seen before reaching T_L is fixed, so that Eve can declare it as r.

In the transformation, we change the order-1 term K into several order-0 terms: K_r for $r \in \{1, \ldots, d, 2d\}$ (where d is a bound on priorities in the considered parity recursion scheme $\mathcal{G}$). These terms generate trees of the same shape as the context C_K generated by K but with some fixed trees substituted in place of the holes of C_K (where originally trees generated by the argument L were attached). The generated trees correspond to particular declarations made by Eve, as described above. Namely, we consider some fixed trees $\bot$ and $\top$ in which Eve loses and wins, respectively. Then, in the tree generated by K_r, the tree $\top$ is placed in holes such that the greatest priority on the path from the root to the hole is not worse than r, and the tree $\bot$ is placed in the remaining holes. In particular, the tree $\bot$ is placed in all holes of the tree generated by K_{2d}, because all priorities actually appearing in the tree are worse than $2d$. Finally, we replace $K\,L$ by $\langle \mathrm{Eve}, 1, K_1^L, K_2^L, \ldots, K_d^L, K_{2d}\rangle$, where $K_r^L = \langle \mathrm{Adam}, 1, K_r, \langle \mathrm{Eve}, r, L\rangle\rangle$. In this way we realize the modified game described above: first Eve chooses a declaration r and then Adam either proceed to K_r or to L after seeing priority r (the latter possibility is disabled for $r = 2d$). The priority 1 of the newly created tree nodes should be seen as a neutral priority; higher priorities visited later will be more important anyway.

When a term K of order 1 takes multiple arguments (instead of one argument L), we proceed in the same way, allowing Eve to make declarations for each of the arguments.

While applying the above-described transformation to recursion schemes, it is possible that the term K considered above contains some nonterminals or variables. Then, in order to realize the transformation, we need to create multiple copies of these nonterminals and variables, corresponding to particular declarations of Eve.

For example, say that in a recursion scheme we have (among others) the following two rules:

$$\mathsf{X} \to \mathsf{Y}\,\mathsf{Z},$$
$$\mathsf{Y}\,\mathsf{z} \to \langle \mathrm{Eve}, 1, \mathsf{z}, \langle \mathrm{Eve}, 2, \mathsf{z}\rangle\rangle.$$

Here X and Z are of type o, and Y is of type $\mathsf{o} \to \mathsf{o}$, so $\mathsf{Y}\,\mathsf{Z}$ is an application that should be replaced by the transformation. Assuming $d = 2$, we should obtain the following rules:

$$\mathsf{X}' \to \langle \mathrm{Eve}, 1, \langle \mathrm{Adam}, 1, \mathsf{Y}_1, \langle \mathrm{Eve}, 1, \mathsf{Z}'\rangle\rangle, \langle \mathrm{Adam}, 1, \mathsf{Y}_2, \langle \mathrm{Eve}, 2, \mathsf{Z}'\rangle\rangle, \mathsf{Y}_4\rangle,$$
$$\mathsf{Y}_1 \to \langle \mathrm{Eve}, 1, \overline{\top}, \langle \mathrm{Eve}, 2, \overline{\top}\rangle\rangle,$$
$$\mathsf{Y}_2 \to \langle \mathrm{Eve}, 1, \underline{\bot}, \langle \mathrm{Eve}, 2, \overline{\top}\rangle\rangle,$$
$$\mathsf{Y}_4 \to \langle \mathrm{Eve}, 1, \underline{\bot}, \langle \mathrm{Eve}, 2, \underline{\bot}\rangle\rangle,$$

where $\underline{\bot}$ and $\overline{\top}$ are nonterminals from which the trees $\bot$ and $\top$ (in which Eve loses and wins, respectively) are generated.

Another possibility is that in the original recursion scheme we have $\mathsf{y}\,\mathsf{Z}$ instead of $\mathsf{Y}\,\mathsf{Z}$:

$$\mathsf{S} \to \mathsf{T}\,\mathsf{Y},$$
$$\mathsf{T}\,\mathsf{y} \to \mathsf{y}\,\mathsf{Z}.$$

Then, the single parameter y gets transformed into three parameters:

$$\mathsf{S}' \to \mathsf{T}'\,\mathsf{Y}_1\,\mathsf{Y}_2\,\mathsf{Y}_4,$$
$$\mathsf{T}'\,\mathsf{y}_1\,\mathsf{y}_2\,\mathsf{y}_4 \to \langle \mathrm{Eve}, 1, \langle \mathrm{Adam}, 1, \mathsf{y}_1, \langle \mathrm{Eve}, 1, \mathsf{Z}'\rangle\rangle, \langle \mathrm{Adam}, 1, \mathsf{y}_2, \langle \mathrm{Eve}, 2, \mathsf{Z}'\rangle\rangle, \mathsf{y}_4\rangle.$$

Formal definition. We now formalize the above intuitions. Fix a parity recursion scheme $\mathcal{G} = (\mathcal{X}, X_0, \Sigma_d, \mathcal{R})$; in particular fix a bound d on priorities appearing in $\mathcal{G}$.

A set D_d of Eve's *declarations* is defined as $D_d = \{1, \dots, d, 2d\}$. For a priority $p \in [d]$ and a declaration $r \in D_d$ we define a *shifted* declaration $r{\restriction_p}$ (obtained from r after seeing priority p):

$$r{\restriction_p} = \begin{cases} p+1 & \text{if } p \text{ is odd and } p > r, \\ p-1 & \text{if } p \text{ is even and } p \geq r, \\ r & \text{otherwise.} \end{cases}$$

We remark that the same definition appears in Tsukada and Ong [30] (where shifts are called left-residuals); a slightly different representation is present also in Salvati and Walukiewicz [26] (with declarations called residuals and shifts called liftings).

The *leader* ("most important priority") of a sequence of priorities π is the greatest priority appearing in π, or 1 if π is empty. A sequence of priorities π *fulfils* a declaration $r \in D_d$ if r is worse or equal than the leader of π (where "worse" refers to the $\preceq$ order defined in Section 2). For example, $1, 4, 2$, and $1, 1, 1$, both fulfil 3, but $1, 5, 4$ does not. The empty sequence fulfils r exactly when r is odd. No sequence of priorities from $[d]$ fulfils $2d$. The following lemma is obtained by a direct analysis (see Appendix B of the full version):

▶ **Lemma 3.2.** *A sequence of priorities $p_1, p_2, \dots, p_k \in [d]$ fulfils a declaration $r \in D_d$ if and only if $p_2, \dots, p_k$ fulfils $r{\restriction_{p_1}}$.*

Having a type, we are interested in cutting off its suffix of order 1. Thus, we use the notation $\alpha_1 \to \cdots \to \alpha_k \Rightarrow \mathsf{o}^\ell \to \mathsf{o}$ for a type $\alpha_1 \to \cdots \to \alpha_k \to \mathsf{o}^\ell \to \mathsf{o}$ such that either $k = 0$ or $\alpha_k \neq \mathsf{o}$. Notice that every type α can be uniquely represented in this form. We remark that some among the types $\alpha_1, \dots, \alpha_{k-1}$ (but not α_k) may be o. For a type α we write $\mathsf{gar}(\alpha)$ ("ground arity") for the number ℓ for which we can write $\alpha = (\alpha_1 \to \cdots \to \alpha_k \Rightarrow \mathsf{o}^\ell \to \mathsf{o})$; we also extend this to terms: $\mathsf{gar}(M) = \mathsf{gar}(\mathsf{tp}(M))$.

We transform terms of type α to terms of type $\alpha^{\dagger_d}$, which is defined by induction:

$$(\alpha_1 \to \cdots \to \alpha_k \Rightarrow \mathsf{o}^\ell \to \mathsf{o})^{\dagger_d} = \left((\alpha_1^{\dagger_d})^{|D_d|^{\mathsf{gar}(\alpha_1)}} \to \cdots \to (\alpha_k^{\dagger_d})^{|D_d|^{\mathsf{gar}(\alpha_k)}} \to \mathsf{o} \right).$$

Thus, we remove all trailing order-0 arguments, and we multiplicate (and recursively transform) remaining arguments. The number of copies depends on the bound d on priorities appearing in the considered parity recursion scheme.

For a finite set S, we write D_d^S for the set of functions $A \colon S \to D_d$. Moreover, we assume some fixed order on functions in D_d^S, and we write $P\,(Q_A)_{A \in D_d^S}$ for an application $P\,Q_{A_1} \dots Q_{A_{|D_d|^{|S|}}}$, where $A_1, \dots, A_{|D_d|^{|S|}}$ are all the functions from D_d^S listed in the fixed order. The only function in $D_d^\varnothing$ is denoted $\varnothing$.

For every variable y and for every function $A \in D_d^{[\mathsf{gar}(y)]}$ we consider a variable $y_A^{\dagger_d}$ of type $(\mathsf{tp}(y))^{\dagger_d}$. Likewise, for every nonterminal X of $\mathcal{G}$ and for every function $A \in D_d^{[\mathsf{gar}(X)]}$ we consider a nonterminal $X_A^{\dagger_d}$ of type $(\mathsf{tp}(X))^{\dagger_d}$. As the new set of nonterminals we take $\mathcal{X}^{\dagger_d} = \{X_A^{\dagger_d} \mid X \in \mathcal{X}, A \in D_d^{[\mathsf{gar}(X)]}\} \cup \{\underline{\bot}, \overline{\top}\}$.

We now define a function tr_d transforming terms. Its value $\mathsf{tr}_d(A, Z, M)$ is defined when M is a term over $(\mathcal{X}, \mathcal{Y}, \Sigma_d)$ for some set of variables $\mathcal{Y}$, and $A \in D_d^{[\mathsf{gar}(M)]}$, and $Z \colon \mathcal{Y} \rightharpoonup D_d$ is a partial function such that $\mathrm{dom}(Z)$ contains only variables of type o. The intention is that A specifies Eve's declarations for trailing order-0 arguments, and Z specifies them for order-0 variables (among those in $\mathrm{dom}(Z)$). The transformation is defined by induction on the structure of M, as follows:

(1) $\mathsf{tr}_d(A, Z, X) = X_A^{\dagger_d}$ for $X \in \mathcal{X}$;
(2) $\mathsf{tr}_d(A, Z, y) = y_A^{\dagger_d}$ for $y \in \mathcal{Y} \setminus \mathrm{dom}(Z)$;
(3) $\mathsf{tr}_d(\varnothing, Z, z) = \overline{\top}$ if $Z(z)$ is odd;
(4) $\mathsf{tr}_d(\varnothing, Z, z) = \underline{\bot}$ if $Z(z)$ is even;
(5) $\mathsf{tr}_d(\varnothing, Z, \langle \wp, p, K_1, \dots, K_k\rangle) = \langle \wp, p, \mathsf{tr}_d(\varnothing, Z{\restriction_p}, K_1), \dots, \mathsf{tr}_d(\varnothing, Z{\restriction_p}, K_k)\rangle$, where $Z{\restriction_p}$ is the function defined by $Z{\restriction_p}(z) = (Z(z)){\restriction_p}$ for all $z \in \mathrm{dom}(Z)$;
(6) $\mathsf{tr}_d(A, Z, K\,L) = \langle \mathrm{Eve}, 1, K_1^L, K_2^L, \dots, K_d^L, K_{2d}\rangle$ if $\mathsf{tp}(K) = (\mathsf{o}^{\ell+1} \to \mathsf{o})$, where $K_r^L = \langle \mathrm{Adam}, 1, K_r, \langle \mathrm{Eve}, r, \mathsf{tr}_d(\varnothing, Z{\restriction_r}, L)\rangle\rangle$ for $r \in [d]$ and $K_r = \mathsf{tr}_d(A[\ell+1 \mapsto r], Z, K)$ for $r \in D_d$;
(7) $\mathsf{tr}_d(A, Z, K\,L) = (\mathsf{tr}_d(A, Z, K))\,(\mathsf{tr}_d(B, Z, L))_{B \in D_d^{[\mathsf{gar}(L)]}}$ if $\mathsf{tp}(K) = (\alpha_1 \to \cdots \to \alpha_k \Rightarrow \mathsf{o}^{\ell} \to \mathsf{o})$ with $k \geq 1$.

In Cases (3), (4), and (5) the term is of type o, so the "A" argument is necessarily $\varnothing$ (a function with an empty domain).

For every rule $X\,y_1 \dots y_k\,z_1 \dots z_\ell \to R$ in $\mathcal{R}$, where $\ell = \mathsf{gar}(X)$, and for every function $A \in D_d^{[\ell]}$, to $\mathcal{R}^{\dagger_d}$ we take the rule

$$X_A^{\dagger_d}\,(y_{1,B}^{\dagger_d})_{B \in D_d^{[\mathsf{gar}(y_1)]}} \dots (y_{k,B}^{\dagger_d})_{B \in D_d^{[\mathsf{gar}(y_k)]}} \to \mathsf{tr}_d(\varnothing, [z_i \mapsto A(\ell+1-i) \mid i \in [\ell]], R).$$

In the function A it is more convenient to count arguments from right to left (then we do not need to shift the domain in Case (6) above), but it is more natural to have variables $z_1, \dots, z_\ell$ numbered from left to right; this is why in the rule for $X_A^{\dagger_d}$ we assign to z_i the value $A(\ell+1-i)$, not $A(i)$. Additionally, in $\mathcal{R}^{\dagger_d}$ we have rules $\underline{\bot} \to \langle \mathrm{Eve}, 1, \underline{\bot}\rangle$ and $\overline{\top} \to \langle \mathrm{Eve}, 2, \overline{\top}\rangle$. Then Eve loses (wins) in the tree $\bot$ ($\top$) generated by $\mathcal{G}^\dagger$ from $\underline{\bot}$ ($\overline{\top}$, respectively).

Finally, the resulting recursion scheme $\mathcal{G}^\dagger$ is $(\mathcal{X}^{\dagger_d}, X_{0,\varnothing}^{\dagger_d}, \Sigma_d, \mathcal{R}^{\dagger_d})$. This finishes the definition of the transformation. In the next section we analyze its complexity, and in Section 5 we justify its correctness.

▶ Remark 3.3. Let us briefly compare our transformation with a transformation by Broadbent et al. [4] reducing the order of a collapsible pushdown automaton by one while preserving the winner of the generated parity game. Although their transformation seems technically more complicated, its overall idea is quite similar to what we do in this paper. Their transformation is split into three independent steps. First, they make the automaton "rank-aware", which means that it knows what was the highest priority visited between creation of a collapse link and its usage. This corresponds to adding the parameters A and Z to our transformation, so that we know whether a declaration is fulfilled when a variable z is used. Second, they eliminate collapse links of order n, which in our case corresponds to removing trailing arguments of order 0 and introducing the gadget asking Eve for a declaration. Third, they reduce the order of the automaton by one, which we also do for recursion schemes.

4 Complexity

In this section we analyze complexity of our transformation. First, we formally define the *size* of a recursion scheme. The size of a term is defined by induction on its structure:

$$|X| = |y| = 1, \qquad |K\,L| = 1 + |K| + |L|,$$
$$|\langle a, K_1, \dots, K_k\rangle| = 1 + |K_1| + \cdots + |K_k|.$$

Then $|\mathcal{G}|$, the size of $\mathcal{G}$, is defined as the sum of $|R| + k$ over all rules $X\,y_1 \dots y_k \to R$ of $\mathcal{G}$. In Asada and Kobayashi [2] such a size is called *Curry-style* size; it does not include sizes of types of employed variables.

We say that a type α *appears in the definition* of a type β if either $\alpha = \beta$, or $\beta = (\beta_1 \to \beta_2)$ and α appears in the definition of β_1 or of β_2. We write $A_{\mathcal{G}}$ for the largest arity of types appearing in the definition of types of nonterminals in a recursion scheme $\mathcal{G}$. Notice that types of other objects used in $\mathcal{G}$, namely variables and subterms of right-hand sides of rules, appear in the definition of types of nonterminals, hence their arity is also bounded by $A_{\mathcal{G}}$. It is reasonable to consider large recursion schemes, consisting of many rules, where simultaneously the maximal arity $A_{\mathcal{G}}$ is respectively small.

While the exponential bound mentioned in Theorem 3.1 is obtained by applying the order-reducing transformation to an arbitrary parity recursion scheme, the complexity becomes slightly better if we first apply a preprocessing step. This is in particular necessary, if we want to obtain linear dependence in the size of $\mathcal{G}$ (and exponential only in the maximal arity $A_{\mathcal{G}}$). The preprocessing, making sure that the recursion scheme is in a *simple form* (defined below), amounts to splitting large rules into multiple smaller rules. A similar preprocessing is present already in prior work [19, 2, 11, 24].

An *application depth* of a term R is defined as the maximal number of applications on a single branch in R, where a compound application $K\,L_1\,\ldots\,L_k$ counts only once. More formally, we define by induction:

$$\mathsf{ad}(\langle a, K_1, \ldots, K_k\rangle) = \max\{\mathsf{ad}(K_i) \mid i \in [k]\},$$
$$\mathsf{ad}(X\,K_1\,\ldots\,K_k) = \mathsf{ad}(y\,K_1\,\ldots\,K_k) = \max(\{0\} \cup \{\mathsf{ad}(K_i) + 1 \mid i \in [k]\}).$$

We say that a recursion scheme $\mathcal{G}$ is in a *simple form* if the right-hand side of each its rule has application depth at most 2. We have the following:

▶ **Lemma 4.1** ([24, Lemma 4.1])**.** *For every recursion scheme $\mathcal{G}$ there exists a recursion scheme $\mathcal{G}'$ being in a simple form, generating the same tree as $\mathcal{G}$, and such that $\mathsf{ord}(\mathcal{G}') = \mathsf{ord}(\mathcal{G})$, and $A_{\mathcal{G}'} \le 2A_{\mathcal{G}}$, and $|\mathcal{G}'| = \mathcal{O}(A_{\mathcal{G}} \cdot |\mathcal{G}|)$. The recursion scheme $\mathcal{G}'$ can be created in time linear in its size.*

We now state and prove the main lemma of this section:

▶ **Lemma 4.2.** *For every parity recursion scheme $\mathcal{G} = (\mathcal{X}, X_0, \Sigma_d, \mathcal{R})$ in a simple form, the recursion scheme $\mathcal{G}^\dagger$ (i.e., the result of the order-reducing transformation) is also in a simple form, and $\mathsf{ord}(\mathcal{G}^\dagger) = \max(0, \mathsf{ord}(\mathcal{G}) - 1)$, and $A_{\mathcal{G}^\dagger} \le A_{\mathcal{G}} \cdot (d+1)^{A_{\mathcal{G}}}$, and $|\mathcal{G}^\dagger| = \mathcal{O}(|\mathcal{G}| \cdot (d+1)^{5 \cdot A_{\mathcal{G}}})$. Moreover, $\mathcal{G}^\dagger$ can be created in time linear in its size.*

Proof. The part about the running time is obvious. It is also easy to see by induction that $\mathsf{ord}(\alpha^{\dagger_d}) = \max(0, \mathsf{ord}(\alpha) - 1)$. It follows that the order of the recursion scheme satisfies the same equality, because nonterminals of $\mathcal{G}^\dagger$ have type $\alpha^{\dagger_d}$ for α being the type of a corresponding nonterminal of $\mathcal{G}$.

Recall that in the type $\alpha^{\dagger_d}$ obtained from $\alpha = (\alpha_1 \to \cdots \to \alpha_k \to \mathsf{o})$, every α_i either disappears or becomes (transformed and) repeated $|D_d|^{\mathsf{gar}(\alpha_i)}$ times, that is, at most $(d+1)^{A_{\mathcal{G}}}$ times. This implies the inequality concerning $A_{\mathcal{G}^\dagger}$.

Every compound application can be written as $f\,K_1\,\ldots\,K_k\,L_1\,\ldots\,L_\ell$, where f is a nonterminal or a variable, and $\ell = \mathsf{gar}(f)$. In such a term, every K_i (after being transformed) gets repeated $|D_d|^{\mathsf{gar}(K_i)}$ times, that is, at most $(d+1)^{A_{\mathcal{G}}}$ times. Then, for every L_i we replicate the outcome $d+1$ times, and we append a small prefix; this replication happens ℓ times (and $\ell \le A_{\mathcal{G}}$). In consequence, we easily see by induction that while transforming a term of application depth c, its size gets multiplicated by at most $O((d+1)^{2c \cdot A_{\mathcal{G}}})$. Moreover, every nonterminal X is repeated $|D_d|^{\mathsf{gar}(X)}$ times, that is, at most $(d+1)^{A_{\mathcal{G}}}$ times. Because the application depth of right-hand sides of rules is at most 2, this bounds the size of the new recursion scheme by $\mathcal{O}(|\mathcal{G}| \cdot (d+1)^{5 \cdot A_{\mathcal{G}}})$.

Looking again at the above description of the transformation, we can notice that the application depth cannot grow; in consequence the property of being in a simple form is preserved. ◀

Thus, if we want to check whether Eve wins in the tree generated by a parity recursion scheme $\mathcal{G}$ of order n, we can first convert $\mathcal{G}$ to a simple form, and then apply the order-reducing transformation n times. This gives us a parity recursion scheme of order 0, which can be seen as a finite parity game with d priorities. Such a game can be solved in time $O(N^4 \cdot 2^d)$, where N is its size [8]. Thus, by Lemmata 4.1 and 4.2, the whole algorithm works in time n-fold exponential in $A_{\mathcal{G}}$ and d, and polynomial (quartic) in $|\mathcal{G}|$.

If $\mathcal{G}$ is created as a product of a recursion scheme $\mathcal{H}$ and an alternating parity automaton $\mathcal{A}$, the running time is n-fold exponential in $A_{\mathcal{H}}$ and $|\mathcal{A}|$, and quartic in $|\mathcal{H}|$ (cf. Appendix A of the full version).

5 Correctness

In this section we finish a proof of Theorem 3.1 by showing that the winner in the tree generated by the recursion scheme $\mathcal{G}^\dagger$ resulting from transforming a recursion scheme $\mathcal{G}$ is the same as in the tree generated by the original recursion scheme $\mathcal{G}$. Our proof consists of three parts. First, we show that reductions performed by $\mathcal{G}$ can be reordered, so that we can postpone substituting for (trailing) variables of order 0. To store such postponed substitutions, called *explicit substitutions*, we introduce *extended trees*. Second, we show that such reordered reductions in $\mathcal{G}$ are in a direct correspondence with reductions in $\mathcal{G}^\dagger$. Finally, we show how winning strategies of particular players from the tree generated by $\mathcal{G}^\dagger$ can be transferred to the tree generated by $\mathcal{G}$.

Extended trees and terms. In the sequel, trees and terms defined previously are sometimes called non-extended trees and non-extended terms, in order to distinguish them from extended trees and extended terms defined below. Having a set $\mathcal{Z}$ of variables of type o and a set of symbols Σ, (potentially infinite) *extended trees* over $(\mathcal{Z}, \Sigma)$ are defined by coinduction: every extended tree over $(\mathcal{Z}, \Sigma)$ is of the form either

- $\langle a, T_1, \dots, T_k \rangle$, where $a \in \Sigma$ and $T_1, \dots, T_k$ are again extended trees over Σ, or
- z for some variable $z \in \mathcal{Z}$, or
- $T(\!|U/z|\!)$, where $z \notin \mathcal{Z}$ is a variable of type o, and T is an extended tree over $(\mathcal{Z} \cup \{z\}, \Sigma)$, and U is an extended tree over $(\mathcal{Z}, \Sigma)$.

The construction of the form $T(\!|U/z|\!)$ is called an *explicit substitution*. Intuitively, it denotes the tree obtained by substituting U for z in T. Notice that the variable z being free in T becomes bound in $T(\!|U/z|\!)$.

Likewise, having a set of typed nonterminals $\mathcal{X}$, a set $\mathcal{Z}$ of variables of type o, and a set of symbols Σ, *extended terms* over $(\mathcal{X}, \mathcal{Z}, \Sigma)$ are defined by induction:

- if $z \notin \mathcal{Z}$ is a variable of type o, and E is an extended term over $(\mathcal{X}, \mathcal{Z} \cup \{z\}, \Sigma)$, and L is a non-extended term of type o over $(\mathcal{X}, \mathcal{Z}, \Sigma)$, then $E(\!|L/z|\!)$ is an extended term over $(\mathcal{X}, \mathcal{Z}, \Sigma)$;
- every non-extended term of type o over $(\mathcal{X}, \mathcal{Z}, \Sigma)$ is an extended term over $(\mathcal{X}, \mathcal{Z}, \Sigma)$.

Notice that explicit substitutions can be placed anywhere inside an extended tree, while in an extended term they are allowed only to surround a non-extended term.

Of course an extended tree over $(\mathcal{Z}, \Sigma)$ can be also seen as an extended tree over $(\mathcal{Z}', \Sigma)$, where $\mathcal{Z}' \supseteq \mathcal{Z}$; likewise for extended terms. In the sequel, such extending of the set of variables is often performed implicitly.

Having a recursion scheme $\mathcal{G} = (\mathcal{X}, X_0, \Sigma, \mathcal{R})$, for every set $\mathcal{Z}$ of variables of type o we define an *ext-reduction* relation $\rightsquigarrow_{\mathcal{G}}$ between extended terms over $(\mathcal{X}, \mathcal{Z}, \Sigma)$, as the least relation such that

- $X\,K_1\,\ldots\,K_k\,L_1\,\ldots\,L_\ell \;\rightsquigarrow_{\mathcal{G}}\; R[K_1/y_1, \ldots, K_k/y_k, z'_1/z_1, \ldots, z'_\ell/z_\ell](\!|L_1/z'_1|\!)\ldots(\!|L_\ell/z'_\ell|\!)$ if $\ell = \mathsf{gar}(X)$, and $\mathcal{R}(X) = (X\,y_1\,\ldots\,y_k\,z_1\,\ldots\,z_\ell \to R)$, and $z'_1, \ldots, z'_\ell$ are fresh variables of type o not appearing in $\mathcal{Z}$.

Then, we define by coinduction the extended tree (over $(\mathcal{Z}, \Sigma)$) *ext-generated* by $\mathcal{G}$ from an extended term E (over $(\mathcal{X}, \mathcal{Z}, \Sigma)$), denoted $\mathsf{BT}^{\mathsf{ext}}_{\mathcal{G}}(E)$:

- if $E \rightsquigarrow^*_{\mathcal{G}} \langle a, F_1, \ldots, F_k\rangle$, then $\mathsf{BT}^{\mathsf{ext}}_{\mathcal{G}}(E) = \langle a, \mathsf{BT}^{\mathsf{ext}}_{\mathcal{G}}(F_1), \ldots, \mathsf{BT}^{\mathsf{ext}}_{\mathcal{G}}(F_k)\rangle$;
- if $E \rightsquigarrow^*_{\mathcal{G}} F(\!|L/z|\!)$, then $\mathsf{BT}^{\mathsf{ext}}_{\mathcal{G}}(E) = \mathsf{BT}^{\mathsf{ext}}_{\mathcal{G}}(F)(\!|\mathsf{BT}^{\mathsf{ext}}_{\mathcal{G}}(L)/z|\!)$;
- otherwise, $\mathsf{BT}^{\mathsf{ext}}_{\mathcal{G}}(E) = \langle\omega\rangle$.

The extended tree ext-generated by $\mathcal{G}$ (without mentioning a term), denoted $\mathsf{BT}^{\mathsf{ext}}(\mathcal{G})$, is defined as $\mathsf{BT}^{\mathsf{ext}}_{\mathcal{G}}(X_0)$. Formally, the ext-generated extended tree is not unique, because arbitrary fresh names may be used for bound variables; we should thus identify extended trees differing only in names of bound variables.

Finally, we say how to convert extended trees to trees, by performing all postponed substitutions. To this end, having fixed a set Σ of symbols, we define a *simplification* relation $\mapsto$ between extended trees over $(\varnothing, \Sigma)$ as the least relation such that

- $\langle a, T_1, \ldots, T_k\rangle(\!|L_1/z_1|\!)\ldots(\!|L_\ell/z_\ell|\!) \mapsto \langle a, T_1(\!|L_1/z_1|\!)\ldots(\!|L_\ell/z_\ell|\!), \ldots, T_k(\!|L_1/z_1|\!)\ldots(\!|L_\ell/z_\ell|\!)\rangle$, and
- $z_i(\!|L_1/z_1|\!)\ldots(\!|L_\ell/z_\ell|\!) \mapsto L_i(\!|L_{i+1}/z_{i+1}|\!)\ldots(\!|L_\ell/z_\ell|\!)$.

Then, we define by coinduction the *expansion* of an extended tree T over $(\varnothing, \Sigma)$, being a tree over Σ, and denoted $\mathsf{BT}^{\mathsf{s}}(T)$:

- if $T \mapsto^* \langle a, T_1, \ldots, T_k\rangle$, then $\mathsf{BT}^{\mathsf{s}}(T) = \langle a, \mathsf{BT}^{\mathsf{s}}(T_1), \ldots, \mathsf{BT}^{\mathsf{s}}(T_k)\rangle$;
- otherwise, $\mathsf{BT}^{\mathsf{s}}(T) = \langle\omega\rangle$.

The following lemma says that instead of generating a tree, we can first ext-generate an extended tree, and then expand all the explicit substitutions:

▶ **Lemma 5.1.** *For every recursion scheme $\mathcal{G}$ it holds that* $\mathsf{BT}(\mathcal{G}) = \mathsf{BT}^{\mathsf{s}}(\mathsf{BT}^{\mathsf{ext}}(\mathcal{G}))$.

The lemma can be proved in a standard way; a proof is contained in Appendix C of the full version (similar lemmata appear in previous work [2, Lemma 18], [24, Lemma 5.1]).

Transforming extended parity trees. An *extended parity tree* is an extended tree whose expansion is a parity tree. We now show how the transformation, defined previously for terms, can be applied to extended parity trees. Namely, we define $\mathsf{tr}^{\mathsf{t}}_d(Z, T)$ when T is an extended tree over $(\mathcal{Z}, \Sigma_d)$ for some set $\mathcal{Z}$ of variables of type o, and $Z\colon \mathcal{Z} \to D_d$ (we do not need an "A" argument, used previously to store declarations for arguments, because extended trees have no arguments). The definition is by coinduction:

(3') $\mathsf{tr}^{\mathsf{t}}_d(Z, z) = \top$ if $Z(z)$ is odd;
(4') $\mathsf{tr}^{\mathsf{t}}_d(Z, z) = \bot$ if $Z(z)$ is even;
(5') $\mathsf{tr}^{\mathsf{t}}_d(Z, \langle \wp, p, K_1, \ldots, K_k\rangle) = \langle \wp, p, \mathsf{tr}^{\mathsf{t}}_d(Z\restriction_p, K_1), \ldots, \mathsf{tr}^{\mathsf{t}}_d(Z\restriction_p, K_k)\rangle$;
(8') $\mathsf{tr}^{\mathsf{t}}_d(Z, T(\!|U/z|\!)) = \langle \mathrm{Eve}, 1, T_1^U, T_2^U, \ldots, T_d^U, T_{2d}\rangle$, where we take $T_r^U = \langle \mathrm{Adam}, 1, T_r, \langle \mathrm{Eve}, r, \mathsf{tr}^{\mathsf{t}}_d(Z\restriction_r, U)\rangle\rangle$ for $r \in [d]$ and $T_r = \mathsf{tr}^{\mathsf{t}}_d(Z[z \mapsto r], T)$ for $r \in D_d$.

Notice that tr_d transforms a term z to nonterminals $\overline{\top}$ or $\underline{\bot}$, while $\mathsf{tr}^{\mathsf{t}}_d$ transforms an extended tree z to trees $\top$ or $\bot$, generated from those nonterminals.

In the next lemma we observe that the tree generated by the transformed recursion scheme $\mathcal{G}^\dagger$ can be obtained by transforming the extended tree ext-generated by the original recursion scheme $\mathcal{G}$:

▶ **Lemma 5.2.** *For every parity recursion scheme $\mathcal{G}$ it holds that* $\mathsf{tr}^{\mathsf{t}}_d(\varnothing, \mathsf{BT}^{\mathsf{ext}}(\mathcal{G})) = \mathsf{BT}(\mathcal{G}^\dagger)$.

The proof is purely syntactical, and is contained in Appendix D of the full version.

Transforming strategies. We finish our correctness proof by showing the following lemma:

▶ **Lemma 5.3.** *Let T be an extended parity tree over $(\varnothing, \Sigma_d)$. If a player $\wp \in \{\text{Adam}, \text{Eve}\}$ wins in $\mathsf{tr}^{\mathsf{t}}_d(\varnothing, T)$, then $\wp$ wins also in $\mathsf{BT}^{\mathsf{s}}(T)$.*

Recall that the goal of this section is to prove that the winner in $\mathsf{BT}(\mathcal{G}^\dagger)$ is the same as in $\mathsf{BT}(\mathcal{G})$, for every parity recursion scheme $\mathcal{G}$. This follows from the above lemma used for $T = \mathsf{BT}^{\mathsf{ext}}(\mathcal{G})$, because $\mathsf{BT}(\mathcal{G}^\dagger) = \mathsf{tr}^{\mathsf{t}}_d(\varnothing, \mathsf{BT}^{\mathsf{ext}}(\mathcal{G}))$ by Lemma 5.2 and $\mathsf{BT}(\mathcal{G}) = \mathsf{BT}^{\mathsf{s}}(\mathsf{BT}^{\mathsf{ext}}(\mathcal{G}))$ by Lemma 5.1.

We now come to a proof of Lemma 5.3. In the sequel we assume a fixed extended parity tree T over $(\varnothing, \Sigma_d)$. Suppose first that it is Eve who wins in $\mathsf{tr}^{\mathsf{t}}_d(\varnothing, T)$; thus, we also fix her winning strategy ρ in this tree. Our goal is to construct Eve's winning strategy ρ' in $\mathsf{BT}^{\mathsf{s}}(T)$.

In the proof, we use two additional notions. First, we say that a sequence of priorities $r_1, \dots, r_k$ is a $\preceq$-*contraction* of a sequence of priorities $p_1, \dots, p_n$ if the latter can be split at some indices $i_0, i_1, \dots, i_k$, where $0 = i_0 \le i_1 \le \cdots \le i_k = n$, so that for every $j \in [k]$ the infix $p_{i_{j-1}+1}, p_{i_{j-1}+2}, \dots, p_{i_j}$ fulfils declaration r_j. Likewise we define $\preceq$-contractions for infinite sequences, only there are infinitely many splitting indices (which necessarily tend to infinity, meaning that the whole infinite sequence is split).

Notice that we allow empty infixes, so one can arbitrarily insert odd numbers r_j (i.e., numbers r_j fulfilled by the empty sequence) to the $\preceq$-contraction. For example, $3, 4, 2$ is a $\preceq$-contraction of $4, 3, 2, 3, 4$ because the empty sequence fulfils 3, and $4, 3$ fulfils 4, and $2, 3, 4$ fulfils 2. On the other hand, $3, 4, 2$ is not a $\preceq$-contraction of $4, 3, 2, 3$. The idea of $\preceq$-contractions is to describe what happens when we move from $\mathsf{BT}^{\mathsf{s}}(T)$ to $\mathsf{tr}^{\mathsf{t}}_d(\varnothing, T)$. Indeed, if T has a subtree of the form $U(\!|V/z|\!)$, then in $\mathsf{tr}^{\mathsf{t}}_d(\varnothing, T)$ the play can continue to V after playing only an Eve's declaration r (skipping completely U), while in $\mathsf{BT}^{\mathsf{s}}(T)$ before reaching V we traverse through U, where visited priorities are intended to fulfil r.

It is easy to see that $\preceq$-contractions are transitive, and that they can make the situation only worse for Eve:

▶ **Lemma 5.4.** *If a sequence π_1 is a $\preceq$-contraction of a sequence π_2, which is in turn a $\preceq$-contraction of a sequence π_3, then π_1 is a $\preceq$-contraction of π_3.*

▶ **Lemma 5.5.** *If an infinite sequence π_1 is a $\preceq$-contraction of an infinite sequence π_2, and the greatest priority appearing infinitely often in π_1 is even, then the greatest priority appearing infinitely often in π_2 is even as well.*

We now introduce the second notion (it concerns only finite sequences, and is relative to the bound d on priorities): for a declaration $r \in D_d$ and two sequences π_1, π_2 of priorities from $[d]$ we say that π_1 is an r-*extension* of π_2 if for every sequence π_3 of priorities from $[d]$ that fulfils the declaration r, the sequence π_1 is a $\preceq$-contraction of the concatenation $\pi_2 \cdot \pi_3$.

For example, the sequence $3, 4, 4$ is a 5-extension of the sequence $4, 3, 6$ (independently from the value of $d \ge 6$), because the empty sequence fulfils 3, and $4, 3$ fulfils 4, and $6, p_1, \dots, p_k$ fulfils 4 whenever $p_1, \dots, p_k$ fulfils 5 (i.e., the maximum among $p_1, \dots, p_k$ is either even or at most 5). Notice, moreover, that every sequence is a $2d$-extension of every sequence, because no sequence of priorities from $[d]$ can fulfil the declaration $2d$.

The following lemma is a direct consequence of the definition and of Lemma 3.2:

▶ **Lemma 5.6.** *If a sequence π is an r-extension of a sequence $p_1, \dots, p_n$, then π is also an $r{\restriction_{p_{n+1}}}$-extension of $p_1, \dots, p_n, p_{n+1}$ for every priority $p_{n+1} \in [d]$.*

Additionally, for a node v (of some parity tree) we write $\pi(v)$ for the sequence of priorities in ancestors of v (not including the priority in v).

We now come back to the proof, showing how to construct the new strategy ρ', winning for Eve in $\mathsf{BT}^{\mathsf{s}}(T)$. In order to describe ρ', we play simultaneously in both trees, $\mathsf{BT}^{\mathsf{s}}(T)$ and $\mathsf{tr}^{\mathsf{t}}_d(\varnothing, T)$, and we use moves in one tree to choose moves in the other tree. Namely, at every moment of the play, we remember

- a current node v in $\mathsf{BT}^{\mathsf{s}}(T)$,
- nodes $w_0, w_1, \dots, w_\ell$ in $\mathsf{tr}^{\mathsf{t}}_d(\varnothing, T)$, for some $\ell \in \mathbb{N}$,
- variables $z_1, \dots, z_\ell$ of type o,
- functions $Z_0, Z_1, \dots, Z_\ell$ storing Eve's declarations, where $Z_i \colon \{z_{i+1}, \dots, z_\ell\} \to D_d$ for every i, and
- extended trees $U_0, U_1, \dots, U_\ell$, where every U_i is over $(\{z_{i+1}, \dots, z_\ell\}, \Sigma_d)$.

They satisfy the following invariant:

(a) $\mathsf{BT}^{\mathsf{s}}(T){\restriction_v} = \mathsf{BT}^{\mathsf{s}}(U_0(\!|U_1/z_1|\!) \dots (\!|U_\ell/z_\ell|\!))$,
(b) $\mathsf{tr}^{\mathsf{t}}_d(\varnothing, T){\restriction_{w_i}} = \mathsf{tr}^{\mathsf{t}}_d(Z_i, U_i)$ for all $i \in \{0, 1, \dots, \ell\}$,
(c) $\pi(w_0)$ is a $\preceq$-contraction of $\pi(v)$, and
(d) $\pi(w_j)$ is a $Z_i(z_j)$-extension of $\pi(w_i)$, for all i, j such that $0 \le i < j \le \ell$.

We start with $\ell = 0$, with v and w_0 at the root of $\mathsf{BT}^{\mathsf{s}}(T)$ and $\mathsf{tr}^{\mathsf{t}}_d(\varnothing, T)$, respectively, with $Z_0 = \varnothing$, and with $U_0 = T$. The invariant is clearly satisfied.

Then, during the play, we have one of three cases, depending on the shape of U_0:

1. First, assume that $U_0 = \langle \wp, p, T_1, \dots, T_k \rangle$. Then

$$\mathsf{BT}^{\mathsf{s}}(T){\restriction_v} = \langle \wp, p, \mathsf{BT}^{\mathsf{s}}(T_1(\!|U_1/z_1|\!) \dots (\!|U_\ell/z_\ell|\!)), \dots, \mathsf{BT}^{\mathsf{s}}(T_k(\!|U_1/z_1|\!) \dots (\!|U_\ell/z_\ell|\!)) \rangle;$$
$$\mathsf{tr}^{\mathsf{t}}_d(\varnothing, T){\restriction_{w_0}} = \langle \wp, p, \mathsf{tr}^{\mathsf{t}}_d(Z_0, T_1), \dots, \mathsf{tr}^{\mathsf{t}}_d(Z_0, T_k) \rangle.$$

If $\wp$ = Adam, Adam chooses some child of v in $\mathsf{BT}^{\mathsf{s}}(T)$, and we choose the same child of w_0 in $\mathsf{tr}^{\mathsf{t}}_d(\varnothing, T)$. If $\wp$ = Eve, Eve chooses some child of w_0 in $\mathsf{tr}^{\mathsf{t}}_d(\varnothing, T)$, according to her strategy ρ, and in ρ' we choose the same child of v. Thus, in both cases, we move both v and w_0 to their c-th child, for some $c \in [k]$. We also take $Z_0{\restriction_p}$ as the new Z_0 and T_c as the new U_0. Lemma 5.6 ensures that Item (d) of the invariant is preserved.

2. Another possibility is that U_0 is a variable, that is, $U_0 = z_c$ for some $c \in [\ell]$. Then $\mathsf{tr}^{\mathsf{t}}_d(\varnothing, T){\restriction_{w_0}}$ (i.e., $\mathsf{tr}^{\mathsf{t}}_d(Z_0, U_0)$) is either $\bot$ or $\top$, depending on the parity of $Z_0(z_c)$. But our play in $\mathsf{tr}^{\mathsf{t}}_d(\varnothing, T)$ follows an Eve's winning strategy, so it will be won by Eve, thus the subtree cannot be $\bot$, in which Eve is losing. In consequence $Z_0(z_c)$ is odd, so the empty sequence fulfils $Z_0(z_c)$. This implies that $\pi(w_c)$, being an $Z_0(z_c)$-extension of $\pi(w_0)$, is its $\preceq$-contraction, and thus also an $\preceq$-contraction of $\pi(v)$ (by Lemma 5.4). We discard w_i, z_i, Z_i, U_i for $i < c$ (so that w_c becomes now w_0, etc.).

3. Finally, assume that $U_0 = V(\!|W/z|\!)$. Then $\mathsf{tr}^{\mathsf{t}}_d(\varnothing, T){\restriction_{w_0}} = \langle \mathrm{Eve}, 1, V_1^W, \dots, V_d^W, V_{2d} \rangle$, where $V_r^W = \langle \mathrm{Adam}, 1, V_r, \langle \mathrm{Eve}, r, \mathsf{tr}^{\mathsf{t}}_d(Z_0{\restriction_r}, W) \rangle \rangle$ for $r \in [d]$ and $V_r = \mathsf{tr}^{\mathsf{t}}_d(Z_0[z \mapsto r], V)$ for $r \in D_d$. In such a node w_0 Eve, according to her strategy ρ, chooses a declaration r by going to an appropriate subtree V_r^W (or V_r if $r = 2d$). We then update our memory as follows:

- We leave v and w_i, z_i, Z_i, U_i for $i \ge 1$ unchanged.
- We move w_0 to the root of V_r (this adds once or twice priority 1 to $\pi(w_0)$, hence Item (c) of the invariant is preserved).

- Let $r' = r$ if $r \in [d]$, and $r' = 1$ if $r = 2d$.
- We add an additional node $w_{0.5}$ between w_0 and w_1 (saying this differently, we shift w_i for $i \geq 1$ by one, and we insert the new node in place of w_1). For $w_{0.5}$ we choose the root of $\mathsf{tr}^{\mathsf{t}}_d(Z_0{\restriction}_{r'}, W)$. Notice that $\pi(w_{0.5})$ is an r-extension of $\pi(w_0)$ (for $r \in [d]$ because $\pi(w_{0.5})$ is obtained from $\pi(w_0)$ by appending the priority $r' = r$, and for $r = 2d$ because no sequence of priorities from $[d]$ fulfils $2d$), and that every $\pi(w_j)$ for $1 \leq j \leq \ell$ is a $Z_0{\restriction}_{r'}(z_j)$-extension of $\pi(w_{0.5})$ (by Lemma 5.6).
- As Z_0, U_0, $z_{0.5}$, $Z_{0.5}$, and $U_{0.5}$ we take $Z_0[z \mapsto r]$, V, z, $Z_0{\restriction}_{r'}$, and W, respectively.

Observe that after finitely many repetitions of Cases 2 and 3 necessarily Case 1 has to occur, where the play advances in $\mathsf{BT}^{\mathsf{s}}(T)$. Indeed, $U_0(\!|U_1/z_1|\!)\dots(\!|U_\ell/z_\ell|\!)$ has to generate the next node of $\mathsf{BT}^{\mathsf{s}}(T)$ in finitely many steps; in particular, the number of explicit substitution at the head of U_0 has to be finite.

We have to prove that the infinite branch ξ of $\mathsf{BT}^{\mathsf{s}}(T)$ obtained this way is won by Eve. To this end, consider the corresponding sequence of "w_0" nodes in the construction and observe that this sequence converges to some infinite branch ζ in $\mathsf{tr}^{\mathsf{t}}_d(\varnothing, T)$. Indeed, whenever the sequence enters to a subtree of the form $\mathsf{tr}^{\mathsf{t}}_d(Z_0, V(\!|W/z|\!))$ and stays there forever, then either it enters to the subtree $V_r = \mathsf{tr}^{\mathsf{t}}_d(Z_0[z \mapsto r], V)$ for some r and stays there forever, or, after some time, it enters to the subtree $\mathsf{tr}^{\mathsf{t}}_d(Z_0{\restriction}_r, W)$ for some r and stays there forever. Moreover, the sequence of priorities on ζ is a $\preceq$-contraction of the sequence of priorities on ξ (the function from elements of the former sequence to infixes of the latter sequence, as needed for $\preceq$-contraction, is obtained as the limit of such functions witnessing that always $\pi(w_0)$ is a $\preceq$-contraction of $\pi(v)$). Since ζ agrees with the strategy ρ, it is won by Eve, hence by Lemma 5.5 also ξ is won by Eve, as required. This finishes the proof in the case of Eve winning in $\mathsf{tr}^{\mathsf{t}}_d(\varnothing, T)$.

Suppose now that it is Adam who wins in $\mathsf{tr}^{\mathsf{t}}_d(\varnothing, T)$. The proof in this case is similar, so we only list differences. First, $\succeq$-*contraction* is defined like $\preceq$-contraction, but for every infix $p_{i_{j-1}+1}, p_{i_{j-1}+2}, \dots, p_{i_j}$ in the split we require that r_j is $\succeq$ (instead of $\preceq$) than the leader of the infix. Second, we say that a sequence π_1 of priorities from $[d]$ is an *r-neg-extension* of a sequence π_2 of priorities from $[d]$ if for every sequence π_3 of priorities from $[d]$ that does NOT fulfil the declaration r, the sequence π_1 is a $\succeq$-contraction of the concatenation $\pi_2 \cdot \pi_3$. In Items (c) and (d) of the invariant we replace $\preceq$-contraction by $\succeq$-contraction, and r-extension by r-neg-extension. Then, in Case 1 of the construction we only swap the role of Eve and Adam. In Case 2 we now have that the play is won by Adam, so $Z_0(z_c)$ is even, that is, not fulfilled by the empty sequence; this implies that $\pi(w_c)$, being an $Z_0(z_c)$-neg-extension of $\pi(w_0)$, is also its $\succeq$-contraction. The main difference is in Case 3. For every $r \in [d]$ we know Adam's decision in the root of V_r^W, according to his winning strategy. Take the worst $r \in [d]$ such that in V_r^W Adam goes to the left subtree, or $r = 2d$ if he goes right everywhere; in both cases, Adam's strategy allows to enter V_r. Let also s be the best among priorities that are worse than r; in V_s^W Adam goes to the right subtree (if there are no priorities worse than r, we choose s arbitrarily, e.g., $s = 1$). Then as the new w_0 we take the root of V_r, and as $w_{0.5}$ we take the root of $\mathsf{tr}^{\mathsf{t}}_d(Z{\restriction}_s, W)$. Notice that $\pi(w_{0.5})$ is an r-neg-extension of $\pi(w_0)$: s is better or equal than the leader of every sequence not fulfilling r (also when r is the worst priority, because no such a sequence exists), which ensures that the invariant is preserved.

6 Final remarks

We have presented a new, simple model-checking algorithm for higher-order recursion schemes. One may ask whether this algorithm can be used in practice. Of course the complexity n-EXPTIME for recursion schemes of order n is unacceptably large (even if we take into

account the fact that we are n-fold exponential only in the arity of types and in the size of an automaton, not in the size of a recursion scheme), but one has to recall that there exist tools solving the considered problem in such a complexity. The reason why these tools work is that the time spent by them on "easy" inputs is much smaller than the worst-case complexity (and many "typical inputs" are indeed easy). Unfortunately, this is not the case for our algorithm: the size of the recursion scheme resulting from our transformation is always large. Moreover, it seems unlikely that any simple analysis of the resulting recursion scheme (like removing useless nonterminals or some control flow analysis) may help in reducing its size. Indeed, one can see that if no nonterminals nor arguments were useless in the original recursion scheme, then also no nonterminals nor arguments are useless in the resulting recursion scheme. Thus, our algorithm is mainly of a theoretical interest.

It seems feasible that a transformation similar to the one presented in this paper can be used to solve the simultaneous unboundedness problem (aka. diagonal problem) [11] for recursion schemes. Developing such a transformation is a possible direction for further work.

References

1 Alfred V. Aho. Indexed grammars – an extension of context-free grammars. *J. ACM*, 15(4):647–671, 1968. doi:10.1145/321479.321488.

2 Kazuyuki Asada and Naoki Kobayashi. Size-preserving translations from order-$(n+1)$ word grammars to order-n tree grammars. In Zena M. Ariola, editor, *5th International Conference on Formal Structures for Computation and Deduction, FSCD 2020, June 29-July 6, 2020, Paris, France (Virtual Conference)*, volume 167 of *LIPIcs*, pages 22:1–22:22. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPIcs.FSCD.2020.22.

3 Luca Breveglieri, Alessandra Cherubini, Claudio Citrini, and Stefano Crespi-Reghizzi. Multi-push-down languages and grammars. *Int. J. Found. Comput. Sci.*, 7(3):253–292, 1996. doi:10.1142/S0129054196000191.

4 Christopher H. Broadbent, Arnaud Carayol, Matthew Hague, Andrzej S. Murawski, C.-H. Luke Ong, and Olivier Serre. Collapsible pushdown parity games. *CoRR*, abs/2010.06361, 2020. arXiv:2010.06361.

5 Christopher H. Broadbent, Arnaud Carayol, C.-H. Luke Ong, and Olivier Serre. Recursion schemes and logical reflection. In *Proceedings of the 25th Annual IEEE Symposium on Logic in Computer Science, LICS 2010, 11-14 July 2010, Edinburgh, United Kingdom*, pages 120–129. IEEE Computer Society, 2010. doi:10.1109/LICS.2010.40.

6 Christopher H. Broadbent and Naoki Kobayashi. Saturation-based model checking of higher-order recursion schemes. In Simona Ronchi Della Rocca, editor, *Computer Science Logic 2013, CSL 2013, September 2-5, 2013, Torino, Italy*, volume 23 of *LIPIcs*, pages 129–148. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2013. doi:10.4230/LIPIcs.CSL.2013.129.

7 Christopher H. Broadbent and C.-H. Luke Ong. On global model checking trees generated by higher-order recursion schemes. In Luca de Alfaro, editor, *Foundations of Software Science and Computational Structures, 12th International Conference, FOSSACS 2009, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2009, York, UK, March 22-29, 2009. Proceedings*, volume 5504 of *Lecture Notes in Computer Science*, pages 107–121. Springer, 2009. doi:10.1007/978-3-642-00596-1_9.

8 Cristian S. Calude, Sanjay Jain, Bakhadyr Khoussainov, Wei Li, and Frank Stephan. Deciding parity games in quasipolynomial time. In Hamed Hatami, Pierre McKenzie, and Valerie King, editors, *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2017, Montreal, QC, Canada, June 19-23, 2017*, pages 252–263. ACM, 2017. doi:10.1145/3055399.3055409.

9 Arnaud Carayol and Olivier Serre. Collapsible pushdown automata and labeled recursion schemes: Equivalence, safety and effective selection. In *Proceedings of the 27th Annual IEEE Symposium on Logic in Computer Science, LICS 2012, Dubrovnik, Croatia, June 25-28, 2012*, pages 165–174. IEEE Computer Society, 2012. doi:10.1109/LICS.2012.73.

10 Lorenzo Clemente, Paweł Parys, Sylvain Salvati, and Igor Walukiewicz. Ordered tree-pushdown systems. In Prahladh Harsha and G. Ramalingam, editors, *35th IARCS Annual Conference on Foundation of Software Technology and Theoretical Computer Science, FSTTCS 2015, December 16-18, 2015, Bangalore, India*, volume 45 of *LIPIcs*, pages 163–177. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2015. doi:10.4230/LIPIcs.FSTTCS.2015.163.

11 Lorenzo Clemente, Paweł Parys, Sylvain Salvati, and Igor Walukiewicz. The diagonal problem for higher-order recursion schemes is decidable. *CoRR*, abs/1605.00371, 2016. arXiv:1605.00371.

12 Łukasz Czajka. Coinductive techniques in infinitary lambda-calculus. *CoRR*, abs/1501.04354, 2015. arXiv:1501.04354.

13 Koichi Fujima, Sohei Ito, and Naoki Kobayashi. Practical alternating parity tree automata model checking of higher-order recursion schemes. In Chung-chieh Shan, editor, *Programming Languages and Systems - 11th Asian Symposium, APLAS 2013, Melbourne, VIC, Australia, December 9-11, 2013. Proceedings*, volume 8301 of *Lecture Notes in Computer Science*, pages 17–32. Springer, 2013. doi:10.1007/978-3-319-03542-0_2.

14 Matthew Hague, Andrzej S. Murawski, C.-H. Luke Ong, and Olivier Serre. Collapsible pushdown automata and recursion schemes. In *Proceedings of the Twenty-Third Annual IEEE Symposium on Logic in Computer Science, LICS 2008, 24-27 June 2008, Pittsburgh, PA, USA*, pages 452–461. IEEE Computer Society, 2008. doi:10.1109/LICS.2008.34.

15 Matthew Hague, Andrzej S. Murawski, C.-H. Luke Ong, and Olivier Serre. Collapsible pushdown automata and recursion schemes. *ACM Trans. Comput. Log.*, 18(3):25:1–25:42, 2017. doi:10.1145/3091122.

16 Teodor Knapik, Damian Niwiński, and Paweł Urzyczyn. Higher-order pushdown trees are easy. In Mogens Nielsen and Uffe Engberg, editors, *Foundations of Software Science and Computation Structures, 5th International Conference, FOSSACS 2002, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2002 Grenoble, France, April 8-12, 2002, Proceedings*, volume 2303 of *Lecture Notes in Computer Science*, pages 205–222. Springer, 2002. doi:10.1007/3-540-45931-6_15.

17 Naoki Kobayashi. Model-checking higher-order functions. In António Porto and Francisco Javier López-Fraguas, editors, *Proceedings of the 11th International ACM SIGPLAN Conference on Principles and Practice of Declarative Programming, September 7-9, 2009, Coimbra, Portugal*, pages 25–36. ACM, 2009. doi:10.1145/1599410.1599415.

18 Naoki Kobayashi. A practical linear time algorithm for trivial automata model checking of higher-order recursion schemes. In Martin Hofmann, editor, *Foundations of Software Science and Computational Structures - 14th International Conference, FOSSACS 2011, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2011, Saarbrücken, Germany, March 26-April 3, 2011. Proceedings*, volume 6604 of *Lecture Notes in Computer Science*, pages 260–274. Springer, 2011. doi:10.1007/978-3-642-19805-2_18.

19 Naoki Kobayashi. Model checking higher-order programs. *J. ACM*, 60(3):20:1–20:62, 2013. doi:10.1145/2487241.2487246.

20 Naoki Kobayashi and C.-H. Luke Ong. A type system equivalent to the modal mu-calculus model checking of higher-order recursion schemes. In *Proceedings of the 24th Annual IEEE Symposium on Logic in Computer Science, LICS 2009, 11-14 August 2009, Los Angeles, CA, USA*, pages 179–188. IEEE Computer Society, 2009. doi:10.1109/LICS.2009.29.

21 Robin P. Neatherway and C.-H. Luke Ong. TravMC2: Higher-order model checking for alternating parity tree automata. In Neha Rungta and Oksana Tkachuk, editors, *2014 International Symposium on Model Checking of Software, SPIN 2014, Proceedings, San Jose, CA, USA, July 21-23, 2014*, pages 129–132. ACM, 2014. doi:10.1145/2632362.2632381.

22 Robin P. Neatherway, Steven J. Ramsay, and C.-H. Luke Ong. A traversal-based algorithm for higher-order model checking. In Peter Thiemann and Robby Bruce Findler, editors, *ACM SIGPLAN International Conference on Functional Programming, ICFP'12, Copenhagen, Denmark, September 9-15, 2012*, pages 353–364. ACM, 2012. doi:10.1145/2364527.2364578.

23 C.-H. Luke Ong. On model-checking trees generated by higher-order recursion schemes. In *21th IEEE Symposium on Logic in Computer Science, LICS 2006, 12-15 August 2006, Seattle, WA, USA, Proceedings*, pages 81–90. IEEE Computer Society, 2006. doi:10.1109/LICS.2006.38.

24 Paweł Parys. Higher-order nonemptiness step by step. In Nitin Saxena and Sunil Simon, editors, *40th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS 2020, December 14-18, 2020, BITS Pilani, K K Birla Goa Campus, Goa, India (Virtual Conference)*, volume 182 of *LIPIcs*, pages 53:1–53:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPIcs.FSTTCS.2020.53.

25 Steven J. Ramsay, Robin P. Neatherway, and C.-H. Luke Ong. A type-directed abstraction refinement approach to higher-order model checking. In Suresh Jagannathan and Peter Sewell, editors, *The 41st Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL '14, San Diego, CA, USA, January 20-21, 2014*, pages 61–72. ACM, 2014. doi:10.1145/2535838.2535873.

26 Sylvain Salvati and Igor Walukiewicz. Krivine machines and higher-order schemes. *Inf. Comput.*, 239:340–355, 2014. doi:10.1016/j.ic.2014.07.012.

27 Sylvain Salvati and Igor Walukiewicz. A model for behavioural properties of higher-order programs. In Stephan Kreutzer, editor, *24th EACSL Annual Conference on Computer Science Logic, CSL 2015, September 7-10, 2015, Berlin, Germany*, volume 41 of *LIPIcs*, pages 229–243. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2015. doi:10.4230/LIPIcs.CSL.2015.229.

28 Sylvain Salvati and Igor Walukiewicz. Simply typed fixpoint calculus and collapsible pushdown automata. *Math. Struct. Comput. Sci.*, 26(7):1304–1350, 2016. doi:10.1017/S0960129514000590.

29 Ryota Suzuki, Koichi Fujima, Naoki Kobayashi, and Takeshi Tsukada. Streett automata model checking of higher-order recursion schemes. In Dale Miller, editor, *2nd International Conference on Formal Structures for Computation and Deduction, FSCD 2017, September 3-9, 2017, Oxford, UK*, volume 84 of *LIPIcs*, pages 32:1–32:18. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2017. doi:10.4230/LIPIcs.FSCD.2017.32.

30 Takeshi Tsukada and C.-H. Luke Ong. Compositional higher-order model checking via ω-regular games over Böhm trees. In Thomas A. Henzinger and Dale Miller, editors, *Joint Meeting of the Twenty-Third EACSL Annual Conference on Computer Science Logic (CSL) and the Twenty-Ninth Annual ACM/IEEE Symposium on Logic in Computer Science (LICS), CSL-LICS '14, Vienna, Austria, July 14 - 18, 2014*, pages 78:1–78:10. ACM, 2014. doi:10.1145/2603088.2603133.

Fluted Logic with Counting

Ian Pratt-Hartmann
Department of Computer Science, University of Manchester, UK
Institute of Computer Science, University of Opole, Poland

Abstract

The fluted fragment is a fragment of first-order logic in which the order of quantification of variables coincides with the order in which those variables appear as arguments of predicates. It is known that the fluted fragment possesses the finite model property. In this paper, we extend the fluted fragment by the addition of counting quantifiers. We show that the resulting logic retains the finite model property, and that the satisfiability problem for its $(m+1)$-variable sub-fragment is in m-NExpTime for all positive m. We also consider the satisfiability and finite satisfiability problems for the extension of any of these fragments in which the fluting requirement applies only to sub-formulas having at least three free variables.

2012 ACM Subject Classification Theory of computation → Complexity theory and logic

Keywords and phrases Fluted fragment, counting quantifiers, satisfiability, complexity

Digital Object Identifier 10.4230/LIPIcs.ICALP.2021.141

Category Track B: Automata, Logic, Semantics, and Theory of Programming

Funding This work was supported by the Polish NCN, grant number 2018/31/B/ST6/03662.

Acknowledgements The author wishes to thank Prof. L. Tendera for valuable discussions.

1 Introduction

The *fluted fragment*, $\mathcal{FL}$, is a fragment of first-order logic in which, *very* roughly, the order of quantification of variables coincides with the order in which those variables appear as arguments of predicates, for example:

No student admires every professor
$$\forall x_1(\mathrm{std}(x_1) \to \neg\forall x_2(\mathrm{prof}(x_2) \to \mathrm{admr}(x_1, x_2))) \tag{1}$$

No lecturer introduces any professor to every student
$$\forall x_1(\mathrm{lectr}(x_1) \to \neg\exists x_2(\mathrm{prof}(x_2) \wedge \forall x_3(\mathrm{std}(x_3) \to \mathrm{intro}(x_1, x_2, x_3)))). \tag{2}$$

More precisely, in fluted formulas, all atoms are of the form $p(x_\ell, \ldots, x_m)$, with a contiguous sequence of variables as their arguments, Boolean combinations can only be formed from formulas whose last free variable is the same, and only the last free variable in a formula may be quantified. Equality is not present. It is known that $\mathcal{FL}$ has the finite model property, and that its m-variable sub-fragment, $\mathcal{FL}^m$, is $\lfloor m/2 \rfloor$-NExpTime-hard for all $m \geq 2$ and in $(m-2)$-NExpTime for all $m \geq 3$ [17]. Hence, the satisfiability problem for $\mathcal{FL}$ is Tower-complete in the system of trans-elementary complexity classes of [24]. (It was incorrectly claimed in [20] that this problem is in NExpTime.)

Counting quantifiers are expressions of the form $\exists_{[\leq M]}$, $\exists_{[\geq M]}$ and $\exists_{[=M]}$, where M is a positive integer, with the interpretations "there exist at most/at least/exactly M ...". We investigate the addition of counting quantifiers and equality to the fluted fragment:

At most three lecturers introduce a professor to at least five students
$$\exists_{[\leq 3]}x_1(\mathrm{lectr}(x_1) \wedge \exists x_2(\mathrm{prof}(x_2) \wedge \exists_{[\geq 5]}x_3(\mathrm{std}(x_3) \wedge \mathrm{intro}(x_1, x_2, x_3)))) \tag{3}$$

Every absent-minded professor introduces some student to himself
$$\forall x_1(\text{abs-mnd}(x_1) \wedge \mathrm{prof}(x_1) \to \exists x_2\exists x_3(\mathrm{std}(x_3) \wedge x_2 = x_3 \wedge \mathrm{intro}(x_1, x_2, x_3))). \tag{4}$$

We denote this extension of $\mathcal{FL}$ by $\mathcal{FLC}$, and its m-variable sub-fragment by $\mathcal{FLC}^m$. We also consider the corresponding fragments $\mathcal{SFC}$ and $\mathcal{SFC}^m$, in which the fluting restriction is waived for sub-formulas with at most two variables. (Formal definitions are given in Sec. 2.)

The definition of $\mathcal{FL}$ employed here is that given by Purdy [19], who traces its origins to Quine [22]. (The term "fluted" is actually Quine's.) While it is unclear whether the quantification patterns specified by Purdy are really those that Quine had in mind, it is Purdy's definition which has established itself, and indeed which is – from the point of view of recent work in computational logic – of greater interest. In particular, two-variable fluted logic with counting, $\mathcal{FLC}^2$ includes (under a simple translation) the description logic $\mathcal{ALCHOQ}$, whose sub-fragments have been the focus of intensive investigation over recent decades; and its semi-fluted extension, $\mathcal{SFC}^2$ coincides with $\mathcal{C}^2$, the two-variable fragment of first-order logic with counting quantifiers, whose satisfiability and finite satisfiability problems are known to be NExpTime-complete [6, 12, 15]. Of course, $\mathcal{FL}$ is not limited in the number of variables formulas can contain, a property it shares with the *guarded fragment* [1], which also has the finite model property, and whose satisfiability problem is in 2-ExpTime [5]. In fact, our logic $\mathcal{SFC}$ extends $\mathcal{C}^2$ with fluted formulas in much the same way as the so-called *triguarded fragment* [23] extends the two-variable fragment of first-order logic (without equality) with guarded formulas. The triguarded fragment has recently been shown to have the finite model property [10]; its satisfiability problem is 2-NExpTime-complete, but becomes undecidable in the presence of equality. We note that negation can be applied freely in $\mathcal{FLC}$ and $\mathcal{SFC}$. Thus, these fragments are not subject to any type of guardedness restrictions: for example, (2) is not guarded or even negation-guarded [2].

In this extended abstract, we show that $\mathcal{FLC}$ has the finite model property, and that the satisfiability problem for $\mathcal{FLC}^{m+1}$ is in m-NExpTime for all $m \geq 1$. We also show that the satisfiability and finite satisfiability problems for $\mathcal{SFC}$ remain decidable.

2 Preliminaries

In the context of fluted formulas, logical variables are taken from the sequence $\bar{x}_\omega = x_1, x_2, \ldots$, and all signatures are purely relational, i.e., there are no individual constants or function symbols; however, we allow 0-ary relations (proposition letters). We employ the syntax of counting quantifiers $\exists_{[\leq M]}$, $\exists_{[\geq M]}$ and $\exists_{[=M]}$, where M is a (numeral denoting a) positive integer, under the expected semantics. A *multiset* over some carrier set X is a function f from X to cardinal numbers, where, for each $x \in X$, $f(x)$ is the *multiplicity* with which x occurs in f. Informally, we identify multisets differing only by elements of multiplicity 0. Almost all mulitplicities we encounter will be finite.

We begin with the syntax of the logics considered here. Define the sets of formulas $\mathcal{FLC}^{[m]}$, for all $m \geq 0$, by simultaneous structural recursion as follows:

- **(i)** any atom $p(x_\ell, \ldots, x_m)$, where $x_\ell, \ldots, x_m$ is a contiguous subsequence of $\bar{x}_\omega$ and p a predicate of arity $m - \ell + 1$, is in $\mathcal{FLC}^{[m]}$;
- **(ii)** $\mathcal{FLC}^{[m]}$ is closed under boolean combinations;
- **(iii)** if φ is in $\mathcal{FLC}^{[m+1]}$, then $\exists x_{m+1}.\varphi$ and $\forall x_{m+1}.\varphi$ are in $\mathcal{FLC}^{[m]}$,
- **(iv)** if φ is in $\mathcal{FLC}^{[m+1]}$ and M a non-negative integer, then $\exists_{[\leq M]} x_{m+1}.\varphi$, $\exists_{[\geq M]} x_{m+1}.\varphi$ and $\exists_{[=M]} x_{m+1}.\varphi$ are in $\mathcal{FLC}^{[m]}$.

It is intended that Clause (i) allows the case $\ell = m + 1$ (empty sequence of arguments), so that the atoms in question are proposition letters; and when $m = \ell + 1$ (exactly two arguments), we allow p to be the equality predicate. We define the sets of formulas $\mathcal{FL}^{[m]}$ similarly, except that we do not allow the equality predicate in Clause (i), and Clause (iv) is

dropped altogether. The *fluted fragment* is the set of formulas $\mathcal{FL} = \bigcup_{m\geq 0} \mathcal{FL}^{[m]}$, and the *fluted fragment with counting and equality*, the set of formulas $\mathcal{FLC} = \bigcup_{m\geq 0} \mathcal{FLC}^{[m]}$. Finally, we define $\mathcal{FL}^m$ to be the fragment of $\mathcal{FL}$ in which at most the variables $x_1, \ldots, x_m$ appear (free or bound); and similarly for $\mathcal{FLC}^m$. Thus, (1) is in $\mathcal{FL}^2$, and (2) in $\mathcal{FL}^3$, while (3) and (4) are in $\mathcal{FLC}^3$. Do not confuse $\mathcal{FLC}^m$ with $\mathcal{FLC}^{[m]}$: all of (1)–(4) are in $\mathcal{FLC}^{[0]}$. By *sentence*, we mean a formula with no free variables.

Denote by $\mathcal{C}^2$ the *two-variable fragment of first-order logic with counting*, i.e. the set of first-order formulas with (equality and) counting quantifiers over a purely relational signature, and featuring only two logical variables. (We may without loss of useful expressive power assume that all predicates have arity at most 2.) Formulas of $\mathcal{C}^2$ are not required to be fluted. For example, $\forall x_1 \exists x_2 . r(x_1, x_2) \wedge \forall x_2 \exists_{[\leq 1]} x_1 . r(x_1, x_2) \wedge \exists x_2 \forall x_1 \neg r(x_1, x_2)$ is a formula of $\mathcal{C}^2$, but not of $\mathcal{FLC}^2$. It is straightforward to see that this formula is satisfiable, but only in infinite models. Thus, $\mathcal{C}^2$ lacks the finite model property. It is well-known that the satisfiability and finite satisfiability problems for the three-variable fragment of first-order logic (even without with counting) are undecidable.

In the context of $\mathcal{FLC}$, the possibility arises of waiving the fluting restrictions on sub-formulas featuring at most two free variables. Define the sets of formulas $\mathcal{SFC}^{[m]}$ in the same way as $\mathcal{FLC}^{[m]}$, but with the additional clauses (for $m \geq 2$):

(v) Any $\mathcal{C}^2$-formula ψ, whose set of free variables is equal to one of $\{x_{m-1}, x_m\}$, $\{x_m\}$ or $\emptyset$, is in $\mathcal{SFC}^{[m]}$.

We then take the *semi-fluted fragment with counting* to be the set of formulas $\mathcal{SFC} = \bigcup_{m\geq 0} \mathcal{SFC}^{[m]}$, denoting its m-variable sub-fragment by $\mathcal{SFC}^m$. If φ is a formula of any of the above fragments, we take its *size*, $\|\varphi\|$, to be the number of bits required to write it, on the understanding that numerical subscripts are encoded as binary strings. Since $\mathcal{SFC}$ contains $\mathcal{C}^2$, it lacks the finite model property. We can now state our main results.

▶ **Theorem 1.** *The logic $\mathcal{FLC}$ has the finite model property. The satisfiability problem for $\mathcal{FLC}^{m+1}$ is in m-NExpTime for all $m \geq 1$.*

▶ **Theorem 2.** *The satisfiability and finite satisfiability problems for $\mathcal{SFC}$ are decidable.*

Assuming, as we shall, that the arity of any predicate is fixed in advance, variables in fluted logic convey no information at all, and therefore can be omitted. (This may have been part of the motivation for Quine [21].) The same applies to fluted formulas with counting quantifiers. For example, (3) and (4) can be written, respectively, as:

$$\exists_{[\leq 3]}(\text{lectr} \wedge \exists(\text{prof} \wedge \exists_{[\geq 5]}(\text{std} \wedge \text{intro}))) \tag{5}$$

$$\forall(\text{abs-mnd} \wedge \text{prof} \rightarrow \exists\exists(\text{std} \wedge = \wedge \text{intro})). \tag{6}$$

It is straightforward to reconstruct (3) and (4) (up to a shift of variable indices) from (5) and (6). Consequently, we employ variable-free notation for $\mathcal{FLC}$ in the sequel, as it is more compact, though formulas such as "std $\wedge = \wedge$ intro" admittedly take some getting used to. It is important to realize that, with variable-free notation, any formula of $\mathcal{FLC}^{[m]}$ is, *without lexical change*, also a formula of $\mathcal{FLC}^{[m+1]}$. For example, the sub-formula $\exists(\text{prof} \wedge \exists_{[\geq 5]}(\text{std} \wedge \text{intro}))$ of (5) may be reconstructed as the $\mathcal{FLC}^{[1]}$-formula $\varphi(x_1) := \exists x_2(\text{prof}(x_2) \wedge \exists_{[\geq 5]} x_3(\text{std}(x_3) \wedge \text{intro}(x_1, x_2, x_3)))$, or alternatively as the $\mathcal{FLC}^{[2]}$-formula $\varphi'(x_1, x_2) := \exists x_3(\text{prof}(x_3) \wedge \exists_{[\geq 5]} x_4(\text{std}(x_4) \wedge \text{intro}(x_2, x_3, x_4)))$, and so on.

Thus, using variable-free notation, the sets $\mathcal{FLC}^{[m]}$ are the minimal family of sets of formulas satisfying:

(i) any predicate p of arity less than or equal to m is in $\mathcal{FLC}^{[m]}$;
(ii) $\mathcal{FLC}^{[m]}$ is closed under boolean combinations;
(iii) if φ is in $\mathcal{FLC}^{[m+1]}$, then $\exists\varphi$ and $\forall\varphi$ are in $\mathcal{FLC}^{[m]}$,
(iv) if φ is in $\mathcal{FLC}^{[m+1]}$ and M a non-negative integer, then $\exists_{[\leq M]}\varphi$, $\exists_{[\geq M]}\varphi$ and $\exists_{[=M]}\varphi$ are in $\mathcal{FLC}^{[m]}$.

We also use the term *sentence* in this context to mean a formula of $\mathcal{FLC}^{[0]}$, and we continue to use the notation $\mathcal{FLC}^m$ to denote the same set of formulas as defined above, but written without variables. We remark that there is no difference between a *quantifier-free* formula of $\mathcal{FLC}^m$ and a *quantifier-free* formula of $\mathcal{FLC}^{[m]}$; it is just a Boolean combination of predicates (including equality) of arity at most m.

If $\varphi \in \mathcal{FLC}^{[m]}$, and $\bar{a} = a_1, \ldots, a_m$ is an m-tuple of elements from some structure $\mathfrak{A}$ interpreting the signature of φ, we write $\mathfrak{A} \models \varphi[\bar{a}]$ as usual to indicate that $\bar{a}$ satisfies φ in $\mathfrak{A}$, under the assignments $x_1 \mapsto a_1, \ldots, x_m \mapsto a_m$. Observe that this notation remains meaningful even with variable-free notation. Indeed, if $\bar{a} \in A^m$ and $a \in A$, then $\mathfrak{A} \models \varphi[\bar{a}]$ if and only if $\mathfrak{A} \models \varphi[a\bar{a}]$. (This trick is important, because it will be used at various points in Sec. 4.) The notation $\forall^m$ stands for a block $\forall \cdots \forall$ of m universal quantifiers.

Of course, we cannot – without further ado – use variable-free notation for $\mathcal{SFC}$.

The analysis of decidable fragments is often simplified by the use of normal forms in the style of [25]. Here, we adapt the normal forms for $\mathcal{FL}$ from [17, Lemma 4.1].

▶ **Lemma 3.** *Let φ be a formula of $\mathcal{FLC}^{m+1}$ $(m \geq 1)$. Then we may compute, in time bounded by a polynomial function of $\|\varphi\|$, an $\mathcal{FLC}^{m+1}$-sentence satisfiable over the same domains as φ, and having the form*

$$\bigwedge_{s\in S} \forall^m(\mu_s \to \exists_{[=M_s]}\zeta_s) \wedge \bigwedge_{t\in T} \forall^m(\nu_t \to \forall\eta_t) \wedge \forall^{m+1}\theta, \tag{7}$$

where S and T are index sets, the μ_s and ν_t are quantifier-free $\mathcal{FLC}^m$-formulas, the ζ_s, η_t and θ are quantifier-free $\mathcal{FLC}^{m+1}$-formulas, and the M_s are positive integers.

Proof. By prepending existential quantifiers if necessary, we may assume that φ is a sentence. Call a quantifier of the form $\exists_{[=M]}$ an *equality quantifier*. Somewhat counter-intuitively, we first remove all equality quantifiers from φ. Let $\varphi_0 := \varphi$, and suppose φ_0 possesses a sub-formula $\theta = \exists_{[=M]}\chi$, such that χ contains no equality quantifiers. Let ℓ be the smallest number such that $\theta \in \mathcal{FLC}^{[\ell]}$. Let p, q be fresh predicates or arity ℓ, and define $\varphi_1 := \varphi_0[\theta/(p \wedge q)]$ and

$$\psi_1 := \forall^\ell(p \leftrightarrow \exists_{[\geq M]}\chi) \wedge \forall^\ell(q \leftrightarrow \exists_{[\leq M]}\chi).$$

It is obvious that φ_0 and $\varphi_1 \wedge \psi_1$ are satisfiable over the same domains Now process φ_1 in the same way to obtain φ_2, until we eventually obtain a sentence φ_n containing no equality quantifiers. This process evidently terminates in polynomial time. Since φ and $\varphi_n \wedge \psi_n \wedge \cdots \wedge \psi_1$ are satisfiable over the same domains, we may simply assume, henceforth, that φ contains no equality quantifiers.

Since there are no equality quantifiers in φ we may move negations inward in the usual way, so that they apply to atomic formulas. At this point, we may follow the proof of the analogous theorem for $\mathcal{FL}$ presented in [17, pp. 174], obtaining, in polynomial time, a sentence satisfiable over the same domains as φ, and having the form

$$\bigwedge_{s\in S} \forall^m(\mu_s \to \exists_{[\bowtie_s M_s]}\zeta_s) \wedge \bigwedge_{t\in T} \forall^m(\nu_t \to \forall\eta_t) \wedge \forall^{m+1}\theta,$$

where each $\bowtie_s$ is one of the symbols $\leq$ or $\geq$. We now eliminate all occurrences of $\leq$ and $\geq$ with $=$. For any conjunct $\forall^m(\mu_s \to \exists_{[\leq M_s]}\zeta_s)$, we let q_s be a new predicate of arity $m+1$ and replace this conjunct with

$$\forall^m(\mu_s \to \exists_{[=M_s]}q_s) \wedge \forall^{m+1}(\zeta_s \to q_s).$$

The case $\geq$ is treated similarly. ◀

We refer to formulas of the forms (7) as *normal-form* formulas of $\mathcal{FLC}^{m+1}$.

The following notions are useful for analysing $\mathcal{FLC}$-formulas. Let Σ be a finite relational signature. A *fluted literal over* Σ is an expression p or $\neg p$, where $p \in \Sigma \cup \{=\}$. (Remember that, under variable-free notation, we think of p as an atom $p(x_\ell, \ldots, x_m)$ of $\mathcal{FLC}^{[m]}$.) The *arity* of the literal is the arity of the underlying predicate. A *fluted m-type* τ (*over* Σ) is a maximal consistent set of fluted literals over Σ having arity at most m. We call τ *reflexive* if it contains the literal $=$. We silently identify fluted m-types with their conjunctions, thus regarding them as quantifier-free $\mathcal{FL}^m$-formulas. Obviously, if τ is a fluted m-type and ψ a quantifier-free $\mathcal{FLC}^m$-formula over the same signature, then either $\models \tau \to \psi$ or $\models \tau \to \neg\psi$. Suppose $\mathfrak{A}$ is a structure over domain A interpreting Σ, and $\bar{a}$ an m-tuple of elements from A ($m \geq 1$). Then there is a unique fluted m-type τ such that $\mathfrak{A} \models \tau[\bar{a}]$. We denote τ by $\mathrm{ftp}^{\mathfrak{A}}[\bar{a}]$, and call it the fluted m-type of $\bar{a}$.

A *fluted star-type* σ of *dimension* m *over* Σ is a multiset of fluted $(m+1)$-types over Σ at most one of which (counting multiplicities) is reflexive. The term "star-type" comes from [15]; in the present paper,we concern ourselves either with *fluted* star-types or *semi-fluted* star-types (defined below). Since we may list the fluted $(m+1)$-types over Σ as $\tau_1, \ldots, \tau_J$ in some fixed order ($J \leq 2^{|\Sigma|+1}$), we can regard any fluted star-type σ over Σ as a vector $(\sigma(\tau_1), \ldots, \sigma(\tau_J))$ of cardinal numbers. We say that σ is *M-bounded* if $|\sigma| = \sigma(\tau_1) + \cdots + \sigma(\tau_J) \leq M$. If ζ is a quantifier-free $\mathcal{FLC}^{m+1}$-formula over Σ, the *retract* of σ to ζ is the fluted star-type $\sigma{\restriction}\zeta$ given by:

$$(\sigma{\restriction}\zeta)(\tau) = \begin{cases} \sigma(\tau) & \text{if } \models \tau \to \zeta \\ 0 & \text{otherwise.} \end{cases}$$

Thus, when performing a retract to ζ, we remove from σ all occurrences of those fluted $(m+1)$-types inconsistent with ζ (i.e. set their multiplicities to 0). We say that σ is a *fluted ζ-star-type* if $\sigma = \sigma{\restriction}\zeta$. Suppose $\mathfrak{A}$ is a structure over domain A interpreting Σ, and $\bar{a}$ an m-tuple of elements from A ($m \geq 1$). If ζ is any quantifier-free formula of $\mathcal{FLC}^{m+1}$ over Σ, we may define a fluted ζ-star-type σ of dimension m by setting, for each fluted $(m+1)$-type τ over Σ, $\sigma(\tau) = |\{b \in A : \mathfrak{A} \models \tau[\bar{a}b] \text{ and } \mathfrak{A} \models \zeta[\bar{a}b]\}|$. We denote σ by $\mathrm{fst}^{\mathfrak{A}}_{\zeta}[\bar{a}]$, and call it the *fluted ζ-star-type of $\bar{a}$ in $\mathfrak{A}$*. As an aide to intuition, think of $\bar{a}$ as emitting various "ζ-rays", each absorbed by some element $b \in A$ such that $\mathfrak{A} \models \zeta[\bar{a}b]$. Every ζ-ray has a "colour" specified by some fluted $(m+1)$-type τ such that $\models \tau \to \zeta$. The fluted star-type $\mathrm{fst}^{\mathfrak{A}}_{\zeta}[\bar{a}]$ simply counts how many rays of each colour arise in this way. To grasp the utility of these notions, let φ be a normal-form formula (7), and suppose $\mathfrak{A} \models \varphi$. Now let $\mathfrak{B}$ be a structure such that, for every m-tuple $\bar{b}$ from B, there exists an m-tuple $\bar{a}$ from A satisfying: (i) $\mathrm{ftp}^{\mathfrak{B}}[\bar{b}] = \mathrm{ftp}^{\mathfrak{A}}[\bar{a}]$; (ii) $\mathrm{fst}^{\mathfrak{B}}_{\zeta_s}[\bar{b}] = \mathrm{fst}^{\mathfrak{A}}_{\zeta_s}[\bar{a}]$ for every s such that $\mathfrak{B} \models \mu_s[\bar{b}]$, and (iii) for every $b' \in B$, there exists $a' \in A$ such that $\mathrm{ftp}^{\mathfrak{B}}[\bar{b}b'] = \mathrm{ftp}^{\mathfrak{A}}[\bar{a}a']$. Then $\mathfrak{B} \models \varphi$.

We mentioned earlier that the logic $\mathcal{C}^2$ lacks the finite model property, but that its satisfiability and finite satisfiability problems are in NExpTime – these being relatively non-trivial results. The following lemma illustrates the comparative weakness of its fluted sub-fragment, $\mathcal{FLC}^2$ by establishing that this has the finite model property. In fact, we adapt a well-known proof (see [8, pp. 77 ff.], [9]) of the corresponding statement for the monadic fragment of first-order logic [11].

▶ **Lemma 4.** *If φ is a satisfiable formula of $\mathcal{FLC}^2$, then φ has a model of size bounded by an exponential function of $\|\varphi\|$. The satisfiablity problem for $\mathcal{FLC}^2$ is in NExpTime.*

Proof. By Lemma 3, we may assume that φ is of the form (7), over a signature Σ, where $m = 1$. We may further assume that Σ features no predicates of arity 0, since their truth-values can be guessed. Let $M = \sum_{s\in S} M_s$, and suppose $\mathfrak{A} \models \varphi$. For each fluted 1-type π (over Σ) realized in $\mathfrak{A}$, let B_π be a set of cardinality $\min(M+1, |\{a \in A \mid \mathrm{ftp}^{\mathfrak{A}}[a] = \pi\}|)$ and let $B = \bigcup_\pi B_\pi$. Thus, $|B| \leq (M+1)2^{|\Sigma|+1}$. We define a structure $\mathfrak{B}$ with domain B and show that $\mathfrak{B} \models \varphi$. For each $b \in B_\pi$, set $\mathrm{ftp}^{\mathfrak{B}}[b] = \pi$. These fluted 1-types involve only unary predicates, and so may be assigned independently of each other. To complete the definition of $\mathfrak{B}$ we fix the extensions of binary predicates so as to determine $\mathrm{ftp}^{\mathfrak{B}}[bb']$ for any ordered pair of elements $\langle b, b'\rangle \in B^2$.

Pick any $b \in B$, and let $a \in A$ be such that $\mathrm{ftp}^{\mathfrak{A}}[a] = \mathrm{ftp}^{\mathfrak{B}}[b] = \pi$, say. Write $S' = \{s \in S : \mathfrak{A} \models \mu_s[a]\}$ and $\zeta' = \bigvee_{s\in S'} \zeta_s$, and let A' be the set of elements $a' \in A \setminus \{a\}$ such that $\mathfrak{A} \models \zeta'[aa']$, say $A' = \{a_1, \ldots, a_k\}$, with $k \leq M$. Thus we may choose a subset $B' = \{b_1, \ldots, b_k\} \subseteq B \setminus \{b\}$ such that, for all i $(1 \leq i \leq k)$, $\mathrm{ftp}^{\mathfrak{B}}[b_i] = \mathrm{ftp}^{\mathfrak{A}}[a_i]$. However this is done, we are guaranteed that, for every $b' \in B \setminus (B' \cup \{b\})$, we can find some $a' \in A \setminus (A' \cup \{a\})$ such that $\mathrm{ftp}^{\mathfrak{A}}[a'] = \mathrm{ftp}^{\mathfrak{B}}[b']$. Now set $\mathrm{ftp}^{\mathfrak{B}}[bb] = \mathrm{ftp}^{\mathfrak{A}}[aa]$ and $\mathrm{ftp}^{\mathfrak{B}}[bb_i] = \mathrm{ftp}^{\mathfrak{A}}[aa_i]$ for all i $(1 \leq i \leq k)$. Further, for all $b' \in B \setminus (B' \cup \{b\})$, pick some $a' \in A \setminus (A' \cup \{a\})$ such that $\mathrm{ftp}^{\mathfrak{A}}[a'] = \mathrm{ftp}^{\mathfrak{B}}[b']$, and set $\mathrm{ftp}^{\mathfrak{B}}[bb'] = \mathrm{ftp}^{\mathfrak{A}}[aa']$. Observe that, in the latter case, $\mathfrak{A} \not\models \zeta'[aa']$, and therefore $\mathfrak{B} \not\models \zeta'[bb']$. Hence, $\mathrm{fst}^{\mathfrak{B}}_{\zeta'}[b] = \mathrm{fst}^{\mathfrak{A}}_{\zeta'}[a]$, so that $\mathrm{fst}^{\mathfrak{B}}_{\zeta_s}[b] = \big(\mathrm{fst}^{\mathfrak{B}}_{\zeta'}[b]\big)\restriction\zeta_s = \big(\mathrm{fst}^{\mathfrak{A}}_{\zeta'}[a]\big)\restriction\zeta_s = \mathrm{fst}^{\mathfrak{A}}_{\zeta_s}[a]$ for every $s \in S'$. By carrying out this construction for every $b \in B$, we fully define $\mathfrak{B}$. Note that the *fluted* 2-types assigned in this process never clash with the fluted 1-types already assigned, and never clash with each other. Thus, for every element $b \in B$ there exists $a \in A$ such that: (i) $\mathrm{ftp}^{\mathfrak{B}}[b] = \mathrm{ftp}^{\mathfrak{A}}[a]$; (ii) $\mathrm{fst}^{\mathfrak{B}}_{\zeta_s}[b] = \mathrm{fst}^{\mathfrak{A}}_{\zeta_s}[a]$ for every s such that $\mathfrak{B} \models \mu_s[b]$, and (iii) for every element $b' \in B$, there exists $a' \in A$ such that $\mathrm{ftp}^{\mathfrak{B}}[bb'] = \mathrm{ftp}^{\mathfrak{A}}[aa']$. Hence $\mathfrak{B} \models \varphi$. ◀

At various points in the ensuing argument, we need to vary the signatures interpreted by structures. The following notation and terminology is standard. If $\mathfrak{A}^+$ is any structure interpreting a signature Σ^+, and $\Sigma \subseteq \Sigma^+$, we denote by $\mathfrak{A}^+\restriction\Sigma$ the structure obtained by forgetting the predicates in $\Sigma^+ \setminus \Sigma$. We call $\mathfrak{A} = \mathfrak{A}^+\restriction\Sigma$ the *reduct* of $\mathfrak{A}^+$ to Σ, and say that $\mathfrak{A}^+$ is an *expansion* of $\mathfrak{A}$.

3 Existential Presburger quantifiers

In view of Lemma 4, a natural strategy for proving Theorem 1 suggests itself: reduce the satisfiability problem for $\mathcal{FLC}^{m+1}$ to that for $\mathcal{FLC}^m$. This is *nearly* the strategy we follow. To make it work, however, we must generalize the notion of counting quantifiers. Denote the natural numbers $\{0, 1, 2, \ldots\}$ by $\mathbb{N}$. A *linear Diophantine inequality* is an expression $a_1v_1 + \cdots + a_nv_n + b \leq c_1v_1 + \cdots + c_nv_n + d$, with coeffecients in $\mathbb{N}$. If $\mathcal{E}(\bar{v})$ is a system of linear Diophantine inequalities in variables $\bar{v}$, a *solution* of $\mathcal{E}$ is an assignment of natural numbers $\bar{a}$ to the variables $\bar{v}$ which make all inequalities of $\mathcal{E}$ true. It was shown in [4] that one can bound the values occurring in the solutions of such systems. (Various such bounds are available: see, e.g. [13].) The following is adequate for our purposes:

▶ **Theorem 5** (from [14], Corollary 1)**.** *Let $\mathcal{E}$ be a system of m linear Diophantine inequalities in n variables, with maximum coefficient M. If $\mathcal{E}$ has a solution, then it has one in which all values are bounded by $(2 + (n+1)M)^{n+m}$.*

By *Presburger arithmetic*, we understand the set of first-order formulas (with equality) over the signature $\{\mathbb{N}, +, \leq, \cdot\}$ whose atomic sub-formulas are linear Diophantine inequalities. We interpret these symbols over the domain $\mathbb{N}$ in the standard way (with the constants $\mathbb{N}$ interpreted as themselves), and say that a tuple of natural numbers $\bar{a}$ *satisfies* a formula of Presburger arithmetic $\Theta(\bar{v})$ if $\mathbb{N} \models \Theta[\bar{a}]$. The *size* of Θ, denoted $\|\Theta\|$, is the number of bits required to write it in the usual way, under the assumption that the individual constants (i.e. coefficients of the linear Diophantine inequalities) are encoded as bit-strings. By *existential Presburger arithmetic*, we mean the set of formulas of Presburger arithmetic of the form $\Theta(\bar{v}) = \exists \bar{u}.\Xi(\bar{v}, \bar{u})$, where Ξ is quantifier-free. Theorem 5 immediately yields (see also, e.g. [7, Table 1]):

▶ **Corollary 6.** *There is a non-deterministic procedure which, given a formula $\Theta(\bar{v})$ of existential Presburger arithmetic and tuple $\bar{a}$ of natural numbers bounded by M with the same arity as $\bar{v}$, has a successful run if and only if $\bar{a}$ satisfies $\Theta(\bar{v})$. This procedure runs in time bounded by a polynomial function of $\|\Theta\| + \log M$.*

Now for our generalization of counting quantifiers.

Fix some $m \geq 1$, and let Σ be a purely relational signature, M a positive integer and Θ a formula of existential Presburger arithmetic in variables $\bar{v} = v_1, \ldots, v_J$ corresponding to the fluted $(m+1)$-types $\tau_1, \ldots, \tau_J$ over Σ. We call an expression $\mathsf{Q}\langle \Sigma, M, \Theta \rangle$ a *fluted existential Presburger quantifier* (or: *fluted ep-quantifier*). If ζ is a quantifier-free formula of $\mathcal{FLC}^{m+1}$, we allow formulas φ of the form $\mathsf{Q}\langle \Sigma, M, \Theta \rangle \zeta$, with semantics given by declaring, for any structure $\mathfrak{A}$ interpreting a signature $\Sigma' \supseteq \Sigma$ and any m-tuple $\bar{a}$ of elements from A:

$$\mathfrak{A} \models \varphi[\bar{a}] \text{ if and only if } \mathrm{fst}_{\zeta}^{(\mathfrak{A} \restriction \Sigma)}[\bar{a}] \text{ is } M\text{-bounded and satisfies } \Theta(\bar{v}). \tag{8}$$

Recall in this connection that we regard a star-type over Σ as a vector with entries in $\mathbb{N}$; the star-type in question is M-bounded if the sum of those entries is at most M. By way of maintaining some contact with familiar territory, the $\mathcal{FLC}$-formula $\exists_{[=M]}\zeta$ can be written in the new syntax as $\mathsf{Q}\langle \Sigma, M, \Theta \rangle \zeta$, where Θ is the single equation $v_1 + \cdots + v_J = M$. (Of course: this equation is just a conjunction of two linear inequalities, and so counts as a formula of Presburger arithmetic). Note that there is no existential quantification in this case.

We define $\mathcal{FLC}^{m+1}_{\mathrm{ep}}$ to be the set of formulas φ given by

$$\bigwedge_{s \in S} \forall^m (\mu_s \to \mathsf{Q}\langle \Sigma, M_s, \Theta_s \rangle \zeta_s) \wedge \bigwedge_{t \in T} \forall^m (\nu_t \to \forall \eta_t) \wedge \forall^{m+1} \theta, \tag{9}$$

where Σ is a relational signature, the μ_s and ν_t are quantifier-free $\mathcal{FLC}^m$-formulas over Σ, the ζ_s, η_t and θ are quantifier-free $\mathcal{FLC}^{m+1}$-formulas over Σ, the M_s are positive integers and the Θ_s are formulas of existential Presburger arithmetic with free variables corresponding to the fluted $(m+1)$-types over Σ. We remark that we have *defined* $\mathcal{FLC}^{m+1}_{\mathrm{ep}}$ directly interms of fomulas of the form (9), rather than establishing an analogue of Lemma 3 for a language extending $\mathcal{FLC}^{m+1}$. This is intentional: formulas in which $\mathsf{Q}\langle \Sigma, M_s, \Theta_s \rangle \zeta$ appears with negative polarity might not be *succinctly* expressible in the form (9). We define the *effective size* of φ, denoted $\#(\varphi)$, to be the quantity $\log(\|\varphi\|) + |S| + |T| + \log M + |\Sigma|$, where $M = \sum_{s \in S} M_s$. Informally: when measuring the effective size of φ, we do not mind if $\|\varphi\|$ becomes exponentially large, as long as Σ, S, T and the number of bits required to write the various M_s do not.

We stress that fluted ep-quantifiers give us no additional expressive power beyond the standard counting quantifiers. Indeed, if ζ is a quantifier-free formula of $\mathcal{FLC}^{m+1}$ over a signature Σ, and supposing the fluted $(m+1)$-types over Σ to be enumerated as $\tau_1, \ldots, \tau_J$, any formula $\mathsf{Q}\langle \Sigma, M, \Theta \rangle \zeta$ is logically equivalent to the huge disjunction

$$\bigvee \left\{ \bigwedge_{j=1}^{J} \exists_{[=\sigma(\tau_j)]}\tau_j \;\middle|\; \sigma \text{ is an } M\text{-bounded fluted } \zeta\text{-star-type over } \Sigma \text{ satisfying } \Theta \right\}. \quad (10)$$

However, fluted ep-quantifiers can be more compact, and we require the added strength of the following routine extension of Lemma 4.

▶ **Lemma 7.** *If φ is a satisfiable formula of $\mathcal{FLC}^2_{\text{ep}}$, then φ has a model of size bounded by an exponential function of $\#(\varphi)$. The satisfiability problem for $\mathcal{FLC}^2_{\text{ep}}$ is in NExpTime, measured in terms of the effective size of the input.*

Proof. For the first statement, let a formula φ of $\mathcal{FLC}^2_{\text{ep}}$ be given. By definition, φ is in the form (9) with $m = 1$. Now construct $\mathfrak{B}$ as in the proof of Lemma 4. Still we have $\mathfrak{B} \models \varphi$, since it does not matter whether the permitted star-types are specified by means of standard counting quantifiers or fluted ep-quantifiers. Further, $|B| \leq (M+1)2^{|\Sigma|+1}$ and thus is bounded by an exponential function of $\#(\varphi)$. For the second statement, we may simply guess a structure $\mathfrak{B}$ subject to this size bound and check that it satisfies φ. It follows from Lemma 5 that, for any $b \in B$ and $s \in S$ such that $\mathfrak{B} \models \mu_s[b]$, we may check, in *non-deterministic* time bounded by an exponential function of $\#(\varphi)$, that $\text{fst}^{\mathfrak{B}}_{\zeta_s}[b]$, satisfies Θ_s. Checking the remaining conditions of φ involves standard model-checking, and requires only (deterministic) time bounded by an exponential function of $\#(\varphi)$. ◀

4 Proof of main result

In this section we prove Theorems 1 and 2, proceeding via the logics $\mathcal{FLC}^m_{\text{ep}}$. Fix some $\mathcal{FLC}^{m+1}_{\text{ep}}$-formula φ as given in (9), with $m \geq 2$. We show how to construct an $\mathcal{FLC}^m_{\text{ep}}$-formula φ', such that φ and φ' are satisfiable over the same domains. The formula φ' employs a signature Σ' formed by removing from Σ all $(m+1)$-ary predicates, while adding a fresh $(m-1)$-ary predicate $p_{S',T'}$ and a fresh m-ary predicate $q_{S',T'}$ for each $S' \subseteq S$ and each $T' \subseteq T$. It is obvious that $|\Sigma'| \leq |\Sigma| + 2^{|S|+|T|+1}$. The maximal arity of predicates in Σ is $m+1$, and in Σ' is m. In addition to Σ and Σ', we consider the signatures $\Sigma^+ = \Sigma \cup \Sigma'$ and $\Sigma^- = \Sigma \cap \Sigma'$. As explained in Sec. 2, we assume some fixed enumeration $\tau_1, \ldots, \tau_J$ of the $J = 2^{|\Sigma|+1}$ fluted $(m+1)$-types over Σ. We write $\tau^+_1, \ldots, \tau^+_{J^+}$ for the corresponding enumeration of fluted $(m+1)$-types over Σ^+; likewise we enumerate the fluted m-types over Σ' as $\tau'_1, \ldots, \tau'_{J'}$, and over Σ^- as $\tau^-_1, \ldots, \tau^-_{J^-}$.

Our essential problem is to get rid of the $(m+1)$-ary predicates appearing in φ without affecting satisfiability. The following device will help. Let ψ be any quantifier-free $\mathcal{FLC}^{m+1}$-formula over Σ. Clearly, there is, up to logical equivalence, a unique strongest quantifier-free $\mathcal{FLC}^m$-formula over Σ^- entailed by ψ, i.e. a quantifier-free $\mathcal{FLC}^m$-formula ψ° over Σ^- satisfying: (i) $\models \psi \to \psi^\circ$; and (ii) for all quantifier-free $\mathcal{FLC}^m$-formulas χ over Σ^- such that $\models \psi \to \chi$, we have $\models \psi^\circ \to \chi$. Indeed, since there are only finitely many such χ (ignoring logical equivalents), we can take ψ° to be their conjunction. Strictly, of course, ψ° is only defined up to logical equivalence, and in fact there are various ways to construct it. Thus, for example, [16, 17] employ resolution theorem-proving; however, the procedure in the following proof requires only basic propositional logic.

▶ **Lemma 8.** *Let ψ be a quantifier-free $\mathcal{FLC}^{m+1}$-formula over Σ. Then we can compute ψ° in time bounded by an exponential function of $\|\psi\| + |\Sigma|$. Moreover, if τ^- is a fluted m-type over Σ^- such that $\models \tau^- \to \psi^\circ$, then there exists a fluted $(m+1)$-type τ over Σ extending τ^- such that $\models \tau \to \psi$.*

Proof. We begin by writing ψ, equivalently, in disjunctive normal form. Thus, $\psi \equiv \bigvee\{\tau \mid \tau \in D\}$, where D is a set of fluted $(m+1)$-types (over Σ). For each $\tau \in D$, let τ° be the fluted m-type over Σ^- obtained by deleting from τ all conjuncts involving predicates not in Σ^-, and define ψ° to be $\bigvee\{\tau^\circ \mid \tau \in D\}$. Note that, since by assumption $m \geq 2$, we will never delete equality-literals. It is evident that this construction can be carried out in time bounded by an exponential function of $\|\psi\|+|\Sigma|$. Since $\models \tau \to \tau^\circ$ for any $(m+1)$-type τ, it is immediate that $\models \psi \to \psi^\circ$. On the other hand, suppose χ is a quantifier-free $\mathcal{FLC}^m$-formula over Σ^- entailed by ψ; without loss of generality, χ is in disjunctive normal form. We claim that τ° is a disjunct of χ for all $\tau \in D$. For if not, we have $\models \psi \to \neg\tau^\circ$, whence $\models \psi \to \neg\tau$, contradicting the supposition that $\psi \equiv \bigvee\{\tau \mid \tau \in D\}$. Hence, $\models \psi^\circ \to \chi$. Given the above construction of ψ°, the second statement of the lemma is completely trivial. ◀

Also in this connection, we define a further operation on fluted star-types. Suppose that σ is an m-dimensional fluted star-type over Σ. The *reduct of σ to Σ^-*, denoted σ/Σ^-, is the $(m-1)$-dimensional fluted star-type over Σ^- given by

$$(\sigma/\Sigma^-)(\tau^-) = \sum\{\sigma(\tau) : \tau \text{ a fluted } (m+1)\text{-type over } \Sigma \text{ such that } \models \tau \to \tau^-\},$$

where τ^- is any fluted m-type over Σ^-. Thus, when forming a reduct to Σ^-, we merge together fluted $(m+1)$-types which look identical in the smaller signature.

To explain the intuition behind the construction of φ', we first describe how a putative model $\mathfrak{A} \models \varphi$ can be expanded to a structure, $\mathfrak{A}^+$, interpreting $\Sigma^+ = \Sigma \cup \Sigma'$. Taking $\mathfrak{A}'$ to be the reduct of $\mathfrak{A}^+$ to Σ' (i.e. with all the $(m+1)$-ary predicates removed), we observe in (13), (14) and (15) that the formulas which will eventually form the conjuncts of φ' are all true in $\mathfrak{A}'$. We begin with the predicates $p_{S',T'}$ for $S' \subseteq S$ and $T' \subseteq T$. Let $\bar{a} \in A^{m-1}$ be any $(m-1)$-tuple of elements, and consider what φ tells us about the relationship of $\bar{a}$ to other elements in the structure. Since φ is fluted, and bearing in mind the form (9), what really matters here are the different subsets $S' \subseteq S$ and $T' \subseteq T$ for which there exists an element a such that $\mathfrak{A} \models \mu_s[a\bar{a}]$ for all $s \in S'$ and $\mathfrak{A} \models \nu_t[a\bar{a}]$ for all $t \in T'$. The $(m-1)$-ary predicates $p_{S',T'}$ will simply record which pairs S', T' are realized in this way. That is, we set, for every $S' \subseteq S$ and $T' \subseteq T$,

$$p^{\mathfrak{A}^+}_{S',T'} = p^{\mathfrak{A}'}_{S',T'} = \{\bar{a} \in A^{m-1} : \text{for some } a \in A,\ \mathfrak{A} \models \mu_s[a\bar{a}] \text{ for all } s \in S' \text{ and } \mathfrak{A} \models \nu_t[a\bar{a}] \text{ for all } t \in T'\}. \quad (11)$$

The predicates $q_{S',T'}$ are only slightly more complicated. Fix $S' \subseteq S$ and $T' \subseteq T$ for the moment, and suppose that $\mathfrak{A}^+ \models p_{S',T'}[\bar{a}]$. By construction, there exists $a \in A$ such that $\mathfrak{A} \models \mu_s[a\bar{a}]$ for all $s \in S'$ and $\mathfrak{A} \models \nu_t[a\bar{a}]$, for all $t \in T'$. So pick *any* such a and denote it by $\dot{a}$. The formula φ then guarantees that, for each $s \in S'$, the m-tuple $\dot{a}\bar{a}$ satisfies the formula $\mathsf{Q}\langle\Sigma, M_s, \Theta_s\rangle\zeta_s$. Defining $B_{\bar{a}} = \{b \in A : \mathfrak{A} \models \zeta_s[\dot{a}\bar{a}b] \text{ for some } s \in S'\}$, we set

$$q^{\mathfrak{A}^+}_{S',T'} = q^{\mathfrak{A}'}_{S',T'} = \{\bar{a}b \in A^m : \mathfrak{A}^+ \models p_{S',T'}[\bar{a}] \text{ and } b \in B_{\bar{a}}\}. \quad (12)$$

Thus, $B_{\bar{a}}$ serves to pick out the witnesses required by the various fluted ep-quantifiers $\mathsf{Q}\langle\Sigma, M_s, \Theta_s\rangle$ for the tuple $\dot{a}\bar{a}$, as s varies over S'. Letting $\zeta_{S'} = \bigvee_{s\in S'}\zeta_s$, we see that $B_{\bar{a}}$ is the set of elements absorbing $\zeta_{S'}$-rays emitted by $\dot{a}\bar{a}$. Observe, in particular, that $|B_{\bar{a}}| \leq \sum_{S'} M_s$. The whole construction is illustrated in Fig. 1. Here we see, arranged in a horizontal strip, the m-tuple $\dot{a}\bar{a}$, which satisfies μ_s for all $s \in S'$ and ν_t for all $t \in T'$. The elements $b \in B_{\bar{a}}$ absorbing the $\zeta_{S'}$-rays emitted by $\dot{a}\bar{a}$, are taken to lie on the periphery of the fan-shaped region. Each of these elements b absorbs a ζ_s-ray, for at least one (in general

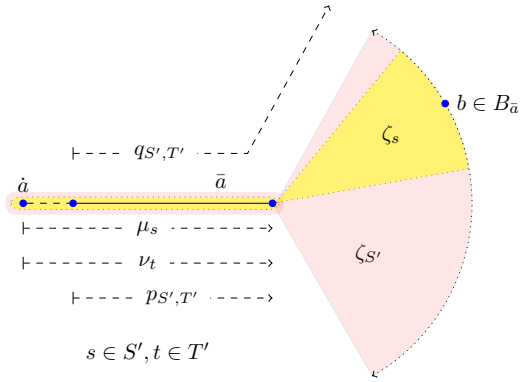


Figure 1 Intended interpretations of the predicates $p_{S',T'}$ and $q_{S',T'}$ in $\mathfrak{A}^+$ (and $\mathfrak{A}'$).

several) values of $s \in S'$. Discarding the element $\dot{a}$, we take the new predicate $p_{S',T'}$ to apply to the $(m-1)$-tuple $\bar{a}$ and the new predicate $q_{S',T'}$ to apply to the m-tuples $\bar{a}b$, where $b \in B_{\bar{a}}$.

We now construct the promised formula φ' in three steps, guided by the structure $\mathfrak{A}'$ just described. The first step is to write formulas reflecting the intended interpretations of the predicates $p_{S',T'}$. Indeed, we see immediately that, for all $S' \subseteq S$ and $T' \subseteq T$,

$$\mathfrak{A}' \models \forall^m \big(\bigwedge_{s \in S'} \mu_s \wedge \bigwedge_{t \in T'} \nu_t \to p_{S',T'} \big). \tag{13}$$

The second step is to write formulas reflecting the intended interpretations of the predicates $q_{S',T'}$, and specifically, the fact that they identify *all* $\zeta_{S'}$-witnesses for m-tuples of interest. Concretely, select an $(m-1)$-tuple $\bar{a}$ from A and suppose $\mathfrak{A} \models p_{S',T}[\bar{a}]$. Now let $\dot{a}$ and $B_{\bar{a}}$ be as chosen in the definition of $q^{\mathfrak{A}'}_{S',T'}$. Then $b \notin B_{\bar{a}}$ implies that $\dot{a}\bar{a}b$ does not satisfy ζ_s for any $s \in S'$. On the other hand, $\dot{a}\bar{a}b$ satisfies η_t for every $t \in T'$ as well as θ. Thus, writing, say, ψ for $(\bigwedge_{s \in S'} \neg\zeta_s \wedge \bigwedge_{t \in T'} \eta_t \wedge \theta)$, we see that $\mathfrak{A} \models \psi[\dot{a}\bar{a}b]$. Recalling that ψ° denotes the strongest quantifier-free formula over Σ^- entailed by ψ, we obviously have $\mathfrak{A}^- \models \psi^\circ[\dot{a}\bar{a}b]$, where $\mathfrak{A}^-$ denotes the reduct $\mathfrak{A}|\Sigma^- = \mathfrak{A}'|\Sigma^-$. But ψ° involves no predicates of arity $m+1$, whence $\mathfrak{A}^- \models \psi^\circ[\bar{a}b]$. (Observe how we are here exploiting variable-free notation: while ψ° is indeed a formula of $\mathcal{FLC}^{[m+1]}$, it involves only predicates in Σ^- , and therefore is simultaneously a formula of $\mathcal{FLC}^{[m]}$, which cannot "see" the element $\dot{a}$ in the tuple $\dot{a}\bar{a}b$.) Thus, since ψ° is also a formula over the signature Σ', we have shown that, for all $S' \subseteq S$ and $T' \subseteq T$,

$$\mathfrak{A}' \models \forall^{m-1} \big(p_{S',T'} \to \forall \big(\neg q_{S',T'} \to \big(\bigwedge_{s \in S'} \neg\zeta_s \wedge \bigwedge_{t \in T'} \eta_t \wedge \theta \big)^\circ \big) \big). \tag{14}$$

The final step in the construction of φ' requires us to define, for all subsets $S' \subseteq S$, a fluted ep-quantifier $\mathsf{Q}\langle \Sigma', M_{S'}, \Theta_{S'} \rangle$. To motivate the definition, let $\bar{a}$ again be any $(m-1)$-tuple from A, let S', T' be such that $\mathfrak{A}^+ \models p_{S',T'}[\bar{a}]$, and let $\dot{a}$ be the element selected in the definition of $q^{\mathfrak{A}^+}_{S',T'}$, so that we have

$$B_{\bar{a}} = \{b \in A : \mathfrak{A} \models \zeta_s[\dot{a}\bar{a}b] \text{ for some } s \in S'\} = \{b \in A : \mathfrak{A}' \models q_{S',T'}[\bar{a}b]\}.$$

Remembering that $\zeta_{S'} = \bigvee_{s \in S'} \zeta_s$, define the fluted star-types $\sigma = \mathrm{fst}^{\mathfrak{A}}_{\zeta_{S'}}[\dot{a}\bar{a}]$ (m-dimensional, over Σ) and $\sigma' = \mathrm{fst}^{\mathfrak{A}'}_{q_{S',T'}}[\bar{a}]$ ($(m-1)$-dimensional, over Σ'). Define in addition the $(m-1)$-dimensional fluted star-type over Σ^- by setting, for any fluted m-type τ^- over Σ^-,

$$\sigma^-(\tau^-) = |\{b \in B_{\bar{a}} : \mathfrak{A}^- \models \tau^-[\bar{a}b]\}|.$$

We see immediately by consideration of the set $B_{\bar{a}}$ that $\sigma/\Sigma^- = \sigma^- = \sigma'/\Sigma^-$. Warning: it will not in general be the case that σ^- arises as $\mathrm{fst}^{\mathfrak{A}^-}_{\zeta^-}[\bar{a}]$ for any quantifier-free formula ζ^- over the signature Σ^-. Indeed, $\mathfrak{A}^-$ interprets neither the predicate $q_{S',T'}$ nor the predicates of $\zeta_{S'}$, and is therefore insensitive to the extension of $B_{\bar{a}}$ used to define σ^-. Nevertheless, we have established:

$$\sigma/\Sigma^- = \sigma'/\Sigma^-. \tag{L1}$$

A little thought shows that σ satisfies various further properties. Fix some $s \in S'$. Since $\models \zeta_s \to \zeta_{S'}$, we see that the retract $\sigma{\restriction}\zeta_s$ is equal to the fluted star-type $\mathrm{fst}^{\mathfrak{A}}_{\zeta_s}[\dot{a}\bar{a}]$ and hence (from the fact that $\mathfrak{A} \models \varphi$), is M_s-bounded and satisfies Θ_s. Thus, we have:

$$\text{for all } s \in S',\ \sigma{\restriction}\zeta_s \text{ is } M_s\text{-bounded and satisfies } \Theta_s. \tag{L2}$$

Now fix some $t \in T'$. Since $\mathfrak{A} \models \varphi$, it follows immediately that, for every $b \in B_{\bar{a}}$, $\mathfrak{A} \models \eta_t[\dot{a}\bar{a}b]$, and indeed $\mathfrak{A} \models \theta[\dot{a}\bar{a}b]$, whence:

$$\text{for all } t \in T \text{ and all } \tau \text{ such that } \sigma(\tau) > 0,\ \models \tau \to \eta_t. \tag{L3}$$

$$\text{for all } \tau \text{ such that } \sigma(\tau) > 0,\ \models \tau \to \theta. \tag{L4}$$

Casting this discussion in terms of $\sigma' = \mathrm{fst}^{\mathfrak{A}'}_{q_{S',T'}}[\bar{a}]$ and writing $M_{S'} = \sum_{s\in S'} M_s$, we see that σ' is $M_{S'}$-bounded and satisfies the property that there exists a fluted star-type σ such that (L1)–(L4) hold. Crucially, this property can be naturally formulated using a formula of existential Presburger arithmetic. Letting $\bar{v}$ be a tuple of variables corresponding to the fluted $(m+1)$-types over Σ and $\bar{v}'$ a tuple of variables corresponding to the fluted m-types over Σ', we see that (L1) is a system of equations $A\bar{v} = A'\bar{v}'$, where A, A' are matrices with entries in $\{0,1\}$ depending only on the fixed ordering of the fluted $(m+1)$-types over Σ and the fixed ordering of the fluted m-types over Σ' and Σ^-. And certainly, (L3) and (L4) can be expressed as a single equation setting certain values in $\bar{v}$ to zero. Let us write $\Lambda(\bar{v},\bar{v}')$ for the conjunction of all the equations expressing (L1), (L3) and (L4). Considering that any retract $\sigma{\restriction}\zeta_s$ amounts to the zeroing of certain entries in σ, we may assume without loss of generality that the corresponding variables do not occur in Θ_s. (If they do, we may replace them by 0.) And in that case, (L2) is expressed by the conjunction $\bigwedge_{s\in S'}\Theta_s(\bar{v})$. Thus, we may formulate the above conditions on σ' as the requirement that (considered as a vector $\bar{v}'$ of length J' over $\mathbb{N}$) it satisfies the formula of Presburger arithmetic $\exists\bar{v}\big(\Lambda(\bar{v},\bar{v}') \wedge \bigwedge_{s\in S'}\Theta_s(\bar{v})\big)$. Writing $\Theta_s(\bar{v})$ as $\exists\bar{u}_s.\Xi_s(\bar{u}_s,\bar{v})$ for each $s \in S$, we obtain, by renaming variables to avoid clashes, the equivalent existential Presburger formula $\Theta_{S'}(\bar{v}') \equiv \exists\bar{v}\bar{u}\big(\Lambda(\bar{v},\bar{v}') \wedge \bigwedge_{s\in S'}\Xi_s(\bar{u}_s,\bar{v})\big)$, where $\bar{u}$ is the concatenation of the (disjoint) tuples $\bar{u}_s$ for $s \in S'$. Thus, we have shown:

$$\mathfrak{A}' \models \forall^{m-1}(p_{S',T'} \to \mathsf{Q}\langle\Sigma', M_{S'}, \Theta_{S'}\rangle q_{S',T'}). \tag{15}$$

Now we are ready to define φ' as the conjunction of the following formulas:

$$\bigwedge_{S'\subseteq S}\bigwedge_{T'\subseteq T}\forall^m\Big(\bigwedge_{s\in S'}\mu_s \wedge \bigwedge_{t\in T'}\nu_t \to p_{S',T'}\Big) \tag{16}$$

$$\bigwedge_{S'\subseteq S}\bigwedge_{T'\subseteq T}\forall^{m-1}(p_{S',T'} \to \mathsf{Q}\langle\Sigma', M_{S'}, \Theta_{S'}\rangle q_{S',T'}) \tag{17}$$

$$\bigwedge_{S'\subseteq S}\bigwedge_{T'\subseteq T}\forall^{m-1}\big(p_{S',T'} \to \forall\big(\neg q_{S',T'} \to \big(\bigwedge_{s\in S'}\neg\zeta_s \wedge \bigwedge_{t\in T'}\eta_t \wedge \theta\big)^\circ\big)\big). \tag{18}$$

By re-ordering of conjuncts, we see that φ' is an $\mathcal{FLC}^m_{\mathrm{ep}}$-formula. Moreover, it follows from (13), (14) and (15) that $\mathfrak{A}' \models \varphi'$. Hence, we have proved:

▶ **Lemma 9.** *If φ is satisfiable over some domain A, then so is φ'.*

We now prove a converse to Lemma 9. Suppose $\mathfrak{B}' \models \varphi'$, where $\mathfrak{B}'$ has domain B. We proceed to construct a model $\mathfrak{B} \models \varphi$ over the same domain. Let $\mathfrak{B}^- = \mathfrak{B}' \restriction \Sigma^-$. Notice that $\mathfrak{B}^-$ features no predicates of arity $m+1$, and none of the "new" predicates $p_{S',T'}$ or $q_{S',T'}$. We shall expand $\mathfrak{B}^-$ to a structure $\mathfrak{B}$ and then show that $\mathfrak{B} \models \varphi$. It suffices to specify, for every $a, b \in B$ and $\bar{a} \in B^{m-1}$, whether the tuple $a\bar{a}b$ is in the extension of each $(m+1)$-ary predicate of Σ. Equivalently, we must specify the fluted $(m+1)$-type of every tuple $a\bar{a}b$ in $\mathfrak{B}$.

Fix $a \in B$ and $\bar{a} \in B^{m-1}$. Let $S' = \{s \in S \mid \mathfrak{B}^- \models \mu_s[a\bar{a}]\}$, $T' = \{t \in T \mid \mathfrak{B}^- \models \nu_t[a\bar{a}]\}$ and $\zeta_{S'} = \bigvee_{s \in S'} \zeta_s$. Since all the μ_s and ν_t are Σ^--formulas, we could equivalently have replaced $\mathfrak{B}^-$ by $\mathfrak{B}'$ in the definitions of S' and T'. Define $\sigma' = (v'_1, \ldots, v'_{J'}) = \mathrm{fst}^{\mathfrak{B}'}_{q_{S',T'}}[\bar{a}]$ and $B_{a\bar{a}} = \{b \in B \mid \mathfrak{B}' \models q_{S',T'}[\bar{a}b]\}$. Notice that, since the sets S' and T' depend on a as well as $\bar{a}$, then so does the set $B_{a\bar{a}}$. It follows from (16) that $\mathfrak{B}' \models p_{S',T'}[\bar{a}]$, and from (17), that σ' is $M_{S'}$-bounded and satisfies the existential Presburger formula $\Theta_{S'}(\bar{v}') \equiv \exists \bar{v} \bar{u} \big(\Lambda(\bar{v}, \bar{v}') \wedge \bigwedge_{s \in S'} \Xi_s(\bar{u}_s, \bar{v}) \big)$. But $\Theta_{S'}(\bar{v}')$ asserts that $\bar{v}'$ is the vector representation of a fluted star-type σ' (over Σ') for which there exists an m-dimensional fluted star-type σ (over Σ) satisfying conditions (L1)–(L4). Letting $\sigma^- = \sigma'/\Sigma^-$, it follows from (L1) that $\sigma^- = \sigma/\Sigma^-$. We proceed to set the fluted $(m+1)$-type of all tuples $a\bar{a}b$, as b ranges over B; we shall do this in such a way that $\mathrm{fst}^{\mathfrak{B}}_{\zeta_{S'}}[a\bar{a}] = \sigma$. The plan is first to find all the required witnesses in the set $B_{a\bar{a}}$, and then to ensure that no unwanted witnesses appear outside this set.

We begin with the elements $b \in B_{a\bar{a}}$. We first partition $B_{a\bar{a}}$ into groups which are indistinguishable from the point of view of the signature Σ^-. Specifically, we write $\sigma^- = (v^-_1, \ldots, v^-_{J^-})$, and for each j^- $(1 \leq j^- \leq J^-)$, we let $B_{j^-} = \{b \in B_{a\bar{a}} \mid \mathfrak{B}^- \models \tau^-_{j^-}[\bar{a}, b]\}$. Writing $J^-_{a\bar{a}}$ for the set of indices j^- for which B_{j^-} is non-empty, we see that $|B_{j^-}| = v_{j^-}$ for all j^- $(1 \leq j^- \leq J^-)$, $v_{j^-} = 0$ for all $j^- \notin J^-_{a\bar{a}}$, and the family of sets $\{B_{j^-} \mid j^- \in J^-_{a\bar{a}}\}$ forms a partition of $B_{a\bar{a}}$.

Now consider just one cell of this partition, say B_{j^-}. We have

$$B_{j^-} = \{b \in B_{a\bar{a}} \mid \mathfrak{B}^- \models \tau^-_{j^-}[\bar{a}, b]\} = \{b \in B_{a\bar{a}} \mid \mathfrak{B}' \models \tau_{j'}[\bar{a}, b] \text{ for some } j' \text{ s.t. } \models \tau'_{j'} \to \tau^-_{j^-}\}.$$

And since $v^-_{j^-} = |B_{j^-}|$, we obtain

$$v^-_{j^-} = \sum \{v_{j'} \mid 1 \leq j' \leq J' \text{ and } \models \tau'_{j'} \to \tau^-_{j^-}\} = \sum \{v_j \mid 1 \leq j \leq J \text{ and } \models \tau_j \to \tau^-_{j^-}\},$$

the second equality arising from the fact that $\sigma/\Sigma^- = \sigma'/\Sigma^-$. Thus, we may choose, for each j such that $\models \tau_j \to \tau^-_{j^-}$, a fresh collection $B_{j^-,j}$ of v_j elements of B_{j^-}, and for each of these elements, b, set $\mathrm{ftp}^{\mathfrak{B}}[a\bar{a}b] = \tau_j$. Because $\models \tau_j \to \tau^-_{j^-}$, the only predicates being defined afresh here have arity $(m+1)$, so that these assignments represent an expansion of $\mathfrak{B}^-$. Once these assignments are made, the set B_{j^-} will contain v_j elements b such that $\mathrm{ftp}^{\mathfrak{B}}[a\bar{a}b] = \tau_j$ for all j such that $\models \tau_j \to \tau^-_{j^-}$. Repeating this procedure for every $j^- \in J^-_{a\bar{a}}$, and writing $J_{a\bar{a}} = \{j \mid 1 \leq j \leq J \text{ and } \models \tau_j \to \tau^-_{j^-} \text{ for some } j^- \in J^-_{a\bar{a}}\}$, we see that the set $B_{a\bar{a}}$ will contain v_j elements b such that $\mathrm{ftp}^{\mathfrak{B}}[a\bar{a}b] = \tau_j$ for all $j \in J_{a\bar{a}}$. On the other hand, $\sigma' = (v'_1, \ldots, v'_{J'}) = \mathrm{fst}^{\mathfrak{B}'}_{q_{S',T'}}[\bar{a}]$, so that $j^- \notin J^-_{a\bar{a}}$ implies $v^-_{j^-} = 0$ and hence

$$\sum \{v_j \mid 1 \leq j \leq J \text{ and } \models \tau_j \to \tau^-_{j^-}\} = 0,$$

since $\sigma \restriction \Sigma^- = \sigma^-$. That is, $v_j = 0$ for all $j \notin J_{a\bar{a}}$, and we have shown that $B_{a\bar{a}}$ contains v_j elements b such that $\mathrm{ftp}^{\mathfrak{B}}[a\bar{a}b] = \tau_j$, for all j $(1 \leq j \leq J)$.

Next, we deal with the elements $b \in B \setminus B_{a\bar{a}}$. By definition, $\mathfrak{B}' \not\models q_{S',T'}[\bar{a}b]$, so that, abbreviating $\bigwedge_{s\in S'} \neg\zeta_s \wedge \bigwedge_{t\in T'} \eta_t \wedge \theta$ by ψ, (18) yields $\mathfrak{B}' \models \psi^\circ[\bar{a}b]$. Writing $\tau^- = \mathrm{ftp}^{\mathfrak{B}^-}[\bar{a}, b]$, and noting that ψ° is a Σ^--formula, we have $\models \tau^- \to \psi^\circ$. By Lemma 8, let $\tilde{\tau}$ be a fluted $(m+1)$-type over Σ extending τ^- such that $\models \tilde{\tau} \to \psi$, and set $\mathrm{ftp}^{\mathfrak{B}}[a\bar{a}b] = \tilde{\tau}$. In particular, $\mathfrak{B} \not\models \zeta_{S'}[a\bar{a}b]$. Thus, we have set the fluted $(m+1)$-type of all $(m+1)$-tuples $a\bar{a}b$, as b ranges over B. Repeating this process for each m-tuple $a\bar{a}$, which we may do independently, the structure $\mathfrak{B}$ will have been completely defined. In the course of this construction, we have shown that, for every $b \notin B_{a\bar{a}}$, $\mathfrak{B} \not\models \zeta_{S'}[a\bar{a}b]$. But we showed above that the number of elements $b \in B_{a\bar{a}}$ such that $\mathfrak{B} \models \tau_j[a\bar{a}b]$ is v_j, where $\sigma = (v_1, \ldots, v_J)$. It follows that $\mathrm{fst}^{\mathfrak{B}}_{\zeta_{S'}}[a\bar{a}] = \sigma$, as required.

In constructing $\mathfrak{B}$, we have secured the following property. Take any m-tuple $a\bar{a}$ with $a \in B$ and $\bar{a} \in B^{m-1}$, and define $S' = \{s \in S \mid \mathfrak{B}^- \models \mu_s[a\bar{a}]\}$, $T' = \{t \in T \mid \mathfrak{B}^- \models \nu_t[a\bar{a}]\}$, $\zeta_{S'} = \bigvee_{s\in S'} \zeta_s$ and $\sigma' = \mathrm{fst}^{\mathfrak{B}'}_{q_{S',T'}}[\bar{a}]$. Then $\mathrm{fst}^{\mathfrak{B}}_{\zeta_{S'}}[a\bar{a}] = \sigma$, where σ is some fluted star-type over Σ satisfying (L1)–(L4), whose existence is guaranteed by the fact that σ' satisfies the existential Presburger formula $\Theta_{S'}$. We have used (L1) in the construction of $\mathfrak{B}$. We now use (L2)–(L4) to ensure that $\mathfrak{B} \models \varphi$.

We first show that, for all $s \in S$, $\mathfrak{B} \models \forall^m(\mu_s \to \mathsf{Q}\langle\Sigma, M_s, \mathcal{L}_s\rangle\zeta_s)$. For consider any m-tuple $a\bar{a}$ with $a \in B$ and $\bar{a} \in B^{m-1}$ such that $\mathfrak{B} \models \mu_s[a\bar{a}]$, and let S', $\zeta_{S'}$, and σ be as just defined. Since $s \in S'$, we have $\models \zeta_s \to \zeta_{S'}$, and hence $\mathrm{fst}^{\mathfrak{B}}_{\zeta_s}[a\bar{a}] = \sigma{\restriction}\zeta_s$, which, by (L2), is M_s-bounded and satisfies $\Theta_{S'}$. We next show that, for all $t \in T$, $\mathfrak{B} \models \forall^m(\nu_t \to \forall\eta_t)$. Again, consider any m-tuple $a\bar{a}$ with $a \in B$ and $\bar{a} \in B^{m-1}$ such that $\mathfrak{B} \models \nu_t[a\bar{a}]$, and let S', T' and σ be as just defined. Pick any $b \in B$. If b is in the set $B_{a\bar{a}}$ used in the construction of $\mathfrak{B}$, then $\sigma(\mathrm{ftp}^{\mathfrak{B}}[a\bar{a}b]) > 0$, whence by (L3), we have $\mathfrak{B} \models \eta_t[a\bar{a}b]$. On the other hand, if $b \notin B_{a\bar{a}}$, then $\mathrm{ftp}^{\mathfrak{B}}[a\bar{a}b]$ was set to some $\tilde{\tau}$ entailing $\bigwedge_{s\in S'} \neg\zeta_s \wedge \bigwedge_{t\in T'} \eta_t \wedge \theta$, so that, again $\mathfrak{B} \models \eta_t[a\bar{a}b]$. Finally, to show that $\mathfrak{B} \models \forall^{m+1}\theta$, we proceed as in the previous case, but using (L4) instead of (L3). Thus we have proved:

▶ **Lemma 10.** *If φ' is satisfiable over some domain B, then so is φ.*

▶ **Lemma 11.** *The formula φ' can be computed in time bounded by an exponential function of $\|\varphi\|$. Moreover, $\#(\varphi')$ is bounded by an exponential function of $\#(\varphi)$.*

Proof. Recalling that φ has the form

$$\bigwedge_{s\in S} \forall^m(\mu_s \to \mathsf{Q}\langle\Sigma, M_s, \Theta_s\rangle\zeta_s) \wedge \bigwedge_{t\in T} \forall^m(\nu_t \to \forall\eta_t) \wedge \forall^{m+1}\theta$$

over signature Σ', let $M = \sum_{s\in S} M_s$. Writing φ' given by (16)–(18) in the same form, over signature Σ', we notice first of all that the sizes of the index sets certainly only increase by an exponential. Let M' be the sum of all the numbers occurring in the fluted ep-quantifiers of φ', i.e. $\sum\{2^{|T|} \cdot M_{S'} \mid S' \subseteq S\}$, whence $\log M' \leq |S| + |T| + \log M \leq \#(\varphi)$. We noted above that $|\Sigma'| \leq |\Sigma| + 2^{|S|+|T|+1} \leq \#(\varphi) + 2^{\#(\varphi)+1}$. To show that $\log\|\varphi'\|$ is also bounded by an exponential function of $\#(\varphi)$, we need only show that $\log\|\Theta_{S'}\|$ is bounded by an exponential function of $\#(\varphi)$, for each $S' \subseteq S$. Fix some $S' \subseteq S$, then, and recall that

$$\Theta_{S'}(\bar{v}') \equiv \exists\bar{v}\{\bar{u}_s\}_{s\in S'}\big(\Lambda(\bar{v}, \bar{v}') \wedge \bigwedge_{s\in S'} \Xi_s(\bar{u}_s, \bar{v})\big).$$

Let e be the maximum number of existentially quantified variables in any Θ_s as s ranges over S; certainly, $e \leq \|\varphi\|$. The number of free variables in $\Theta_{S'}$ is simply the number of fluted m-types over Σ' and hence at most $2^{|\Sigma'|+1}$. The number of existentially quantified variables in in $\Theta_{S'}$ is bounded by $|S'|.e + 2^{|\Sigma|+1}$. Moreover, Λ consists of $2^{|\Sigma^-|+1}$ equations

featuring at most $2^{|\Sigma|+1} + 2^{|\Sigma'|+1}$ terms. Finally we evidently have $\|\bigwedge_{s\in S'} \Theta_s\| \leq |S'| \cdot \|\varphi\|$. Adding all of these together, we see that $\log\|\Theta_{S'}\|$ is bounded by an exponential function of $\#(\varphi)$, as required. ◀

▶ **Lemma 12.** *Let $m \geq 1$. Any satisfiable formula of $\mathcal{FLC}^{m+1}_{\text{ep}}$ has a finite model. Moreover, the satisfiability problem for $\mathcal{FLC}^{m+1}_{\text{ep}}$ is in m-NExpTime, measured in terms of the effective size of the input.*

Proof. The case $m = 1$ is Lemma 4. The inductive step is from Lemmas 9, 10 and 11. ◀

We may now prove Theorem 1. Let a formula φ of $\mathcal{FLC}^{m+1}$ be given. By Lemma 3, we may suppose without loss of generality that φ is in normal form (7). Further, we may replace any sub-formula $\exists_{[=M_s]}\zeta_s$ in φ by the equivalent sub-formula $\mathsf{Q}\langle\Sigma, M_s, \Theta_s\rangle\zeta_s$, where Θ_s is simply the equation $v_1 + \cdots + v_J = M_s$, thus obtaining, in time bounded by an exponential function of $\|\varphi\|$, an $\mathcal{FLC}^{m+1}_{\text{ep}}$-formula φ_0 equivalent to φ. Note that $\#(\varphi_0)$ is bounded by a polynomial function of $\|\varphi\|$. The result then follows from Lemma 12.

5 The semi-fluted fragment

Finally, we consider the semi-fluted fragment with counting, $\mathcal{SFC}$. The analysis proceeds largely as for $\mathcal{FLC}$. We remind the reader that variable-free notation is no longer available in this case. Of course, we could employ predicate-functor-style syntax here; however, for the little material remaining, this seems excessive.

Let Σ be a purely relational signature and m a positive integer. A *semi-fluted m-atom* over Σ is a formula of the form $p(\bar{x})$ or its negation where either: (i) $p \in \Sigma$ and $\bar{x}$ is a contiguous sequence of variables $x_\ell, \ldots, x_m$, or (ii) $p \in \Sigma \cup \{=\}$ and $\bar{x}$ is a sequence of at most two variables chosen (repeats allowed) from the set $\{x_{m-1}, x_m\}$. A *semi-fluted m-literal* is either a semi-fluted m atom or its negation. Thus, we have the same restriction on argument patterns as in fluted logic generally, except that semi-fluted m-literals of arity at most 2 may feature the variables x_{m-1} and x_m in any order we like. A *semi-fluted m-type* (over Σ) is a maximal consistent set of semi-fluted m-literals (over Σ). For these purposes, consistency takes account of the special meaning of the equality predicate: thus, $\{p(x_1), \neg p(x_2), x_1 = x_2\}$ is not consistent. As before, where convenient, we identify a semi-fluted m-type τ with the conjunction of its members and call τ *reflexive* if it contains $x_{m-1} = x_m$. We remark that semi-fluted 1- and 2-types are simply maximal consistent sets of literals (atomic formulas or their negations) in the variables x_1 and x_2, and are usually referred to in the literature simply as 1- and 2-*types*. If τ is a semi-fluted literal of arity $m \geq 2$, define $\text{tp}_1(\tau)$ to be the set of literals in τ featuring only the variable x_{m-1}. (Note that replacing the variable x_{m-1} in $\text{tp}_1(\tau)$ by x_1 would yield a 1-type.) A *semi-fluted star-type of dimension m over Σ* is a multiset of semi-fluted $(m+1)$-types over Σ at most one of which is reflexive, subject to the additional condition that the value $\text{tp}_1(\tau)$ is the same for all τ occurring (i.e. having non-zero multiplicity) in σ. A semi-fluted star-type is M-bounded if the sum of its multiplicities is M. By enumerating the semi-fluted $(m+1)$-types over Σ in some fixed order, we may regard any semi-fluted star-type as a vector of cardinal numbers.

If $\mathfrak{A}$ is a structure interpreting Σ, and $\bar{a}$ is an m-tuple of elements from A (repeats allowed), there is a unique semi-fluted m-type satisfied by $\bar{a}$; we denote this by $\text{sftp}^{\mathfrak{A}}[\bar{a}]$. If, in addition, ζ is a quantifier-free formula of $\mathcal{SFC}^{m+1}$, then we may define a semi-fluted star-type σ of dimension m by setting, for each semi-fluted $(m+1)$-type τ over Σ, $\sigma(\tau) = |\{b \in A : \mathfrak{A} \models \tau[\bar{a}b] \text{ and } \mathfrak{A} \models \zeta[\bar{a}b\}]|$. Notice incidentally that for all τ occurring in σ, we have $\mathfrak{A} \models \text{tp}_1(\tau)[a]$, where a is the last element of $\bar{a}$, whence these 1-types are indeed

all the same. We call σ the *semi-fluted ζ-star-type of $\bar{a}$ in $\mathfrak{A}$*, and denote it by $\mathrm{sfst}^{\mathfrak{A}}_{\zeta}[\bar{a}]$. As with fluted ζ-star-types, so with their semi-fluted cousins, we can think of the tuple $\bar{a}$ as "emitting" various "ζ-rays", which are "absorbed" by various elements of $\mathfrak{A}$. The principal difference is that the (non-empty) semi-fluted ζ-star-type of $\bar{a}$ gives us the 1-type satisfied by the final element a of $\bar{a}$, and indeed of the full 2-types (not just the fluted 2-types) of the pairs ab, for b an element absorbing one of the ζ-rays emitted by $\bar{a}$.

Normal forms analogous to those of Lemma 3 are easily obtainable:

▶ **Lemma 13.** *Let φ be a formula of $\mathcal{SFC}^{m+1}$ ($m \geq 1$). Then we may compute, in time bounded by a polynomial function of $\|\varphi\|$, an $\mathcal{SFC}^{m+1}$-sentence satisfiable over the same domains as φ, and having the form*

$$\bigwedge_{s\in S} \forall x_1 \ldots \forall x_m(\mu_s \to \exists_{[=M_s]} x_{m+1}.\zeta_s) \wedge \bigwedge_{t\in T} \forall x_1 \ldots \forall x_m(\nu_t \to \forall \eta_t) \wedge \forall x_{m+1}.\theta, \tag{19}$$

where S and T are index sets, the μ_s and ν_t are quantifier-free $\mathcal{SFC}^m$-formulas, the ζ_s, η_t and θ are quantifier-free $\mathcal{SFC}^{m+1}$-formulas, and the M_s are positive integers.

The proof proceeds as for Lemma 3, almost verbatim.

Extending the notion of ep-quantifiers to the semi-fluted case is again routine. If ζ is a quantifier-free formula of $\mathcal{SFC}^{m+1}$, we allow formulas φ of the form $\mathsf{Q}\langle\Sigma, M, \Theta\rangle\zeta$. For any structure $\mathfrak{A}$ interpreting a signature $\Sigma' \supseteq \Sigma$ and any m-tuple $\bar{a}$ of elements from A, we declare:

$$\mathfrak{A} \models \varphi[\bar{a}] \text{ if and only if } \mathrm{sfst}^{(\mathfrak{A}\restriction\Sigma)}_{\zeta}[\bar{a}] \text{ is } M\text{-bounded and satisfies } \Theta(\bar{v}), \tag{20}$$

just as with (8). We then define $\mathcal{SFC}^{m+1}_{\mathrm{ep}}$ to be the set of formulas φ given by

$$\bigwedge_{s\in S} \forall x_1 \ldots \forall x_m(\mu_s \to \mathsf{Q}\langle\Sigma, M_s, \Theta_s\rangle x_{m+1}.\zeta_s) \wedge \bigwedge_{t\in T} \forall x_1 \ldots \forall x_m(\nu_t \to \forall x_{m+1}.\eta_t) \wedge \forall x_1 \ldots \forall x_{m+1}.\theta, \tag{21}$$

where Σ is a relational signature, the μ_s and ν_t are quantifier-free $\mathcal{SFC}^m$-formulas over Σ, the ζ_s, η_t and θ are quantifier-free $\mathcal{SFC}^{m+1}$-formulas over Σ, the M_s are positive integers and the Θ_s are formulas of existential Presburger arithmetic with free variables corresponding to the semi-fluted $(m+1)$-types over Σ. Of course, this parallels the definition of $\mathcal{FLC}^{m+1}_{\mathrm{ep}}$ given in (9). Again, semi-fluted ep-quantifiers give us no expressive power beyond the ordinary counting quantifiers, since the translation (10) (with the counting quantifier taken to bind the variable x_{m+1}) holds also when ζ is only semi-fluted. We may define the *effective size*, $\#(\varphi)$ of an $\mathcal{SFC}^m_{\mathrm{ep}}$-formula φ exactly as with $\mathcal{FLC}^m_{\mathrm{ep}}$-formulas.

At this point, we are in a position to sketch the proof of Theorem 2. Let an $\mathcal{SFC}^{m+1}$-formula φ be given ($m \geq 1$). By Lemma 13, we may assume without loss of generality that φ is in the form (19). Clearly, this may be converted, in time bounded by an exponential function of $\|\varphi\|$, and with at most a polynomial increase in $\#(\varphi)$, into an $\mathcal{SFC}^{m+1}_{\mathrm{ep}}$-formula of the form (21). The reduction described in Sec. 4 can then be repeated almost verbatim, since the transformation of φ into φ' never affects the two final variables. Lemmas 9, 10 and 11 then continue to hold. This allows us to transform any formula of $\mathcal{SFC}^{m+1}$ ($m \geq 2$) eventually into a formula of $\mathcal{SFC}^2_{\mathrm{ep}}$ satisfiable over the same domains. But we have already argued that any $\mathcal{SFC}^2_{\mathrm{ep}}$-formula can be translated into equivalent formula of $\mathcal{C}^2$. Since the satisfiability and finite satisfiability problems for $\mathcal{C}^2$ are in NExpTime, we thus obtain Theorem 2, as promised.

The complexity bound we may extract from the above argument is rather weak. The translation given above from a $\mathcal{SFC}^2_{\text{ep}}$-formula φ to an equivalent $\mathcal{C}^2$-formula ψ' runs in time bounded by a doubly exponential function of $\|\varphi\|$, and hence a triply exponential function of $\#(\varphi)$. Since the satisfiability and finite satisfiability problems for $\mathcal{C}^2$ are in NExpTime, this means that the corresponding problems for $\mathcal{SFC}^2_{\text{ep}}$ are in non-deterministic time bounded by quadruply exponential time as a function of $\#(\varphi)$. This results in an upper complexity bound of $(m+3)$-NExpTime for the satisfiability and finite satisfiability problems for $\mathcal{SFC}^{m+1}_{\text{ep}}$ $(m \geq 1)$. We claim that the satisfiability and finite satisfiability problems for $\mathcal{SFC}^{m+1}$ in fact remain in m-NExpTime. However, the proof appears to require a modified version of existing proofs of the complexity bounds for $\mathcal{C}^2$, in order to accommodate semi-fluted ep-quantifiers. Such a reconstruction is beyond the scope of the current paper.

6 Discussion

For $m \geq 2$, the upper complexity bound of m-NExpTime for $\mathcal{FLC}^{m+1}$ in Theorem 1 is laxer than the corresponding upper complexity bound of $(m-1)$-NExpTime for $\mathcal{FL}^{m+1}$ from [17]. The best known lower complexity bound on satisfiability for both logics is $\lfloor (m+1)/2 \rfloor$-NExpTime-hard, from the same source. It is currently not known how to close this gap. It is plausible that, for $m \geq 2$, the upper bound given in this extended abstract for $\mathcal{FLC}^{m+1}$ could be reduced by one exponential, by adapting the procedure of in [17] for $\mathcal{FL}^3$. The probable difficulty of doing so coupled with the fact that a complexity gap would remain for $\mathcal{FLC}^5$ and above acts, however, as a deterrent to trying.

It is shown in [18] that the satisfiability and finite satisfiability problems for the fluted fragment, $\mathcal{FL}$ remain decidable even in the presence of a distinguished binary predicate required to be interpreted as a transitive relation (equality is also permitted); with just two transitive relations (or three transitive relations without equality), however, decidability is lost. The question arises as to whether a single transitive relation can be added to $\mathcal{FLC}$ without losing decidability of satisfiability. The argumentation of Sec. 4 will reduce this problem (with blow-up given by the same towers of exponentials) to the corresponding problem for $\mathcal{FLC}^2$ with a single transitive relation. However, this latter problem appears to be open.

We noted above that the fluted ep-quantifiers introduced here do not extend the expressive power of ordinary counting quantifiers. In this regard, they do less work than the "existential Presburger formulas" of [3], which strictly extend the expressive power of $\mathcal{C}^2$, permitting, saliently, counting *modulo* k for $k \geq 2$. The objects referred to as "*behaviours*" in that paper play a role very similar to the fluted star-types considered here, except that the values they assign are not integers, but *semi-linear sets* of integers. Unifying these approaches seems therefore to be a natural line of future enquiry.

References

1 H. Andréka, J. van Benthem, and I. Németi. Modal languages and bounded fragments of predicate logic. *Journal of Philosophical Logic*, 27:217–274, 1998.

2 V. Bárány, B. ten Cate, and L. Segoufin. Guarded negation. *Journal of the ACM*, 62(3):22:1–26, 2015.

3 M. Benedikt, E. Kostylev, and T. Tan. Two-variable logic with ultimately periodic counting. In A. Dawar A. Czumaj and E. Merelli, editors, *Proc. of ICALP 2020*, LIPIcs, pages 112:1–112:16. Schloß Dagstuhl - Leibniz-Zentrum für Informatik, 2020.

4 I. Borosh and L. Treybig. Bounds on the positive integral solutions of linear Diophantine equations. *Proceedings of the American Mathematical Society*, 55(2):299–304, 1976.

5 E. Grädel. On the restraining power of guards. *Journal of Symbolic Logic*, 64:1719–1742, 1999.
6 E. Grädel, M. Otto, and E. Rosen. Two-variable logic with counting is decidable. In *Logic in Computer Science*, pages 306–317. IEEE, 1997.
7 C. Haase. A survival guide to Presburger arithmetic. *ACM SIGLOG News*, 5(3):67–82, 2018.
8 D. Hilbert and W. Ackermann. *Grundzüge der mathematischen Logik*. Springer, Berlin, 1st edition, 1928.
9 D. Hilbert and W. Ackermann. *Principles of Mathematical Logic*. Chelsea, New York, 1950.
10 E. Kieroński and S. Rudolph. Finite model theory of the triguarded fragment and related logics. To appear, *Logic in Computer Science* (LICS), 2021; arXiV preprint `arXiv:2101.08377v2`.
11 L. Löwenheim. Über Möglichkeiten im Relativkalkül. *Math. Annalen*, 76:447–470, 1915.
12 L. Pacholski, W. Szwast, and L. Tendera. Complexity results for two-variable first-order logic with counting. *SIAM Journal of Computing*, 29:1083–1117, 1999.
13 C. Papadimitriou. On the complexity of integer programming. *Journal of the ACM*, 28(4):765–768, 1981.
14 L. Pottier. Minimal solutions of linear Diophantine systems : bounds and algorithms. In R. Book, editor, *Proc. Rewriting Techniques and Applications*, volume 488 of *LNCS*, pages 162–173, Berlin, 1991. Springer.
15 I. Pratt-Hartmann. Complexity of the two-variable fragment with counting quantifiers. *Journal of Logic, Language and Information*, 14(3):369–395, 2005.
16 I. Pratt-Hartmann. The finite satisfiability problem for two-variable, first-order logic with one transitive relation is decidable. *Mathematical Logic Quarterly*, 64(3):218–248, 2018.
17 I. Pratt-Hartmann, W. Szwast, and L. Tendera. The fluted fragment revisited. *Journal of Symbolic Logic*, 84(3):1020–1048, 2019.
18 Ian Pratt-Hartmann and Lidia Tendera. The fluted fragment with transitive relations, 2020. `arXiv:2006.11169`.
19 W. Purdy. Fluted formulas and the limits of decidability. *Journal of Symbolic Logic*, 61(2):608–620, 1996.
20 W. Purdy. Complexity and nicety of fluted logic. *Studia Logica*, 71:177–198, 2002.
21 W. V. Quine. Variables explained away. *Proceedings of the American Philosophical Society*, 104(3):343–347, 1960.
22 W. V. Quine. On the limits of decision. In *Proceedings of the 14th International Congress of Philosophy*, volume III, pages 57–62. University of Vienna, 1969.
23 S. Rudolph and M. Šimkus. The triguarded fragment of first-order logic. In *Proceedings of 22nd International Conference for Programming, Artificial Intelligence and Reasoning (LPAR-22)*, volume 57, pages 604–619. EPiC Series in Computing, 2018.
24 S. Schmitz. Complexity hierarchies beyond Elementary. *ACM Transactions on Computation Theory*, 8(1:3):1–36, 2016.
25 D. Scott. A decision method for validity of sentences in two variables. *Journal Symbolic Logic*, 27:477, 1962.

Guarded Kleene Algebra with Tests: Coequations, Coinduction, and Completeness

Todd Schmid
Department of Computer Science, University College London, UK

Tobias Kappé
Department of Computer Science, Cornell University, Ithaca, NY, USA

Dexter Kozen
Department of Computer Science, Cornell University, Ithaca, NY, USA

Alexandra Silva
Department of Computer Science, University College London, UK

Abstract

Guarded Kleene Algebra with Tests (GKAT) is an efficient fragment of KAT, as it allows for almost linear decidability of equivalence. In this paper, we study the (co)algebraic properties of GKAT. Our initial focus is on the fragment that can distinguish between unsuccessful programs performing different actions, by omitting the so-called *early termination axiom*. We develop an operational (coalgebraic) and denotational (algebraic) semantics and show that they coincide. We then characterize the behaviors of GKAT expressions in this semantics, leading to a coequation that captures the covariety of automata corresponding to these behaviors. Finally, we prove that the axioms of the reduced fragment are sound and complete w.r.t. the semantics, and then build on this result to recover a semantics that is sound and complete w.r.t. the full set of axioms.

2012 ACM Subject Classification Theory of computation → Program reasoning

Keywords and phrases Kleene algebra, program equivalence, completeness, coequations

Digital Object Identifier 10.4230/LIPIcs.ICALP.2021.142

Category Track B: Automata, Logic, Semantics, and Theory of Programming

Related Version *Full Version*: https://arxiv.org/abs/2102.08286

Funding *Tobias Kappé*: DARPA grant HR001120C0107 (Pronto)
Dexter Kozen: NSF grant CCF-2008083
Alexandra Silva: ERC Consolidator Grant AutoProbe (101002697) and a Royal Society Wolfson Fellowship

1 Introduction

Kleene algebra with tests (KAT) [17] was introduced in the early 90's as an extension of Kleene algebra (KA), the algebra of regular expressions. The core idea of the extension was simple: consider regular languages over a two-sorted alphabet, in which one sort represents Boolean tests and the other denotes basic program actions. This seemingly simple extension enables an important application for regular languages in reasoning about imperative programs with basic control flow structures like branches (**if-then-else**) and loops (**while**). KAT largely inherited the properties of KA: a language model [22], a Kleene theorem [19], a sound and complete axiomatization [22], and a PSPACE decision procedure for equivalence [8].

In 2014, a specialized KAT called NetKAT [4] was proposed to program software-defined networks. NetKAT was later extended with a probabilistic choice operator that enabled the modelling of randomized protocols [9]. Interestingly, there exists a decision procedure for NetKAT program equivalence that enables practical verification of reachability in networks

with thousands of nodes and links, which seems to scale almost linearly despite the PSPACE-completeness of this problem [10, 35]. This raised the question: do practical NetKAT programs belong to a fragment of KAT that has more favorable properties than the full language?

Recently, this question was answered positively [34], in the form of *Guarded Kleene Algebra with Tests* (GKAT), a fragment of KAT obtained by adding a Boolean guard to the non-deterministic choice and iteration operators so that they correspond exactly to the standard **if-then-else** and **while** constructs. GKAT is expressive enough to capture all programs used in network verification while allowing for almost linear time[1] decidability of equivalence, thereby explaining the experimental results observed in NetKAT.

The use of GKAT as a framework for program analysis also raises further questions about recovering the properties of KAT on the level of GKAT. Is there a class of automata that provides a Kleene theorem? Is there a sound and complete axiomatization of GKAT equivalence? The original paper [34] gave incomplete answers to these questions. First, it proposed a class of *well-nested* automata that can be used to describe the semantics of all GKAT programs, but left open whether this class covered all automata that accept the behaviors of GKAT programs. Second, GKAT was axiomatized under the assumption of *early termination*: intuitively, referring to a semantics of imperative programs where programs that fail immediately are equated to programs that fail eventually. This semantics, though useful, is too coarse in contexts where program behavior prior to failure matters.

In this paper, we take a new perspective on the semantics of GKAT programs and their corresponding automata, using coequations. Coequations provide the right tool to characterize fragments of languages as they enable a precise way to remove unwanted traces. We are then able to give a precise characterization of the behaviors of GKAT programs and prove a completeness theorem for each of the fragments of interest.

Our contributions. In a nutshell, the contributions of this paper are the following:

1. We give a denotational model for GKAT without early termination by representing the behavior as a certain kind of tree. This allows us to design two coequations: one characterizing the behaviors denoted by GKAT expressions, and another capturing only the behaviors of GKAT expressions that terminate early.
2. We obtain two completeness results for GKAT: one for the model of the previous item and the axiomatization of [34] without the early termination axiom; and building on this, another for the full axiomatization. The former is new; the latter provides an alternative proof to the completeness theorem presented in [34].
3. A concrete example of a well-nested GKAT automaton with a non-well-nested quotient. This settles an open question of [34] and closes the door on an alternative proof of completeness based on well-nested automata.

2 Guarded Kleene Algebra with Tests

At its heart, *Guarded Kleene Algebra with Tests* (GKAT) is an algebraic theory of imperative programs. Expressions in GKAT are concise formulas for WHILE programs [23], which are built inductively from actions and tests with sequential composition and the classic programming constructs of branches and loops: **if** b **then** e **else** f and **while** b **do** e.

[1] $O(n\alpha(n))$, where $\alpha(n)$ is the inverse of Ackermann's function

Union Axioms

U1. $e +_b e \equiv e$

U2. $e +_b f \equiv f +_{\bar{b}} e$

U3. $(e +_b f) +_c g \equiv e +_{b \wedge c} (f +_c g)$

U4. $e +_b f \equiv b \cdot e +_b f$

U5. $e \cdot g +_b f \cdot g \equiv (e +_b f) \cdot g$

Sequence Axioms

S1. $(e \cdot f) \cdot g \equiv e \cdot (f \cdot g)$

S2. $0 \cdot e \equiv 0$

S3. $e \cdot 0 \equiv 0$

S4. $1 \cdot e \equiv e$; S5. $e \equiv e \cdot 1$

S6. $b \cdot c \equiv b \wedge c$

Loop Axioms

W1. $e^{(b)} \equiv e \cdot e^{(b)} +_b 1$

W2. $(ce)^{(b)} \equiv (e +_c 1)^{(b)}$

W3. $\dfrac{E(e) \equiv 0 \quad g \equiv eg +_b f}{g \equiv e^{(b)} \cdot f}$

Figure 1 Axioms for GKAT-expressions. Here, $e, f, g \in \mathrm{Exp}$ and $b, c \in \mathrm{BExp}$.

Formally, these expressions are drawn from a two-sorted language of *tests* and *programs*. The tests are built from a finite set of ***primitive tests*** T, as follows:

$$\mathrm{BExp} \ni b, c ::= 0 \mid 1 \mid t \in T \mid \bar{b} \mid b \wedge c \mid b \vee c.$$

Here, 0 and 1 are understood as the constant tests **false** and **true** respectively, $\bar{b}$ denotes the negation of b, and $\wedge$ and $\vee$ are conjunction and disjunction, respectively. We will use A to denote the set of ***atomic tests*** (or just ***atoms***), Boolean expressions of the form $d_1 \wedge \cdots \wedge d_l$, where $d_i \in \{t_i, \bar{t}_i\}$ for each $i \leqslant l$ and $\{t_i \mid i \leqslant l\}$ is a fixed enumeration of T. It is well known that any $b \in \mathrm{BExp}$ can be written equivalently as the disjunction of the atoms $a \in A$ that imply b under the laws of Boolean algebra. We will often identify each Boolean expression $b \in \mathrm{BExp}$ with this set of atoms and write $b \subseteq A$ or $a \in b$.

Programs are built from tests and a finite set of ***primitive programs*** or ***actions*** Σ, disjoint from T. Formally, programs are generated by the grammar

$$\mathrm{Exp} \ni e, f ::= b \in \mathrm{BExp} \mid p \in \Sigma \mid e \cdot f \mid e +_b f \mid e^{(b)}$$

Here, a test b abbreviates the statement **assert** $\boldsymbol{b}$, the operator $\cdot$ is sequential composition, $e +_b f$ is shorthand for **if** $\boldsymbol{b}$ **then** $\boldsymbol{e}$ **else** $\boldsymbol{f}$ and $e^{(b)}$ is shorthand for **while** $\boldsymbol{b}$ **do** $\boldsymbol{e}$.

GKAT programs satisfy standard properties of imperative programs. For instance, swapping the branches of an **if-then-else** construct should not make a difference, provided that we also negate the condition; that is, the semantics of $e +_b f$ should coincide with that of $f +_{\bar{b}} e$. The rules in Figure 1 axiomatize equivalences between programs. Together with the axioms of Boolean algebra, these generate a congruence $\equiv$ on Exp.

Some remarks are in order for axiom W3. The right-hand premise states that an expression g has some self-similarity in the sense that it is equivalent to checking whether b holds, in which case it runs e followed by recursing at g, and otherwise running f. Intuitively, this says that g is loop-like, matching the conclusion that g is equivalent to $e^{(b)} \cdot f$. However, this conclusion may not make sense when based on just the second premise. Specifically, if we choose e, f, g and b to be 1, we can show that the premise holds and derive $1 \equiv 1^{(1)} \cdot 1$, which is to say that **assert true** is equivalent to **(while true do assert true); assert true**. Intuitively, this should be false: the first program terminates successfully and immediately, but the second program does not. The problem is that the loop body does not perform any actions that affect the state and make progress towards the end of the loop.

This is remedied by the left-hand premise, which distinguishes loop bodies that can accept immediately from those that cannot. It plays the same role as the *empty word property* in Salomaa's axiomatization of the algebra of regular events [31]. Formally, given $e \in \mathrm{Exp}$, the Boolean expression $E(e)$ is defined inductively by setting $E(p) = 0$, $E(b) = b$, and

$$E(e \cdot f) = E(e) \wedge E(f) \qquad E(e +_b f) = (b \wedge E(e)) \vee (\bar{b} \wedge E(f)) \qquad E(e^{(b)}) = \bar{b}$$

We call e ***productive*** if $E(e) \equiv 0$. Axioms W2 and W3 are analogues of Salomaa's axioms A_{11} and R2 [31]. Specifically, W2 says that non-productive loop iterations do not contribute to the semantics. This allows the use of W3 to reason about loops in general, for instance to prove $e^{(b)} \equiv e^{(b)} \cdot \overline{b}$, which says that the loop condition is false when a loop ends [34].

Axiom S3 identifies a program that fails eventually with the program that fails immediately. As a consequence, $\equiv$ cannot distinguish between processes that loop forever, like $p^{(1)}$ and $q^{(1)}$, even though they perform different actions [34]. Consequently, GKAT can be seen as a theory of *computation* schemata, i.e., programs that need to halt successfully to be meaningful.

In contrast, it is also useful to be able to reason about *process* schemata, i.e., programs that perform meaningful tasks, even when they do not terminate successfully. To this end, we define the ***reduced congruence*** $\equiv_0$ generated by the axioms of Figure 1 except S3.

Let $[\![-]\!] : \mathrm{Exp} \to S$ be a semantics of GKAT. We say that $[\![-]\!]$ is ***sound w.r.t.*** $\equiv$ if for all $e, f \in \mathrm{Exp}$ with $e \equiv f$, it holds that $[\![e]\!] = [\![f]\!]$. Similarly, $[\![-]\!]$ is ***sound w.r.t.*** $\equiv_0$ if $e \equiv_0 f$ implies that $[\![e]\!] = [\![f]\!]$.

Since $\equiv$ encodes common program laws, one might wonder whether there is a single interpretation in which programs are related by $\equiv$ if and only if they have the same image. Such an interpretation is called ***free w.r.t.*** $\equiv$. This question is not just of theoretical interest: a free interpretation can help decide whether programs are provably equivalent, and hence the same under any sound interpretation, by checking whether their free semantics coincide. Naturally, the same question can be asked for $\equiv_0$: is there a semantics that is ***free w.r.t.*** $\equiv_0$, i.e., where $e \equiv_0 f$ if and only if e and f have the same interpretation?

The remainder of this paper is organized as follows. In Section 3, we describe the operational structure for GKAT expressions in terms of GKAT-automata, as in [34]. In Section 4, we provide an explicit construction of a GKAT-automaton in which all other automata can be uniquely interpreted. We then build a semantics that is sound w.r.t. $\equiv_0$ in Section 5. In Section 6 we relate our coequational description of GKAT expressions to the *well-nested* GKAT-*automata* of [34]. In Section 7, we prove that this semantics is in fact complete w.r.t. $\equiv_0$ and, building on this, obtain a semantics that is complete w.r.t. $\equiv$. Omitted proofs are included in the extended version [32].

3 An operational model: GKAT-automata

In this section we discuss the small-step operational model for GKAT programs from [34]. The operational perspective provides us with the tools to describe a semantics that is complete w.r.t. $\equiv_0$ and paves the way to a decision procedure.

We can think of a GKAT-program as a machine that evolves as it reads a string of atomic tests. Depending on the most recently observed atomic test, the program either accepts, rejects, or emits an action label and changes to a new state. For example, feeding **if** b **do** p **else** q an atomic test $a \in b$ causes it to perform the action p and then terminate successfully.

▶ **Definition 3.1.** *A* GKAT**-automaton** *[34, 23] is a pair $\mathcal{X} = (X, \delta)$, where X is a set of* **states** *and $\delta : X \times A \to 2 + \Sigma \times X$ is a* **transition function**. *We use $x \xrightarrow{a|p}_{\mathcal{X}} x'$ as a notation for $\delta(x, a) = (p, x')$. Similarly, $x \Rightarrow_{\mathcal{X}} a$ denotes that $\delta(x, a) = 1$, and $x \downarrow_{\mathcal{X}} a$ denotes that $\delta(x, a) = 0$. We drop the subscript $\mathcal{X}$ when the automaton is clear from context.*

Intuitively, X represents the states of an abstract machine running a GKAT program, with dynamics encoded in δ. When the machine is in state $x \in X$ and observes $a \in A$, there are three possibilities: if $x \downarrow a$, the machine rejects; if $x \Rightarrow a$, it accepts; and if $x \xrightarrow{a|p} x'$, it performs the action p followed by a transition to the state x'.

$$\frac{a \in b}{b \Rightarrow a} \qquad \frac{}{p \xrightarrow{a|p} 1} \qquad \frac{a \in b \qquad e \Rightarrow a}{e +_b f \Rightarrow a} \qquad \frac{a \in \bar{b} \qquad f \Rightarrow a}{e +_b f \Rightarrow a} \qquad \frac{a \in b \qquad e \xrightarrow{a|p} e'}{e +_b f \xrightarrow{a|p} e'} \qquad \frac{a \in \bar{b} \qquad f \xrightarrow{a|p} f'}{e +_b f \xrightarrow{a|p} f'}$$

$$\frac{e \Rightarrow a \qquad f \Rightarrow a}{e \cdot f \Rightarrow a} \qquad \frac{e \Rightarrow a \qquad f \xrightarrow{a|p} f'}{e \cdot f \xrightarrow{a|p} f'} \qquad \frac{e \xrightarrow{a|p} e'}{e \cdot f \xrightarrow{a|p} e' \cdot f} \qquad \frac{a \in b \qquad e \xrightarrow{a|p} e'}{e^{(b)} \xrightarrow{a|p} e' \cdot e^{(b)}} \qquad \frac{a \in \bar{b}}{e^{(b)} \Rightarrow a}$$

Figure 2 The transition structure of $\mathcal{E}$. Here, $e, e', f, f' \in \mathrm{Exp}$, $b \subseteq A$, $a \in A$, and $p \in \Sigma$. Transitions that are not explicitly defined above are assumed to be failed termination.

▶ Remark 3.2. The reader familiar with coalgebra will recognize that GKAT-automata are precisely coalgebras for the functor $G = (2 + \Sigma \times \mathsf{Id})^A$ [34]. Indeed, the notions relating to GKAT-automata, such as homomorphism, bisimulation, and semantics to follow are precisely those that arise from G as prescribed by universal coalgebra [27].

We can impose an automaton structure on Exp yielding the ***syntactic* GKAT-*automaton*** $\mathcal{E} = (\mathrm{Exp}, D)$, where D is the transition map given by Brzozowski derivatives [34] as specified in Figure 2. For instance, the operational behavior of $p^{(b)}$ as a state of $\mathcal{E}$ could be drawn as follows, where $x \xrightarrow{b|p} y$ denotes that $x \xrightarrow{a|p} y$ for every $a \in b$ and rejecting transitions $x \downarrow a$ are left implicit:

$$\bar{b} \Longleftarrow p^{(b)} \xrightarrow{b|p} 1 \cdot p^{(b)} \Longrightarrow \bar{b} \qquad (1)$$

The operational structure of $\mathcal{E}$ is connected to $\equiv_0$ as follows.

▶ **Theorem 3.3** (Fundamental theorem of GKAT). *For any $e \in \mathrm{Exp}$, $e \equiv_0 1 +_{E(e)} D(e)$ where*

$$D(e) = \bigplus_{e \xrightarrow{a|p_a} e_a} p_a \cdot e_a \qquad \text{and} \qquad \bigplus_{a \in b} e_a = \begin{cases} 0 & \text{if } b = 0, \\ e_a +_a \left(\bigplus_{a' \in b \setminus a} e_{a'} \right) & \text{some } a \in b, \text{ otherwise.} \end{cases}$$

The generalized guarded union above is well defined, in that the order of atoms does not matter up to $\equiv_0$. See [34] for more details about the generalised guarded union.

States of GKAT-automata have the same behavior if reading the same sequence of atoms leads to the same sequence of actions, acceptance, or rejection. This happens when one state mimics the moves of the other, performing the same actions in response to the same stimuli. For instance, consider the GKAT-automaton in (1): the behavior of $p^{(b)}$ can be replicated by the behavior of $1 \cdot p^{(b)}$, in that both either consume an $a \in \bar{b}$ and terminate or consume $a \in b$ and emit p before transitioning to $1 \cdot p^{(b)}$. This can be made precise.

▶ **Definition 3.4.** *Let $R \subseteq X \times Y$ be a relation between the state spaces of* GKAT*-automata $\mathcal{X}$ and $\mathcal{Y}$. Then R is a* **bisimulation** *if for any $(x, y) \in R$ and $a \in A$,*
(1) *$x \downarrow_{\mathcal{X}} a$ if and only if $y \downarrow_{\mathcal{Y}} a$; and **(2)** $x \Rightarrow_{\mathcal{X}} a$ if and only if $y \Rightarrow_{\mathcal{Y}} a$; and*
(3) *if $x \xrightarrow{a|p}_{\mathcal{X}} x'$ and $y \xrightarrow{a|q}_{\mathcal{Y}} y'$ for some x' and y', then $p = q$ and $(x', y') \in R$.*

If a pair of states $(x, y) \in X \times Y$ is contained in a bisimulation, we say that x and y are **bisimilar**. *If a bisimulation R is the graph of a function $\varphi : X \to Y$, we write $\varphi : \mathcal{X} \to \mathcal{Y}$ and call φ a* GKAT**-automaton homomorphism** *[27].*

Indeed, bisimulations are designed to formally witness behavioral equivalence. We use the term ***behavior*** as a synonym for the phrase *bisimilarity (equivalence) class.*

4 The final GKAT-automaton

One way of assigning semantics to GKAT expressions is to find a sufficiently large GKAT-automaton $\mathcal{Z}$ that contains the behavior of every other GKAT-automaton. In this section, we provide a concrete explicit description of such a "semantic" GKAT-automaton – this is a crucial step towards being able to devise a completeness proof.

Concretely, $\mathcal{Z}$ represents the behavior of a state as a tree that holds information about acceptance, rejection, and transitions to other states (which are subtrees). Essentially, this tree is an unfolding of the transition graph from that state.

We describe these trees using partial functions. Let us write A^+ for the set of all non-empty words consisting of atoms. The state space Z of $\mathcal{Z}$ is the set of all partial functions $t : A^+ \rightharpoonup 2 + \Sigma$ with $A \subseteq \mathrm{dom}(t)$, such that the following hold for all $a \in A$ and $x \in A^+$.

$$\frac{w \in \mathrm{dom}(t) \quad t(w) \in \Sigma}{wa \in \mathrm{dom}(t)} \qquad \frac{w \in \mathrm{dom}(t) \quad t(w) \in 2}{wx \notin \mathrm{dom}(t)}$$

The transition structure of $\mathcal{Z}$ is defined by the inferences

$$\frac{t(a) = 0}{t \downarrow a} \qquad \frac{t(a) = 1}{t \Rightarrow a} \qquad \frac{t(a) = p \in \Sigma}{t \xrightarrow{a|p} \lambda w.t(aw)}$$

When $t(w) \in \Sigma$, we will write $\partial_w t$ for $\lambda u.t(wu)$. We can think of $t \in Z$ as a tree where the root has leaves for atoms $a \in A$ with $t(a) = 1$, and a subtree for every $a \in A$ with $t(a) \in \Sigma$.

▶ Remark 4.1. Trees correspond to *deterministic* (possibly *infinite*) *guarded languages* [34, 23]. More precisely, every tree can be identified with a language $L \subseteq (A \cdot \Sigma)^* \cdot A \cup (A \cdot \Sigma)^\omega$ satisfying (i) if $wap\sigma, waq\sigma' \in L$, then $p = q$; and (ii) if $wa \in L$, then $wap\sigma \notin L$ for any $p\sigma$. We forgo a description in terms of guarded languages in favor of trees because these trees have the constraint about determinism built in.

A ***node*** of t is a word $w \in A^*$ such that either $w = \epsilon$ (the empty word), or $w \in \mathrm{dom}(t)$ and $t(w) \in \Sigma$. We write $\mathrm{Node}(t)$ for the set of nodes of t. A ***subtree*** of t is a tree t' such that $t' = \partial_w t$ for some $w \in \mathrm{Node}(t)$. A ***leaf*** of t is a word $w \in \mathrm{dom}(t)$ such that $t(w) \in 2$.

Next, we specialize Definition 3.4 to $\mathcal{Z}$ (c.f. [28, Theorem 3.1]).

▶ **Lemma 4.2.** *$R \subseteq Z \times Z$ is a bisimulation on $\mathcal{Z}$ iff for any $(t, s) \in R$ and $a \in A$, (1) $t(a) = s(a)$; and (2) if either $\partial_a t$ or $\partial_a s$ is defined, then both are defined and $(\partial_a t, \partial_a s) \in R$.*

We can now prove that bisimilar trees in Z coincide.

▶ **Lemma 4.3** (Coinduction)**.** *If $s, t \in Z$ are bisimilar, then $s = t$.*

Thus, to show that two trees are equal, it suffices to demonstrate a bisimulation that relates them. This proof method is called ***coinduction***. We can also use Lemma 4.2 to define algebraic operations on Z, and such definitions are said to be ***coinductive***. Many of the results in the sequel are argued using coinduction, and many of the constructions are coinductive. With this in mind, we are now ready to prove that $\mathcal{Z}$ contains every behavior that can be represented by a GKAT-automaton, as follows.

▶ **Theorem 4.4.** *$\mathcal{Z}$ is the final GKAT-automaton. In other words, for every GKAT-automaton $\mathcal{X}$, there exists a unique GKAT-automaton homomorphism $!_{\mathcal{X}}$ from $\mathcal{X}$ to $\mathcal{Z}$.*

Given a GKAT-automaton $\mathcal{X}$, the unique map $!_{\mathcal{X}}$ assigns a tree from Z to each of its states. In particular, recalling that the syntactic GKAT-automaton $\mathcal{E}$ has Exp as its set of states, $!_{\mathcal{E}}$ is a semantics of GKAT programs in terms of trees. The following lemma states that bisimulation is sound and complete with respect to this semantics.

▶ **Lemma 4.5.** *States x and x' of a GKAT-automaton $\mathcal{X}$ are bisimilar iff $!_{\mathcal{X}}(x) = !_{\mathcal{X}}(x')$.*

5 Trees form an algebra

So far, we have seen that the behavior of a GKAT-program is naturally interpreted as a certain kind of tree, and that each such tree is the state of the final GKAT-automaton $\mathcal{Z}$. In this section, we show that the trees in Z can themselves be manipulated and combined using the programming constructs of GKAT. These operations satisfy all of the axioms that build $\equiv_0$, but fail the *early-termination axiom* S3. This gives rise to an inductive semantics of GKAT-programs $[\![-]\!] : \mathrm{Exp} \to Z$ that is sound w.r.t. $\equiv_0$. As a matter of fact, we will see that $[\![-]\!]$ coincides with the unique GKAT-automaton homomorphism $!_{\mathcal{E}} : \mathrm{Exp} \to Z$.

We begin by interpreting the tests. Given $b \subseteq A$, we define $[\![b]\!]$ as the characteristic function of b as a subset of A^+, i.e., $[\![b]\!](a) = 1$ if $a \in b$, and $[\![b]\!](a) = 0$ otherwise.

On the other hand, primitive action symbols denote programs that perform an action in one step and then terminate successfully in the next. For $p \in \Sigma$, this behavior is described by the unique tree $[\![p]\!]$ such that $[\![p]\!](a) = p$ and $\partial_a [\![p]\!] = [\![1]\!]$ for any $a \in A$. When context can disambiguate, we write b in place of $[\![b]\!]$ and p in place of $[\![p]\!]$.

Each operation is defined using a ***behavioral differential equation (BDE)*** consisting of a set of ***initial conditions*** $t(a) = \xi_a \in 2 + \Sigma$ indexed by $a \in A$ and a set of ***step equations*** $\partial_a t = s_a$ indexed by the $a \in A$ with $t(a) \in \Sigma$. This is possible because every BDE describes a unique automaton, which (by Theorem 4.4) has a unique interpretation in Z [28]. Each BDE below can be read more or less directly from Figure 2.

The first operation that we interpret in Z is sequential composition. For any $s, t \in Z$, the tree $s \cdot t$ models sequential composition of programs by replacing each non-zero leaf of s by the nodal subtree of t given by the corresponding atomic test. This can formally be defined as the unique operation satisfying the following behavioral differential equation.

$$(s \cdot t)(a) = \begin{cases} t(a) & \text{if } s(a) = 1, \\ s(a) & \text{otherwise} \end{cases} \qquad \partial_a(s \cdot t) = \begin{cases} \partial_a t & \text{if } s(a) = 1, \\ \partial_a s \cdot t & \text{otherwise.} \end{cases}$$

Here, $\partial_a s \cdot t = (\partial_a s) \cdot t$. Using this operation, we define $[\![e \cdot f]\!] = [\![e]\!] \cdot [\![f]\!]$.

To interpret the guarded union operation, define $+_b$ to be the unique operation such that

$$(s +_b t)(a) = \begin{cases} s(a) & \text{if } a \in b, \\ t(a) & \text{otherwise} \end{cases} \qquad \partial_a(s +_b t) = \begin{cases} \partial_a s & \text{if } a \in b, \\ \partial_a t & \text{otherwise.} \end{cases}$$

As before, we define $[\![e +_b f]\!] = [\![e]\!] +_b [\![f]\!]$.

Finally, we interpret the guarded exponential operation. Following Figure 2, $t^{(b)}$ can be defined as the unique tree satisfying

$$t^{(b)}(a) = \begin{cases} 1 & \text{if } a \notin b, \\ t(a) & \text{if } a \in b \text{ and } t(a) \in \Sigma, \\ 0 & \text{otherwise.} \end{cases} \qquad \partial_a(t^{(b)}) = \partial_a t \cdot t^{(b)}$$

Similar to the other operators, we set $[\![e^{(b)}]\!] = [\![e]\!]^{(b)}$. This completes our definition of the algebraic homomorphism $[\![-]\!] : \mathrm{Exp} \to Z$.

As it happens, $[\![-]\!]$ is also a GKAT automaton homomorphism from $\mathcal{E}$ to $\mathcal{Z}$. By uniqueness of such homomorphisms (Theorem 4.4), we can conclude that $[\![-]\!]$ and $!_{\mathcal{E}}$ are the same.

▶ **Proposition 5.1.** *For any $e \in \mathrm{Exp}$, $[\![e]\!] = !_{\mathcal{E}}(e)$.*

This allows us to treat the algebraic and coalgebraic semantics as synonymous. Using Lemma 4.5, we can then show soundness w.r.t. $\equiv_0$ by arguing that $\equiv_0$ is a bisimulation on $\mathcal{E}$.

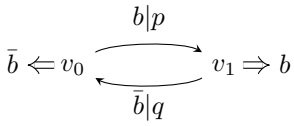


Figure 3 A GKAT-automaton without GKAT behaviors.

▶ **Theorem 5.2.** *The semantics* $[\![-]\!]$ *is sound w.r.t.* $\equiv_0$.

On the other hand, Z does not satisfy S3. For instance, $[\![p \cdot 0]\!] \neq [\![0]\!]$ for any $p \in \Sigma$. We will adapt the model to overcome this in Section 7.3.

6 Well-nested automata and nested behavior

Not all behaviors expressible in terms of finite GKAT-automata occur in $\mathcal{E}$. For example, the two-state automaton in Figure 3 fails to exhibit any behavior of the form $[\![e]\!]$, with $e \in \mathrm{Exp}$, when $b, \bar{b} \neq 0$. This is proven in the extended version [32] where we show that no branch of a GKAT behavior can accept both b and $\bar{b}$ infinitely often. For another example, see [23], where a particular three-state automaton is shown to exhibit no GKAT behavior.

Intuitively, both of the examples above fail to exhibit the behaviors of GKAT programs because GKAT lacks a **goto**-statement that allows control to transfer to an arbitrary position in the program; instead, GKAT automata corresponding to GKAT expressions are structured by branches and loops. The question then arises: can we characterize the "shapes" of automata whose behavior is **goto**-free, i.e., described by a GKAT expression?

In [34], the authors proposed the class of *well-nested* GKAT automata, consisting of automata built inductively by applying a series of operations designed to mimic the structural effects of loops. It was shown that the behavior of every GKAT expression can be described by some well-nested automaton. Moreover, they proved that the class of well-nested automata constitutes a sufficient condition: the behavior of a well-nested GKAT automaton is described by a GKAT expression. Whether this condition is also *necessary*, i.e., whether every automaton with behavior corresponding to a GKAT expression is well-nested, was left open.

Thus, a positive answer to the latter question amounts to showing that every GKAT automaton whose behavior is the same as a well-nested GKAT automaton is itself well-nested. Such a class of automata closed under behavioral equivalence is known as a ***covariety***. Covarieties have desirable structural properties. In particular, they are closed under homomorphic images [27, 12, 3]. Unfortunately, well-nested automata do not satisfy this property: we have found a well-nested automaton whose homomorphic image is not well-nested, depicted in Figure 4. In other words, there exists a non-well-nested automaton whose behavior is still described by a GKAT expression. This also closes the door on a simpler approach to completeness described in [34].

Thus, well-nested automata do not constitute a characterization of the GKAT automata that correspond to GKAT expressions. To obtain such a characterization, we take a slightly different approach: rather than describing shapes of these automata, we describe the shapes of the trees that they denote. We refer to a set of trees $\mathsf{U} \subseteq Z$ as a ***coequation***, and treat it as a predicate: a GKAT-automaton $\mathcal{X}$ ***satisfies*** U, written $\mathcal{X} \models \mathsf{U}$, if every behavior present in $\mathcal{X}$ appears in U – in other words, if $!_{\mathcal{X}}$ factors through U. We write $\mathrm{Cov}(\mathsf{U})$ to denote the class of all GKAT-automata that satisfy U. It is easily shown that $\mathrm{Cov}(\mathsf{U})$ is a covariety.

The coequation that we give to describe the covariety of automata whose behavior corresponds to a GKAT expression is driven by the intuition behind well-nested automata: the trees in this coequation are built using compositions that enforce **while**-like behavior,

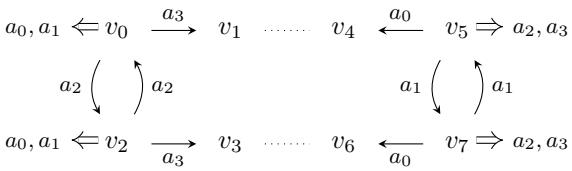


Figure 4 As depicted, this automaton is well-nested. However, identifying v_1 with v_4, and v_3 with v_6, we obtain an automaton that is not well-nested.

and do not permit the construction of **goto**-like behavior. To this end, we need to define a new *continuation* operation, as follows. Given $s, t \in Z$, the ***continuation*** $s \rhd t$ of s along t is the unique tree satisfying the behavioral differential equation

$$(s \rhd t)(a) = \begin{cases} t(a) & \text{if } s(a) = 1, \\ s(a) & \text{otherwise} \end{cases} \qquad \partial_a(s \rhd t) = \begin{cases} \partial_a t \rhd t & \text{if } s(a) = 1, \\ \partial_a s \rhd t & \text{otherwise.} \end{cases}$$

Intuitively, $s \rhd t$ is the tree that attaches infinitely many copies of t to s. This operation can be thought of as the dual to Kleene's original $*$-operation [16], which loops on its first argument some number of times before continuing in the second.

▶ **Definition 6.1.** *The* **nesting** *coequation* W *is the smallest subset of* Z *containing the* **discrete** *coequation* $\mathsf{D} := \{[\![b]\!] \mid b \subseteq A\}$ *and closed under the* **nesting** *rules below:*

$$\frac{t, s \in \mathsf{W}}{t \cdot s \in \mathsf{W}} \qquad \frac{(\forall a \in A)\ t(a) \in \Sigma \implies \partial_a t \in \mathsf{W}}{t \in \mathsf{W}} \qquad \frac{t, s \in \mathsf{W}}{t \rhd s \in \mathsf{W}}$$

The first and third nesting rules say that W is closed under composition and continuation; the second rule says that integrals over nested trees are nested.

It is not too hard to see that W is a subautomaton of $\mathcal{Z}$. In other words, if $t \in \mathsf{W}$, then the derivatives of t are in W as well. In fact, W is a subalgebra of Z in that it is closed under the operations of GKAT. This can be seen from the following observations: first, $\partial_a p = 1$ for all $a \in A$, so $p \in \mathsf{W}$ for any $p \in \Sigma$ by the second nesting rule. Second, W is closed under sequential composition by definition. Third, if $s, t \in \mathsf{W}$ and $b \subseteq A$, then every derivative of $s +_b t$ is either a derivative of s or a derivative of t. Lastly, closure under the guarded exponential is a consequence of the identity

$$t^{(b)} = 1 \rhd (\tilde{t} +_b 1), \qquad \text{where} \qquad \tilde{t} := +_{t \xrightarrow{a|p_a} t_a} p_a \cdot t_a.$$

This identity can be shown to hold for all $t \in Z$ and $b \subseteq A$ using a coinductive argument. It follows that the nesting coequation contains the image of $[\![-]\!]$. A similar argument can be used to establish the reverse containment as well, which leads to the following.

▶ **Proposition 6.2.** W *is the set of* GKAT *program behaviors, i.e,* $\mathsf{W} = \{[\![e]\!] \mid e \in \mathrm{Exp}\}$.

Proposition 6.2 characterizes W as the the set of behavioral patterns exhibited by GKAT expressions: the states of a GKAT-automaton $\mathcal{X}$ behave like GKAT programs if and only if $\mathcal{X}$ satisfies W, or, in other words, if $\mathcal{X}$ can be found in the covariety $\mathrm{Cov}(\mathsf{W})$. Since every well-nested automaton has the behavior of some GKAT expression [34], it must satisfy W.

▶ **Proposition 6.3.** *Well-nested* GKAT*-automata satisfy the nesting coequation.*

7 Completeness

This section contains two completeness theorems for GKAT . As in [34], we need to assume that W3 is generalized to arbitrary (linear) systems of equations. This *uniqueness axiom*, discussed in Section 7.1, will allow us to prove that the semantics $[\![-]\!]$ from Section 5 is free with respect to $\equiv_0$ – that is, $[\![e]\!] = [\![f]\!]$ implies $e \equiv_0 f$ – in Section 7.2. This will then provide an alternative route to completeness for GKAT in Section 7.3.

7.1 Uniqueness of solutions for Salomaa systems

In part, W3 from Figure 1 ensures that the equation $g \equiv e \cdot g +_b f$ with indeterminate g has at most one solution in $\mathrm{Exp}/\equiv_0$ for any $e, f \in \mathrm{Exp}$ under the condition that e denotes a productive program. In fact, we could have stated the axiom this way from the beginning, as W1 provides the existence of a solution to this equation (even without the restriction on productivity). As we will see, the uniqueness axiom makes a more general statement than W3 about *systems* of equations with an arbitrary number of indeterminates.

▶ **Definition 7.1.** *A* **system of (n left-affine) equations** *is a sequence of n equations of the form $x_i = e_{i1} \cdot x_1 +_{b_{i1}} \cdots +_{b_{i(n-1)}} e_{in} \cdot x_n +_{b_{in}} c_i$, indexed by $i \leqslant n$, such that (1) x_i is an indeterminate variable; (2) $(b_{ij})_{j \leqslant n}$ is a sequence of* **disjoint** *Boolean expressions, i.e. $b_{ij} \wedge b_{ik} \equiv 0$ for any $j \neq k$; (3) c_i is a Boolean expression disjoint from b_{ij} for all $j \leqslant n$; and (4) e_{ij} is a* GKAT *expression for any $j \leqslant n$.*

Given any congruence $\doteqdot$ satisfying the axioms of $\equiv_0$, a **solution in $\mathrm{Exp}/\doteqdot$** *to such a system is an n-tuple of* GKAT *expressions $(g_i)_{i \leqslant n}$ such that the equivalence $g_i \doteqdot e_{i1} \cdot g_1 +_{b_{i1}} \cdots +_{b_{i(n-1)}} e_{in} \cdot g_n +_{b_{in}} c_i$ holds for all $i \leqslant n$.*

For example, the equation in the premise of W3 is a system of one left-affine equation, and the conclusion prescribes a unique solution (in $\mathrm{Exp}/\equiv_0$) to the premise. Every finite GKAT-automaton $\mathcal{X}$ gives rise to a system of equations with variables indexed by $X = \{x_i \mid i \leqslant n\}$ and coefficients indexed by the transition map, as follows:

$$e_{ij} = \bigplus_{x_i \xrightarrow{a|p_a} x_j} p_a \qquad c_i = \{a \in A \mid x_i \Rightarrow a\} \qquad b_{ij} = \{a \in A \mid x_i \xrightarrow{a|p} x_j\}.$$

Solving this system of equations uncovers the GKAT-constructs the automaton implements.

The uniqueness axiom states that certain systems of equations, like the one in the premise of W3, admit at most one solution. Choosing which systems the axiom should apply to must be done carefully for the same reason that necessitates the side-condition on W3. Crucially, we require that the system have *productive coefficients*, i.e. $E(e_{ij}) \equiv 0$ for all $i, j \leqslant n$, to admit a unique solution. As this condition is analogous to Salomaa's *empty word property* [31], a system of equations with productive coefficients is called ***Salomaa*** [34]. The ***uniqueness axiom (for $\doteqdot$)*** states that every Salomaa system of equations has at most one solution in $\mathrm{Exp}/\doteqdot$. It is sound with respect to the semantics $[\![-]\!]$ from Section 5.

▶ **Theorem 7.2.** *For any $i, j \leqslant n$, let $s_{ij} \in Z$ satisfy $s_{ij}(a) \neq 1$ for any $a \in A$, $(b_{ij})_{j \neq n}$ be a sequence of disjoint Boolean expressions for any $i \leqslant n$, and $c_i \subseteq A$ be disjoint from b_{ij} for each $i \leqslant n$. The system of equations $x_i = s_{i1} \cdot t_1 +_{b_{i1}} \cdots +_{b_{i(n-1)}} s_{in} \cdot t_n +_{b_{in}} c_i$, indexed by $i \leqslant n$ has a unique solution in Z^n.*

7.2 Completeness with respect to $\equiv_0$

Next, we present a completeness theorem w.r.t. $\equiv_0$. We have already seen that the behavior of a program takes the form of a tree, and that the programming constructs of GKAT apply to trees in such a way that equivalence up to the axioms of $\equiv_0$ is preserved (Theorem 5.2). The completeness theorem in this section shows that up to $\equiv_0$-equivalence, GKAT programs can be identified with the trees they denote.

▶ **Theorem 7.3** (Completeness for $\equiv_0$). *Assume the uniqueness axiom for $\equiv_0$ and let $e, f \in \mathrm{Exp}$. If $[\![e]\!] = [\![f]\!]$, then $e \equiv_0 f$.*

Proof sketch. Since $[\![e]\!] = [\![f]\!]$, e and f are bisimilar as expressions. This bisimulation gives rise to a Salomaa system of equations, which can be shown to admit both the derivatives of e and f as solutions. By the unique solutions axiom, it then follows that $e \equiv_0 f$. ◀

7.3 Completeness with respect to $\equiv$

Having found a semantics that is sound and complete w.r.t. $\equiv_0$, we proceed to extend this result to find a semantics that is sound and complete w.r.t. $\equiv$. Recall that the only difference between these equivalences was S3, which equates programs that fail eventually with programs that fail immediately. To coarsen our semantics, we need an operation on labelled trees that forces early termination in case an accepting state cannot be reached.

▶ **Definition 7.4.** *We say $t \in Z$ is* **dead** *when for all $w \in \mathrm{dom}(t)$ it holds that $t(w) \neq 1$. The* **normalization** *operator is defined coinductively, as follows:*

$$t^\wedge(a) = \begin{cases} 0 & t(a) \in \Sigma \wedge \partial_a t \text{ is dead,} \\ t(a) & \text{otherwise} \end{cases} \qquad \partial_a(t^\wedge) = (\partial_a t)^\wedge.$$

▶ **Example 7.5.** Normalizing the tree $[\![p +_b p \cdot 0]\!]$ prunes the branch corresponding to $\bar{b}$, since it has no accepting leaves. This yields the tree $[\![b \cdot p]\!]$.

We can compose the normalization operator with the semantics $[\![-]\!]$ to obtain a new semantics $[\![-]\!]^\wedge$, which replaces dead subtrees with early termination. Composing normalization with the earlier semantics of GKAT, we obtain the ***normalized semantics*** $[\![-]\!]^\wedge$. This semantics is sound w.r.t. $\equiv$.

▶ **Proposition 7.6.** *If $e \equiv f$, then $[\![e]\!]^\wedge = [\![f]\!]^\wedge$.*

For the corresponding completeness property, we need a way of "normalizing" a given expression in Exp. The following observation gives us a way to do this.

▶ **Lemma 7.7.** W *is closed under normalization.*

When $e \in \mathrm{Exp}$, we have that $[\![e]\!] \in \mathsf{W}$. Moreover, by the above, $[\![e]\!]^\wedge \in \mathsf{W}$, which means that there is an $e' \in \mathrm{Exp}$ such that $[\![e']\!] = [\![e]\!]^\wedge$. We write $e^\wedge$ for this ***normalized expression***. As it turns out, we can derive the equivalence $e^\wedge \equiv e$ from the uniqueness axiom for $\equiv$. This gives an alternative proof of the completeness result of [34] that highlights the role of coequational methods in reasoning about failure modes.

▶ **Corollary 7.8** ([34]). *Assume the uniqueness axiom for $\equiv$ and $\equiv_0$. If $[\![e]\!]^\wedge = [\![f]\!]^\wedge$, then $e \equiv f$.*

Proof sketch. If $[\![e]\!]^\wedge = [\![f]\!]^\wedge$, then $[\![e^\wedge]\!] = [\![f^\wedge]\!]$. By completeness of $\equiv_0$ w.r.t. $[\![-]\!]$, we can then derive that $e \equiv e^\wedge \equiv_0 f^\wedge \equiv f$, and since $\equiv_0$ is contained in $\equiv$, also $e \equiv f$. ◀

By normalizing the trees in W, we obtain the coequation $\mathsf{W}^\wedge = \{t^\wedge \mid t \in \mathsf{W}\}$. This coequation precisely characterizes GKAT programs with forced early termination. In particular, since $\mathsf{W}^\wedge \subseteq \mathsf{W}$, neither state in Figure 3 has a semantics described by $[\![e]\!]^\wedge$ for some $e \in \mathrm{Exp}$.

8 Related work

This paper builds on [34], where GKAT was proposed together with a language semantics based on guarded strings [15] and an axiomatization closely related to Salomaa's axiomatization of regular expressions based on unique fixpoints [31]. Note that the language of *propositional while programs* from [23, 20] is closely related to GKAT in terms of semantics, although the compact syntax and axiomatization were only introduced in [34].

Some GKAT-automata have behavior that does not correspond to any GKAT expression, such as the example in [23]. The upshot is that the Böhm-Jacopini theorem [6, 13], which states that every deterministic flowchart corresponds to a WHILE program, does not hold propositionally, i.e., when we abstract from the meaning of individual actions and tests [23].

In contrast with [34, 23], our work provides a precise characterization of the behaviors denoted by GKAT programs using trees. In other words, we characterize the image of the semantic map inside the space of all behaviors. This explicit characterization was essential for proving completeness of the full theory of GKAT, including the early termination axiom. KAT equivalence without early termination has been investigated by Mamouras [24].

Brzozowski derivatives [7] appear in the completeness proof of KA [18, 21, 14]. We were more directly inspired by Silva's coalgebraic analogues of Brzozowski derivatives used in the context of completeness [33]. Rutten [28] and Pavlovic and Escardo [26] document the connection between the differential calculus of analysis and coalgebraic derivatives.

Coequations have appeared in the coalgebra literature in a variety of contexts, e.g. [3, 1, 5, 29, 30], and notably in the proof of generalized Eilenberg theorems [36, 2]. The use of coequations in completeness proofs is, as far as we are aware, new.

9 Discussion

GKAT was introduced in [23] under the name *propositional while programs* and extensively studied in [34] as an algebraic framework to reason about simple imperative programs. We presented a new perspective on the theory of GKAT, which allowed us to isolate a fragment of the original axiomatization that captures the purely behavioral properties of GKAT programs. We solved an open problem from [34], providing a proof that well-nested automata are not closed under homomorphisms, thereby making it unlikely that these automata can be used in a completeness proof that does not rely on uniqueness axioms. Finally, we proved completeness for the full theory, respecting the early-termination property, in which programs that fail immediately are equated with programs that fail eventually.

There are several directions for future work that are worth investigating. First, it was conjectured in [34] that the uniqueness axiom follows from the other axioms of GKAT. This remains open, but at the time of writing we think this conjecture might be false. Secondly, the technique we use, based on coequations, can serve as basis for a general approach to completeness proofs. We plan to investigate other difficult problems where our technique might apply. Of particular interest is an open problem posed by Milner in [25], which consists of showing that a certain set of axioms are complete w.r.t. bisimulation equivalence for regular expressions. Recently, Grabmeyer and Fokkink [11] provided a partial solution. We believe our technique can simplify their proofs and shed further light on Milner's problem.

We have chosen to adopt the axiomatization from [34], which can be described as a Salomaa-style axiomatization – the loop is a unique fixpoint satisfying a side condition on termination. We would like to generalize the results of the present paper to an axiomatization in which the loop is a least fixpoint w.r.t. an order. The challenge is that there is no natural order in the language because the $+$ of Kleene Algebra has been replaced by $+_b$. However, we hope to devise an order $\leqslant$ directly on expressions and extend the characterizations that we have to the new setting. This new axiomatization would have the advantage of being algebraic (that is, sound under arbitrary substitution), which makes it more suitable for verification purposes as the number of models of the language would increase.

References

1 Jirí Adámek. A logic of coequations. In *CSL*, pages 70–86, 2005. doi:10.1007/11538363_7.

2 Jirí Adámek, Stefan Milius, Robert S. R. Myers, and Henning Urbat. Generalized Eilenberg theorem: Varieties of languages in a category. *ACM Trans. Comput. Log.*, 20(1):3:1–3:47, 2019. doi:10.1145/3276771.

3 Jirí Adámek and Hans-E. Porst. On varieties and covarieties in a category. *Math. Struct. Comput. Sci.*, 13(2):201–232, 2003. doi:10.1017/S0960129502003882.

4 Carolyn Jane Anderson, Nate Foster, Arjun Guha, Jean-Baptiste Jeannin, Dexter Kozen, Cole Schlesinger, and David Walker. NetKAT: semantic foundations for networks. In *POPL*, pages 113–126, 2014. doi:10.1145/2535838.2535862.

5 Adolfo Ballester-Bolinches, Enric Cosme-Llópez, and Jan J. M. M. Rutten. The dual equivalence of equations and coequations for automata. *Inf. Comput.*, 244:49–75, 2015. doi:10.1016/j.ic.2015.08.001.

6 Corrado Böhm and Giuseppe Jacopini. Flow diagrams, Turing machines and languages with only two formation rules. *Commun. ACM*, 9(5):366–371, 1966. doi:10.1145/355592.365646.

7 Janusz A. Brzozowski. Derivatives of regular expressions. *J. ACM*, 11(4):481–494, 1964. doi:10.1145/321239.321249.

8 Ernie Cohen, Dexter Kozen, and Frederick Smith. The complexity of Kleene algebra with tests. Technical Report TR96-1598, Cornell University, July 1996. URL: https://hdl.handle.net/1813/7253.

9 Nate Foster, Dexter Kozen, Konstantinos Mamouras, Mark Reitblatt, and Alexandra Silva. Probabilistic NetKAT. In *ESOP*, pages 282–309, 2016. doi:10.1007/978-3-662-49498-1_12.

10 Nate Foster, Dexter Kozen, Matthew Milano, Alexandra Silva, and Laure Thompson. A coalgebraic decision procedure for NetKAT. In *POPL*, pages 343–355, 2015. doi:10.1145/2676726.2677011.

11 Clemens Grabmayer and Wan J. Fokkink. A complete proof system for 1-free regular expressions modulo bisimilarity. In *LICS*, pages 465–478, 2020. doi:10.1145/3373718.3394744.

12 H. Gumm. Elements of the general theory of coalgebras, 2000.

13 David Harel. On folk theorems. *Commun. ACM*, 23(7):379–389, 1980. doi:10.1145/358886.358892.

14 Bart Jacobs. A bialgebraic review of deterministic automata, regular expressions and languages. In *Algebra, Meaning, and Computation, Essays Dedicated to Joseph A. Goguen on the Occasion of His 65th Birthday*, pages 375–404, 2006. doi:10.1007/11780274_20.

15 Donald M. Kaplan. Regular expressions and the equivalence of programs. *J. Comput. Syst. Sci.*, 3(4):361–386, 1969. doi:10.1016/S0022-0000(69)80027-9.

16 Stephen C. Kleene. Representation of events in nerve nets and finite automata. In Claude E. Shannon and John McCarthy, editors, *Automata Studies*, pages 3–41. Princeton University Press, 1956.

17 Dexter Kozen. Kleene algebra with tests and commutativity conditions. In *TACAS*, pages 14–33, 1996. doi:10.1007/3-540-61042-1_35.

18 Dexter Kozen. Myhill-Nerode relations on automatic systems and the completeness of Kleene algebra. In *STACS*, pages 27–38, 2001. doi:10.1007/3-540-44693-1_3.

19 Dexter Kozen. Automata on guarded strings and applications. *Matematica Contemporanea*, 24:117–139, 2003.

20 Dexter Kozen. Nonlocal flow of control and Kleene algebra with tests. In *LICS*, pages 105–117, 2008. doi:10.1109/LICS.2008.32.

21 Dexter Kozen. On the coalgebraic theory of Kleene algebra with tests. In Can Başkent, Lawrence S. Moss, and Ramaswamy Ramanujam, editors, *Rohit Parikh on Logic, Language and Society*, volume 11 of *Outstanding Contributions to Logic*, pages 279–298. Springer, 2017. doi:10.1007/978-3-319-47843-2_15.

22 Dexter Kozen and Frederick Smith. Kleene algebra with tests: Completeness and decidability. In *CSL*, pages 244–259, 1996. doi:10.1007/3-540-63172-0_43.

23 Dexter Kozen and Wei-Lung Dustin Tseng. The Böhm-Jacopini theorem is false, propositionally. In *MPC*, pages 177–192, 2008. doi:10.1007/978-3-540-70594-9_11.

24 Konstantinos Mamouras. Equational theories of abnormal termination based on Kleene algebra. In *FOSSACS*, volume 10203 of *Lecture Notes in Computer Science*, pages 88–105, 2017. doi:10.1007/978-3-662-54458-7_6.

25 Robin Milner. A complete inference system for a class of regular behaviours. *J. Comput. Syst. Sci.*, 28(3):439–466, 1984. doi:10.1016/0022-0000(84)90023-0.

26 Dusko Pavlovic and Martín Hötzel Escardó. Calculus in coinductive form. In *LICS*, pages 408–417, 1998. doi:10.1109/LICS.1998.705675.

27 Jan J. M. M. Rutten. Universal coalgebra: a theory of systems. *Theor. Comput. Sci.*, 249(1):3–80, 2000. doi:10.1016/S0304-3975(00)00056-6.

28 Jan J. M. M. Rutten. Behavioural differential equations: a coinductive calculus of streams, automata, and power series. *Theor. Comput. Sci.*, 308(1-3):1–53, 2003. doi:10.1016/S0304-3975(02)00895-2.

29 Julian Salamanca, Adolfo Ballester-Bolinches, Marcello M. Bonsangue, Enric Cosme-Llópez, and Jan J. M. M. Rutten. Regular varieties of automata and coequations. In *MPC*, pages 224–237, 2015. doi:10.1007/978-3-319-19797-5_11.

30 Julian Salamanca, Marcello M. Bonsangue, and Jurriaan Rot. Duality of equations and coequations via contravariant adjunctions. In Ichiro Hasuo, editor, *CMCS*, pages 73–93, 2016. doi:10.1007/978-3-319-40370-0_6.

31 Arto Salomaa. Two complete axiom systems for the algebra of regular events. *J. ACM*, 13(1):158–169, 1966. doi:10.1145/321312.321326.

32 Todd Schmid, Tobias Kappé, Dexter Kozen, and Alexandra Silva. Guarded Kleene algebra with tests: Coequations, coinduction, and completeness, 2021. arXiv:2102.08286.

33 Alexandra Silva. *Kleene coalgebra*. PhD thesis, Radboud University, Nijmegen, 2010. URL: https://hdl.handle.net/2066/83205.

34 Steffen Smolka, Nate Foster, Justin Hsu, Tobias Kappé, Dexter Kozen, and Alexandra Silva. Guarded Kleene algebra with tests: Verification of uninterpreted programs in nearly linear time. In *POPL*, 2020. doi:10.1145/3371129.

35 Steffen Smolka, Praveen Kumar, David M. Kahn, Nate Foster, Justin Hsu, Dexter Kozen, and Alexandra Silva. Scalable verification of probabilistic networks. In *PLDI*, pages 190–203, 2019. doi:10.1145/3314221.3314639.

36 Henning Urbat, Jirí Adámek, Liang-Ting Chen, and Stefan Milius. Eilenberg theorems for free. In *MFCS*, 2017. doi:10.4230/LIPIcs.MFCS.2017.43.

Analytical Differential Calculus with Integration

Han Xu
Department of Computer Science and Technology, Peking University, Beijing, China

Zhenjiang Hu
Key Laboratory of High Confidence Software Technologies (MoE),
Department of Computer Science and Technology, Peking University, Beijing, China

Abstract

Differential lambda-calculus was first introduced by Thomas Ehrhard and Laurent Regnier in 2003. Despite more than 15 years of history, little work has been done on a differential calculus with integration. In this paper, we shall propose a differential calculus with integration from a programming point of view. We show its good correspondence with mathematics, which is manifested by how we construct these reduction rules and how we preserve important mathematical theorems in our calculus. Moreover, we highlight applications of the calculus in incremental computation, automatic differentiation, and computation approximation.

2012 ACM Subject Classification Software and its engineering → General programming languages; Software and its engineering → Functional languages

Keywords and phrases Differential Calculus, Integration, Lambda Calculus, Incremental Computation, Adaptive Computing

Digital Object Identifier 10.4230/LIPIcs.ICALP.2021.143

Category Track B: Automata, Logic, Semantics, and Theory of Programming

Related Version *Full Version*: `https://arxiv.org/abs/2105.02632`

1 Introduction

Differential calculus has more than 15 years of history in computer science since the pioneer work by Thomas Ehrhard and Laurent Regnier [9]. It is, however, not well-studied from the perspective of programming languages; we would expect the profound connection of differential calculus with important fields such as incremental computation, automatic differentiation and self-adjusting computation just like how mathematical analysis connects with mathematics. We want to understand what is the semantics of the derivative of a program and how we can use these derivatives to write a program. That is, we wish to have a clear description of derivatives and introduce integration to compute from operational derivatives to the program.

The two main lines of the related work are the differential lambda-calculus [9, 8] and the change theory [7, 4, 5]. On one hand, the differential lambda-calculus uses linear substitution to represent the derivative of a term. For example, given a term $x * x$ (i.e., x^2), with the differential lambda-calculus, we may use the term $\frac{\partial x * x}{\partial x} \cdot 1$ to denote its derivative at 1. As there are two alternatives to substitute 1 for x in the term $x * x$, it gives $(1 * x) + (x * 1)$ (i.e., $2x$) as the derivative (where $+$ denotes "choice").

Despite that the differential lambda-calculus provides a concise way to analyze the alternatives of linear substitution on a lambda term, there is a gap between analysis on terms and computation on terms. For instance, let $+'$ denote our usual addition operator, and $+$ denote the choice of linear substitution. Then we have that $\frac{\partial x +' x}{\partial x} \cdot 1 = (1 +' x) + (x +' 1)$, which is far away from the expected $1 +' 1$. Moreover, it offers no method to integrate over a derivative, say $\frac{\partial t}{\partial x} \cdot y$.

On the other hand, the change theory gives a systematic way to define and propagate (transfer) changes. The main idea is to define the change of function f as *Derive* f, satisfying

$$f(x \oplus \Delta x) = f(x) \oplus (\textit{Derive } f)\ x\ \Delta x.$$

where $\oplus$ denotes an updating operation. It reads that the change over the input x by Δx results in the change over the result of $f(x)$ by (*Derive* f) x Δx. While change theory provides a general way to describe changes, the changes it described are differences (deltas) instead of derivatives. It is worth noting that derivative is not the same as delta. For example, by change theory, we can deduce that $f(x)$ will be of the form of $x * x + C$ if we know (*Derive* f) x $\Delta x = 2 * x * \Delta x + \Delta x * \Delta x$, but we cannot deduce this form if we just know that its derivative is $2 * x$, because change theory has no concept of integration or limits.

Although a bunch of work has been done on derivatives [9, 8, 7, 4, 19, 16, 21, 10, 1], there is unfortunately, as far as we are aware, little work on integration. It may be natural to ask what a derivative really means if we cannot integrate it. If there is only a mapping from a term to its derivative without its corresponding integration, how can we operate on derivatives with a clear understanding of what we actually have done?

In this paper[1], we aim at a new differential framework, having dual mapping between derivatives and integrations. With this framework, we can manifest the power of this dual mapping by proving, among others, three important theorems, namely the Newton-Leibniz formula, the Chain Rule and the Taylor's theorem.

Our key idea can be illustrated by a simple example. Suppose we have a function f mapping from an n-dimensional space to an m-dimensional space. Then, let x be $(x_1, x_2, ..., x_n)^T$, and $f(x)$ be $(f_1(x), f_2(x), ..., f_m(x))^T$. Mathematically, we can use a Jacobian matrix A to represent its derivative, which satisfies the equation

$$f(x + \Delta x) - f(x) = A\Delta x + o(\Delta x),\ \text{where } A\ = \begin{pmatrix} \frac{\partial f_1}{\partial x_1} & \frac{\partial f_1}{\partial x_2} & \cdots & \frac{\partial f_1}{\partial x_n} \\ \frac{\partial f_2}{\partial x_1} & \frac{\partial f_2}{\partial x_2} & \cdots & \frac{\partial f_2}{\partial x_n} \\ \cdots & \cdots & \cdots & \cdots \\ \frac{\partial f_m}{\partial x_1} & \frac{\partial f_m}{\partial x_2} & \cdots & \frac{\partial f_m}{\partial x_n} \end{pmatrix}$$

However, computer programs usually describe computation over data of some structure, rather than just scalar data or matrix. In this paper, we extend the idea and propose a new calculus that enables us to perform differentiation and integration on data structures. Our main contributions are summarized as follows.

- To our knowledge, we have made the first attempt of designing a calculus that provides both derivative and integral. It is an extension of the lambda-calculus with five new operators including derivatives and integrations. We give clear semantics and typing rules, and prove that it is sound and strongly normalizing. (Section 2)
- We prove three important theorems and highlight their practical application for incremental computation, automatic differentiation, and computation approximation.
  - We prove the Newton-Leibniz formula: $\int_{t_1}^{t_2} \frac{\partial t}{\partial y}|_x dx = t[t_2/y] \ominus t[t_1/y]$, which is also known as Second Fundamental Theorem of Calculus. It shows the duality between derivatives and integrations, and can be used for incremental computation. (Section 3)
  - We prove the Chain Rule: $\frac{\partial f(g\ x)}{\partial x}|_{t_1} * t = \frac{\partial f\ y}{\partial y}|_{g\ t_1} * (\frac{\partial g\ z}{\partial z}|_{t_1} * t)$. It says $\forall x, \forall x_0, (f(g(x)))' * x_0 = f'(g(x)) * g'(x) * x_0$, and can be used for incremental computation and automatic differentiation. (Section 4)

[1] A full version of this paper is available at `https://arxiv.org/abs/2105.02632`.

Terms	t	::=	c	constants of interpretable type
		|	x	variable
		|	$\lambda x : T. t$	lambda abstraction
		|	$t\ t$	function application
		|	$(t_1, t_2, \ldots, t_n) \mid \pi_j\ t$	n-tuple and projection
		|	$t \oplus t$	addition
		|	$t \ominus t$	subtraction
		|	$t * t$	multiplication
		|	$\frac{\partial t}{\partial x}\|_t$	derivative
		|	$\int_t^t t\ dx$	integration
		|	$inl\ t \mid inr\ t$	left/right injection
		|	$case\ t\ of\ inl\ x_1 \Rightarrow t \mid inr\ x_2 \Rightarrow t$	case analysis
		|	$fix\ t$	fix point
Types	T	::=	B	base type
		|	$(T_1, T_2, \ldots, T_n)$	product type
		|	$T \to T$	function type
		|	$T + T$	sum type
Contexts	Γ	::=	$\emptyset$	empty context
		|	$\Gamma, x : T$	variable binding

Figure 1 Calculus Syntax.

- We prove the Taylor's Theorem: $f\ t = \sum_{k=0}^{\infty} \frac{1}{k!}(f^{(k)}\ t_0) * (t \ominus t_0)^k$. Different from that one of the differential lambda-calculus [9], this Taylor's theorem manifests results of computation instead of analysis on occurrence of terms. It can be used for approximation of a function computation. (Section 5)

2 Calculus

In this section, we shall give a clear definition of our calculus with both derivatives and integration. We explain important insights in our design, and prove some useful properties and theorems that will be used later.

2.1 Syntax

Our calculus, as defined in Figure 1, is an extension of the simply-typed lambda calculus [20]. Besides the usual constant, variable, lambda abstraction, function application, and tuple, it introduces five new operations: addition $\oplus$, subtraction $\ominus$, multiplication $*$, derivative $\frac{\partial t}{\partial x}|_t$ and integration $\int_t^t t\ dx$. The three binary operations, namely $\oplus$, $\ominus$, and $*$, are generalizations of those from our mathematics. Intuitively, $x \oplus \Delta$ is for updating x with change Δ, $\ominus$ for canceling updates, and * for distributing updates. We build up terms from terms of base types (such as $\mathbb{R}$, $\mathbb{C}$), and on each base type we require these operations satisfy the following properties:

$$\frac{c : T \in \Gamma}{\Gamma \vdash c : T}\ (\textsc{TCon}) \qquad \frac{x : T \in \Gamma}{\Gamma \vdash x : T}\ (\textsc{TVar})$$

$$\frac{\Gamma \vdash t : T_1}{\Gamma \vdash inl\ t : T_1 + T_2}\ (\textsc{TInl}) \qquad \frac{\Gamma \vdash t : T_2}{\Gamma \vdash inr\ t : T_1 + T_2}\ (\textsc{TInr})$$

$$\frac{\Gamma \vdash t_1 : T^* \quad \Gamma \vdash t_2 : T^*}{\Gamma \vdash t_1 \oplus t_2 : T^*}\ (\textsc{TAdd}) \qquad \frac{\Gamma \vdash t_1 : T^* \quad \Gamma \vdash t_2 : T^*}{\Gamma \vdash t_1 \ominus t_2 : T^*}\ (\textsc{TSub})$$

$$\frac{\Gamma, x : T_1 \vdash t : T_2}{\Gamma \vdash \lambda x : T_1 . t : T_1 \to T_2}\ (\textsc{TAbs}) \qquad \frac{\Gamma \vdash t_1 : T_1 \to T_2 \quad \Gamma \vdash t_2 : T_1}{\Gamma \vdash t_1\ t_2 : T_2}\ (\textsc{TApp})$$

$$\frac{\Gamma \vdash t : T \to T}{\Gamma \vdash fix\ t : T}\ (\textsc{TFix}) \qquad \frac{\Gamma \vdash t_1 : T_1 \quad \Gamma, x : T_1 \vdash t_2 : T_2}{\Gamma \vdash \frac{\partial t_2}{\partial x}|_{t_1} : \frac{\partial T_2}{\partial T_1}}\ (\textsc{TDer})$$

$$\frac{\forall j \in [1,n], \Gamma \vdash t_j : T_j}{\Gamma \vdash (t_1, t_2, ..., t_n) : (T_1, T_2, ..., T_n)}\ (\textsc{TPair}) \qquad \frac{\forall j \in [1,n], \Gamma \vdash t : (T_1, T_2, ..., T_n)}{\Gamma \vdash \pi_j\ t : T_j}\ (\textsc{TProj})$$

$$\frac{\Gamma \vdash t_1 : \frac{\partial T^*}{\partial T} \quad \Gamma \vdash t_2 : T}{\Gamma \vdash t_1 * t_2 : T^*}\ (\textsc{TMul})$$

$$\frac{\Gamma \vdash t_1 : T \quad \Gamma \vdash t_2 : T \quad \Gamma, x : T \vdash t : \frac{\partial T^*}{\partial T}}{\Gamma \vdash \int_{t_1}^{t_2} t\ dx : T^*}\ (\textsc{TInt})$$

$$\frac{\Gamma, x_1 : T_1 \vdash t_1 : T \quad \Gamma, x_2 : T_2 \vdash t_2 : T \quad \Gamma \vdash t : T_1 + T_2}{\Gamma \vdash case\ t\ of\ inl\ x_1 \Rightarrow t_1 \mid inr\ x_2 \Rightarrow t_2 : T}\ (\textsc{TCase})$$

Figure 2 Typing Rules.

- The addition and multiplication are associative and commutative, i.e., $(a \oplus b) \oplus c = a \oplus (b \oplus c)$, $a \oplus b = b \oplus a$, $(a * b) * c = a * (b * c)$, $a * b = b * a$.
- The addition and the subtraction are cancellable, i.e., $(a \oplus b) \ominus b = a$ and $(a \ominus b) \oplus b = a$.
- The multiplication is distributive over addition, i.e., $a * (b \oplus c) = a * b \oplus a * c$.

▶ **Example 1** (Basic Operations on Real Numbers). For real numbers $r_1, r_2 \in \mathbb{R}$, we have the following definitions.

$$\begin{aligned} r_1 \oplus r_2 &= r_1 + r_2 \\ r_1 \ominus r_2 &= r_1 - r_2 \\ r_1 * r_2 &= r_1\ r_2 \end{aligned}$$

We use $\frac{\partial t_1}{\partial x}|_{t_2}$ to denote derivative of t_1 over x at point t_2, and $\int_{t_1}^{t_2} t\ dx$ to denote integration of t over x from t_1 to t_2.

2.2 Typing

As defined in Figure 1, we have base types (denoted by B), tuple types, function types, and sum type. To make our later typing rules easy to understand, we introduce the following type notations.

$$\begin{array}{llll} \text{Type} & T^* ::= & B & \text{base type} \\ & \mid & (T^*, T^*, ..., T^*) & \text{product type} \\ & \mid & T \to T^* & \text{arrow type} \end{array}$$

T^* means the types that are addable (i.e., updatable through $\oplus$). We view the addition between functions, tuples and base type terms as valid, which will be showed by our reduction rules later. But here, we forbid the addition and subtraction between sum types because we view updates such as $inl\ 0 \oplus inr\ 1$ as invalid. If we want to update the change to a term of sum types anyway, we may do case analysis such as $case\ t\ of\ inl\ x_1 \Rightarrow inl\ (x_1 \oplus ...) \mid inr\ x_2 \Rightarrow (x_2 \oplus ...)$.

Next, we introduce two notations for derivatives on types:

$$\frac{\partial T}{\partial \mathsf{B}} = T,$$

$$\frac{\partial T}{\partial (T_1, T_2, ..., T_n)} = (\frac{\partial T}{\partial T_1}, \frac{\partial T}{\partial T_2}, ..., \frac{\partial T}{\partial T_n}).$$

The first notation says that with the assumption that differences (subtraction) of values of base types are of base types, the derivative over base types has no effect on the result type. And, the second notation resembles partial differentiation. Note that we do not consider derivatives on functions because even for functions on real numbers, there is no good mathematical definition for them yet. Therefore, we do not have a type notation for $\frac{\partial T}{\partial (T_1 \to T_2)}$. Besides, because we forbid the addition and subtraction between the sum types, we will iew the differentiation of the sum types as invalid, so we do not have notations for $\frac{\partial T}{\partial (T_1 + T_2)}$ either.

Figure 2 shows the typing rules for the calculus. The typing rules for constant, variable, lambda abstraction, function application, tuple, and projection are nothing special. The typing rules for addition and subtraction are natural, but the rest three kinds of rules are more interesting. Rule TMul the typing rule for $t_1 * t_2$. If t_1 is a derivative of T_1 over T_2, and t_2 is of type T_2, then multiplication will produce a term of type T_1. This may be informally understood from our familiar equation $\frac{\triangle Y}{\triangle X} * \triangle X = \triangle Y$. Rule TDer shows introduction of the derivative type through a derivative operation, while Rule TInt cancellation of the derivative type through an integration operation.

2.3 Semantics

We will give a two-stage semantics for the calculus. At the first stage, we assume that all the constants (values and functions) over the base types are *interpretable* in the sense there is a default well-defined interpreter to evaluate them. At the second stage, the important part of this paper, we define a set of reduction rules and use the full reduction strategy to compute their normal form, which enjoys good properties of soundness, confluence, and strong normalization.

More specifically, after the full reduction of a term in our calculus, every subterm (now in a normal form of interpretable types) outside the lambda function body will be interpretable on base types, which will be proved in the full version. In other words, our calculus helps to reduce a term to a normal form which is interpretable on base types, and leave the remaining evaluations to interpretation on base types. We will not give reduction rules to the operations on base types because we do not want to touch on implementations of primitive functions on base types.

For simplicity, in this paper we will assume that the important properties such as the Newton-Leibniz formula, the Chain Rule, and the Taylor's theorem, are satisfied by all the primitive functions and their closures through addition, subtraction, multiplication, derivative and integration. This assumption may seem too strong, since not all primitive functions on base types meet this assumption. However, it would make sense to start with the primitive functions meeting these requirements to build our system, and extend it later with other primitive functions.

$$\frac{t_0 : \mathsf{B}}{\frac{\partial(t_1,t_2,...,t_n)}{\partial x}|_{t_0} \rightarrow (\frac{\partial t_1}{\partial x}|_{t_0}, \frac{\partial t_2}{\partial x}|_{t_0}, ..., \frac{\partial t_n}{\partial x}|_{t_0})} \quad \text{(EAppDer1)}$$

$$\frac{t_0 : \mathsf{B}}{\frac{\partial inl/inr\ t}{\partial x}|_{t_0} \rightarrow inl/inr\ \frac{\partial t}{\partial x}|_{t_0}} \quad \text{(EAppDer2)}$$

$$\frac{t_0 : \mathsf{B}}{\frac{\partial(\lambda y{:}T.t)}{\partial x}|_{t_0} \rightarrow \lambda y : T.\frac{\partial t}{\partial x}|_{t_0}} \quad \text{(EAppDer3)}$$

$$\frac{\forall i \in [1,n],\ t_{i*} = (t_1, t_2..., t_{i-1}, x_i, t_{i+1}..., t_n)}{\frac{\partial t}{\partial x}|_{(t_1,t_2,...,t_n)} \rightarrow (\frac{\partial t[t_{1*}/x]}{\partial x_1}|_{t_1}, \frac{\partial t[t_{2*}/x]}{\partial x_2}|_{t_2}, ..., \frac{\partial t[t_{n*}/x]}{\partial x_n}|_{t_n})} \quad \text{(EAppDer4)}$$

$$\frac{t_1, t_2 : \mathsf{B}}{\int_{t_1}^{t_2}(t_{11}, t_{12}, ...t_{1n})dx \rightarrow (\int_{t_1}^{t_2} t_{11}dx, \int_{t_1}^{t_2} t_{12}dx, ..., \int_{t_1}^{t_2} t_{1n}dx)} \quad \text{(EAppInt1)}$$

$$\frac{t_1, t_2 : \mathsf{B}}{\int_{t_1}^{t_2} inl/inr\ t\ dx \rightarrow inl/inr\ \int_{t_1}^{t_2} t\ dx} \quad \text{(EAppInt2)}$$

$$\frac{t_1, t_2 : \mathsf{B}}{\int_{t_1}^{t_2} \lambda y : T_2.tdx \rightarrow \lambda y : T_2.\int_{t_1}^{t_2} tdx} \quad \text{(EAppInt3)}$$

$$\frac{\forall i \in [1,n], t_{i*} = (t_{21}..., t_{2i-1}, x_i, t_{1i+1}..., t_{1n})}{\int_{(t_{11},t_{12},...t_{1n})}^{(t_{21},t_{22},...,t_{2n})} tdx \rightarrow \int_{t_{11}}^{t_{21}} \pi_1(t[t_{1*}/x])dx_1 \oplus ... \oplus \int_{t_{1n}}^{t_{2n}} \pi_n(t[t_{n*}/x])dx_n} \quad \text{(EAppInt4)}$$

Figure 3 Reduction Rules for Derivative and Integration.

2.4 Reduction Rules

Our calculus is an extension of simply-typed lambda-calculus. Our lambda abstraction and application are nothing different from the simply-typed lambda calculus, and we have the reduction rule:

$$(\lambda x : T.\, t)t_1 \rightarrow t[t_1/x].$$

We use an n-tuple to model structured data and projection π_j to extract j-th component from a tuple, and we have the following reduction rule:

$$\pi_j(t_1, t_2, ...t_n) \rightarrow t_j.$$

Similarly, we have reduction rules for the case analysis:

$$case\ (inl\ t)\ of\ inl\ x_1 \Rightarrow t_1 \mid inr\ x_2 \Rightarrow t_2 \rightarrow t_1[t/x_1]$$

$$case\ (inr\ t)\ of\ inl\ x_1 \Rightarrow t_1 \mid inr\ x_2 \Rightarrow t_2 \rightarrow t_2[t/x_2]$$

Besides, we introduce fix-point operator to deal with recursion:

$$fix\ f \rightarrow f\ (fix\ f)$$

It is worth noting that tuples, having a good correspondence in mathematics, should be understood as structured data instead of high-dimensional vectors because there are some operations that are different from those in mathematics. As will be seen later, there is difference between our multiplication and matrix multiplication, and derivative and integration on tuples of tuples has no correspondence to mathematical objects.

The core reduction rules in our calculus are summarized in Figure 3, which define three basic cases for both reducing derivative terms and integration terms. For derivative, we use $\frac{\partial t}{\partial x}|_{t_0}$ to denote the derivative of t over x at point t_0, and we have four reduction rules:

- Rule EAppDer1 is to distribute point $t_0 : \mathsf{B}$ into a tuple. This resembles the case in mathematics; if we have a function f defined by $f(x) = (f_1(x), f_2(x), \ldots, f_m(x))^T$, its derivative will be $(\frac{\partial f_1}{\partial x}, \frac{\partial f_2}{\partial x}, \ldots, \frac{\partial f_m}{\partial x})^T$. For example, if we have a function $f : \mathbb{R} \to (\mathbb{R}, \mathbb{R})$ defined by $f(x) = (x, x * x)$, then its derivative will be $(1, 2 * x)$.
- Rule EAppDer2 is similar to Rule EAppDer1.
- Rule EAppDer3 is to distribute point $t_0 : \mathsf{B}$ into a lambda abstraction. Again this is very natural in mathematics. For example, for function $f(x) = \lambda y : B.\, x * y$, then we would have its derivative on x as $\lambda y : B.y$.
- Rule EAppDer4 is to deal with partial differentiation, similar to the Jacobian matrix in mathematics (as shown in the introduction). For example, if we have a function that maps a pair (x, y) to $(x * x, x * y \oplus y)$, which may be written as $\lambda z : (\mathsf{B}, \mathsf{B}).\, (\pi_1 z * \pi_1 z, (\pi_1 z * \pi_2 z \oplus \pi_2 z))$ then we would have its derivative $\frac{\partial (f\ z)}{\partial z}|_{(x,y)}$ as $((2 * x, y), (0, x \oplus 1))$.

Similarly, we can define four reduction rules for integration. Rules EAppInt1,EAppInt2 and EAppInt3 are simple. Rule EAppInt4 is worth more explanation. It is designed to establish the Newton-Leibniz formula

$$\int_{t_1}^{t_2} \frac{\partial t}{\partial y}|_x dx = t[t_2/y] \ominus t[t_1/y]$$

when t_1 and t_2 are tuples:

$$\int_{(t_{11}, t_{12}, \ldots, t_{1n})}^{(t_{21}, t_{22}, \ldots, t_{2n})} \frac{\partial t}{\partial y}|_x dx = t[(t_{21}, t_{22}, ..., t_{2n})/y] \ominus t[(t_{11}, t_{12}, ..., t_{1n})/y].$$

So we design the rule to have

$$\int_{t_{1j}}^{t_{2j}} \frac{\partial t[(t_{21}, ..., t_{2(j-1)}, x'_j, t_{1(j+1)}, ..., t_{1n})/y]}{\partial x'_j}|_{x_j} dx_j = \int_{(t_{21}, \ldots, t_{2(j-1)}, t_{1j}, t_{1(j+1)}, \ldots, t_{1n})}^{(t_{21}, \ldots, t_{2(j-1)}, t_{2j}, t_{1(j+1)}, \ldots, t_{1n})} \frac{\partial t}{\partial y}|_x dx.$$

Notice that under our evaluation rules on derivative, $\pi_j(\frac{\partial t}{\partial x}|_{x=(x_1, x_2, \ldots, x_n)})$ will be equal to the derivative of t to its j-th parameter x_j, so the integration will lead us to the original t.

Finally, we discuss the reduction rules for the three new binary operations, as summarized in Figure 4. The addition $\oplus$ is introduced to support the reduction rule of integration. It is also useful in proving the theorem and constructing the formula. We can understand the two reduction rules for addition as the addition of high-dimension vectors and functions respectively. Similarly, we can have two reduction rules for subtraction $\ominus$. The operator $*$ was introduced as a powerful tool for constructing the Chain Rule and the Taylor's theorem. The first two reduction rules can be understood as multiplications of a scalar with a function and a high-dimension vector respectively, while the last one can be understood as the multiplication on matrix. For example, we have

$$((1, 4), (2, 5), (3, 6)) * (7, 8, 9) = (50, 122)$$

which corresponds to the following matrix multiplication.

$$\begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{pmatrix} \begin{pmatrix} 7 \\ 8 \\ 9 \end{pmatrix} = \begin{pmatrix} 50 \\ 122 \end{pmatrix}$$

$$(t_{11},...,t_{1n}) \oplus (t_{21},...,t_{2n}) \ \to \ (t_{11} \oplus t_{21},...,t_{1n} \oplus t_{2n}) \quad \text{(EAppAdd1)}$$

$$(\lambda x : T.\, t_1) \oplus (\lambda y : T.\, t_2) \ \to \ \lambda x : T.\, (t_1 \oplus t_2[y/x]) \quad \text{(EAppAdd2)}$$

$$(t_{11},...,t_{1n}) \ominus (t_{21},...,t_{2n}) \ \to \ (t_{11} \ominus t_{21},...,t_{1n} \ominus t_{2n}) \quad \text{(EAppSub1)}$$

$$(\lambda x : T.\, t_1) \ominus (\lambda y : T.\, t_2) \ \to \ \lambda x : T.\, (t_1 \ominus t_2[y/x]) \quad \text{(EAppSub2)}$$

$$\frac{t_0 : \mathsf{B}}{(t_1, t_2, ..., t_n) * t_0 \ \to \ (t_1 * t_0, t_2 * t_0, ..., t_n * t_0)} \quad \text{(EAppMul1)}$$

$$\frac{t_0 : \mathsf{B}}{(\lambda x : T.t) * t_0 \ \to \ \lambda x : T.(t * t_0)} \quad \text{(EAppMul2)}$$

$$\frac{t_0 : \mathsf{B}}{(inl/inr\ t) * t_0 \ \to \ inl/inr\ (t * t_0)} \quad \text{(EAppMul3)}$$

$$\frac{t_1 : (t_{11}, t_{12}, ...t_{1n}), t_2 : (t_{21}, t_{22}, ...t_{2n})}{t_1 * t_2 \ \to \ (t_{11} * t_{21}) \oplus (t_{12} * t_{22}) \oplus ... \oplus (t_{1n} * t_{2n})} \quad \text{(EAppMul4)}$$

Figure 4 Reduction Rules for Addition, Subtraction and Multiplication.

It is worth noting that while they are similar, $*$ is different from the matrix multiplication operation. For example, we cannot write x as an m-dimensional vector (or $m * 1$ matrix) in Taylor's theorem because no matrix A is well-performed under $A * x * x$, but we can write Taylor's Theorem easily under our framework. In the matrix representation, the number of rows of the first matrix and the number of columns of the second matrix must be equal so that we can perform multiplication on them. This means, we can only write case $m = 1$'s Taylor's theorem in matrices, while our version can perform for any tuples.

2.5 Properties

Next, we prove some properties of our calculus. The proof is rather routine with some small variations.

▶ **Lemma 2** (Properties)**.** This calculus has the properties of progress, preservation and confluence. Moreover, if a term t does not contain subterms $fix\ t'$, then t is strong normalizable.

Proof. Full proof is in the full version, which is adapted from the standard proof. ◀

2.6 Term Equality

We need to talk a bit more on equality because we do not consider reduction or calculation on primitive functions. This notion of equality has little to do with our evaluation but has a lot to do with the equality of primitive functions. Using this notion of equality, we can compute the result from completely different calculations. This will be used in our later proof of the three theorems.

Since we have proved the confluence property, we know that every term has at most one normal form after reduction. Thus, we can define our equality based on their normal forms; the equality between unnormalizable terms is undefined.

▶ **Definition 3** (Term Equality). An open term t_1 is said to be equal to a term t_2, if and only if for all free variables $x_1, x_2, ..., x_n$ in t_1 and t_2, for all closed and weak-normalizable term u_i whose type is the same as that of x_i, we have $t_1[u_1/x_1, ..., u_n/x_n] = t_2[u_1/x_1, ..., u_n/x_n]$.

A closed-term $t_1 = t_2$, if their normal forms n_1 and n_2 have the relation that $n_1 = n_2$, where a normal form n_1 is said to be equal to another normal form n_2, if they satisfy one of the following rules:

- (1) n_1 is a of type iB (Type iB is used to capture terms with base type constants and functions, being defined by $iB = iB \to iB \mid B$). A normal form of type iB is interpretable by the base type interpreter. Detailed proof is in the full version), then n_2 has to be of the same type, and under the base type interpretation, n_1 is equal to n_2;
- (2) n_1 is $(t_1, t_2, ..., t_n)$, then n_2 has to be $(t'_1, t'_2, ..., t'_n)$, and $\forall j \in [1, n], t_j$ is equal to t'_j;
- (3) n_1 is $\lambda x : T.t$, then n_2 has to be $\lambda y : T.t'$ (y can be x), and $n_1\ x$ is equal to $n_2\ x$.
- (4) n_1 is $inl\ t'_1$, then n_2 has to be $inl\ t'_2$, and t'_1 is equal to t'_2.
- (5) n_1 is $inr\ t'_1$, then n_2 has to be $inr\ t'_2$, and t'_1 is equal to t'_2.

▶ **Lemma 4.** The equality is reflexive, transitive and symmetric for weak-normalizable terms.

Proof. Based on the equality of terms of base types, we can prove it by induction. ◀

▶ **Lemma 5.** The equality is consistent, e.g., we can not prove equality between arbitrary two terms.

Proof. Notice that except for the equality introduced by the base type interpreter, other equality inferences all preserve the type. So for arbitrary t_1 of type (B, B) and t_2 of type B, we can not prove equality between them. ◀

Next we give some lemmas that will be used later in our proof. It is relatively unimportant to the mainline of our calculus, so we put their proofs in the full version.

▶ **Lemma 6.** If $t_1 \rho^* t'_1, t_2 \rho^* t'_2$, then $t_1[t_2/x]\rho^* t'_1[t'_2/x]$.

▶ **Lemma 7.** If $t_1 = t'_1$, $t_2 = t'_2$, then $t_1 \oplus t_2 = t'_1 \oplus t'_2$.

▶ **Lemma 8.** For a term t, for any subterm s, if the term s'=s, then t[s'/s]=t. (We only substitute the subterm s, but not other subterms same as s)

▶ **Lemma 9.** If $t_1 * (t_2 \oplus t_3)$ and $(t_1 * t_2) \oplus (t_1 * t_3)$ are weak-normalizable, then $t_1 * (t_2 \oplus t_3) = (t_1 * t_2) \oplus (t_1 * t_3)$.

▶ **Lemma 10.** If $(t_1 \ominus t_2) \oplus (t_2 \ominus t_3)$ and $t_1 \ominus t_3$ are weak-normalizable, then $(t_1 \ominus t_2) \oplus (t_2 \ominus t_3) = t_1 \ominus t_3$.

3 Newton-Leibniz's Formula

The first important theorem we will give is the Newton-Leibniz's formula, which ensures the duality between derivatives and integration. This theorem lays a solid basis for our calculus.

▶ **Theorem 11** (Newton-Leibniz). Let t contain no free occurrence of x, and both $\int_{t_1}^{t_2} \frac{\partial t}{\partial y}|_x dx$ and $t[t_2/y] \ominus t[t_1/y]$ are well-typed and weak-normalizable. Then we have

$$\int_{t_1}^{t_2} \frac{\partial t}{\partial y}|_x dx = t[t_2/y] \ominus t[t_1/y].$$

Proof. If t_1, t_2 or t is not closed, then we need to prove $\forall u_1, ..., u_n$, we have

$$(\int_{t_1}^{t_2} \frac{\partial t}{\partial y}|_x dx)[u_1/x_1, ..., u_n/x_n] = (t[t_2/y] \ominus t[t_1/y])[u_1/x_1, ..., u_n/x_n].$$

By freezing $u_1, ..., u_n$, we can apply the substitution $[u_1/x_1, ..., u_n/x_n]$ to make every term closed. So, for simplicity, we will assume t, t_1 and t_2 to be closed.

We prove this by induction on types.

- Case: t_1,t_2 and t are of base types. By the confluence lemma, we know there exists the normal form t', t'_1 and t'_2 of the term t, t_1 and t_2. Also, we know $\int_{t_1}^{t_2} \frac{\partial t}{\partial y}|_x dx = \int_{t'_1}^{t'_2} \frac{\partial t'}{\partial y}|_x dx$ and $t[t_2/y] \ominus t[t_1/y] = t'[t'_2/y] \ominus t'[t'_1/y]$. Since on base types we have $\int_{t'_1}^{t'_2} \frac{\partial t'}{\partial y}|_x dx = t'[t'_2/y] \ominus t'[t'_1/y]$, we have $\int_{t_1}^{t_2} \frac{\partial t}{\partial y}|_x dx = t[t_2/y] \ominus t[t_1/y]$.
- Case: t_1,t_2 are of base types, t is of type $(T_1, T_2, ..., T_n)$. By the confluence lemmas, there exist a normal form $(t'_{11}, t'_{12}, ..., t'_{1n})$ for t. Using Rules (EAppInt1) and (EAppDer1), we know

$$\begin{array}{rcl} \int_{t_1}^{t_2} \frac{\partial t}{\partial y}|_x dx & = & \int_{t_1}^{t_2} \frac{\partial (t'_{11}, t'_{12}, ..., t'_{1n})}{\partial y}|_x dx \\ & = & \int_{t_1}^{t_2} (\frac{\partial t'_{11}}{\partial y}|_x, \frac{\partial t'_{12}}{\partial y}|_x, ..., \frac{\partial t'_{1n}}{\partial y}|_x) dx \\ & = & (\int_{t_1}^{t_2} \frac{\partial t'_{11}}{\partial y}|_x dx, \int_{t_1}^{t_2} \frac{\partial t'_{12}}{\partial y}|_x dx, ..., \int_{t_1}^{t_2} \frac{\partial t'_{1n}}{\partial y}|_x dx) \end{array}$$

  On the other hand, we have

$$\begin{array}{l} t[t_2/y] \ominus t[t_1/y] \\ \quad = (t'_{11}[t_2/y], t'_{12}[t_2/y], ..., t'_{1n}[t_2/y]) \ominus (t'_{11}[t_1/y], t'_{12}[t_1/y], ..., t'_{1n}[t_1/y]) \\ \quad = (t'_{11}[t_2/y] \ominus t'_{11}[t_1/y], t'_{12}[t_2/y] \ominus t'_{12}[t_1/y], ..., t'_{1n}[t_2/y] \ominus t'_{1n}[t_1/y]) \end{array}$$

  By induction, we have $\forall j \in [1, n], \int_{t_1}^{t_2} \frac{\partial t'_{1j}}{\partial y}|_x dx = t'_{1j}[t_2/y] \ominus t'_{1j}[t_1/y]$, so we have proven the case.
- Case: t_1,t_2 are of base types, t is of type $A \to B$. By Lemma 8, we can use $\lambda z : A.t\ z$ (for simiplicity, we use $\lambda z : A.t'$ where $t' = t\ z$) to substitute for t, where z is a fresh variable. Now, we have for any u,

$$\begin{array}{rcl} (\int_{t_1}^{t_2} \frac{\partial t}{\partial y}|_x dx)\ u & = & (\int_{t_1}^{t_2} \frac{\partial \lambda z:A.t'}{\partial y}|_x dx)\ u \\ & = & \lambda z : A.(\int_{t_1}^{t_2} \frac{\partial t'}{\partial y}|_x dx)\ u \\ & = & \int_{t_1}^{t_2} \frac{\partial t'[u/z]}{\partial y}|_x dx \end{array}$$

  and on the other hand, since z is free in t_1 and t_2, we have

$$\begin{array}{rcl} (t[t_2/y] \ominus t[t_1/y])\ u & = & ((\lambda z : A.t')[t_1/y] \ominus (\lambda z : A.t')[t_2/y])\ u \\ & = & \lambda z : A.(t'[t_2/y] \ominus t'[t_1/y])\ u \\ & = & (t'[t_2/y] \ominus t'[t_1/y])[u/z] \\ & = & (t'[u/z])[t_2/y] \ominus (t'[u/z])[t_1/y] \end{array}$$

  By induction (on B), we know $\int_{t_1}^{t_2} \frac{\partial t'[u/z]}{\partial y}|_x dx = (t'[u/z])[t_2/y] \ominus (t'[u/z])[t_1/y]$, thus we have proven the case.
- Case: t_1,t_2 are of base types, t is of type $T_1 + T_2$. This case is impossible because the righthand term is not well-typed.

- Case: t_1,t_2 are of type $(T_1, T_2, ..., T_n)$, t is of any type T. By using the confluence lemma, we know there exist the normal forms $(t'_{11}, t'_{12}, ..., t'_{1n})$ and $(t'_{21}, t'_{22}, ..., t'_{2n})$ for t_1 and t_2 respectively.
  Applying Rules (EAppDer3) and (EAppInt3), we have

$$\begin{aligned}\int_{t_1}^{t_2} \tfrac{\partial t}{\partial y}|_x dx &= \int_{(t'_{11},t'_{12},...,t'_{1n})}^{(t'_{21},t'_{22},...,t'_{2n})} \tfrac{\partial t}{\partial y}|_x dx \\ &= \int_{t'_{11}}^{t'_{21}} \pi_1(\tfrac{\partial t}{\partial y}|_x[(x_1, t'_{12}, ...t'_{1n})/x])dx_1 \oplus \cdots \oplus \\ &\quad \int_{t'_{1n}}^{t'_{2n}} \pi_n(\tfrac{\partial t}{\partial y}|_x[(t'_{21}, t'_{22}, ..., x_n)/x])dx_n\end{aligned}$$

  Notice that there is no occurrence of x in t, so we have

$$\begin{aligned}&\int_{t'_{1j}}^{t'_{2j}} \pi_j(\tfrac{\partial t}{\partial y}|_x[(t'_{21}, t'_{22}, ..., t'_{2(j-1)}, x_j, t'_{1(j+1)}, ..., t'_{1n})/x])dx_j \\ &= \int_{t'_{1j}}^{t'_{2j}} \pi_j(\tfrac{\partial t}{\partial y}|_{(t'_{21},t'_{22},...,t'_{2(j-1)},x_j,t'_{1(j+1)},...,t'_{1n})})dx_j \\ &= \int_{t'_{1j}}^{t'_{2j}} \pi_j(\tfrac{\partial t[t_{1*}/y]}{\partial x_1}|_{t'_{21}}, \tfrac{\partial t[t_{2*}/y]}{\partial x_2}|_{t'_{22}}, ..., \tfrac{\partial t[t_{(j-1)*}/y]}{\partial x_{j-1}}|_{t'_{2(j-1)}}, \\ &\qquad \tfrac{\partial t[t_{j*}/y]}{\partial x'_j}|_{x_j}, \tfrac{\partial t[t_{(j+1)*}/y]}{\partial x_{j+1}}|_{t'_{1(j+1)}}, ..., \tfrac{\partial t[t_{n*}/y]}{\partial x_n}|_{t'_{1n}})dx_j \\ &= \int_{t'_{1j}}^{t'_{2j}} \tfrac{\partial t[(t'_{21},t'_{22},...,t'_{2(j-1)},x'_j,t'_{1(j+1)},...,t'_{1n})/y]}{\partial x'_j}|_{x_j} dx_j\end{aligned}$$

  By induction (on the case where t_1, t_2 are of type T_j, t is of type T), we have

$$\begin{aligned}&\int_{t'_{1j}}^{t'_{2j}} \tfrac{\partial t[(t'_{21},t'_{22},...,t'_{2(j-1)},x'_j,t'_{1(j+1)},...,t'_{1n})/y]}{\partial x'_j}|_{x_j} dx_j \\ &= (t[(t'_{21}, t'_{22}, ..., t'_{2(j-1)}, x'_j, t'_{1(j+1)}, ..., t'_{1n})/y])[t'_{2j}/x'_j] \ominus \\ &\qquad (t[(t'_{21}, t'_{22}, ..., t'_{2(j-1)}, x'_j, t'_{1(j+1)}, ..., t'_{1n})/y])[t'_{1j}/x'_j] \\ &= (t[(t'_{21}, t'_{22}, ..., t'_{2(j-1)}, t'_{2j}, t'_{1(j+1)}, ..., t'_{1n})/y]) \ominus \\ &\qquad (t[(t'_{21}, t'_{22}, ..., t'_{2(j-1)}, t'_{1j}, t'_{1(j+1)}, ..., t'_{1n})/y])\end{aligned}$$

  Note that the last equation holds because x'_j is a fresh variable and t has no occurrence of x'_j.
  Now we have the following calculation.

$$\begin{aligned}&\int_{t_1}^{t_2} \tfrac{\partial t}{\partial y}|_x dx \\ =\ & \{ \text{ all the above } \} \\ & ((t[(t'_{21}, t'_{12}, ..., t'_{1n})/y]) \ominus (t[(t'_{11}, t'_{12}, ..., t'_{1n})/y])) \oplus \\ & \quad ((t[(t'_{21}, t'_{22}, ..., t'_{1n})/y]) \ominus (t[(t'_{21}, t'_{12}, ..., t'_{1n})/y])) \oplus \cdots \oplus \\ & \quad\quad ((t[(t'_{21}, t'_{22}, ..., t'_{2n})/y]) \ominus (t[(t'_{21}, t'_{22}, ..., t'_{1n})/y])) \\ =\ & \{ \text{ Lemma 10 } \} \\ & (t[(t'_{21}, t'_{22}, ..., t'_{2n})/y]) \ominus (t[(t'_{11}, t'_{12}, ..., t'_{1n})/y]) \\ =\ & \{ \text{ Lemma 6 } \} \\ & t[t_2/y] \ominus t[t_1/y]\end{aligned}$$

  Thus we have proven the theorem. ◀

Application: Incremental Computation

A direct application is incrementalization [17, 7, 11]. Given a function $f(x)$, if the input x is changed by Δ, then we can obtain its incremental version of $f(x)$ by $f'(x, \Delta)$.

$$f(x \oplus \Delta) = f(x) \oplus f'(x, \Delta)$$

where f' satisfies that

$$f'(x,\Delta) = \int_x^{x\oplus\Delta} \frac{\partial f(x)}{\partial x}|_x \; dx.$$

▶ **Example 12** (Averaging a Pair of Real numbers)**.** As a simple example, consider the average of a pair of real numbers

$$\begin{array}{lcl} average & :: & (\mathbb{R},\mathbb{R}) \to \mathbb{R} \\ average & = & \lambda x.(\pi_1(x)+\pi_2(x))/2 \end{array}$$

Suppose that we want to get an incremental computation of *average* at $x = (x_1, x_2)$ when the first element x_1 is changed to $x_1 + d$ while the second component x_2 is kept the same. The incremental computation is defined by

$$inc(x,d) = average(x,(d,0)) = \int_x^{x\oplus(d,0)} \frac{\partial average(x)}{\partial x}|_x \; dx = \frac{d}{2}$$

which is efficient.

4 Chain Rule

The Chain Rule is another important theorem of the relation between function composition and derivatives. This Chain Rule in our calculus has many important applications in automatic differentiation and incremental computation.

▶ **Theorem 13** (Chain Rule)**.** Let $f : T_1 \to T$, $g : T_2 \to T_1$. If both $\frac{\partial f(g\ x)}{\partial x}|_{t_1} * t$ and $\frac{\partial f\ y}{\partial y}|_{(g\ t_1)} * (\frac{\partial g\ z}{\partial z}|_{t_1} * t)$ are well-typed and weak-normalizable. Then for any $t, t_1 : T_2$, we have

$$\frac{\partial f(g\ x)}{\partial x}|_{t_1} * t = \frac{\partial f\ y}{\partial y}|_{(g\ t_1)} * (\frac{\partial g\ z}{\partial z}|_{t_1} * t).$$

Proof. Like in the proof of Theorem 11, for simplicity, we assume that f, g, t and t_1 are closed. Furthermore, we assume that t and t_1 are in normal form. We prove this by induction on types.

- Case T, T_2 are base types, and T_1 is any type. To be well-typed, T_1 must contain no $\to$ or $+$ type. So for simplicity, we suppose T_1 to be $(B, B, B, ..., B)$ of n-tuples, but the technique below can be applied to any T_1 type (such as tuples of tuples) that makes the term well-typed.
  First we notice that

$$\begin{array}{lcl} g\ z & = & (\pi_1(g\ z), \pi_2(g\ z), ..., \pi_n(g\ z)) \\ & = & ((\lambda b' : B.\pi_1(g\ b'))\ z, (\lambda b' : B.\pi_2(g\ b'))\ z, ..., (\lambda b' : B.\pi_n(g\ b'))\ z) \end{array}$$

  and for any j, we notice that $\pi_j(g\ b')$ has only one free variable of base type, so it can be reduced to a normal form, say E_j, of base type. Let g_j be $\lambda b' : B.E_j$, then we have $g\ z = (g_1\ z, g_2\ z, ..., g_n\ z)$.
  Next, we deal with the term f:

$$\begin{array}{lcl} f & = & \lambda a : T_1.(f\ a) \\ & = & \lambda a : T_1.((\lambda y_1 : B.\, \lambda y_2 : B., ... \lambda y_n : B.\, (f\ (y_1, y_2, ..., y_n)))\ \pi_1(a)\ \pi_2(a) ...\ \pi_n(a)) \end{array}$$

and we know that $(f\ (y_1, y_2, ..., y_n))$ only contains base type free variables, so it can be reduced to a base type normal form, say N, so we have

$$f = \lambda a : T.\ ((\lambda y_1 : B.\lambda y_2 : B., ...\lambda y_n : B.\ N)\ \pi_1(a)\ \pi_2(a)...\ \pi_n(a)).$$

Now, we can calculate as follows:

$$\begin{aligned}
&\tfrac{\partial f\ (g\ x)}{\partial x}|_{t_1} * t \\
&\quad = \tfrac{\partial (\lambda a{:}T.(\lambda y_1{:}B.\lambda y_2{:}B.,...\lambda y_n{:}B.N)\ \pi_1(a)\ \pi_2(a)...\ \pi_n(a))\ (g_1\ x, g_2\ x, ..., g_n\ x)}{\partial x}|_{t_1} * t \\
&\quad = \tfrac{\partial (\lambda y_1{:}B.\lambda y_2{:}B.,...\lambda y_n{:}B.N)\ (g_1\ x)\ (g_2\ x)...\ (g_n\ x)}{\partial x}|_{t_1} * t \\
&\quad = \tfrac{\partial N[(g_1\ x)/y_1, (g_2\ x)/y_2, ...(g_n\ x)/y_n]}{\partial x}|_{t_1} * t
\end{aligned}$$

$$\begin{aligned}
&\tfrac{\partial f\ y}{\partial y}|_{(g\ t_1)} * (\tfrac{\partial g\ z}{\partial z}|_{t_1} * t) \\
&\quad = \tfrac{\partial f\ y}{\partial y}|_{(g_1\ t_1, g_2\ t_1, ..., g_n\ t_1)} * (\tfrac{\partial (g_1\ z, g_2\ z, ..., g_n\ z)}{\partial z}|_{t_1} * t) \\
&\quad = \tfrac{\partial f\ y}{\partial y}|_{(g_1\ t_1, g_2\ t_1, ..., g_n\ t_1)} * (\tfrac{\partial g_1\ z}{\partial z}|_{t_1} * t, \tfrac{\partial g_2\ z}{\partial z}|_{t_1} * t, ..., \tfrac{\partial g_n\ z}{\partial z}|_{t_1} * t) \\
&\quad = \tfrac{\partial (\lambda y_1{:}B.\lambda y_2{:}B.,...\lambda y_n{:}B.N)\ \pi_1(y)\ \pi_2(y)...\ \pi_n(y)}{\partial y}|_{(g_1\ t_1, g_2\ t_1, ..., g_n\ t_1)} * \\
&\qquad\qquad (\tfrac{\partial g_1\ z}{\partial z}|_{t_1} * t, \tfrac{\partial g_2\ z}{\partial z}|_{t_1} * t, ..., \tfrac{\partial g_n\ z}{\partial z}|_{t_1} * t) \\
&\quad = (\tfrac{\partial N[y_1'/y_1, g_2\ t_1/y_2, ..., g_n\ t_1/y_n]}{\partial y_1'}|_{g_1\ t_1}, ..., \tfrac{\partial N[g_1\ t_1/y_1, g_2\ t_1/y_2, ..., y_n'/y_n]}{\partial y_n'}|_{g_n\ t_1}) * \\
&\qquad\qquad (\tfrac{\partial g_1\ z}{\partial z}|_{t_1} * t, \tfrac{\partial g_2\ z}{\partial z}|_{t_1} * t, ..., \tfrac{\partial g_n\ z}{\partial z}|_{t_1} * t) \\
&\quad = (\tfrac{\partial N[y_1'/y_1, g_2\ t_1/y_2, ..., g_n\ t_1/y_n]}{\partial y_1'}|_{g_1\ t_1} * (\tfrac{\partial g_1\ z}{\partial z}|_{t_1} * t)) \oplus ... \oplus \\
&\qquad\qquad (\tfrac{\partial N[g_1\ t_1/y_1, g_2\ t_1/y_2, ..., y_n'/y_n]}{\partial y_n'}|_{g_n\ t_1} * (\tfrac{\partial g_n\ z}{\partial z}|_{t_1} * t))
\end{aligned}$$

Notice that by the base type interpretation, $f(g_1(x), g_2(x), ..., g_n(x)) = f_1'(g_1(x), g_2(x), ..., g_n(x)) * g_1'(x) + f_2'(g_1(x), g_2(x), ..., g_n(x)) * g_2'(x) + ... + f_n'(g_1(x), g_2(x), ..., g_n(x)) * g_n'(x)$ where f_j' means the derivative of f to its j-th parameter, so we get the following and prove the case.

$$\begin{aligned}
&\tfrac{\partial N[(g_1\ x)/y_1, (g_2\ x)/y_2, ...(g_n\ x)/y_n]}{\partial x}|_{t_1} * t \\
&\quad = (\tfrac{\partial N[y_1'/y_1, g_2\ t_1/y_2, ..., g_n\ t_1/y_n]}{\partial y_1'}|_{g_1\ t_1} * (\tfrac{\partial g_1\ z}{\partial z}|_{t_1} * t)) \oplus ... \oplus \\
&\qquad (\tfrac{\partial N[g_1\ t_1/y_1, g_2\ t_1/y_2, ..., y_n'/y_n]}{\partial y_n'}|_{g_n\ t_1} * (\tfrac{\partial g_n\ z}{\partial z}|_{t_1} * t))
\end{aligned}$$

- Case T_2 is base type, T_1 is any type, T is $A \to B$. We prove that for any u of type A, we have $(\tfrac{\partial f\ (g\ x)}{\partial x}|_{t_1} * t)\ u = (\tfrac{\partial f\ y}{\partial y}|_{(g\ t_1)} * (\tfrac{\partial g\ z}{\partial z}|_{t_1} * t))\ u$.
  First, let $f' = \lambda x : T_1.(f\ x)\ u$, $g' = g$, then by induction we have

  $$\tfrac{\partial f'(g'\ x)}{\partial x}|_{t_1} * t = \tfrac{\partial f'\ y}{\partial y}|_{(g'\ t_1)} * (\tfrac{\partial g'\ z}{\partial z}|_{t_1} * t)$$

  that is, we have

  $$(\tfrac{\partial f\ (g\ x)\ u}{\partial x}|_{t_1} * t) = (\tfrac{\partial f\ y\ u}{\partial y}|_{(g\ t_1)} * (\tfrac{\partial g\ z}{\partial z}|_{t_1} * t))$$

  Then, we prove $(\tfrac{\partial f\ (g\ x)\ u}{\partial x}|_{t_1} * t) = (\tfrac{\partial f\ (g\ x)}{\partial x}|_{t_1} * t)\ u$ by the following calculation.

  $$\begin{aligned}
  (\tfrac{\partial f\ (g\ x)}{\partial x}|_{t_1} * t)\ u &= (\tfrac{\partial \lambda a{:}A.(f\ (g\ x))\ a}{\partial x}|_{t_1} * t)\ u \\
  &= (\lambda a : A.(\tfrac{\partial (f\ (g\ x))\ a}{\partial x}|_{t_1} * t))\ u \\
  &= (\tfrac{\partial f\ (g\ x)\ u}{\partial x}|_{t_1} * t)
  \end{aligned}$$

Next, we prove $(\frac{\partial f\ y}{\partial y}|_{(g\ t_1)} * (\frac{\partial g\ z}{\partial z}|_{t_1} * t))\ u = \frac{\partial f\ y\ u}{\partial y}|_{(g\ t_1)} * (\frac{\partial g\ z}{\partial z}|_{t_1} * t)$. For simplicity, we assume T_1 to be $(B, B, B, ..., B)$ of n-tuples (the technique below can be applied to any T_1 type which makes the term well-typed).
On one hand, by substituting $(g_1\ z, g_2\ z, ..., g_n\ z)$ for $g\ z$, we have

$$\begin{aligned}
&(\frac{\partial f\ y}{\partial y}|_{(g\ t_1)} * (\frac{\partial g\ z}{\partial z}|_{t_1} * t))\ u \\
&\quad = \frac{\partial f\ y}{\partial y}|_{(g_1\ t_1, g_2\ t_1, ..., g_n\ t_1)} * (\frac{\partial (g_1\ z, g_2\ z, ..., g_n\ z)}{\partial z}|_{t_1} * t)\ u \\
&\quad = (\frac{\partial f(y_1, g_2\ t_1, ..., g_n\ t_1)}{\partial y_1'}|_{g_1\ t_1} * (\frac{\partial g_1\ z}{\partial z}|_{t_1} * t) \oplus ... \oplus \\
&\qquad\qquad \frac{\partial f(g_1\ t_1, g_2\ t_1, ..., y_n)}{\partial y_n'}|_{g_n\ t_1} * (\frac{\partial g_n\ z}{\partial z}|_{t_1} * t))\ u
\end{aligned}$$

Since

$$\begin{aligned}
&f(g_1\ t_1, g_2\ t_1, ..., g_{j-1}\ t_1, y_j, g_{j+1}\ t_1, ..., g_n\ t_1) \\
&\quad = \lambda a : A. f\ (g_1\ t_1, g_2\ t_1, ..., g_{j-1}\ t_1, y_j, g_{j+1}\ t_1, ..., g_n\ t_1)\ a
\end{aligned}$$

which will be denoted as $\lambda a : A.t_j^*$, we continue the calculation as follows.

$$\begin{aligned}
&(\frac{\partial f\ y}{\partial y}|_{(g\ t_1)} * (\frac{\partial g\ z}{\partial z}|_{t_1} * t))\ u \\
&\quad = (\frac{\partial \lambda a : A.t_1^*}{\partial y_1}|_{g_1\ t_1} * (\frac{\partial g_1\ z}{\partial z}|_{t_1} * t) \oplus ... \oplus \frac{\partial \lambda a : A.t_n^*}{\partial y_n}|_{g_n\ t_1} * (\frac{\partial g_n\ z}{\partial z}|_{t_1} * t))\ u \\
&\quad = \lambda a : A.(\frac{\partial t_1^*}{\partial y_1}|_{g_1\ t_1} * (\frac{\partial g_1\ z}{\partial z}|_{t_1} * t)) \oplus ... \oplus (\frac{\partial t_n^*}{\partial y_n}|_{g_n\ t_1} * (\frac{\partial g_n\ z}{\partial z}|_{t_1} * t))\ u \\
&\quad = \frac{\partial t_1^*[u/a]}{\partial y_1}|_{g_1\ t_1} * (\frac{\partial g_1\ z}{\partial z}|_{t_1} * t) \oplus ... \oplus \frac{\partial t_n^*[u/a]}{\partial y_n}|_{g_n\ t_1} * (\frac{\partial g_n\ z}{\partial z}|_{t_1} * t)
\end{aligned}$$

On the other hand, we have

$$\begin{aligned}
&(\frac{\partial f\ y\ u}{\partial y}|_{(g\ t_1)} * (\frac{\partial g\ z}{\partial z}|_{t_1} * t)) \\
&\quad = \frac{\partial f\ y\ u}{\partial y}|_{(g_1\ t_1, g_2\ t_1, ..., g_n\ t_1)} * (\frac{\partial (g_1\ z, g_2\ z, ..., g_n\ z)}{\partial z}|_{t_1} * t) \\
&\quad = \frac{\partial f\ (y_1, g_2\ t_1, ..., g_n\ t_1)\ u}{\partial y_1}|_{g_1\ t_1} * (\frac{\partial g_1\ z}{\partial z}|_{t_1} * t) \oplus ... \oplus \\
&\qquad\qquad \frac{\partial f\ (g_1\ t_1, g_2\ t_1, ..., y_n)\ u}{\partial y_n}|_{g_n\ t_1} * (\frac{\partial g_n\ z}{\partial z}|_{t_1} * t) \\
&\quad = \frac{\partial (\lambda a : A.t_1^*)\ u}{\partial y_1}|_{g_1\ t_1} * (\frac{\partial g_1\ z}{\partial z}|_{t_1} * t) \oplus ... \oplus \frac{\partial (\lambda a : A.t_n^*)\ u}{\partial y_n}|_{g_n\ t_1} * (\frac{\partial g_n\ z}{\partial z}|_{t_1} * t) \\
&\quad = (\frac{\partial t_1^*[u/a]}{\partial y_1}|_{g_1\ t_1} * (\frac{\partial g_1\ z}{\partial z}|_{t_1} * t)) \oplus ... \oplus (\frac{\partial t_n^*[u/a]}{\partial y_n}|_{g_n\ t_1} * (\frac{\partial g_n\ z}{\partial z}|_{t_1} * t))
\end{aligned}$$

Therefore, we have proven the case.

- Case T_2 is base type, T_1 is any type, T is $(T_1, T_2, ..., T_n)$. We need to prove that for all j, we have $\pi_j(\frac{\partial f(g\ x)}{\partial x}|_{t_1} * t) = \pi_j(\frac{\partial f\ y}{\partial y}|_{(g\ t_1)} * (\frac{\partial g\ z}{\partial z}|_{t_1} * t))$. We may follow the proof for the case when T has type $A \to B$. Let $f' = \lambda x : T_1.\pi_j(f\ x), g' = g$, by induction, we have

$$\frac{\partial \pi_j(f(g\ x))}{\partial x}|_{t_1} * t = \frac{\partial \pi_j(f\ y)}{\partial y}|_{(g\ t_1)} * (\frac{\partial g\ z}{\partial z}|_{t_1} * t)$$

  The rest of the proof is similar to that for the case when $T = A \to B$.
- Case T_2 is base type, T_1 is any type, T is $T_1 + T_2$. Notice that T_1 has to be base type to be well-typed. But either the case, the proof is similar to the case when $T = A \to B$.
- Case T_2 , T_1 and T are any type. Notice that T_2 does not contain no $\to$ or $+$ to be well-typed (i.e., no derivative over function types). We have proved the case when T_2 is base type, and we assume that T_2 has type $(T_1, T_2, ..., T_n)$. Suppose the normal form of t_1 is $(t_{11}', t_{12}', ..., t_{1n}')$ and the normal form of t is $(t_{21}', t_{22}', ..., t_{2n}')$, Then

$$\begin{aligned}
&\frac{\partial f(g\ x)}{\partial x}|_{t_1} * t \\
&\quad = \frac{\partial f(g\ x)}{\partial x}|_{(t_{11}', t_{12}', ..., t_{1n}')} * (t_{21}', t_{22}', ..., t_{2n}') \\
&\quad = (\frac{\partial f(g\ (x_1, t_{12}', ..., t_{1n}'))}{\partial x_1}|_{t_{11}'}, ..., \frac{\partial f(g\ (t_{11}', t_{12}', ..., x_n))}{\partial x_n}|_{t_{1n}'}) * (t_{21}', ..., t_{2n}') \\
&\quad = (\frac{\partial f(g\ (x_1, t_{12}', ..., t_{1n}'))}{\partial x_1}|_{t_{11}'} * t_{21}') \oplus ... \oplus (\frac{\partial f(g\ (t_{11}', t_{12}', ..., x_n))}{\partial x_n}|_{t_{1n}'} * t_{2n}')
\end{aligned}$$

On the other hand, we can use Lemma 9 (i.e., $t_1 * (t_2 \oplus t_3) = (t_1 * t_2) \oplus (t_1 * t_3)$) to do the following calculation.

$$\begin{aligned}
&\frac{\partial f\ y}{\partial y}|_{(g\ t_1)} * (\frac{\partial g\ z}{\partial z}|_{t_1} * t) \\
&\quad = \frac{\partial f\ y}{\partial y}|_{(g\ t_1)} * ((\frac{\partial g\ (x_1, t'_{12}, \ldots, t'_{1n})}{\partial x_1}|_{t'_{11}} * t'_{21}) \oplus \ldots \oplus (\frac{\partial g\ (t'_{11}, t'_{12}, \ldots, x_n)}{\partial x_n}|_{t'_{1n}} * t'_{2n})) \\
&\quad = \frac{\partial f\ y}{\partial y}|_{(g\ t_1)} * (\frac{\partial g\ (x_1, t'_{12}, \ldots, t'_{1n})}{\partial x_1}|_{t'_{11}} * t'_{21}) \oplus \ldots \oplus \\
&\qquad\quad \frac{\partial f\ y}{\partial y}|_{(g\ t_1)} * (\frac{\partial g\ (t'_{11}, t'_{12}, \ldots, x_n)}{\partial x_n}|_{t'_{1n}} * t'_{2n})
\end{aligned}$$

Now by induction using $f' = f, g' = \lambda x : T_j.g\ (t'_{11}, t'_{12}, ..., t'_{1(j-1)}, x, t'_{1(j+1)}, ..., t'_{1n})$, we have

$$\begin{aligned}
&\frac{\partial f(g\ (t'_{11}, t'_{12}, \ldots, t'_{1(j-1)}, x_j, t'_{1(j+1)} \cdots, t'_{1n})}{\partial x_j}|_{t'_{1j}} * t'_{2j} \\
&\quad = \frac{\partial f\ y}{\partial y}|_{(g'\ t'_{1j})} * (\frac{\partial g\ (t'_{11}, t'_{12}, \ldots, t'_{1(j-1)}, x_j, t'_{1(j+1)} \cdots, t'_{1n})}{\partial x_j}|_{t'_{1j}} * t'_{2j}) \\
&\quad = \frac{\partial f\ y}{\partial y}|_{(g\ t_1)} * (\frac{\partial g\ (t'_{11}, t'_{12}, \ldots, t'_{1(j-1)}, x_j, t'_{1(j+1)} \cdots, t'_{1n})}{\partial x_j}|_{t'_{1j}} * t'_{2j})
\end{aligned}$$

Therefore by Lemma 7, we have proven the case.
Thus we have proven the theorem. ◀

Application: Automatic Differentiation

The Chain Rule provides another way to compute the derivatives. There are many applications of the chain rule, and here we give an example of how to associate it with the automatic differentiation [10].

▶ **Example 14** (AD). This is an example from [10]. Let *sqr* and *magSqr* be defined as follows.

$$\begin{aligned}
&sqr && :: && \mathbb{R} \to \mathbb{R} \\
&sqr\ a && = && a * a \\
&magSqr && :: && (\mathbb{R}, \mathbb{R}) \to \mathbb{R} \\
&magSqr\ (a, b) && = && sqr\ a \oplus sqr\ b
\end{aligned}$$

First of all, let t_1 and t_2 two pairs, then it is easy to prove that $\frac{\partial (t_1 \oplus t_2)}{\partial x}|_{t_3} = \frac{\partial t_1}{\partial x}|_{t_3} \oplus \frac{\partial t_2}{\partial x}|_{t_3}$. Next, we can perform automatic differentiation on *magSqr* by the following calculation.

$$\begin{aligned}
&\frac{\partial (magSqr\ x)}{\partial x}|_{(a,b)} * t \\
&\quad = \frac{\partial (sqr(\pi_1 x) \oplus sqr(\pi_2 x))}{\partial x}|_{(a,b)} * t \\
&\quad = \frac{\partial (sqr\ y)}{\partial y}|_{\pi_1(a,b)} * (\frac{\partial (\pi_1 x)}{\partial x}|_{(a,b)} * t) \oplus \frac{\partial (sqr\ y)}{\partial y}|_{\pi_2(a,b)} * (\frac{\partial (\pi_2 x)}{\partial x}|_{(a,b)} * t) \\
&\quad = 2 * a * ((1, 0) * t) \oplus 2 * b * ((0, 1) * t)
\end{aligned}$$

Now, because the theorem applies for any t of pair type, we use $(1, 0)$ and $(0, 1)$ to substitute for t respectively, and we will get $\frac{\partial (magSqr\ x)}{\partial x}|_{(a,b)} = (2 * a, 2 * b)$, which means its derivative to a is $2 * a$ and its derivative to b is $2 * b$.

5 Taylor's Theorem

In this section, we discuss Taylor's Theorem, which is useful to give an approximation of a k-order differentiable function around a given point by a polynomial of degree k. In programming, it is important and has many applications in approximation and incremental computation.

First of all, we introduce some high-order notations.

$$\begin{array}{lcllcl} \frac{\partial^0 t_1}{\partial x^0}|_{t_2} & = & t_1 & \frac{\partial^n t_1}{\partial x^n}|_{t_2} & = & \frac{\partial \frac{\partial^{n-1} t_1}{\partial x^{n-1}}|_x}{\partial x}|_{t_2} \\ t * t_1^0 & = & t & t * t_1^n & = & (t * t_1) * t_1^{n-1} \\ f^0 & = & f & f^n & = & (f')^{n-1} \\ (\lambda x : T.\, t)' & = & \lambda x : T.\, \frac{\partial t}{\partial x}|_x & & & \end{array}$$

Now our Taylor's Theorem can be expressed as follows.

▶ **Theorem 15** (Taylor's Theorem). If both $f\ t$ and $\sum_{k=0}^{\infty} \frac{1}{k!}(f^{(k)}\ t_0) * (t \ominus t_0)^k$ are weak-normalizable, then

$$f\ t = \sum_{k=0}^{\infty} \frac{1}{k!}(f^{(k)}\ t_0) * (t \ominus t_0)^k.$$

Proof. Like in the proof of Theorem 11, for simplicity, we assume that f, g, t and t_1 are closed. Furthermore, we assume that t and t_1 are in normal form. We prove it by induction on the type of $f : T \to T'$.

- Case T' is a base type. T must contain no $\to$ by our typing, so for simplicity, we suppose T to be $(B, B, ..., B)$. Using the same technique in Theorem 13, we assume f to be

$$f = \lambda x : T.\, (\lambda x_1 : B.\, \lambda x_2 : B., ...\lambda x_n : B.\, N)\ \pi_1(x)\ \pi_2(x)...\ \pi_n(x)$$

(denoted by $f = \lambda x : T.\, t_2$ later), t to be $(t_{11}, t_{12}, ..., t_{1n})$, and t_0 to be $(t_{21}, t_{22}, ..., t_{2n})$, where each t_{ij} is a normal form of base type. Then we have

$$\begin{aligned} &(f^{(n)}\ t_0) * (t \ominus t_0)^n \\ &\quad = \frac{\partial^n t_2}{\partial x^n}|_{t_0} * (t \ominus t_0)^n \\ &\quad = (\frac{\partial \frac{\partial^{n-1} t_2}{\partial x^{n-1}}|_{(x_1, t_{22}, ..., t_{2n})}}{\partial x_1}|_{t_{21}}, ..., \frac{\partial \frac{\partial^{n-1} t_2}{\partial x^{n-1}}|_{(t_{21}, t_{22}, ..., x_n)}}{\partial x_n}|_{t_{2n}}) * (t \ominus t_0)^n \\ &\quad = (\frac{\partial \frac{\partial^{n-1} t_2}{\partial x^{n-1}}|_{(x_1, t_{22}, ..., t_{2n})}}{\partial x_1}|_{t_{21}} * (t_{11} \ominus t_{21}) \oplus ... \oplus \frac{\partial \frac{\partial^{n-1} t_2}{\partial x^{n-1}}|_{(t_{21}, t_{22}, ..., x_n)}}{\partial x_n}|_{t_{2n}} \\ &\qquad\qquad * (t_{1n} \ominus t_{2n})) * (t \ominus t_0)^{n-1} \\ &\quad = ((\frac{\partial (\frac{\partial \frac{\partial^{n-2} t_2}{\partial x^{n-2}}|_{(x_1, t_{22}, ..., t_{2n})}}{\partial x_1}|_{x_1}, ..., \frac{\partial \frac{\partial^{n-2} t_2}{\partial x^{n-2}}|_{(x_1, t_{22}, ..., x_n)}}{\partial x_n}|_{t_{2n}})}{\partial x_1}|_{t_{21}}) * (t_{11} \ominus t_{21}) \oplus ... \oplus \\ &\qquad\qquad (\frac{\partial (\frac{\partial \frac{\partial^{n-2} t_2}{\partial x^{n-2}}|_{(x_1, t_{22}, ..., x_n)}}{\partial x_1}|_{t_{21}}, ..., \frac{\partial \frac{\partial^{n-2} t_2}{\partial x^{n-2}}|_{(t_{21}, t_{22}, ..., x_n)}}{\partial x_n}|_{x_n})}{\partial x_n}|_{t_{2n}}) \\ &\qquad\qquad * (t_{1n} \ominus t_{2n})) * (t \ominus t_0)^{n-1} \\ &\quad = ((\frac{\partial \frac{\partial \frac{\partial^{n-2} t_2}{\partial x^{n-2}}|_{(x_1, t_{22}, ..., t_{2n})}}{\partial x_1}|_{x_1}}{\partial x_1}|_{t_{21}}) * (t_{11} \ominus t_{21})^2 \oplus ... \oplus \\ &\qquad\qquad ((\frac{\partial \frac{\partial \frac{\partial^{n-2} t_2}{\partial x^{n-2}}|_{(x_1, t_{22}, ..., x_n)}}{\partial x_1}|_{t_{21}}}{\partial x_n}|_{t_{2n}}) * (t_{1n} \ominus t_{2n})) * (t_{11} \ominus t_{21}), \end{aligned}$$

$$
\begin{aligned}
&((\frac{\partial \frac{\partial^{n-2} t_2}{\partial x^{n-2}}|_{(x_1,x_2,\ldots,t_{2n})}}{\partial x_2}|_{t_{22}}}{\partial x_1}|_{t_{21}}) * (t_{11} \ominus t_{21})) * (t_{12} \ominus t_{22}) \oplus ... \oplus \\
&\qquad ((\frac{\partial \frac{\partial^{n-2} t_2}{\partial x^{n-2}}|_{(t_{11},x_2,\ldots,x_n)}}{\partial x_2}|_{t_{22}}}{\partial x_n}|_{t_{2n}}) * (t_{1n} \ominus t_{2n})) * (t_{12} \ominus t_{22}), \\
&... \\
&((\frac{\partial \frac{\partial^{n-2} t_2}{\partial x^{n-2}}|_{(x_1,t_{22},\ldots,x_n)}}{\partial x_n}|_{t_{2n}}}{\partial x_1}|_{t_{21}}) * (t_{11} \ominus t_{21})) * (t_{1n} \ominus t_{2n}) \oplus ... \oplus \\
&\qquad (\frac{\partial \frac{\partial^{n-2} t_2}{\partial x^{n-2}}|_{(t_{11},t_{22},\ldots,x_n)}}{\partial x_n}|_{x_n}}{\partial x_n}|_{t_{2n}}) * (t_{1n} \ominus t_{2n})^2) * (t \ominus t_0)^{n-2} \\
&= ...
\end{aligned}
$$

As seen in the above, every time we decompose a $\frac{\partial}{\partial x_i}|_{(...)}$, apply Rule EAppDer1, and then make reduction with Rule EAppMul3 to lower down the exponent of $(t \ominus t_0)^n$. Finally, we will decompose the last derivative and get the term t_2 in the form of $t_2[t'_{21}/x_1, t'_{22}/x_2, ..., t'_{2n}/x_n]$ where $\forall j \in [1, n], t'_{2j}$ is either t_{2j} or x_j.
Note that on base type we assume that we have Taylor's Theorem:

$$f(x_0 + h) = f(x_0) + \sum_{k=1}^{\infty} \frac{1}{k!} (\sum_{i=1}^{n} h_i \frac{\partial}{\partial x_i})^k f(x_0)$$

where x_0 and h is an n-dimensional vector, and x_j, h_j is its projection to its j-th dimension.
So we have $(f^{(k)}\ t_0) * (t \ominus t_0)^k$ corresponds to the k-th addend $\frac{1}{k!}(\sum_{i=1}^{n} h_i \frac{\partial}{\partial x_i})^k f(x_0)$.

- Case: T' is function type $A \to B$. Similar to the proof in Theorem 13, for all u of type A, we define $f^* = \lambda x : T.\, f\ x\ u$, and by using the inductive result on type B, we can prove the case simiarly as that in Theorem 13.
- Case: T' is a tuple type $(T_1, T_2, T_3, ...)$. Just define $f^* = \lambda x : T.\, \pi_j(f\ x)$ to use inductive result. The rest is simple.
- Case: T' is a tuple type $T_1 + T_2$. This case is impossible because the righthand is not well-typed.

Thus we have proven the theorem. ◀

Application: Polynomial Approximation

Taylor's Theorem has many applications. Here we give an example of using Taylor's Theorem for approximation. Suppose there is a point $(1, 0)$ in the polar coordinate system, and we want to know where the point will be if we slightly change the radius r and the angle θ. Since it is extremely costive to compute functions such as sin() and cos(), Taylor's Theorem enables us to make a fast polynomial approximation.

▶ **Example 16.** Let function *polar2catesian* be defined by

$$
\begin{array}{lcl}
\mathit{polar2cartesian} & :: & (\mathbb{R}, \mathbb{R}) \to (\mathbb{R}, \mathbb{R}) \\
\mathit{polar2cartesian}(r, \theta) & = & (r * \cos(\theta), r * \sin(\theta))
\end{array}
$$

We show how to expand $\mathit{polar2cartesian}(r, \theta)$ at $(1, 0)$ up to 2nd-order derivative. Since

$$\begin{aligned}\frac{\partial(\mathit{polar2cartesian}(x))}{\partial x}|_{(1,0)} &= \frac{\partial(\pi_1 x * \mathit{cos}(\pi_2 x), \pi_1 x * \mathit{sin}(\pi_2 x)}{\partial x}|_{(1,0)} \\ &= (\frac{\partial(x_1 * \cos(0), x_1 * \sin(0))}{\partial x_1}|_1, \frac{\partial(1 * \cos(x_2), 1 * \sin(x_2))}{\partial x_2}|_0) \\ &= ((1, 0), (0, 1))\end{aligned}$$

we have

$$\frac{\partial(\mathit{polar2cartesian}(x))}{\partial x}|_{(1,0)} * (\Delta r, \Delta\theta) = (\Delta r, \Delta\theta).$$

Again, we have

$$\begin{aligned}\frac{1}{2}\frac{\partial^2(\mathit{polar2cartesian}(x))}{\partial x^2}|_{(1,0)} * (\Delta r, \Delta\theta)^2 &= (((0, 0), (0, 1)), ((0, 1), (-1, 0))) * (\Delta r, \Delta\theta)^2 \\ &= (-\frac{1}{2}\Delta\theta^2, \Delta r * \Delta\theta).\end{aligned}$$

Combining the above, we can use $(1 \oplus \Delta r \ominus \frac{1}{2}\Delta\theta^2, \Delta\theta \oplus \Delta r * \Delta\theta)$ to make an approximation to $\mathit{polar2cartesian}(1 + \Delta r, \Delta\theta)$.

6 Related Work

Differential Calculus and The Change Theory. The differential lambda-calculus [9, 8] has been studied for computing derivatives of arbitrary higher-order programs. In the differential lambda-calculus, derivatives are guaranteed to be linear in its argument, where the incremental lambda-calculus does not have this restriction. Instead, it requires that the function should be differentiable. The big difference between our calculus and differential lambda calculus is that we perform computation on terms instead of analysis on terms.

The idea of performing incremental computation using derivatives has been studied by Cai et al. [7], who give an account using change structures. They use this to provide a framework for incrementally evaluating lambda calculus programs. It is shown that the work can be enriched with recursion and fix-point computation [4]. The main difference between our work and change theory is that we describe changes as mathematical derivatives while the change theory describe changes as (discrete) deltas.

Incremental/Self-Adaptive Computation. Paige and Koenig [19] present derivatives for a first-order language with a fixed set of primitives for incremental computation. Blakeley et al. [16] apply these ideas to a class of relational queries. Koch [14] guarantees asymptotic speedups with a compositional query transformation and delivers huge speedups in realistic benchmarks, though still for a first-order database language. We have proved Taylor's theorem in our framework, which provides us with another way to perform finite difference on the computation.

Self-adjusting computation [2] or adaptive function programming [3] provides a dynamic approach to incrementalization. In this approach, programs execute on the original input in an enhanced runtime environment that tracks the dependencies between values in a dynamic dependence graph; intermediate results are memoized. Later, changes to the input propagate through dependency graphs from changed inputs to results, updating both intermediate and final results; this processing is often more efficient than recomputation. Mathematically, self-adjusting computations corresponds to differential equations (The derivative of a function can be represented by the computational result of function), which may be a future work of our calculus.

Automatic Differentiation. Automatic differentiation [12] is a technique that allows for efficiently computing the derivative of arbitrary programs, and can be applied to probabilistic modeling [15] and machine learning [6]. This technique has been successfully applied to some higher-order languages [21, 10]. As pointed out in [4], while some approaches have been suggested [18, 13], a general theoretical framework for this technique is still a matter of open research. We prove the chain rule inside our framework, which lays a foundation for our calculus to perform automatic differentiation. And with more theorems in our calculus, we expect more profound applications in differential calculus.

7 Conclusion

In this paper, we propose an analytical differential calculus which is equipped with integration. This calculus, as far as we are aware, is the first one that has well-defined integration, which has not appeared in both differential lambda calculus and the change theory. Our calculus enjoys many nice properties such as soundness and strong normalizing, and has three important theorems, which have profound applications in computer science. Also, our calculus is highly extendable, it would be easy for users to add new features and prove more theorems inside our calculus.

References

1 Martín Abadi and Gordon D. Plotkin. A simple differentiable programming language. *Proceedings of the ACM on Programming Languages*, 4(POPL):1–28, January 2020. doi:10.1145/3371106.

2 Umut A. Acar, Amal Ahmed, and Matthias Blume. Imperative self-adjusting computation. In George C. Necula and Philip Wadler, editors, *Proceedings of the 35th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2008, San Francisco, California, USA, January 7-12, 2008*, pages 309–322. ACM, 2008.

3 Umut A. Acar, Guy E. Blelloch, and Robert Harper. Adaptive functional programming. In John Launchbury and John C. Mitchell, editors, *Conference Record of POPL 2002: The 29th SIGPLAN-SIGACT Symposium on Principles of Programming Languages, Portland, OR, USA, January 16-18, 2002*, pages 247–259. ACM, 2002.

4 Mario Alvarez-Picallo, Alex Eyers-Taylor, Michael Peyton Jones, and C.-H. Luke Ong. Fixing incremental computation - derivatives of fixpoints, and the recursive semantics of datalog. In Luís Caires, editor, *Programming Languages and Systems - 28th European Symposium on Programming, ESOP 2019, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2019, Prague, Czech Republic, April 6-11, 2019, Proceedings*, volume 11423 of *Lecture Notes in Computer Science*, pages 525–552. Springer, 2019.

5 Mario Alvarez-Picallo and C. H. Luke Ong. The difference lambda-calculus: A language for difference categories, 2020. arXiv:2011.14476.

6 Atilim Gunes Baydin, Barak A. Pearlmutter, Alexey Andreyevich Radul, and Jeffrey Mark Siskind. Automatic differentiation in machine learning: a survey. *J. Mach. Learn. Res.*, 18:153:1–153:43, 2017.

7 Yufei Cai, Paolo G. Giarrusso, Tillmann Rendel, and Klaus Ostermann. A theory of changes for higher-order languages: incrementalizing λ-calculi by static differentiation. In Michael F. P. O'Boyle and Keshav Pingali, editors, *ACM SIGPLAN Conference on Programming Language Design and Implementation, PLDI '14, Edinburgh, United Kingdom - June 09 - 11, 2014*, pages 145–155. ACM, 2014.

8 Thomas Ehrhard. An introduction to differential linear logic: proof-nets, models and antiderivatives. *Math. Struct. Comput. Sci.*, 28(7):995–1060, 2018.

9 Thomas Ehrhard and Laurent Regnier. The differential lambda-calculus. *Theor. Comput. Sci.*, 309(1-3):1–41, 2003.

10 Conal Elliott. The simple essence of automatic differentiation. *Proc. ACM Program. Lang.*, 2(ICFP):70:1–70:29, 2018.

11 Paolo G. Giarrusso, Yann Régis-Gianas, and Philipp Schuster. Incremental \lambda -calculus in cache-transfer style - static memoization by program transformation. In Luís Caires, editor, *Programming Languages and Systems - 28th European Symposium on Programming, ESOP 2019, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2019, Prague, Czech Republic, April 6-11, 2019, Proceedings*, volume 11423 of *Lecture Notes in Computer Science*, pages 553–580. Springer, 2019.

12 Andreas Griewank and Andrea Walther. *Evaluating derivatives - principles and techniques of algorithmic differentiation, Second Edition.* SIAM, 2008.

13 Robert Kelly, Barak A. Pearlmutter, and Jeffrey Mark Siskind. Evolving the incremental λ calculus into a model of forward automatic differentiation (AD). *CoRR*, abs/1611.03429, 2016. `arXiv:1611.03429`.

14 Christoph Koch. Incremental query evaluation in a ring of databases. In Jan Paredaens and Dirk Van Gucht, editors, *Proceedings of the Twenty-Ninth ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems, PODS 2010, June 6-11, 2010, Indianapolis, Indiana, USA*, pages 87–98. ACM, 2010.

15 Alp Kucukelbir, Dustin Tran, Rajesh Ranganath, Andrew Gelman, and David M. Blei. Automatic differentiation variational inference. *J. Mach. Learn. Res.*, 18:14:1–14:45, 2017.

16 Per-Åke Larson and Jingren Zhou. Efficient maintenance of materialized outer-join views. In Rada Chirkova, Asuman Dogac, M. Tamer Özsu, and Timos K. Sellis, editors, *Proceedings of the 23rd International Conference on Data Engineering, ICDE 2007, The Marmara Hotel, Istanbul, Turkey, April 15-20, 2007*, pages 56–65. IEEE Computer Society, 2007.

17 Yanhong A. Liu. Efficiency by incrementalization: An introduction. *High. Order Symb. Comput.*, 13(4):289–313, 2000. `doi:10.1023/A:1026547031739`.

18 Oleksandr Manzyuk. A simply typed λ-calculus of forward automatic differentiation. *Electronic Notes in Theoretical Computer Science*, 286:257–272, 2012. Proceedings of the 28th Conference on the Mathematical Foundations of Programming Semantics (MFPS XXVIII).

19 Robert Paige and Shaye Koenig. Finite differencing of computable expressions. *ACM Trans. Program. Lang. Syst.*, 4(3):402–454, 1982.

20 Benjamin C. Pierce. *Types and Programming Languages.* The MIT Press, 2002.

21 Jeffrey Mark Siskind and Barak A. Pearlmutter. Nesting forward-mode AD in a functional framework. *High. Order Symb. Comput.*, 21(4):361–376, 2008.